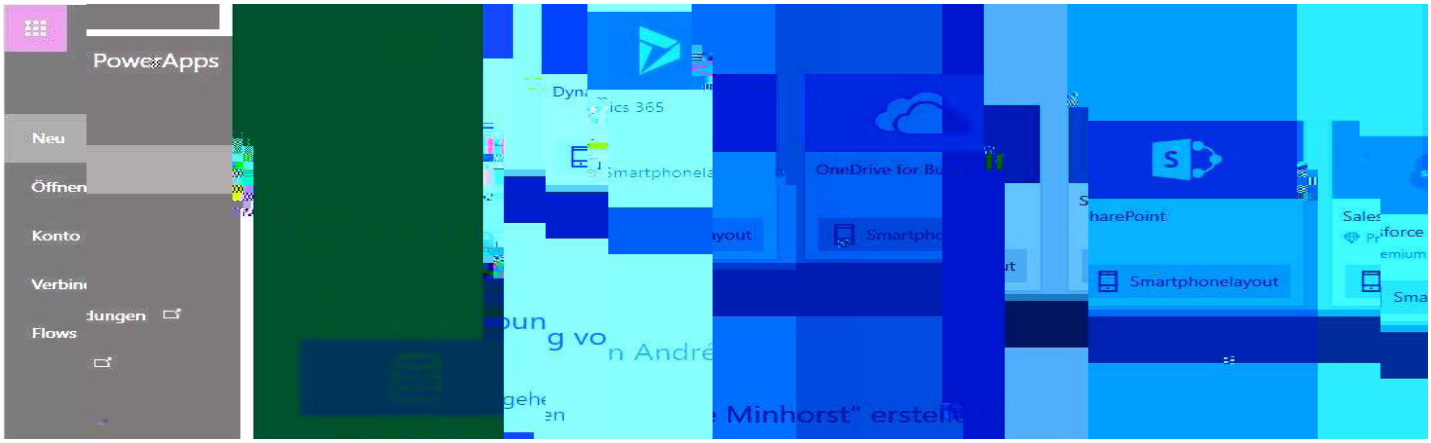
AUSGABE 06/2018

DATENBANK ENTWICKLER

MAGAZIN FÜR DIE DATENBANKENTWICKLUNG MIT VISUAL STUDIO FÜR DESKTOP, WEB UND CO.



TOP-THEMEN:

André Minhorst Verlag

Impressum

DATENBANKENTWICKLER
© 2018 André Minhorst Verlag
Borkhofer Str. 17
47137 Duisburg

Redaktion: Dipl.-Ing. André Minhorst

SQL Server-Datenbank ins Web mit SQL Azure

In einigen vorangehenden Artikeln haben wir gezeigt, wie Sie eine SQL Server-Datenbank auf Basis des Entity Data Models eines Projekts erstellen und unter einem Azure-Konto über eine Webseite veröffentlichen. In diesem Artikel wollen wir uns ansehen, wie Sie eine bestehende SQL Server-Datenbank ohne Entity Data Model als Azure-Datenbank anlegen. Auf diese Datenbank können Sie dann sowohl von Desktop-Anwendungen aus zugreifen als auch von PowerApps, die wir ebenfalls in dieser Ausgabe vorstellen.

Microsoft bietet die Möglichkeit, eigene SQL Server-Datenbanken über das Internet verfügbar zu machen. Dazu benötigen Sie ein Azure-Konto bei Microsoft, unter dem Sie dann einige teils kostenlose und teils kostenpflichtige Dienste in Anspruch nehmen können. Den Einstieg finden Sie unter dem folgenden Link:

```
https://azure.microsoft.
com/de-de/free/
```

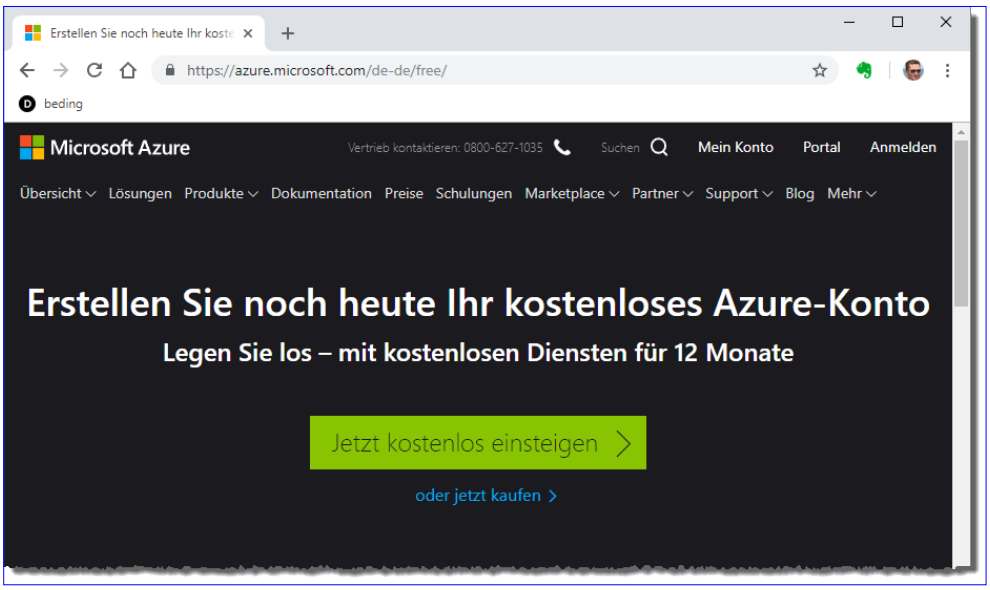


Bild 1: Einstieg mit dem kostenlosen Azure-Konto

Dieser führt Sie zu einer Seite, wo Sie ein zunächst kostenloses Konto anlegen können (siehe Bild 1). Eines vorneweg: Wenn Sie SQL Server-Datenbanken betreiben wollen, sind damit Kosten verbunden. Ich sehe keine Möglichkeit, einen solchen Dienst kostenlos in Anspruch zu nehmen – auch wenn Sie einen eigenen Server mieten und dort einen SQL Server betreiben, fallen dabei Kosten an.

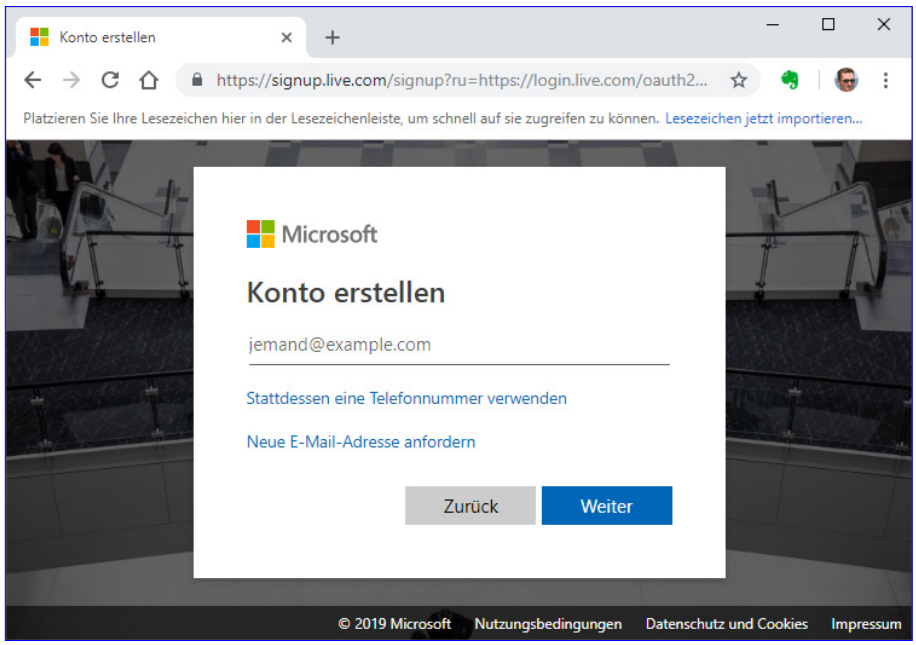


Bild 2: Anmelden bei live.com

Als Nächstes legen Sie nun ein Konto auf der Seite **live.com** an, wo Sie

ein paar Schritte später landen. Nach der Eingabe der E-Mail-Adresse auf der Seite aus Bild 2 geben Sie auch noch ein Kennwort an.

Danach fragt Microsoft noch ein paar weitere Daten ab wie das Land und das Geburtsdatum (siehe Bild 3).

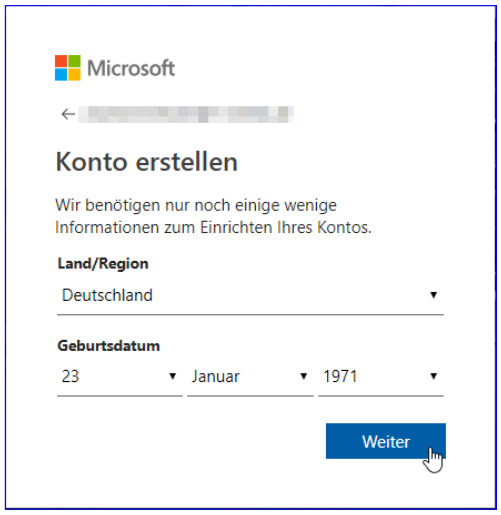


Bild 3: Weitere Schritte zum Erstellen des Microsoft-Kontos

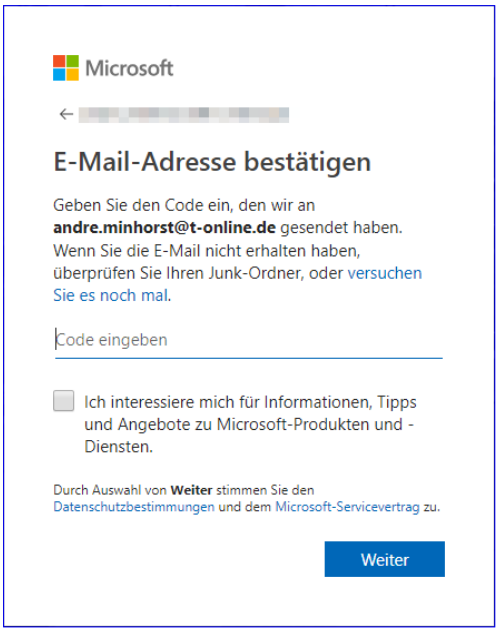


Bild 4: Abschluss der Anmeldung bei Microsoft

Anschließend müssen Sie die E-Mail-Adresse bestätigen, indem Sie einen an die angegebene Adresse gesendeten Code eingeben (siehe Bild 4). Schließlich folgt noch eine Prüfung, ob das Konto von einer realen Person erstellt wird und welche Telefonnummer zur Wiederherstellung des Kontos genutzt werden soll.

Azure-Konto registrieren

Danach folgt die Registrierung des Azure-Kontos. Dazu geben Sie im ersten Dialog die persönlichen Informationen an, also Land, Vorname, Nachname, E-Mail-Adresse und so weiter (siehe Bild 5). Hier geben Sie dann die E-Mail-Adresse ein, mit der Sie das Microsoft-Konto erstellt haben (wenn Sie nicht schon eines besessen haben).

Danach folgt eine Identitätsprüfung per Telefon, bei der eine Nachricht an die zuvor angegebene Telefonnummer geschickt wird. Nachdem Sie die Nachricht erhalten haben, geben Sie

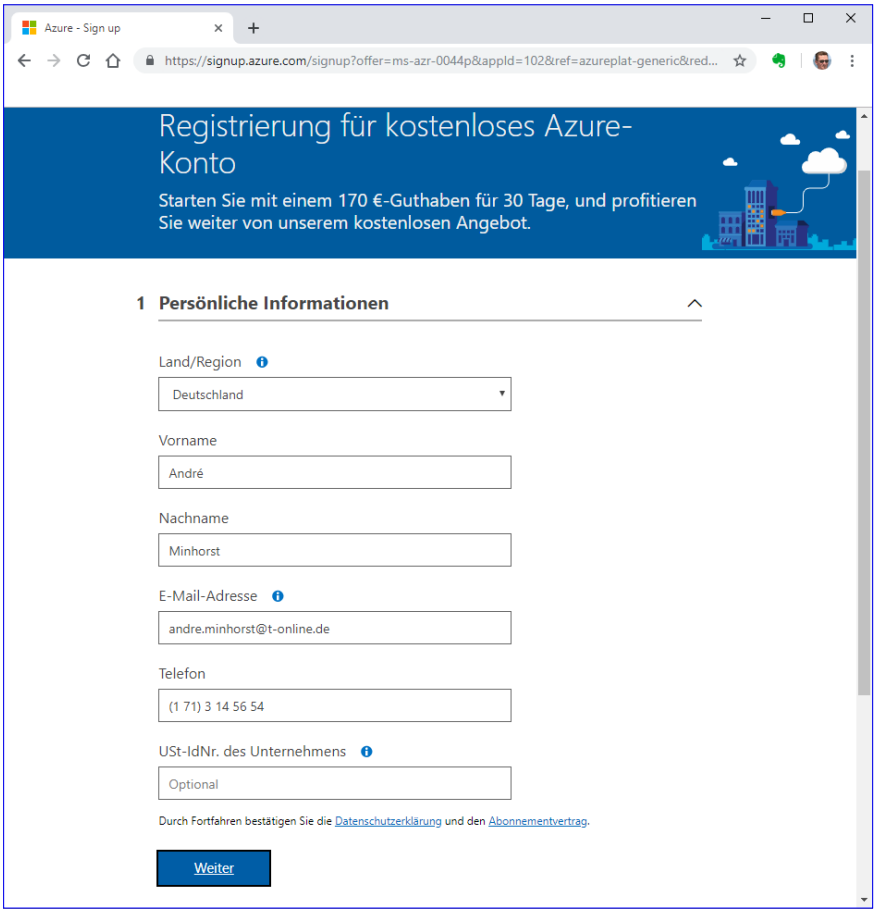


Bild 5: Registrierung für ein Azure-Konto

den enthaltenen Prüfcode in das entsprechende Textfeld ein. Im nächsten Schritt erfolgt eine weitere Identitätsüberprüfung über die Kreditkarte.

Diese wird nach der Angabe auf der Webseite erst nach einem Upgrade auf einen kostenpflichtigen Dienst belastet. Nachdem Sie die Kartendaten angegeben und einige Optionen wie die Datenschutzerklärung abgehakt haben, können Sie die Registrierung abschließen.

Das Azure-Portal

Danach gelangen Sie entweder direkt oder über den folgenden Link zum Azure-Portal, wo Sie sich gegebenenfalls noch einloggen müssen:

https://portal.azure.com/

Das Portal erwartet Sie mit der Übersicht aus Bild 6. Hier klicken Sie auf den Eintrag **SQL-Datenbanken**.

Damit gelangen Sie zum Bereich **SQL-Datenbanken**. Wenn Sie für Ihr Konto zuvor noch keine Datenbank erstellt haben, finden Sie hier die Meldung **Keine SQL-Datenbanken zum Anzeigen** vor.

Das ist kein Problem, denn direkt darunter erscheint der Link **SQL-Datenbank erstellen** (siehe Bild 7). Diesen klicken Sie an, um zum Dialog zum Erstellen einer neuen SQL-Datenbank zu gelangen.

