Projekt Auftragsverwaltung

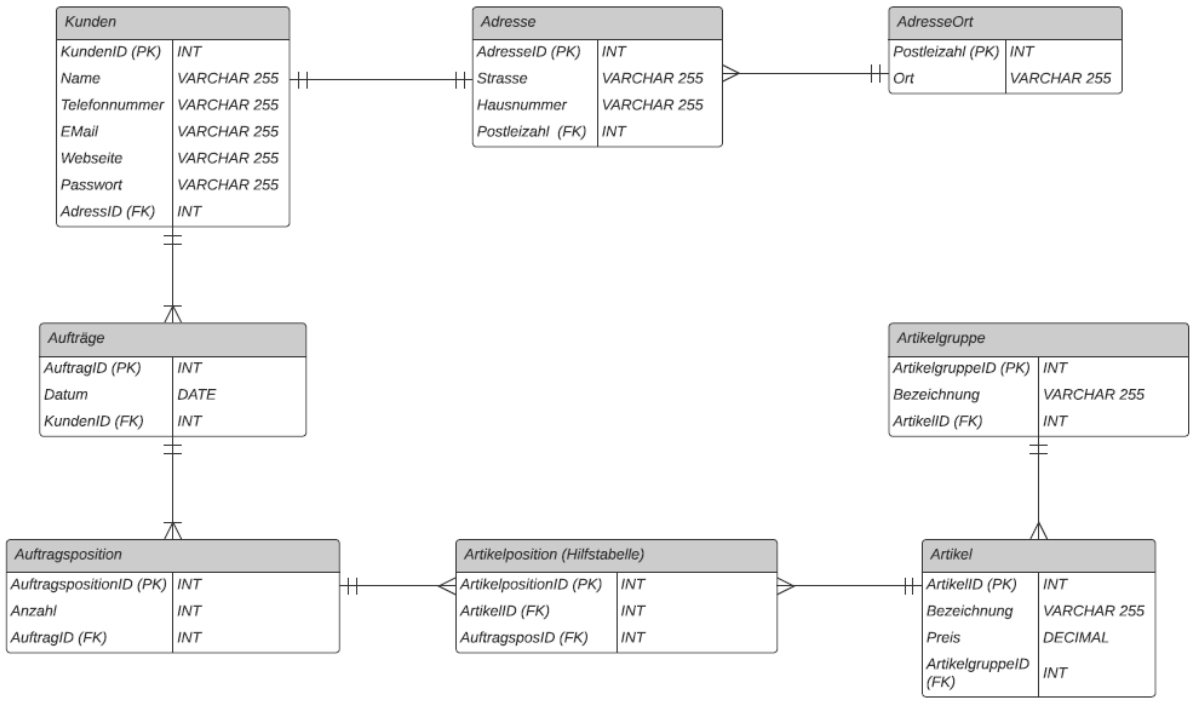
Konzeption

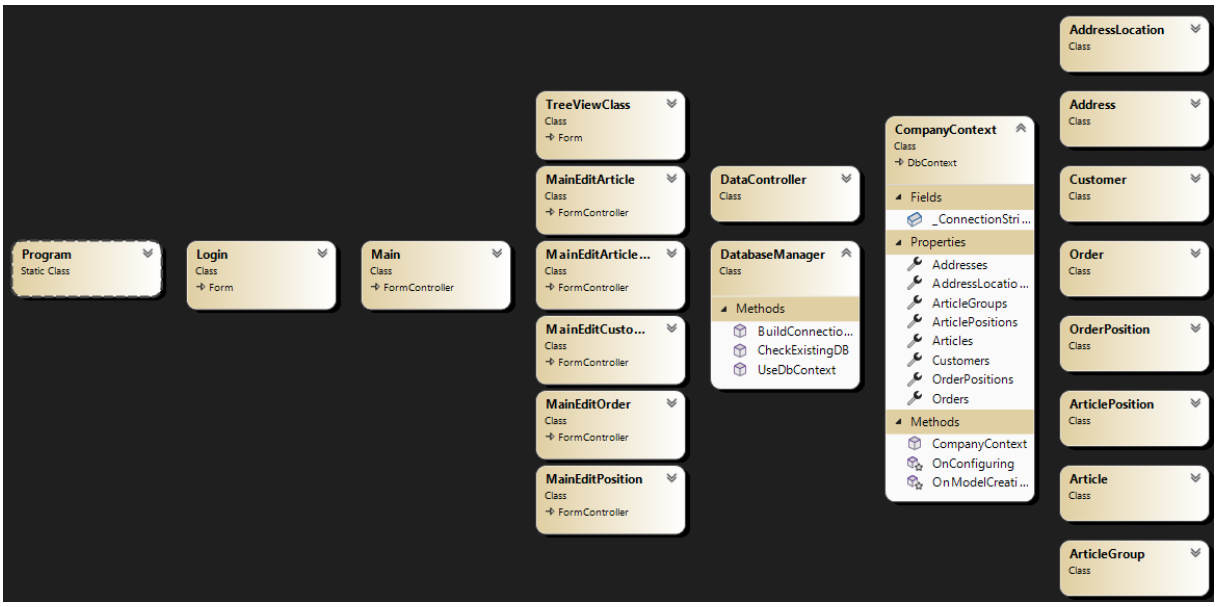
Das Ziel dieses Projekts ist es, eine Business-Applikation für die Muster AG zu entwickeln, die es ermöglicht, Kunden, Adressen, Artikel, Artikelgruppen sowie Aufträge und Auftragspositionen effektiv zu verwalten. Die Applikation wird mit dem Code-First-Ansatz erstellt und mit einer lokalen Datenbank verknüpft. Die Benutzeroberfläche soll benutzerfreundlich, einfach gestaltet sein und alle erforderlichen Funktionen abdecken. Es ist jedoch klar, dass in einem realen Projekt diese Oberfläche noch optimiert werden muss und deshalb nicht final ist.

Betrieb der Applikation

Durch die Eingabe des eigene Connection String wird eine Verbindung zum lokalen Datenbankserver erstellt. Der Datenbank Name kann neu gewählt werden, dadurch wird eine neue Datenbank erstellt oder es wird die gleichnamige Datenbank überschrieben. Nachdem der Login erfolgreich war, öffnet sich das Hauptfenster. In diesem ist es möglich, Daten einzusehen, welche in einem Grid-View dargestellt sind und in Tabs gegliedert sind. Um CRUD auszuführen, klickt man auf den entsprechenden Button, welcher ein neues Fenster öffnet und dort Bearbeitungen der Daten zulässt. Nach Beendigung der Task wird man wieder zum Hauptfenster geführt.

ERM / Klassenmodel





Vorgehen

Wir haben uns für .NET Core Applikation mit WinForms entschieden, weil WPF erst später behandelt wurde und wir sowieso an unsere Grenzen stossen. Um den Code First Ansatz zu lösen haben wir das EF Core Framework benutzt. Dazu mussten wir zuallererst die dazu benötigten NugGet Packete installieren.

NuGet

Microsoft.EntityFrameworkCore

Dieses Packet hilft zur Entwurfszeit bei Entwicklungsaufgaben. Diese werden hauptsächlich verwendet, um Migrationen zu verwalten und ein Gerüst für DbContext und Entitätstypen durch Reverse Engineering eines Datenbankschemas zu erstellen.

Microsoft.EntityFrameworkCore.SqlServer

Zunächst müssen wir das NuGet-Paket für den Anbieter der Datenbank installieren, auf die wir zugreifen möchten. In diesem Fall möchten wir auf die MS SQL Server-Datenbank zugreifen.

Microsoft.EntityFrameworkCore.Tools

Um EF Core-Befehle über die Befehlszeilenschnittstelle (CLI) von .NET Core ausführen wird diese NuGet Packet verwendet. Darüber haben wir nach längerer Zeit auch die Migration gemacht.