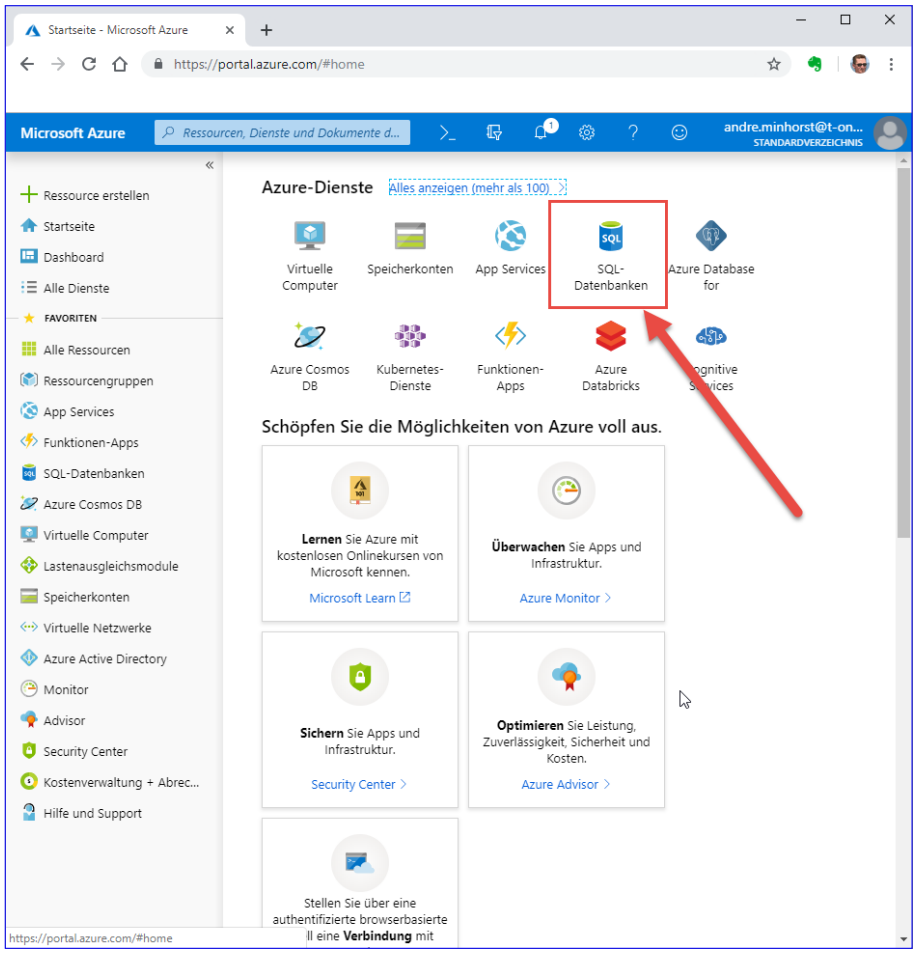


Bild 6: Das Azure-Portal

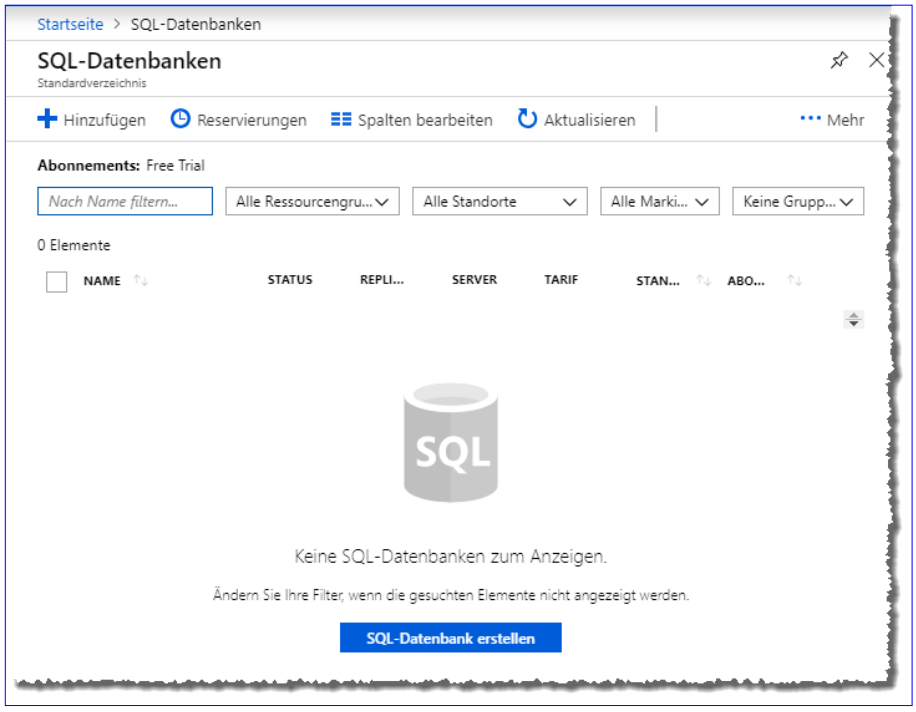


Bild 7: Dialog zum Verwalten der Datenbanken

Ressourcengruppe erstellen

Hier taucht gleich die erste Herausforderung auf: Es wird (bei einem neuen Account) zwar das Abonnement namens **Free Trial** voreingestellt, aber darunter findet sich die Option **Ressourcengruppe**, mit der wir zunächst einmal nichts anfangen können. Ein Klick auf das Auswahlfeld mit dem Text **Vorhandene auswählen...** führt nicht zum Erfolg, da hier bei einem neuen Konto noch keine Einträge vorhanden sind. Also verwenden wir den Link **Neues Element erstellen** direkt darunter (siehe Bild 8).

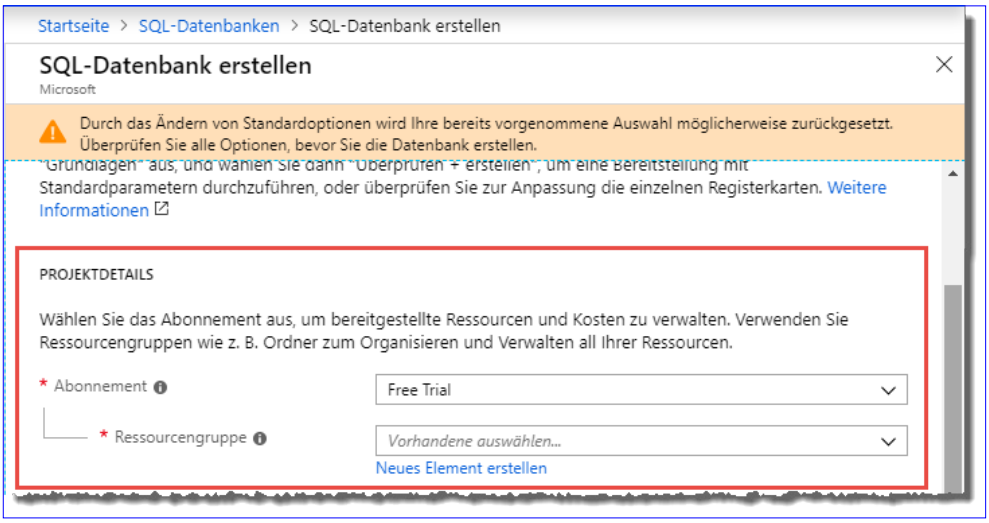


Bild 8: Anlegen einer neuen SQL-Datenbank

Die Ressourcengruppe ist eine Art Container, in der Sie verschiedene Ressourcen verwalten können, also etwa eine Webanwendung und eine SQL-Datenbank. Wir benötigen im Rahmen des aktuellen Artikels nur eine SQL-Datenbank für ein Beispiel, daher benennen wir die Ressourcengruppe einfach **Beispiel-SQL-Datenbank** (siehe Bild 9).

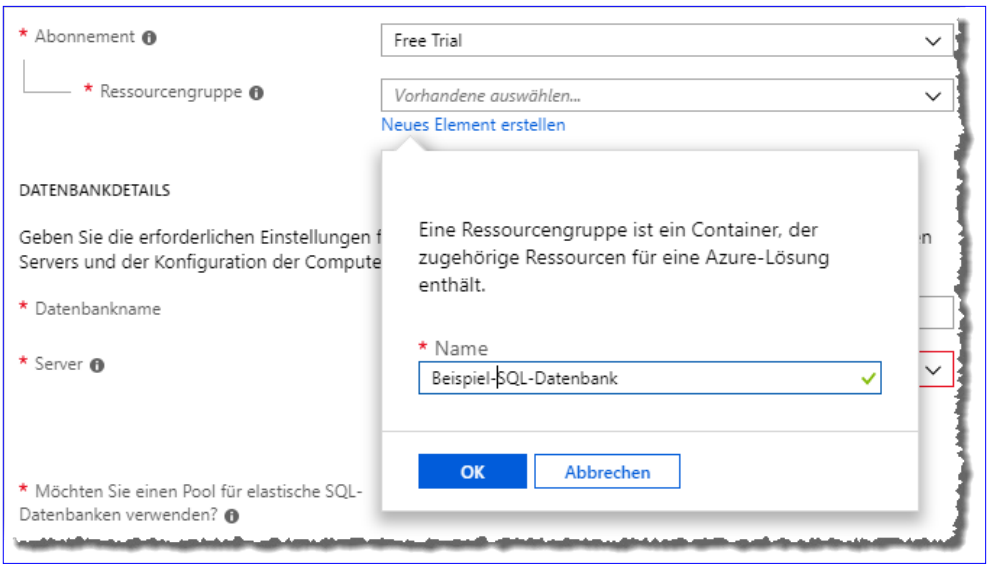


Bild 9: Anlegen einer neuen Ressource

Datenbankdetails festlegen

Nun können wir uns den weiter unten im Dialog einzugebenden Informationen

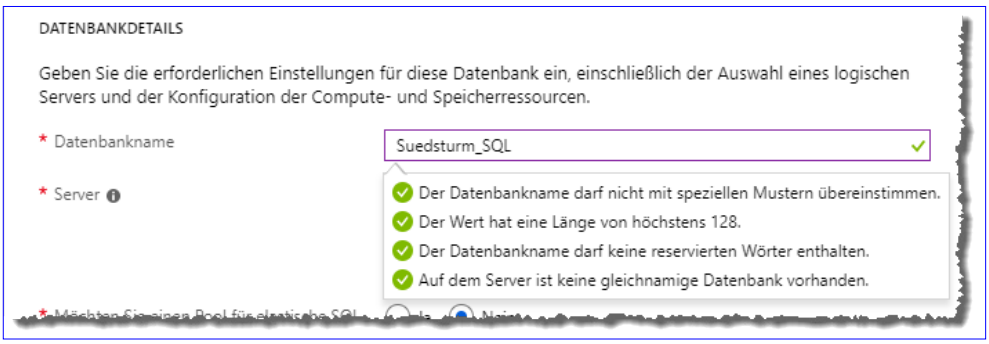


Bild 10: Eingabe des Datenbanknamens

zuwenden. Hier benötigen wir einen Datenbanknamen und einen Server. Der Datenbankname ist schnell gefunden – wir wollen die Südsturm-Datenbank, die wir in weiteren Artikeln verwenden, als Azure-Datenbank erstellen. Als Abgrenzung zur ursprünglich unter Access verwendeten Südsturm-Datenbank fügen wir hier noch den Zusatz **SQL** hinzu, also **Suedsturm_SQL**. Bezüglich des Datenbanknamens gibt es einige Einschränkungen, die aber bei der Eingabe eingeblendet werden (siehe Bild 10).

Server erstellen

Auch die Option **Server** bietet den Eintrag **Wählen Sie einen Server aus**, allerdings ist auch dieses Auswahlfeld leer. Also betätigen wir auch hier den Link **Neu erstellen**. Es erscheint ein Popup, mit dem wir den Servernamen, den Namen für die Administratoranmeldung und das Kennwort eingeben können. Als Standort wurde **Australien Ost** angegeben, was sich in unserem Fall auch nicht ändern ließ. Also haben wir den Wert so beibehalten. Nach dem Eintragen der Informationen legen Sie den neuen Server mit einem Klick auf **Auswählen** an (siehe Bild 11). Danach dauert es einige Sekunden, bis die Datenbank erstellt wurde.

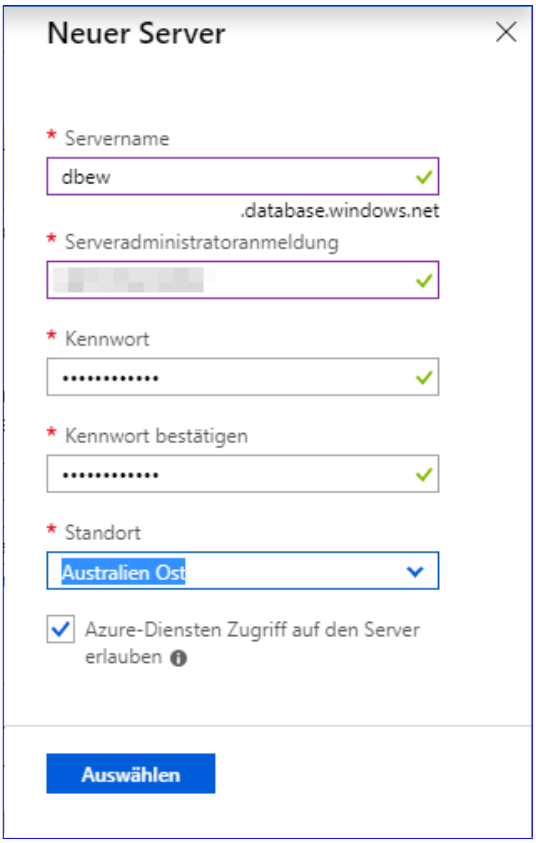


Bild 11: Anlegen eines neuen Servers

Kosten im Griff

Wichtig ist noch, die richtige Datenbankversion zu wählen. Klicken Sie auf **Datenbank konfigurieren**, erscheint rechts ein neuer Bereich, der die verschiedenen Versionen (Basic, Standard und Premium) anbietet (siehe Bild 12). Üblicherweise ist **Standard** aktiviert. Für ein Beispiel benötigen Sie aber vermutlich kaum mehr als zwei Gigabyte Speicher, also wechseln wir zu **Basic**. Damit entstehen Kosten von mindestens 4,63 EUR pro Monat, wobei wir beim Erstellen unseres Azure-Accounts eine Gutschrift von 170 EUR erhalten haben – Sie können damit also theoretisch eine Weile experimentieren. Allerdings stehen Ihnen die 170

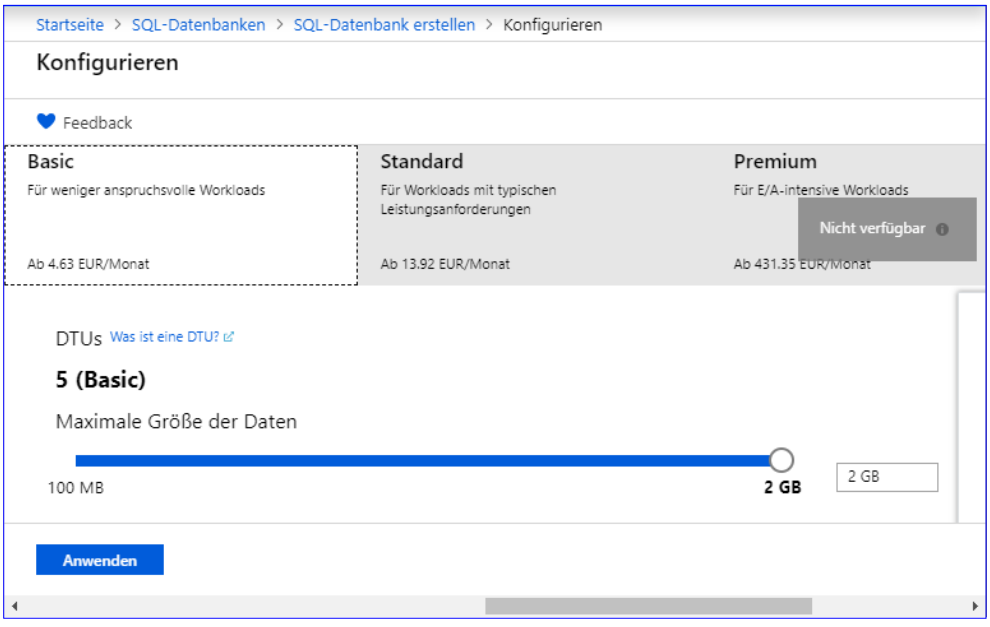


Bild 12: Auswahl des Preismodells

EUR nur für einen Zeitraum von 30 Tagen zur Verfügung. Informationen über das kostenlose Konto und die Bedingungen erfahren Sie unter folgendem Link:

`https://azure.microsoft.com/de-de/free/free-account-faq/`

Bild 13: Start der Konfiguration der Datenbank

Dort erfahren Sie auch, dass Sie nach Ablauf der 30 Tage oder nach dem Aufbrauchen der 170 EUR informiert werden und die Dienste dann durch den Abschluss eines Azure-Abonnements mit nutzungsbasierter Abrechnung weiter nutzen können. Da Sie nur 30 Tage zum Aufbrauchen der 170 EUR haben, könnten Sie also auch ein teureres Modell für die Datenbanknutzung wählen.

Datenbank mit Tabellen und Daten füllen

Damit haben wir alle Informationen unter Azure angegeben, die für die Verwendung einer SQL-Datenbank erforderlich sind – mit Ausnahme der Tabellen und Daten. Diese können Sie hinzufügen, wenn Sie weiter unten auf **Weiter: Zusätzliche Einstellungen** klicken (siehe Bild 13). Im nun erscheinenden Bereich können Sie die Datenquelle auswählen, und zwar auf zwei Arten:

- Auswahl der Option **Sicherung**: Hier können Sie eine Sicherungsdatei angeben, auf deren Basis dann die gesicherte Datenbank wiederhergestellt wird. Allerdings können Sie hier nur Sicherungen auswählen, die unter diesem Abonnement hergestellt wurden. Da wir das Konto gerade neu erstellt haben, sind wir hier in einer Sackgasse gelandet.

- Auswahl der Option **Beispiel**: Das ist die einfachste Methode, mit der Sie einfach die Microsoft-Beispieldatenbank **AdventureWorksLT** anlegen.

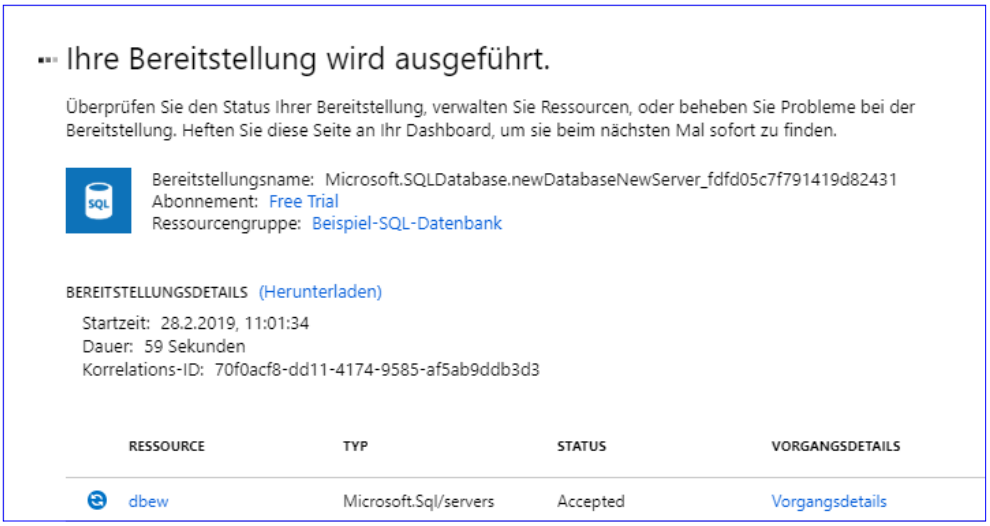


Bild 14: Ausführen der Bereitstellung

Wir können aber auch, wenn wir die Verbindungszeichenfolge kennen, etwa von Visual Studio aus die Tabellen der Datenbank hinzufügen oder dies von SQL Server Management Studio

EDM: m:n-Beziehungen per Code First

Genau wie 1:n-Beziehungen können Sie mit Code First auch m:n-Beziehungen abbilden. Wer denkt, dass dazu wie in Datenbanksystem eine Art Verknüpfungsklasse notwendig ist, hat Recht – diese ist aber nur in bestimmten Fällen notwendig. Welche das sind und wie Sie die verschiedenen Arten einer m:n-Beziehung unter Code First abbilden, erfahren Sie in diesem Artikel. Außerdem zeigen wir, welche Datenmodelle beim Migrieren der Klassen in ein Datenbanksystem entstehen.

Vorbereitungen

Die Vorbereitungen laufen wie im Artikel **EDM: 1:n-Beziehungen per Code First** beschrieben.

Verschiedene Arten von m:n-Beziehungen

Wir kümmern uns in diesem Artikel um zwei verschiedene Arten von m:n-Beziehungen. Die erste Art geht davon aus, dass nur die Beziehung zwischen den Elementen zweier Klassen abgebildet werden soll. Das kann zum Beispiel die Beziehung zwischen Artikeln und Kategorien sein. Sie können jedem Artikel eine oder mehrere Kategorien zuweisen und jeder Kategorie ein oder mehrere Artikel. Dabei sind für eine Beziehung keine weiteren Informationen notwendig. Die zweite Art der Beziehung enthält zusätzliche Informationen zu jeder Kombination der beteiligten Klassen. Dabei kann es sich etwa die Beziehung zwischen einer Bestellung und den darin enthaltenen Artikeln handeln. Jede Bestellung kann mehrere Artikel enthalten und umgekehrt – insofern unterscheidet sich dies nicht von der einfachen m:n-Beziehung. Allerdings wollen wir in diesem Fall zu jeder Kombination noch weitere Informationen speichern, im einfachsten Fall die Menge der bestellten Artikel in einer Bestellung. Hier würden wir im Datenbanksystem einfach ein weiteres Feld zu der Verknüpfungstabelle hinzufügen. In der Modellierung der Klassen ergeben sich weitreichendere Unterschiede, wie die folgenden Beispiele zeigen werden.

Klassen für eine einfache m:n-Beziehung

Wir wollen zunächst eine einfache m:n-Beziehung abbilden, wie sie zwischen Artikeln und Kategorien vorgenommen wird. Dazu legen wir als Erstes die einfachen Klassen für diese beiden Entitäten an – in unserem Beispiel im Verzeichnis **Data**. Die Klasse enthält die Felder **ID**, **Name** und **Price**:

```
namespace EDMFluentAPI.Data {
    public class Article {
        public int ID { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
    }
}
```

Die Klasse für die Kategorien halten wir noch einfacher – sie soll nur das Feld **ID** und das Feld **Name** enthalten:

```
public class Category {
    public int ID { get; set; }
```

```
    public string Name { get; set; }
}
```

Wie stellen wir nun die Beziehung her? Das gelingt ganz einfach, indem wir beiden Klassen jeweils eine **ICollection**-Auflistung mit dem Typ des jeweils zu verknüpfenden Elements hinzufügen. Zunächst für die Klasse **Article**:

```
public class Article {
    ...
    public ICollection<Category> Categories { get; set; }
}
```

Für die Klasse **Category** sieht das andersherum aus:

```
public class Category {
    ...
    public ICollection<Article> Articles { get; set; }
}
```

Wir dürfen natürlich nicht vergessen, beide Klassen über entsprechende **DbSet**-Auflistungen zum Datenbankkontext hinzuzufügen:

```
public class CustomerManagementContext : DbContext {
    ...
    public virtual DbSet<Article> Articles { get; set; }
    public virtual DbSet<Category> Categories { get; set; }
    ...
}
```

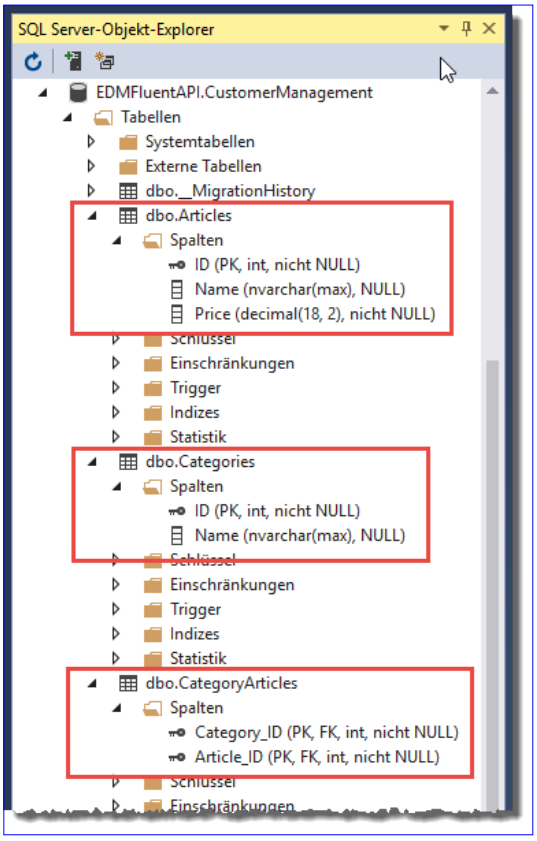


Bild 1: Anlegen einer einfachen m:n-Beziehung

Datenmodell erstellen

Wie überführt Entity Framework diese Angaben in das Datenmodell? Mit der Anweisung **add-migration AddArticlesCategories** in der Paket-Manager-Konsole erstellen wir das Skript für die Erstellung von Tabellen, Feldern und Beziehungen. Dieses wird direkt angezeigt, damit Sie es kontrollieren können. Dann führen Sie es mit der Anweisung **update-database** aus und schauen uns uns das Ergebnis im SQL Server-Objekt-Explorer an. Hier finden wir nun gleich drei neue Tabellen vor. Die Tabellen **Articles** und **Categories** kennen wir, die Tabelle **CategoryArticles** ist verknüpft die Tabellen **Articles** und **Categories** über die beiden Fremdschlüsselfelder **Category_ID** und **Article_ID**, welche jeweils mit den Tabellen **Categories** und **Articles** verknüpft sind (siehe Bild 1). Die Beziehung wurde sogar direkt mit einem zusammengesetzten Primärschlüssel erzeugt, sodass die zumeist vorliegende Anforderung, dass jede Kombination zweier Datensätze nur einmal in der Verknüpfungstabelle vorkommen darf, erfüllt ist.

Beziehung mit Verknüpfungsinformationen

Nun schauen wir uns den anderen Fall an, bei dem wir der Verknüpfung weitere Informationen hinzufügen wollen – in diesem Fall die Menge eines Artikels in einer Bestellung. Dazu legen wir zunächst wieder die beiden Klassen an, die durch die Bezie-

SQLite Code First mit EF Core

Wir haben bereits einige Male SQLite als Datenbanksystem verwendet. Allerdings hatten wir damals jeweils eine Datenbank mit SQLite erstellt und dann ein Entity Data Model dafür erstellt. In den letzten Ausgaben haben wir uns jedoch in Zusammenhang mit SQL Server-Datenbanken mit Code First-Migrationen befasst, also den Aufbau der Datenbank durch Klassen beschrieben und dann erst die Datenbank erstellen lassen. Das wollen wir nun auch mit SQLite machen, was erstmals durch Entity Framework Core möglich ist. Allerdings sind dazu ein paar Schritte mehr nötig, als es mit dem SQL Server der Fall ist.

Die .NET Core-Vorlagen sind noch überschaubar. Derzeit gibt es für den Desktop noch gar keine Vorlage, sondern nur für Konsolen, Klassenbibliotheken und etwa für ASP.NET-Anwendungen (aktueller Stand: **.NET Core 2.1**).

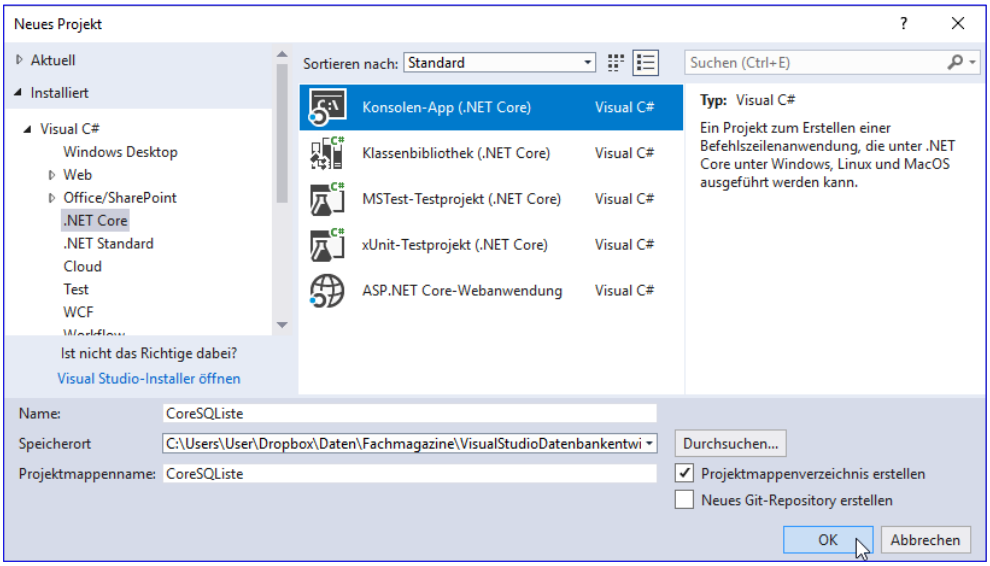


Bild 1: Anlegen eines Projekts auf Basis von .NET Core

Letztere haben wir ja schon programmiert, nun machen wir uns mit den anderen beiden Typen vertraut. Zum Testen verwenden wir zunächst die Vorlage **Konsolen-App (.NET Core)** – siehe Bild 1.

Nachdem wir die Konsolen-App erstellt haben, finden wir die dort übliche Klassenbibliothek **Program.cs** vor. Und Sie sehen es richtig: Für diese Funktion, also den Zugriff per Entity Framework auf eine SQLite-Datenbank, können wir aktuell nicht nur nicht mit WPF arbeiten, sondern die Konsolen-App bietet auch noch keine Unterstützung der Programmiersprache Visual Basic.

Zum Glück haben wir uns in den ersten Ausgaben von **DATENBANKENTWICKLER** auch mit C# beschäftigt, sodass wir einige wenige Codezeilen leicht damit programmieren können. Um die Unterstützung für das Entity Framework hinzuzufügen, klicken Sie mit der rechten Maustaste auf das Projekt-Element im Projektmappen-Explorer und wählen aus dem dann erscheinenden Kontextmenü den Eintrag **NuGet-Pakete verwalten** aus. Hier wechseln Sie dann zum Bereich **Durchsuchen** und geben als Suchbegriff **EntityFrameworkCore.Sqlite** ein.

Wählen Sie hier die stabilste aktuelle Version aus und klicken Sie auf **Installieren** (siehe Bild 2). Danach werden Sie noch aufgefordert, der Lizenz zuzustimmen.

Den gleichen Vorgang wiederholen Sie anschließend für das Paket **Microsoft.EntityFrameworkCore.Design**. Schließlich fügen wir noch das Package **Microsoft.EntityFrameworkCore.Tools** hinzu.

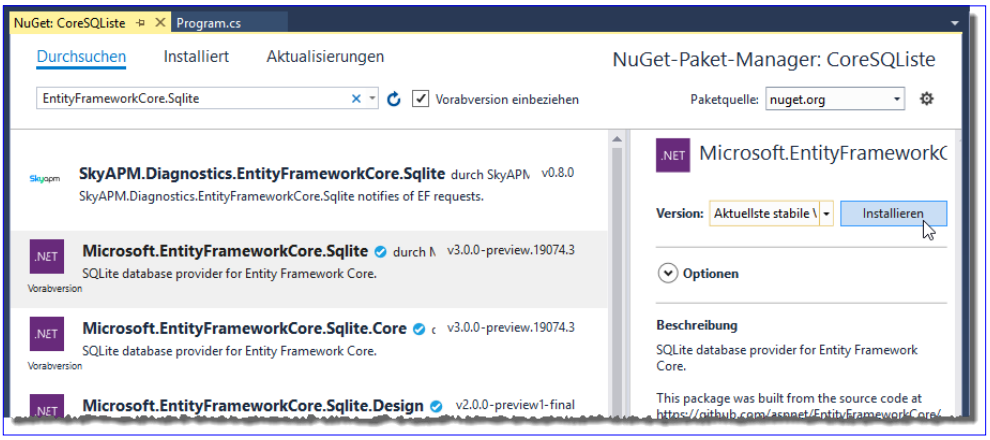


Bild 2: Hinzufügen der NuGet-Pakete

Modell erstellen

Nun benötigen wir das Entity Data Model. Dazu legen Sie eine neue Klasse namens **BestellverwaltungContext.cs** an. Dieser fügen wir zunächst einige Verweise auf Namespaces hinzu:

```
using System;
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using Microsoft.EntityFrameworkCore;
```

Der Code der Klasse selbst sieht wie folgt aus. Die Klasse implementiert die Schnittstelle **DbContext**.

Als Erstes fügen wir eine Methode namens **OnModelCreating** hinzu, welche die eindeutigen Indizes für die betroffenen Felder hinzufügt:

```
namespace CoreSQListe {
    class BestellverwaltungContext : DbContext {
        //eindeutigen Index für EntityFrameworkCore:
        protected override void OnModelCreating(ModelBuilder builder) {
            builder.Entity<Anrede>()
                .HasIndex(a => a.Name)
                .IsUnique(true);
            builder.Entity<Bestellposition>()
                .HasIndex(b => new { b.BestellungID, b.ProduktID })
                .IsUnique(true);
        }
```

Die nächste Methode heißt **OnConfiguring** und legt den Namen der zu erstellenden beziehungsweise zu verwendenden SQLite-Datenbank fest. Das erledigen wir mit der Methode **UseSqlite** des mit dem Parameter **optionsBuilder** übergebenen Elements.