DbContext

Diese Klasse dient dazu, die Datenbank mit dem übergebenen Connection String sowie allen Tabellen und ihren Beziehungen zu erstellen. Wir haben lange versucht, die Migration direkt im Code durchzuführen. Leider hat das nicht funktioniert und nach mehreren Stunden und weiteren Diskussionen haben wir uns für die Migration über die CLI entschieden, was dann auch sehr gut funktioniert hat. Der Denkfehler, den Migration-Befehl auf jedem einzelnen Rechner auszuführen, hat uns eine ganze Zeit beschäftigt.

Connection zur Datenbank

Das Programm soll auf eine lokale Datenbank zugreifen, weil wir keine Server besitzen. Um die Applikation auf jedem Rechner starten zu können und auf den lokalen MS SQL Server zugreifen zu können, haben wir uns überlegt, wie wir das am benutzerfreundlichsten lösen können. Der erste Gedanke war, dass alle Connection Strings in einer App.config-Datei gespeichert werden und dann über die GUI abgerufen werden können. Doch nach der Recherche haben wir festgestellt, dass es eigentlich nicht gut ist, diese Daten in der App.config zu speichern. Außerdem haben neue Nutzer das Problem, dass sie zuerst in einer Konfigurationsdatei Einträge machen müssen. Somit haben wir uns dafür entschieden, die Connection Strings beim Ausführen des Programms in einem Login-Fenster einzugeben, welche geprüft werden und wenn diese gültig sind, eine Datenbank-Session eröffnen und eine Datenbank erstellen. Dies ist nicht der sicherste Weg, aber benutzerfreundlich. Damit die Verbindung auch auf jedem Rechner funktioniert, mussten wir die Connection auf "Trusted" setzen und die Encryption auf "false" setzen.

GUI

Diese wird in Windows Forms gemacht. Die Möglichkeit, die GUI in WPF zu machen, war für uns zwar interessant, aber keine Option. WPF wurde zu einem späten Zeitpunkt des Projekts behandelt, und weil uns das Projekt sonst schon stark gefordert hat, haben wir uns für das bekannte Windows Forms entschieden. Durch ein Login-Fenster verbindet man die Datenbank mit der Appli-kation. Ist dies erfolgreich geschehen, kommt man auf das Hauptfenster. Hier kann man über verschiedene Tabs die Tabellen einsehen. Durch Buttons wird man auf ein neues Fenster geleitet, wo Daten mit CRUD bearbeitet werden können.

Arbeitsjournal

Wie lange	Was	Gelerntes	Wer
7	Gui erstellen und optimieren	Gui bauen, Verhalten von separierten Guis	Nick
12	Auftragsverwaltung	Verwaltung von Daten mit LINQ, GridView, Abhängigkeiten von Tabellen in einer DB	Nick
3	Jahresvergleich	Anwenden von Window Functions und Pivot-Tabellen	Nick
10	Implementierung ModelView	Abkapselung von View und Modelling	Nick
6	Besprechungen	Zusammenarbeit im Team, Git-Merging	Nick
12	Tabellen erstellen mit allen Beziehungen	Fluent-API, Navigation Properties, DbContext erstellen	Marco
4	Testdaten erstellen	Welche Möglichkeit es gibt um Testdaten einzufügen	Marco
14	Rekursive CTE erstellen	Wie man ein rekursives CTE erstellt und es in einer TreeView in Forms ausgibt	Marco
6	Besprechungen	Zusammenarbeit im Team, Git-Merging	Marco
4	DbContext	Überarbeiten, Fluent-API	Gabor
5	Migration	Migration erstellt/ erster versuch über Software selbst nachher mit CLI Tools gelöst	Gabor
3	Dokumentation / ERM	Zusammentragen, bearbeiten und überarbeiten	Gabor
6	Window Functions	Alles in SQL SM Studio getestet.	Gabor
1	DatabaseManager	Klasse die im Nachhinein nicht nötig gewesen wäre	Gabor
2	Login Klasse	Login auf Datenbank überprüft und Verbindung hergestellt	Gabor