Der Parameter dieser Methode heißt **Data Source** und soll in diesem Fall den Wert **Bestellverwaltung.db** annehmen:

```
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
            optionsBuilder.UseSqlite("Data Source=Bestellverwaltung.db");
        }
```

Nun folgenden die Deklarationen der **DbSet**-Elemente für die einzelnen Auflistungen beziehungsweise Tabellen:

```
        public DbSet<Anrede> Anreden { get; set; }
        public DbSet<Bestellposition> Bestellpositionen { get; set; }
        public DbSet<Bestellung> Bestellungen { get; set; }
        public DbSet<Kategorie> Kategorien { get; set; }
        public DbSet<Kunde> Kunden { get; set; }
        public DbSet<Mehrwertsteuersatz> Mehrwertsteuersaetze { get; set; }
        public DbSet<Produkt> Produkte { get; set; }
    }
```

Im Anschluss an die Definition der **DbContext**-Klasse legen wir die Klassen für die einzelnen Entitäten an. Diese erhalten jeweils mit der DataAnnotation den Namen der zu erstellenden Tabelle. Außerdem werden mit weiteren DataAnnotations wie **StringLength**, **Required** et cetera weitere Eigenschaften für die zu erstellenden Felder angegeben. Wir schauen uns an dieser Stelle nur die Definition der beiden Entitäten **Anrede** und **Bestellposition** an:

```
    [Table("Anreden")]
    public class Anrede {
        public int Id { get; set; }
        [StringLength(255)]
        [Required]
        public string Name { get; set; }
    }
    [Table("Bestellpositionen")]
    public class Bestellposition {
        public int ID { get; set; }
        [Required]
        public int BestellungID { get; set; }
        [Required]
        public int ProduktID { get; set; }
        [Required]
        public decimal Einzelpreis { get; set; }
        [Required]
        public decimal Mehrwertsteuersatz { get; set; }
        [Required]
        public decimal Rabatt { get; set; }
        [Required]
```

C#-DLL in VB-Projekt nutzen

Einige Techniken werden von Microsoft erst für die Benutzung unter C# bereitgestellt. Das ist aktuell etwa der Fall für einige der Core-Produkte wie Entity Framework Core. Wenn Sie diese dennoch frühzeitig nutzen wollen, aber normalerweise mit Visual Basic arbeiten, können Sie das auf folgende Art erledigen: Sie bauen eine DLL auf Basis der Vorlage für C# und binden diese dann in das Visual Basic-Projekt, in dem Sie die Hauptarbeit verrichten. Auf die Objekte, Eigenschaften und Methoden der C#-Bibliothek können Sie dann einfach zugreifen.

Um zu zeigen, wie der Einsatz etwa einer C#-DLL innerhalb einer Visual Basic-Konsolenanwendung funktioniert, erstellen wir zunächst ein Projekt des Typs **Visual BasicIKonsolen-App** (siehe Bild 1). Diese App nennen wir **VisualBasicKonsole**.

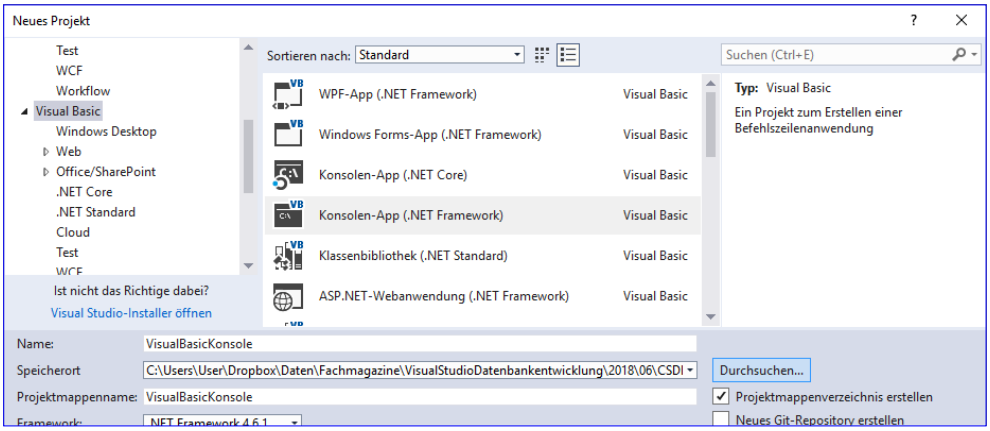


Bild 1: Visual Basic-Konsolenanwendung erstellen

Danach markieren wir im Projektmappen-Explorer den Eintrag mit der Projektmappe und wählen aus dem Kontextmenü den Befehl **HinzufügenINeues Projekt...** aus (siehe Bild 2).

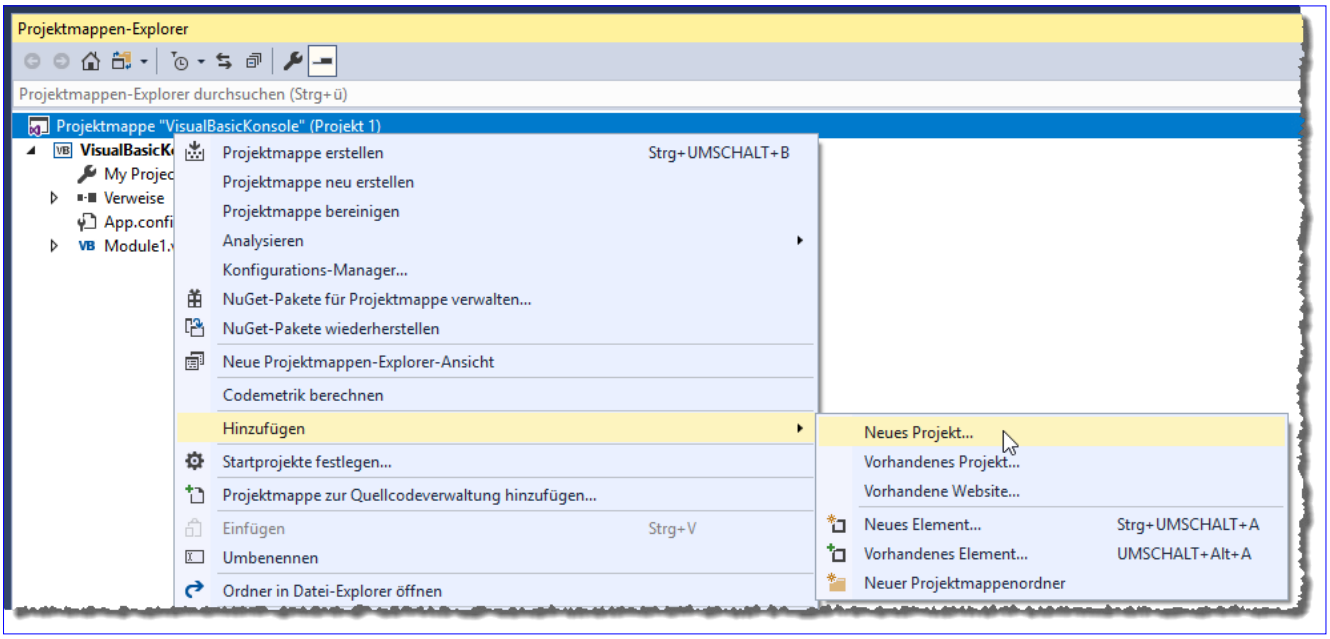


Bild 2: Neues Projekt zu Projektmappe hinzufügen

Im nun erscheinenden Dialog **Neues Projekt hinzufügen** wählen wir die Vorlage **Visual C#\Klassenbibliothek** aus (siehe Bild 3) und nennen diese **CSKlassenbibliothek**.

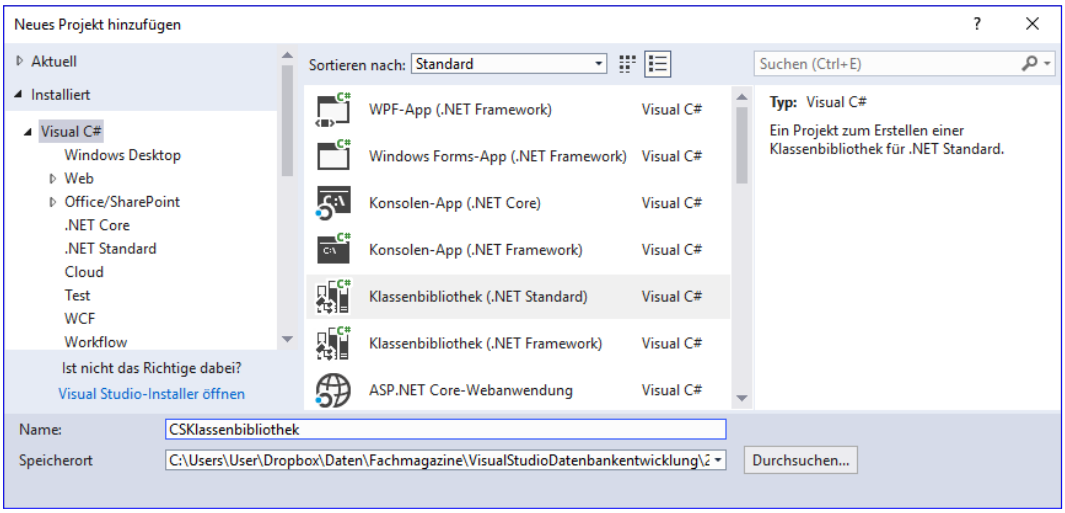


Bild 3: Projekttyp für zusätzliches Projekt festlegen

Im Projektmappen-Explorer sehen wir nun zwei Projekte, von denen eines in fetter Schrift dargestellt ist (das zuerst angelegte) und eines in normaler Schrift (siehe Bild 4).

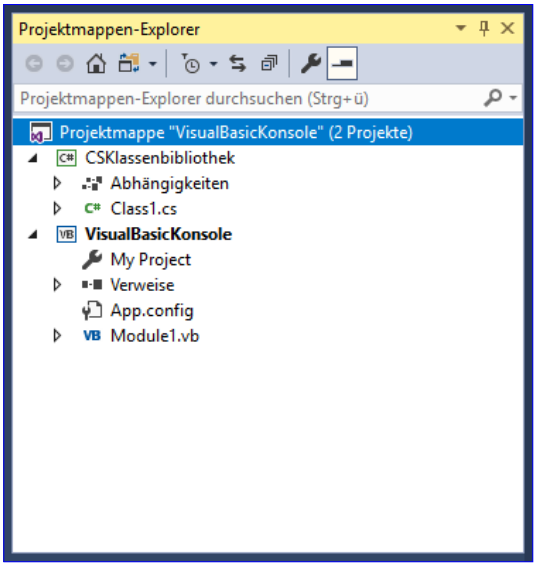


Bild 4: Projektmappe mit zwei Projekten

Das Projekt in fetter Schrift ist allerdings nur zufällig das zuerst angelegte: Eigentlich markiert die fette Schrift nämlich das Startprojekt, also das Projekt, das beim Start ausgeführt wird. Sie können ein anderes Projekt als Startprojekt auswählen, indem Sie den Kontextmenü-Eintrag **Als Startprojekt festlegen** eines aktuell nicht als Startprojekt festgelegten Projekts auswählen.

Verweis auf DLL hinzufügen

Damit Sie nun vom Startprojekt, also **VisualBasicKonsole**, auf die Elemente der DLL **CSKlassenbibliothek** zugreifen können, fügen Sie einen Verweis auf die DLL zur Konsolenanwendung hinzu. Dazu wählen Sie den Kontextmenü-Eintrag **Verweis hinzufügen...** des Eintrags **Verweise** des Projekts **VisualBasicKonsole** aus (siehe Bild 5).

Es erscheint der Dialog **Verweis-Manager**. Dieser zeigt praktischerweise direkt das andere Projekt der aktuellen Projektmappe im Bereich **Projekte\Projektmappe** an.

Also markieren wir den Eintrag **CSKlassenbibliothek** und schließen den Dialog mit der **OK**-Schaltfläche (siehe Bild 6).

Nach dem Schließen des Dialogs **Verweis-Manager** erscheint der neu hinzugefügte Verweis auch direkt unter dem Ordner **Verweise** der Konsolenanwendung (siehe Bild 7).

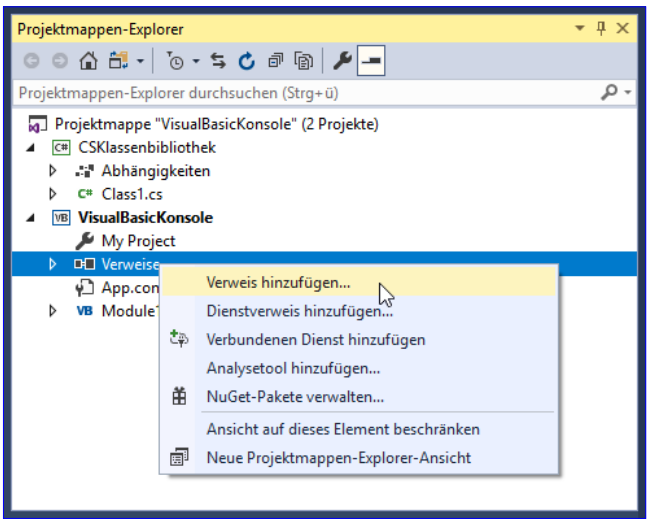


Bild 5: Verweis zu Projekt hinzufügen

Einstieg in PowerApps

PowerApps sind eine neue Möglichkeit, Anwendungen für Web, Desktop und mobile Geräte zu programmieren. Dabei geht es weniger darum, native Apps etwa für ein iPhone oder iPad zu programmieren. Stattdessen wählt Microsoft den Ansatz, eine Art Container-App zur Verfügung zu stellen, mit der man dann speziell für diese programmierte Anwendungen öffnen kann. Das gelingt allein über eine recht einfache Online-Entwicklungsumgebung. Aber es gibt auch Einschränkungen: Zum Beispiel kann nicht jeder eine solche App im App-Store erwerben. Wer die App nutzen will, muss die PowerApps-App auf seinem Gerät haben und muss für die Nutzung der App freigeschaltet sein. Dieser Artikel zeigt die Grundlagen rund um die Erstellung von PowerApps.

Voraussetzungen

Die erste Voraussetzung für die kommerzielle Nutzung der PowerApps oder die Nutzung in Teams ist das Vorhandensein einer Lizenz für Office 365, Dynamics 365 oder Power Apps Plan 1 oder 2.

Für die beiden letzteren gibt es kostenlose Testversionen, von denen wir in diesem Artikel die Variante **PowerApps Plan 1** nutzen wollen (zu finden unter **https://powerapps.microsoft.com/de-de/pricing/**).

Zum Glück gibt es wie von Visual Studio und SQL Server auch hier wieder eine Community-Version, die Sie hier aufrufen können: **https://powerapps.microsoft.com/de-de/communityplan/**

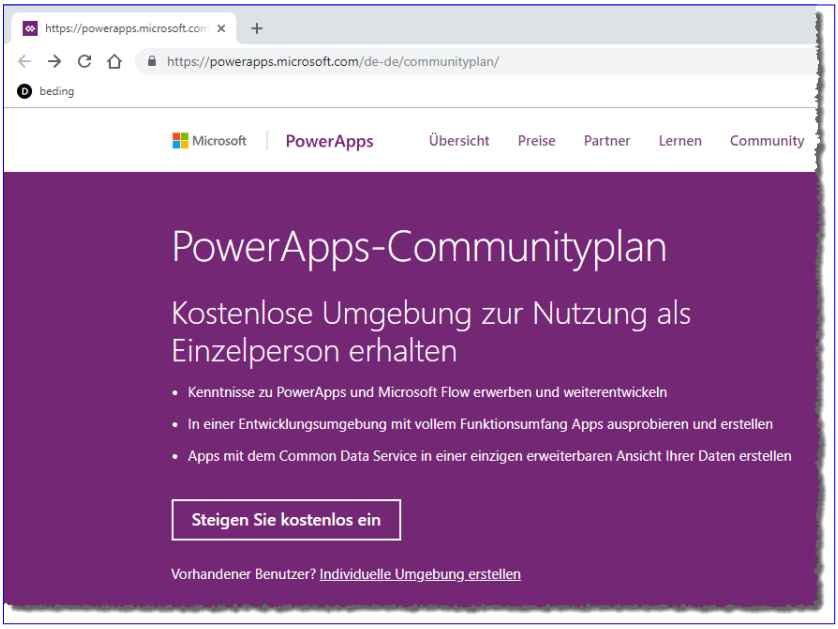


Bild 1: Einstieg in die Community-Version der PowerApps

Wir schauen uns diese Version der PowerApps an und klicken gleich auf **Steigen Sie kostenlos ein** (siehe Bild 1). Danach melden Sie sich mit einem Microsoft-Konto an oder erstellen ein neues. Nach ein paar weiteren organisatorischen Schritten stehen Sie vor der ersten Entscheidung: Sie können dann verschiedene Vorlagen für das Erstellen einer eigenen App nutzen. Es gibt die folgenden Möglichkeiten:

- **Canvas-App ohne Vorlage**: Hier entwerfen Sie die komplette App und verknüpfen diese mit den gewünschten Datenquellen.
- **Modellgesteuerte Apps ohne Vorlage**: Sie erstellen ein Modell Ihrer Daten und lassen dann auf Basis dieses Modells eine PowerApp erstellen.

- **Mit Daten beginnen**: Dies erstellt eine App auf Basis von Daten, die aus drei Bildschirmen besteht, die sie anschließend anpassen können.

- **PowerApps Training for Office**

Eine Canvas-App ohne Vorlage erstellen

Wir wollen zuerst die erste Vorlage ausprobieren. Dazu fahren Sie mit der Maus über das Element und wählen zunächst aus, ob die App für Smartphones oder für Tablets optimiert werden soll. Wir wählen Smartphone, da dieses im Gegensatz zu einem Tablet bei jedem Leser vorhanden sein sollte. Danach klicken Sie auf den Link **Diese App erstellen** (siehe Bild 2).

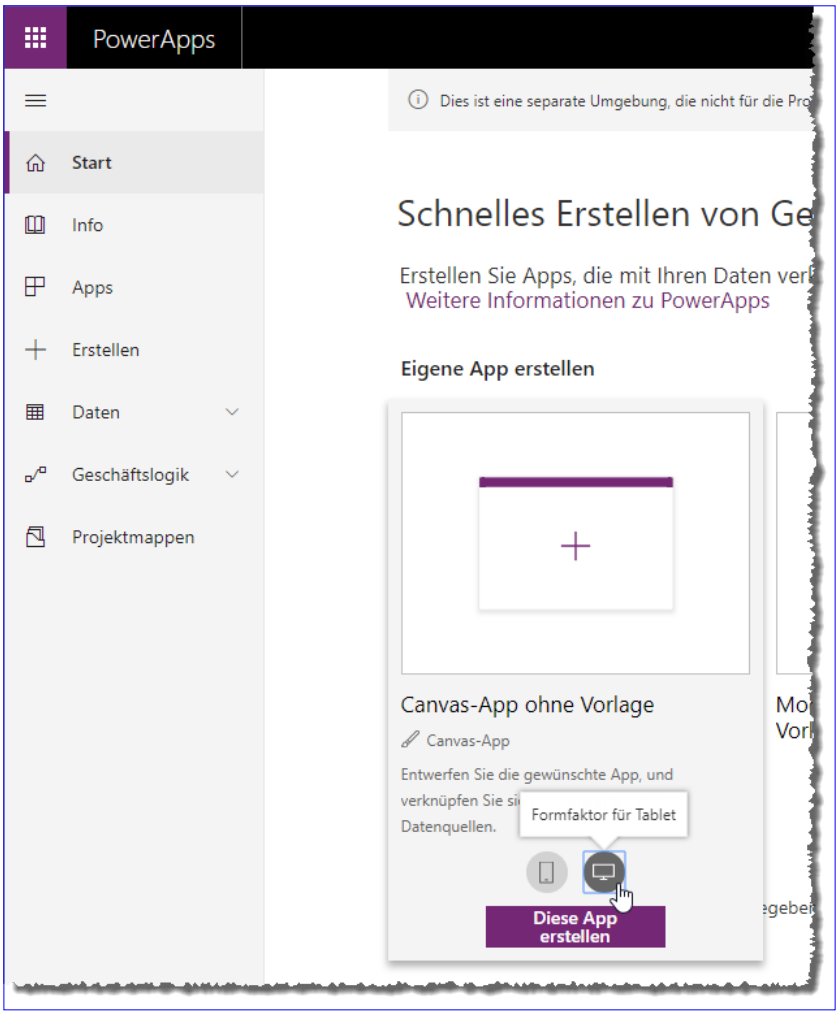


Bild 2: Erstellen einer Canvas-App ohne Vorlage

Danach öffnet sich die PowerApps-Entwicklungsumgebung im Browser. Hier können Sie noch auswählen, ob Sie ein Formular erstellen wollen oder einen Katalog. Außerdem finden Sie hier eine interaktive Tour (siehe Bild 3).

Formular erstellen bedeutet, eine Detailansicht von einem Datensatz anzuzeigen. **Katalog erstellen** heißt, eine Übersicht mit mehreren Datensätzen zu bauen.

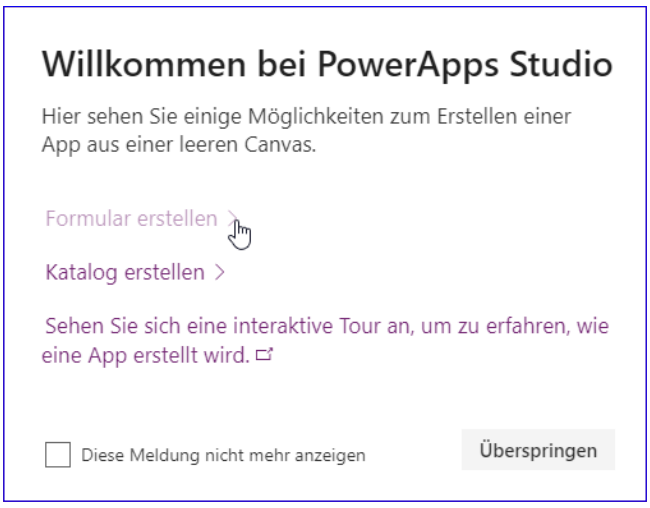


Bild 3: Möglichkeiten beim Anlegen der App

Wir wollen erst einmal eine Anwendung nach einer Vorlage erstellen, um uns grundlegende Schritte anzusehen. Dazu nutzen wir die Option **Überspringen**, um die App direkt ohne Unterstützung zu erstellen und zu weiteren Vorlagen zu gelangen. Nun erscheint die Entwicklungsoberfläche für PowerApps (siehe Bild 4). Hier klicken wir links oben auf die Schaltfläche **Datei**.

Klicken Sie diesen Befehl an, erscheint ein neuer Bereich, der verschiedene Möglichkeiten zum Erstellen einer PowerApp anbietet – und diesmal etwas konkreter mit der Angabe

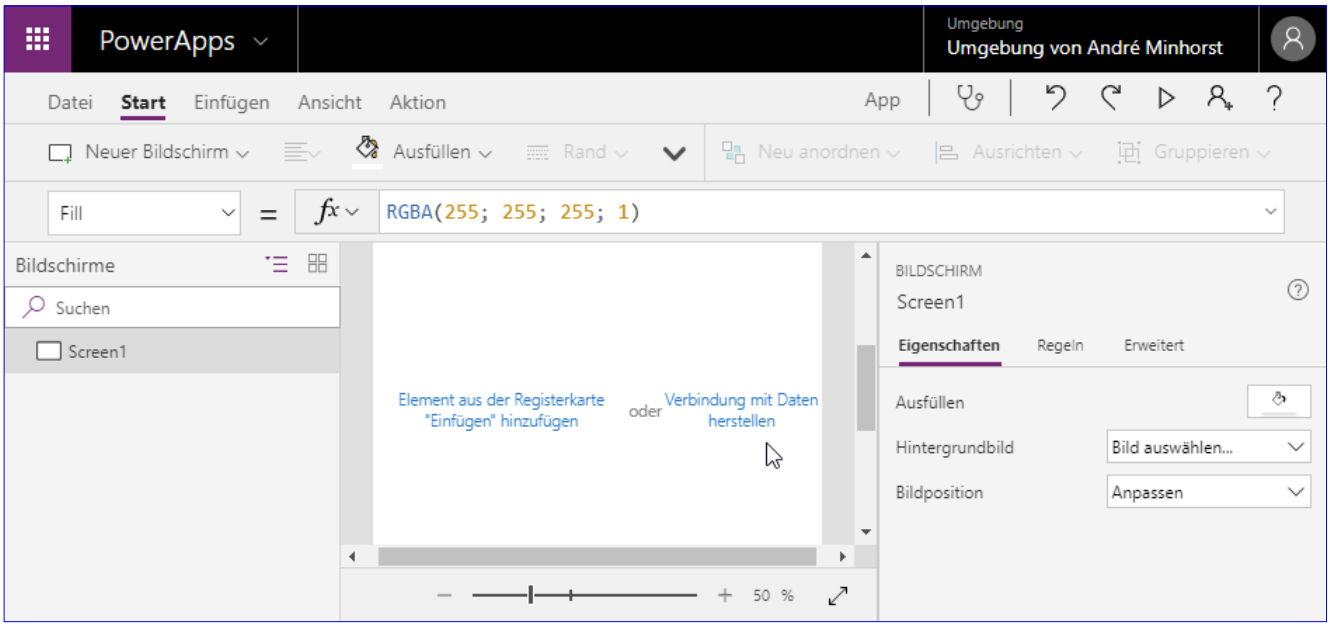


Bild 4: Entwicklungsumgebung für die PowerApps

verschiedener Datenquellen oder auch mit einer leeren App oder einer App-Vorlage. Es erscheint ein Bereich ähnlich dem Backstage-Bereich, den Sie von den Office-Anwendungen her kennen. Hier finden Sie auch einen Befehl namens **Neu** (siehe Bild 5). Hier klicken wir auf die Schaltfläche **Smartphonelayout** des Bereichs **App-Vorlagen**.

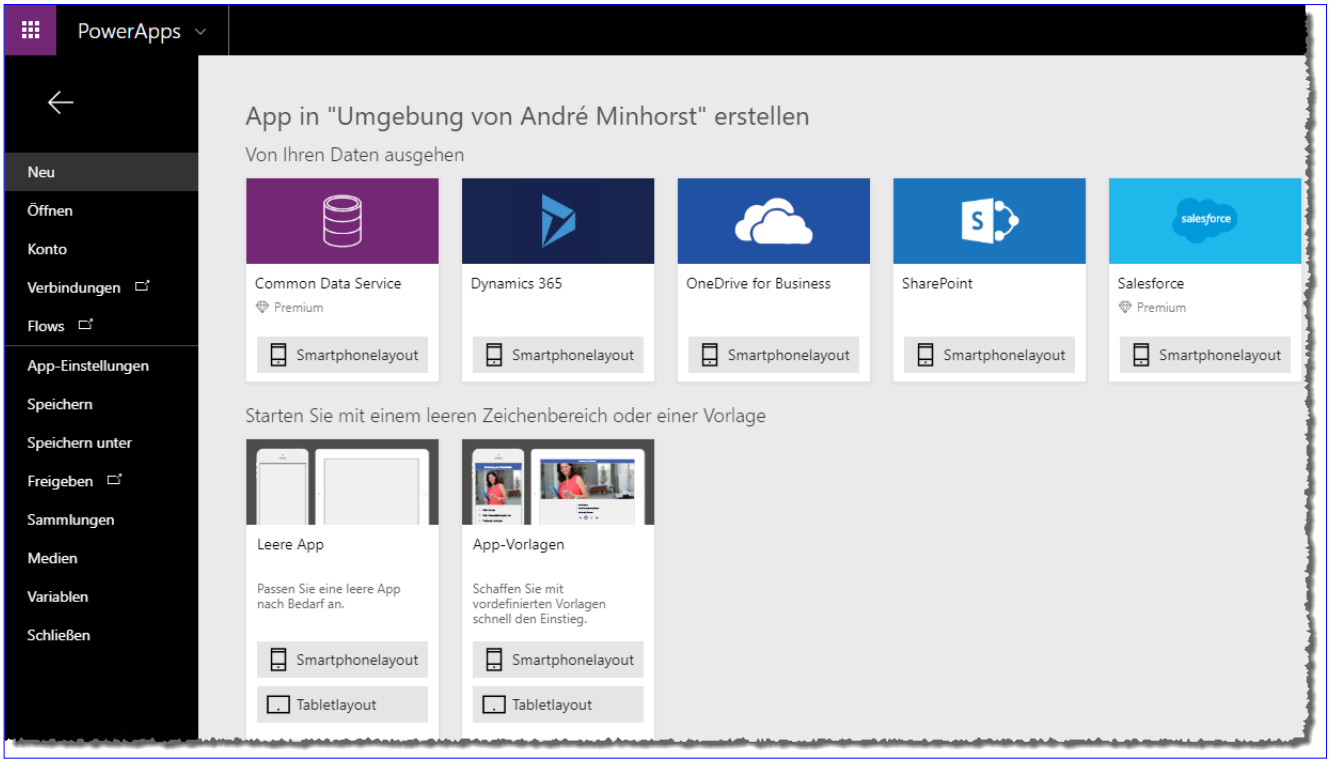


Bild 5: Backstage-Bereich mit dem **Neu**-Befehl

Vorlage auswählen

Wir entscheiden uns an dieser Stelle für den Eintrag **Budget Tracker** (siehe Bild 6). Außerdem müssen wir einen Speicherort für die Daten dieser App auswählen.

Klicken wir unten auf den Link **Auswählen**, erscheint ein Popup-Fenster mit Möglichkeiten wie **Dropbox**, **Google Drive** oder **OneDrive**.

Hier wälhen wir den Wert **Dropbox** und geben in einem weiteren Dialog den Zugriff auf die Dropbox frei.

Nach der Auswahl klicken wir auf die Schaltfläche **Verwendung**. Die Entwicklungsumgebung lädt anschließend die Vorlage, was einige Sekunden dauert.

Danach erscheint die erste Seite des Projekts, in diesem Fall die Seite **landing** (siehe Bild 7). Hier können Sie auf der linken Seite die einzelnen Elemente anklicken und diese mit den Eigenschaften auf der rechten Seite bearbeiten.

Bevor wir uns diese Features genauer ansehen, wollen wir die Anwendung einmal zum Laufen bringen.

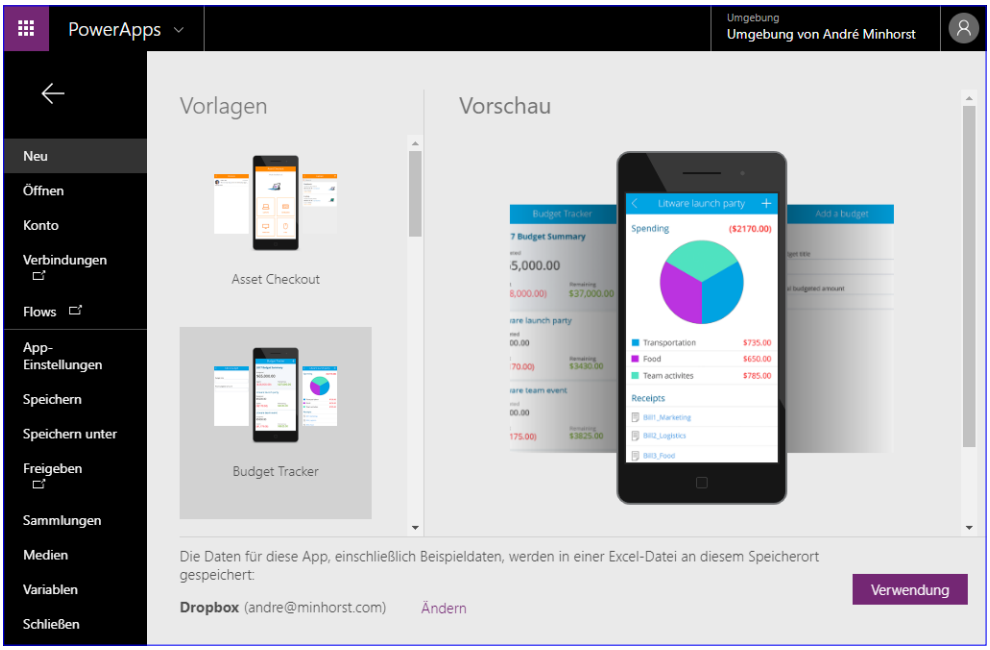


Bild 6: Auswahl der Vorlage

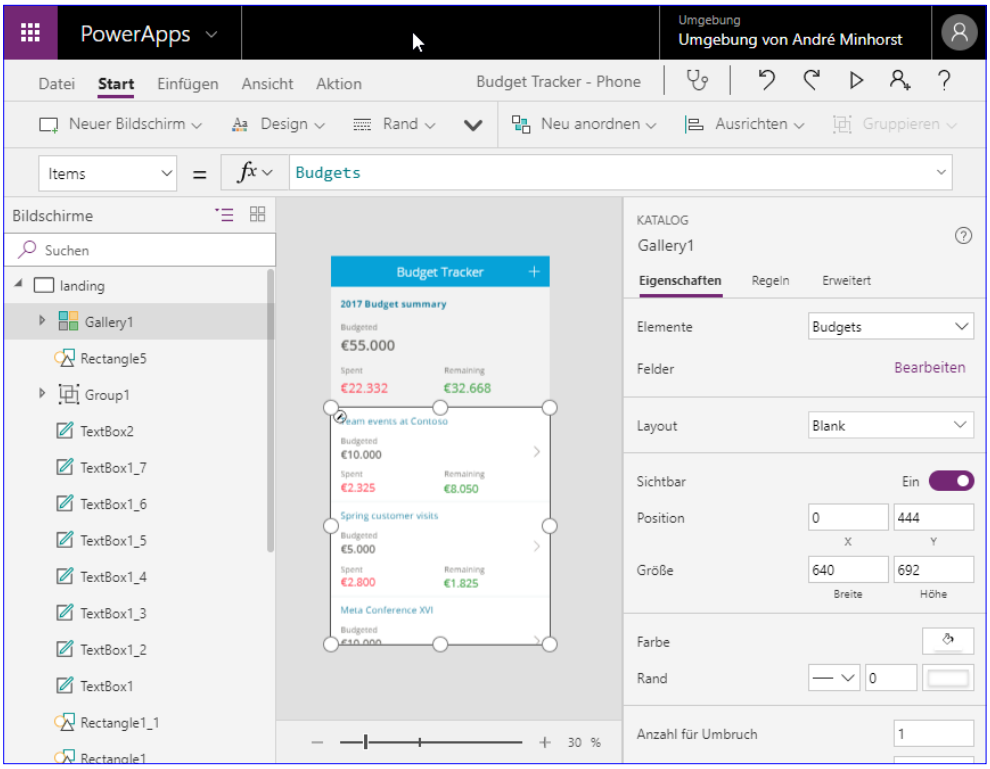


Bild 7: Bearbeiten der einzelnen Seiten der Vorlage

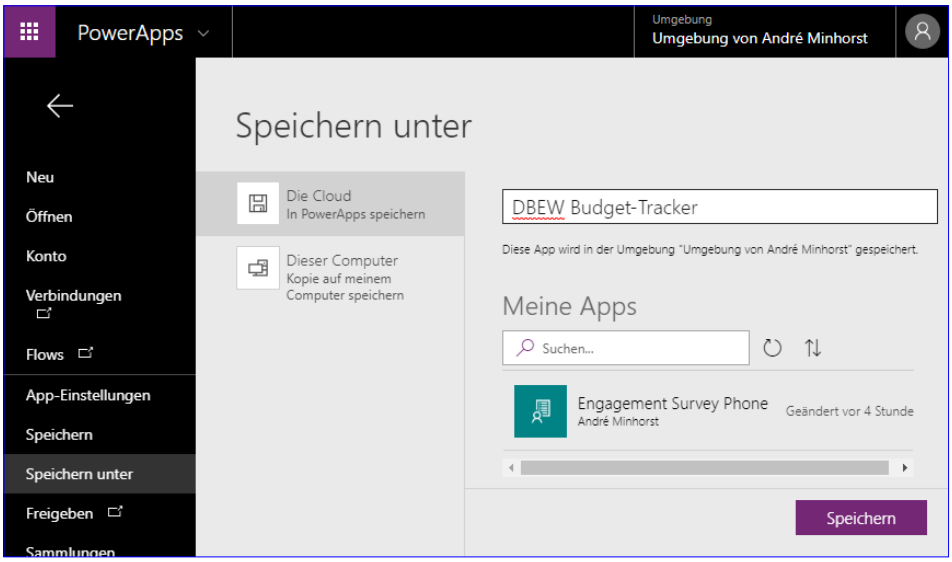


Bild 8: Speichern der PowerApp

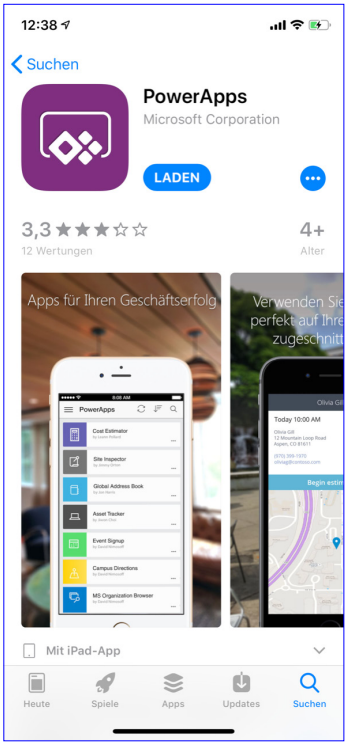


Bild 9: Installieren der PowerApps-App

Anwendung freigeben

Um die Anwendung zur Benutzung freizugeben, speichern wir diese nun in der Cloud. Dazu klicken Sie wieder links oben auf **Datei** und wählen dann links den Befehl **Speichern** aus. Im nun erscheinenden Bereich geben Sie den Namen ein, unter dem die App gespeichert werden soll – in diesem Fall **DBEW Budget-Tracker** (siehe Bild 8). Dieser Vorgang dauert wiederum einige Sekunden. Anschließend erscheint die Bestätigung, dass die App gespeichert wurde. Hier finden Sie dann auch eine Schaltfläche namens **Diese App freigeben**. Betätigen Sie diese Schaltfläche, erscheint ein neuer Dialog in einem neuen Tab des Internet-Browsers, auf dem Sie die App für weitere Personen freigeben können. Das wollen wir in diesem Moment nicht tun – wir wollen die App nur über unseren eigenen Account nutzen.

Damit können wir diesen Schritt auslassen, die App ist nämlich bereits für den erstellenden Benutzer freigegeben.

PowerApps auf Smartphone verfügbar machen

Wenn Sie die PowerApp nun auf Ihrem Smartphone verfügbar machen wollen, benötigen

Bild 10: Anmelden mit dem Benutzeraccount

Bild 11: Auswahl des zu verwendenden Kontos bei mehrere Konten

PowerApp mit Datenbank erstellen

Im Artikel »Einstieg in PowerApps« haben wir uns angesehen, wie wir eine der Vorlagen in eine PowerApp umwandeln. Diese verwendet eine Excel-Datei als Datenquelle. Das ist schon ganz cool, aber wir wollen natürlich noch einen Schritt weitergehen – mit einer SQL Server-Datenbank als Datenquelle. Natürlich funktionieren auch noch eine ganze Reihe anderer Datenquellen, zum Beispiel MySQL. Wir schauen uns an, wie Sie eine PowerApp erstellen und dieser direkt die gewünschte Datenquelle hinzufügen.

Voraussetzungen

Voraussetzung für das Nachvollziehen der Beispiele dieses Artikels ist das Vorhandensein einer SQL Server-Instanz, die über das Internet erreichbar ist – zum Beispiel eine solche, wie wir sie im Artikel **SQL Server-Datenbank ins Web mit SQL Azure** erstellt haben. Die zweite Voraussetzung ist – wie auch schon für Azure – ein Microsoft-Konto und die Anmeldung zumindest an die Community-Version der PowerApps. Mehr darüber erfahren Sie im Artikel **Einstieg in PowerApps**.

Erstellen der PowerApp

Wenn Sie die oben genannten Voraussetzungen erfüllt haben, können Sie eine neue, leere PowerApp erstellen. Dazu öffnen Sie die Webseite **powerapps.microsoft.com**. Hier melden Sie sich über den **Anmelden**-Link an. Danach landen Sie in der Entwicklungsumgebung für PowerApps, die Sie ja auch schon im Artikel **Einstieg in PowerApps** kurz kennengelernt haben.

Nach dem Anmelden finden Sie die Startseite vor, auf der Sie verschiedene Vorlagen zum Anlegen neuer PowerApps vorfinden. Wir wählen die Vorlage **Canvas-App ohne Vorlage** aus, und zwar für Smartphones (siehe Bild 1).

Den nun erscheinenden Dialog **Willkommen bei PowerApps Studio** schließen wir mit der Schaltfläche **Überspringen**. Danach finden Sie das PowerApps Studio wie in Bild 2 vor.

Verbindung mit Daten herstellen

Die Entwicklungsumgebung zeigt im Arbeitsbereich die beiden Möglichkeiten **Elemente aus der Registerseite "Einfügen" hinzufügen** und **Verbindung mit Daten her-**

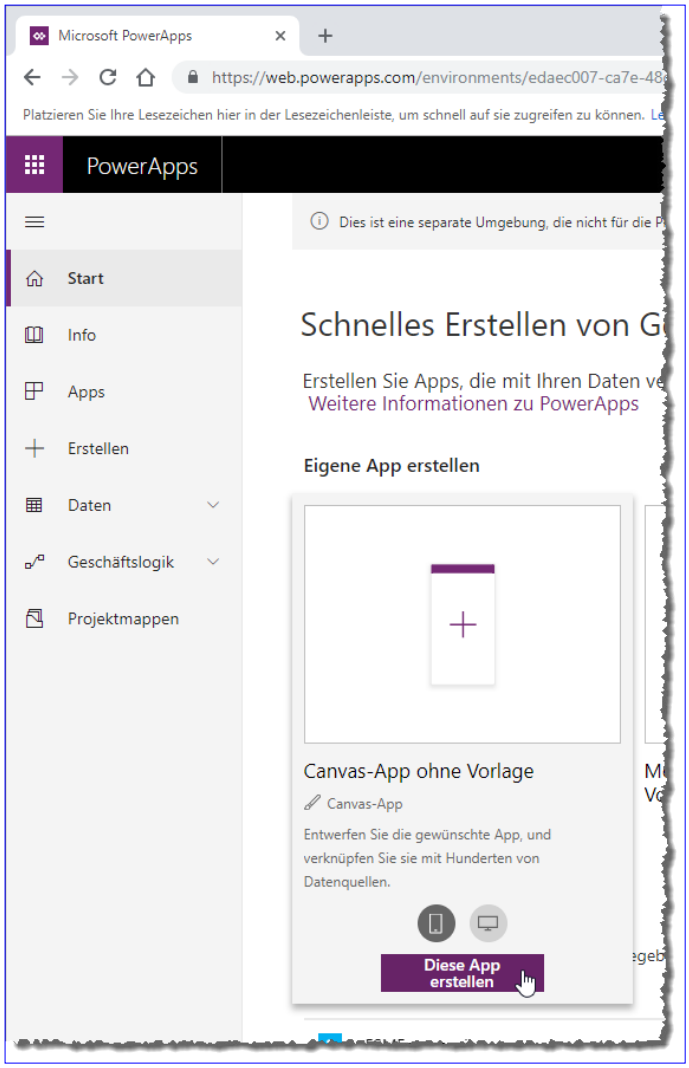


Bild 1: Erste Ansicht der Entwicklungsumgebung

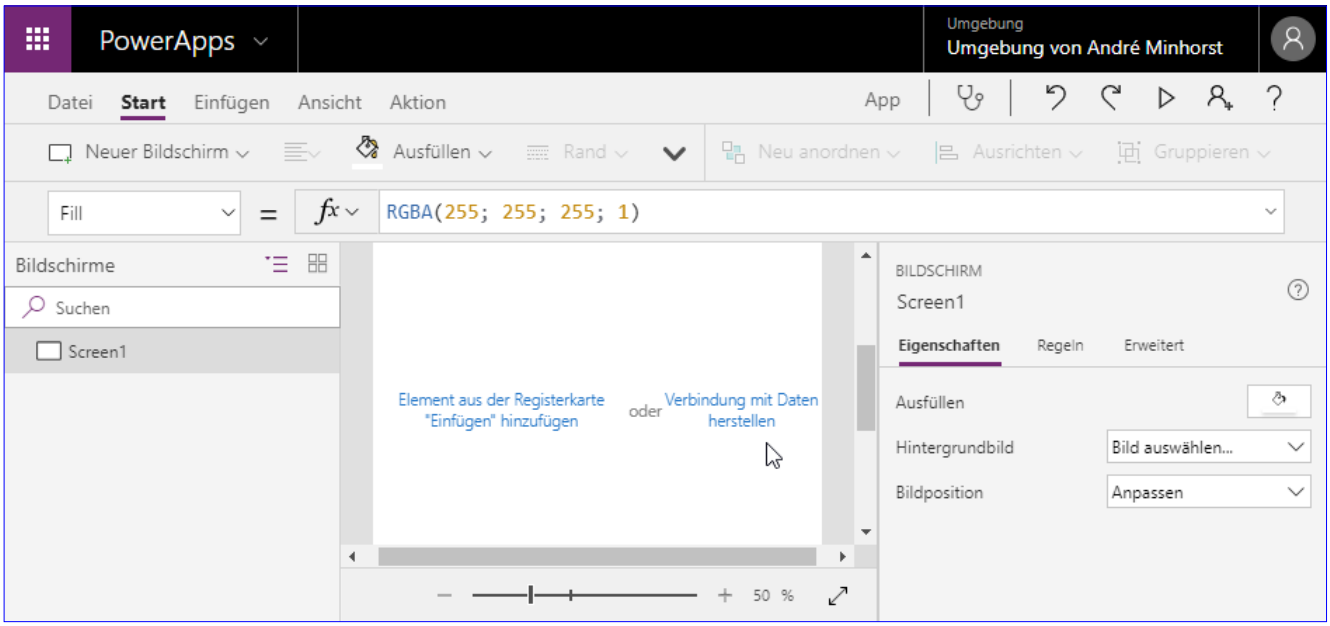


Bild 2: PowerApps-Studio

stellen an. Wir wollen eine Verbindung mit Daten herstellen, also klicken wir auf **Verbindung mit Daten herstellen**.

Hier bietet ein kleiner Dialog namens **Daten** die Möglichkeit, statische Daten aus Excel zu importieren (siehe Bild 3). Wir wollen aber direkt einen Schritt weitergehen und die Daten aus einer SQL Server-Tabelle einer Azure-Datenbank laden.

Also klicken wir auf **Neue Verbindung** und finden dann eine sehr lange Liste verschiedenster Datenquellen vor. Uns interessiert jedoch der SQL Server, den wir am schnellsten finden, indem wir **SQL** oben ins Suchfeld eingeben. Es erscheinen dann die Einträge aus Bild 4.

Wie Sie eine Azure-Datenbank anlegen, haben wir ja im Artikel **SQL Server-Datenbank ins Web mit SQL Azure** schon beschrieben. Dort sollten Sie sich einige Daten gemerkt haben wie die Adresse des SQL Servers, den Datenbanknamen, den Namen des Benutzers sowie das Kennwort des Benutzers.

Diese können Sie nun im folgenden Dialog eingeben, der genau diese Informationen abfragt (siehe Bild 5). Hier finden Sie auch noch die Option **Verbindung mithilfe eines lokalen Datengateways**. Sie können damit nach einigen weiteren Maßnahmen auch auf einen SQL Server zugreifen, der auf ihrem lokalen Rechner liegt, aber von außen zugreifbar sein

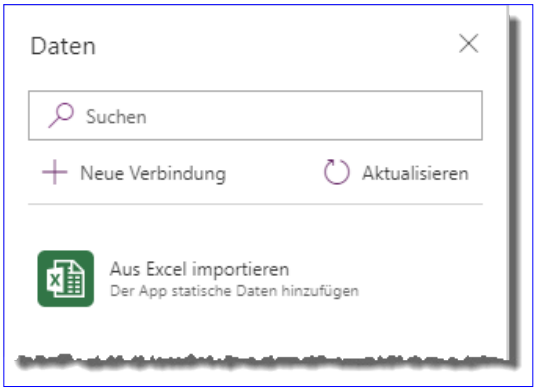


Bild 3: Auswahl der Datenquelle

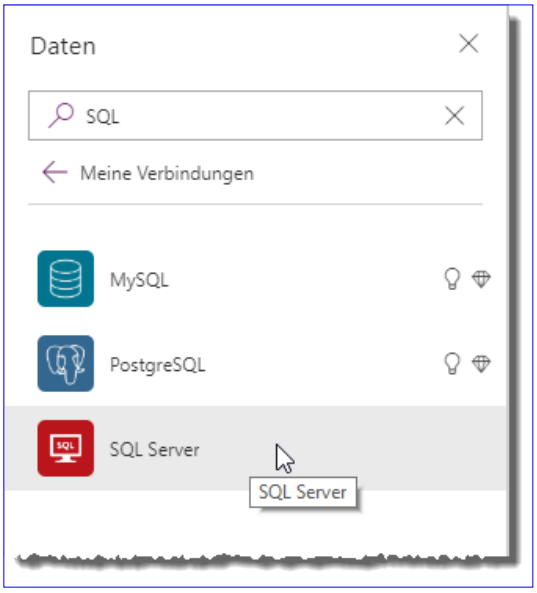


Bild 4: Filtern der Datenquellen

muss. Da diese Schritte relativ aufwendig sind, verzichten wir an dieser Stelle auf ihre Beschreibung. Wenn Sie kein Azure-Programm beginnen möchten, können Sie auch einen MySQL-Server etwa auf einem Webserver ansprechen, sollten Sie Zugriff auf einen solchen haben.

Nachdem Sie die Verbindungsdaten eingegeben haben und auf die Schaltfläche **Erstellen** geklickt haben, können Sie die Tabellen auswählen, mit der Sie die PowerApp verknüpfen wollen – in diesem Fall alle hier markierten Tabellen (siehe Bild 6).

Schließlich erscheint im gleichen Bereich eine Liste von Datenquellen, und zwar für jede Tabelle mit einem eigenen Eintrag (siehe Bild 7). Diese Elemente können wir nun über das Kontextmenü, das Sie mit der Schaltfläche mit den drei Punkten rechts neben dem Eintrag aufrufen, aktualisieren und auch wieder entfernen.

Nun haben wir zwar eine Reihe von Datenquellen in einer Liste, aber wie bekommen wir diese nun auf eine Bildschirmseite unserer Anwendung? Dazu wollen wir nur ein ganz einfaches Beispiel liefern.

Daten testweise ausgeben

Die erste und einzige Seite dieser PowerApp heißt **Screen1**. Diese könnten Sie nun umbenennen, aber für dieses kurze Beispiel lohnt es sich nicht. Um die Daten schnell in der PowerApp anzuzeigen, klicken Sie oben auf den Reiter **Einfügen** und dann auf **Datentabelle** (siehe Bild 8).

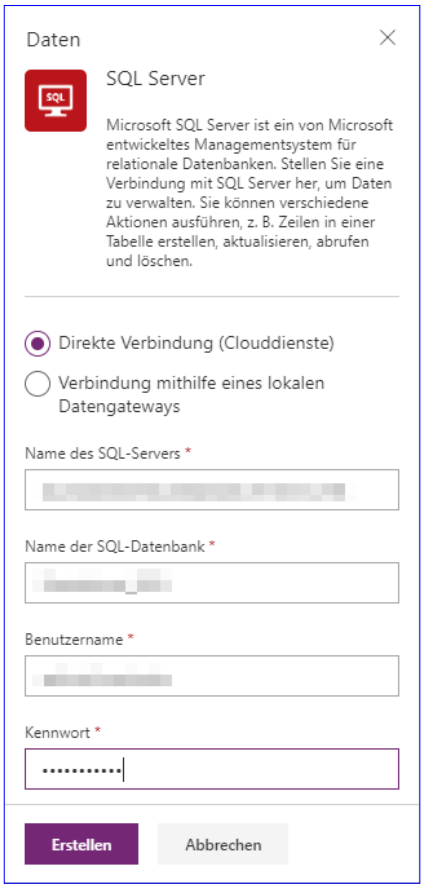


Bild 5: Eingabe der Verbindungsdaten

Bild 6: Auswahl der Tabellen

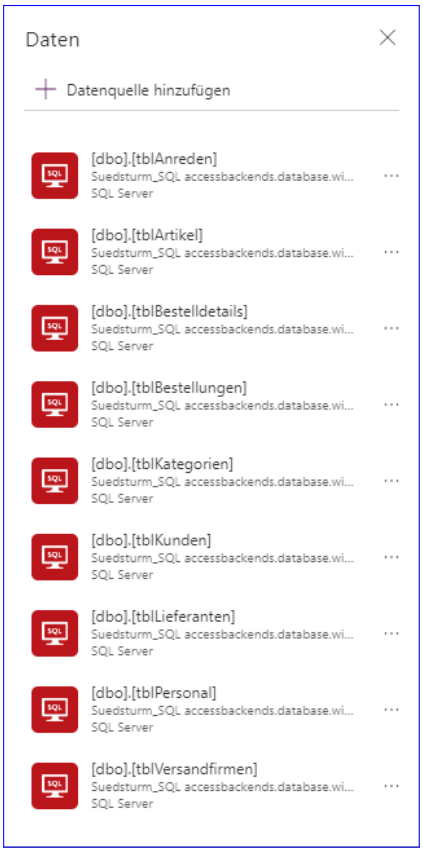


Bild 7: Datenquellen

PowerApps: Artikel verwalten

In den vorherigen Artikeln haben wir uns schon eine PowerApp anhand einer Vorlage angesehen und eine kleine, an die Daten einer Tabelle gebundene PowerApp programmiert. Nun gehen wir einen Schritt weiter und erstellen eine PowerApp mit den für die Anzeige von Daten wichtigen Steuerelementen und zeigen, wie Sie die Steuerelemente mit Daten füllen, zwischen verschiedenen Seiten navigieren und Daten anlegen, bearbeiten oder auch löschen.

Voraussetzungen

Um die Beispiele dieses Artikels nachvollziehen zu können, benötigen Sie eine SQL Server-, MySQL- oder PostgreSQL-Datenbank, die über das Internet erreichbar ist. Sie können dazu eine Datenbank von einem Webserver nehmen, den Sie selbst betreiben oder auch einen Microsoft Azure-Account mit einer SQL Server-Datenbank anlegen (siehe auch im Artikel **SQL Server-Datenbank ins Web mit SQL Azure**).

Außerdem setzen wir auf der PowerApp auf, die wir im Artikel **PowerApp mit Datenbank erstellen angelegt** haben. Dort haben wir der PowerApp zum Beispiel einige Tabellen der Datenbank **Suedsturm_SQL** eines SQL Servers hinzugefügt. Den Start machen wir über die Webseite **powerapps.microsoft.com**, wo Sie sich mit Ihren Zugangsdaten einloggen und die gewünschte Anwendung aufrufen oder neu anlegen.

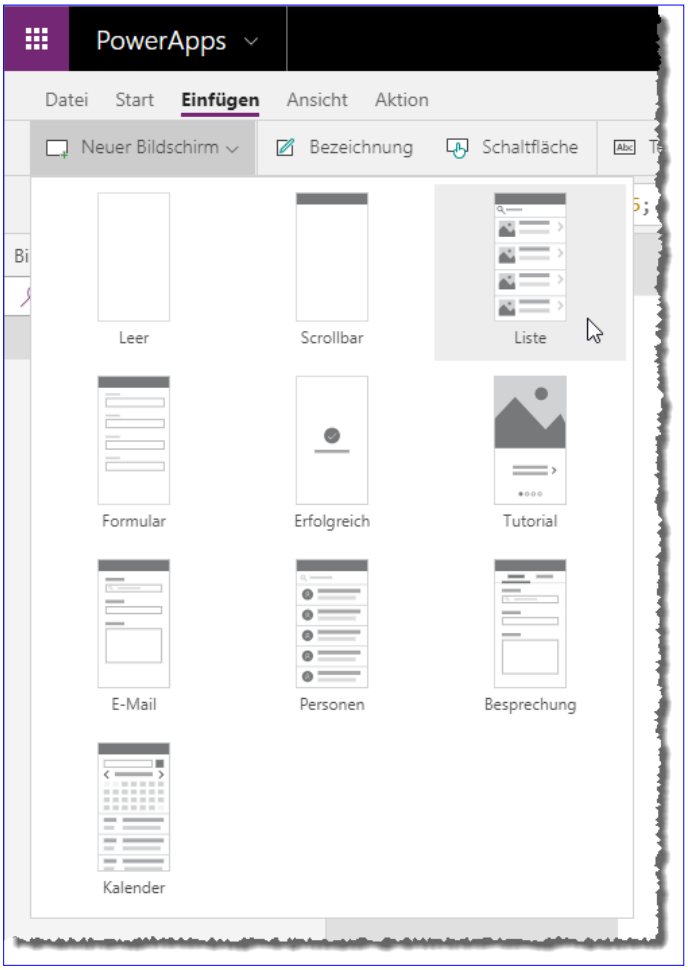


Bild 1: Einfacher Link

Liste erstellen

Wir beginnen mit einer ersten Seite, die eine Liste von Elementen einer unserer Tabellen enthalten soll. Dazu wählen wie in der Entwicklungsumgebung unter **Einfügen** den Eintrag **Neuer Bildschirm\|Liste** aus (siehe Bild 1).

Danach zeigt die Entwicklungsumgebung, die wir uns hier nochmal genauer anschauen, die neue Seite direkt mit den von der Vorlage vorgesehenen Elementen an (siehe Bild 2). Diese sehen wir im mittleren Bereich. Die einzelnen Elemente dieser Seite finden wir im Bereich links unter **Screen2**. Das Element **BrowseGallery1** ist das eigentliche Element zur Anzeige mehrerer Datensätze der Datenquelle, das im Prinzip wie eine Art Endlosformular wie unter Access arbeitet. Die Anzeige im Entwurf ist etwas irreführend, denn eigentlich enthält das Steuerelement jedes der abgebildeten untergeordneten Steuerelemente nur einmal. Hier werden allerdings direkt Platzhalterdaten abgebildet.

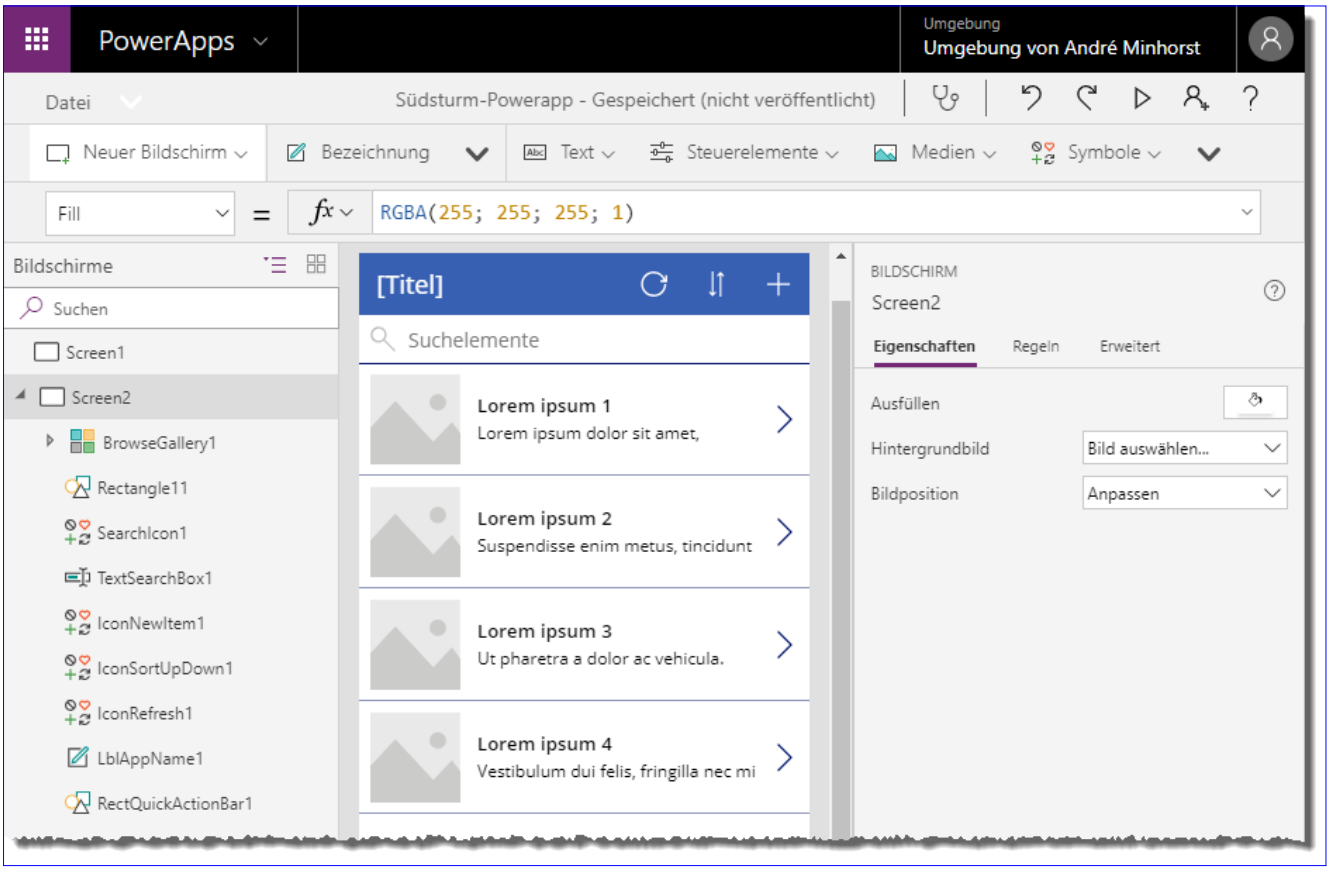


Bild 2: Neue Seite in der Entwicklungsumgebung

Wenn Sie auf der linken Seite **BrowseGallery1** aufklappen, finden Sie die für ein Element dieser Endlosansicht angelegten Steuerelemente (siehe Bild 3) – einen Separator, ein Image, eine Schaltfläche sowie zwei Bezeichnungsfelder für Texte aus der Datenquelle.

Hier können wir uns schon einiges abschauen, aber wir wollen mit einer komplett leeren Seite beginnen. Dazu fügen wir eine neue Seite hinzu, die wir diesmal auf Basis der Formularvorlage **Leer** erstellen. Dementsprechend ist die Seite auch komplett leer. Wir wollen eine Übersicht aller Artikelnamen anzeigen, für die es oben ein Textfeld für eine Schnellsuche gibt. Dieses

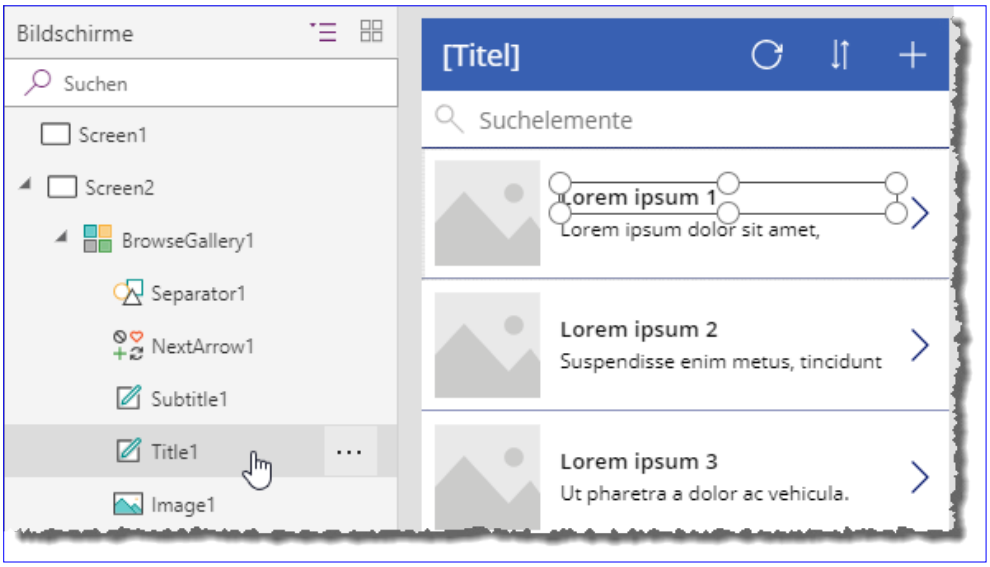


Bild 3: Untersuchen der einzelnen Elemente der Gallery

soll nach der Eingabe eines jeden Zeichens das Suchergebnis filtern.

Dazu fügen wir der neuen Seite, die wir zuvor in **Artikeluebersicht** umbenennen, als Erstes ein leeres Katalog-Element hinzu, das wir im Bereich **Einfügen** unter **Katalog|Leerzeichen vertikal** vorfinden (siehe Bild 4).

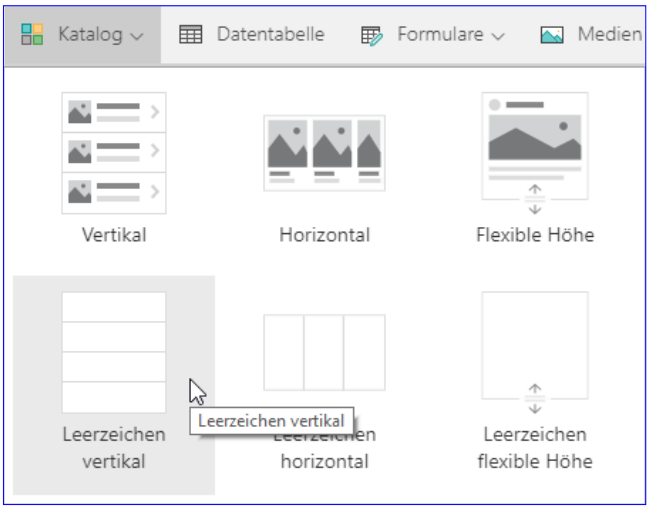


Bild 4: Einfügen eines leeren Katalogs mit vertikalen Elementen

Danach legen wir noch ein Bezeichnungsfeld namens **lblSuche** und ein Textfeld namens **txtSuche** über dem Katalog-Element namens **galArtikel** an (siehe Bild 5).

Datenquelle für den Katalog hinzufügen

Die Datenquelle fügen wir nun hinzu, indem wir das Katalog-Element markieren und dann rechts in den Eigenschaften die gewünschte Tabelle auswählen – in diesem Fall **tblArtikel** (siehe Bild 6). Dies legt auch noch den Wert der Eigenschaft **Items** im Bereich **Daten** der Registerseite **Erweitert** auf **'[dbo].[tblArtikel]'** fest.

Dann fügen wir bei markiertem Katalog-Element ein weiteres Bezeichnungsfeld hinzu, dass dann in das Katalog-Element eingefügt wird. Das Bezeichnungsfeld kann – im Gegensatz zu Access – auch an eine Datenherkunft gebunden werden und dient so etwa zur Anzeige von gebundenen Werten, die aktuell nicht vom Benutzer bearbeitet werden sollen. Dazu würden wir dann das Textfeld nutzen. Es geschieht etwas Überraschendes: Sobald wir im Bereich **Einfügen** auf das Element **Bezeichnung** geklickt haben, um dieses hinzuzufügen, enthält dieses bereits die Werte des Feldes **Artikelname** der Tabelle **tblArtikel**!

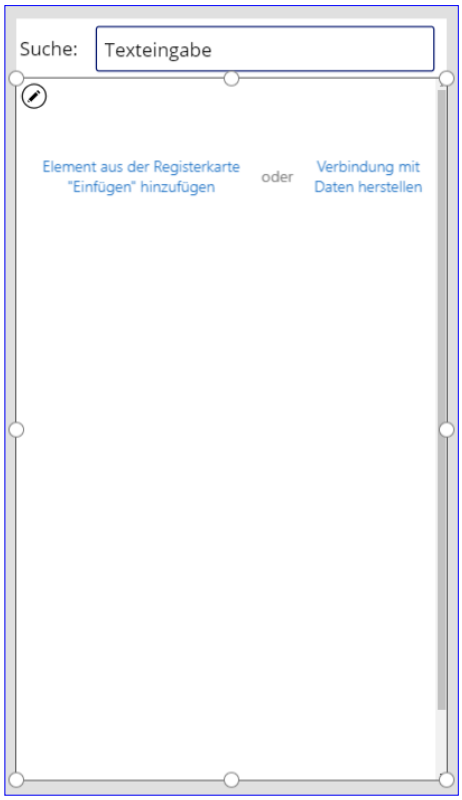


Bild 5: Steuerelemente für die Artikelübersicht

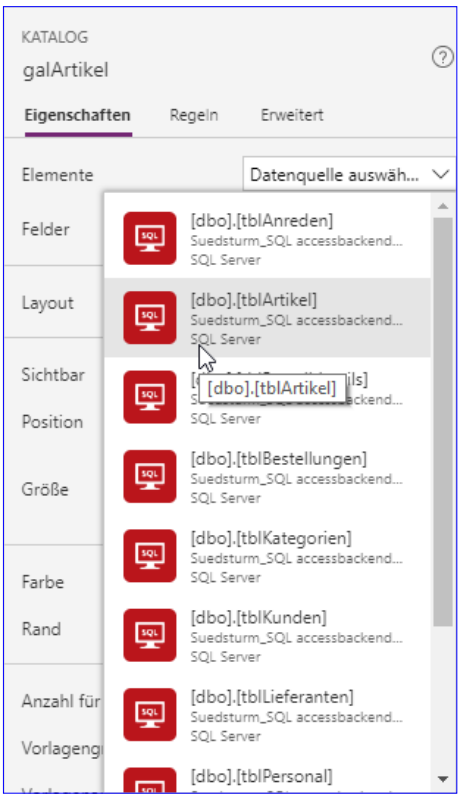


Bild 6: Auswahl der Datenquelle

Diese Anzeige sieht, nachdem wir die Höhe des obersten Bereichs des Katalog-Elements etwas verkleinert haben, etwa wie in Bild 7 aus. Offensichtlich hat

die Entwicklungsumgebung das erste Textfeld der Datenquelle ermittelt und dieses als Datenherkunft für das Bezeichnungsfeld eingestellt. Schauen wir also nach, welche Eigenschaft mit einem entsprechenden Wert ausgestattet wurde.

Die passende Eigenschaft finden wir dann auf der Seite **Erweitert** der Eigenschaften des Bezeichnungsfeldes. Die Bindung erfolgt über die Eigenschaft **Text**, der automatisch der Wert **ThisItem.Artikelname** zugewiesen wurde (siehe Bild 8). **ThisItem** referenziert dabei offensichtlich die für die Eigenschaft **Elemente** des Katalog-Steuerelements angegebene Datenquelle.

Bild 7: Anzeige der Daten in der Vorschau

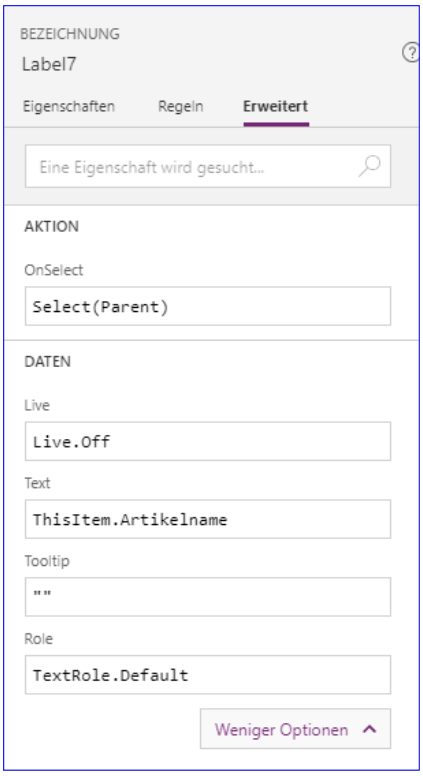


Bild 8: Eigenschaft zum Einstellen der Datenherkunft eines Steuerelements

Suche hinzufügen

Nun wollen wir das Such-Textfeld in die Anzeige der Daten im Katalog-Steuerelement einbinden. Dazu müssen wir den Wert der Eigenschaft **Items** des Katalog-Steuerelements anpassen. Bisher hat diese Eigenschaft den Wert **'[dbo].[tblArtikel]'**.

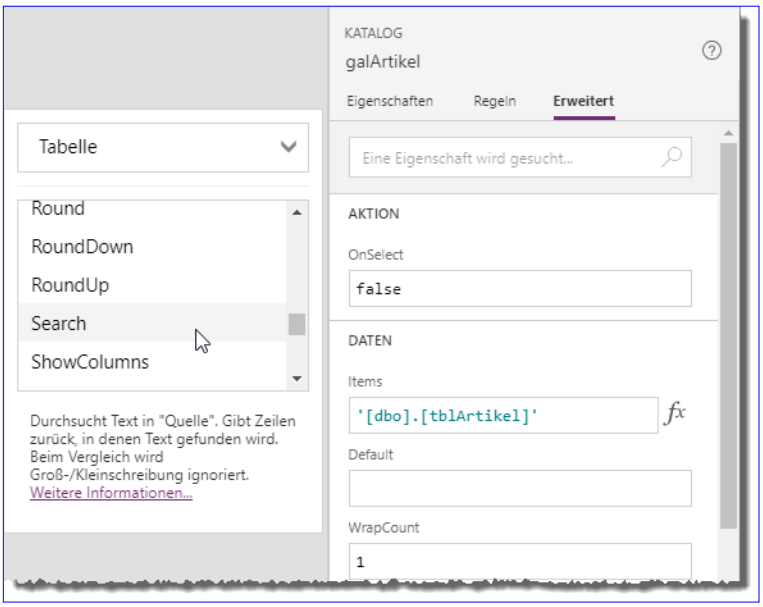


Bild 9: Auswählen einer Funktion zum Suchen von Daten

Nun haben PowerApps einen speziellen Dialog von Ausdrücken und Funktionen, die am ehesten mit denen von Excel zu vergleichen sind. Diese können Sie einfach auswählen, wenn Sie die Eigenschaft **Items** markieren und dann auf die rechts davon erscheinende Schaltfläche mit dem Funktions-Icon klicken. Es erscheint dann links daneben ein Bereich zur Auswahl der zur Verfügung stehenden Funktionen. Hier wählen wir oben den Eintrag **Tabelle** aus und scrollen dann zur Funktion **Search** (siehe Bild 9).

Klicken wir doppelt auf diesen Eintrag, wird der erste Teil der Funktion schon einmal in die Eigenschaft **Items** eingetragen (siehe Bild 10). Außerdem erscheint direkt ein Popup mit Vorschlä-

gen für den ersten Parameter, für den wir die Tabelle **dbo.tblArtikel** auswählen. Die **Search**-Funktion erwartet mindestens drei Parameter:

- Name der Datenquelle, hier **'[dbo].[tblArtikel]'**
- Steuerelement, das den Vergleichswert liefert. Dabei muss auch die Eigenschaft angegeben werden, die den Wert liefert, in diesem Fall also etwa **txtSuche.Text**.
- Schließlich folgen ein oder mehrere Felder der Tabelle, in denen nach dem mit dem zweiten Parameter angegebenen Wert gesucht werden soll.

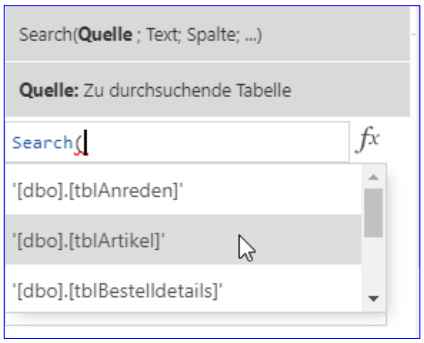


Bild 10: Eingabe der Funktionsparameter

Unser Suchausdruck sieht nun wie folgt aus (siehe Bild 11):

```
Search('[dbo].[tblArtikel]';txtSuche.Text;"Artikelname")
```

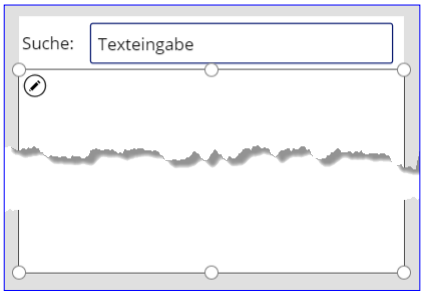


Bild 11: Leeres Suchergebnis

Warum aber sind nach der Eingabe dieses Ausdrucks plötzlich alle Einträge des Katalog-Elements leer? Die Lösung ist einfach: Für das Textfeld ist der Standardwert **Texteingabe** festgelegt. Wenn Sie doppelt in das Textfeld klicken und dieses leeren, erscheinen auch alle Einträge der Tabelle **tblArtikel** wieder (siehe Bild 12).

Wenn wir nun einen Suchbegriff eingeben, wird die Liste der Artikel auch direkt gefiltert (siehe Bild 13).

Damit demnächst immer direkt alle Einträge angezeigt werden, leeren Sie die Eigenschaft **Default** des Steuerelements **txtSuche** am besten noch. Das funktioniert allerdings nicht durch vollständiges Leeren, sondern Sie müssen eine leere Zeichenkette (**""**) für diese Eigenschaft eintragen.

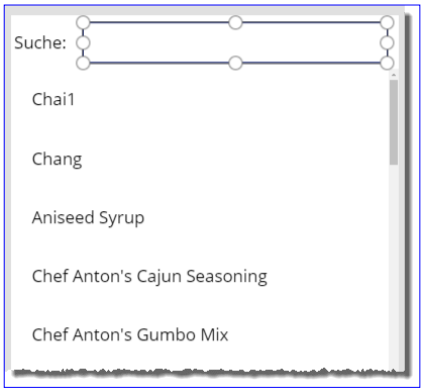


Bild 12: Leeres Suchfeld

Suchfunktion auf dem Smartphone testen

Das wollen wir nun auf dem Smartphone ausprobieren. Auf diesem muss dazu die App **PowerApps** installiert sein, außerdem melden Sie sich unter den gleichen Daten an, unter denen Sie auch die PowerApp erstellt haben.

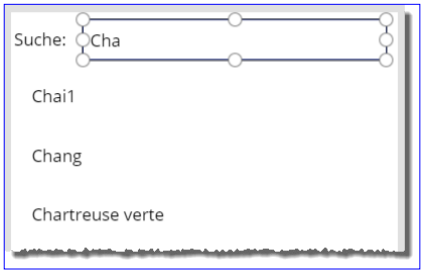


Bild 13: Suchergebnisse

Bevor wir zum Smartphone wechseln, müssen wir allerdings noch folgende Schritte durchführen:

- Unter Umständen ist die neu erstellte Seite nicht die erste Seite in der Liste der Seiten. In diesem Fall bewegen Sie diese mit dem Kontextmenü-Befehl **Nach oben** an die erste Position.
- Danach müssen wir die App noch speichern. Das gelingt ganz einfach mit dem Befehl **DateiSpeichern**.

- Schließlich veröffentlichen Sie die PowerApp noch mit der Schaltfläche **Veröffentlichen**, die nach dem Speichern im gleichen Bereich erscheint.

Danach können Sie die Anwendung in der PowerApps-App auf Ihrem Smartphone öffnen – und sie funktioniert wie gewünscht! Ich finde das ein sehr bemerkenswertes Ergebnis gemessen daran, dass wir – wenn die Schritte etwas eingeübt sind – wohl nur wenige Minuten für die Erstellung dieser App benötigen werden.

Wenn Sie kein Smartphone zur Hand haben, können Sie die App übrigens auch direkt auf dem Rechner ausprobieren. Dazu klicken Sie auf der Übersichtsseite für die PowerApps einfach auf den Eintrag mit dem Namen der zu testenden App.

Details anzeigen

Als nächstes wollen wir der Katalog-Seite Elemente hinzufügen, die es dem Benutzer ermöglichen, die Details zu jedem der angezeigten Datensätze zu öffnen. Dazu brauchen wir natürlich nicht nur eine entsprechende Schaltfläche je Eintrag, sondern auch noch eine Seite zur Anzeige der Details. Diese wollen wir zuerst erstellen.

Dazu fügen wir über den Befehl **Einfügen|Neuer Bildschirm** einen weiteren neuen Bildschirm hinzu, wieder mit der Vorlage **Leer**. Diese Seite nennen wir **Artikeldetails**.

Dann fügen wir über den Menüeintrag **Formulare|Anzeige** ein Formular zur Seite hinzu, das die Anzeige der Details zum gewählten Artikel erlauben soll. Auch dieses zeigt wieder an, dass es noch mit Daten verbunden werden muss. Also wählen wir als Datenquelle wieder die Tabelle **[dbo].[tblArtikel]** aus (siehe Bild 14) und klicken auf die Schaltfläche **Verbinden**.

Danach wird es richtig komfortabel: Für die Eigenschaft **Felder** finden Sie einen Link namens **Felder bearbeiten** (siehe Bild 15). Klicken Sie auf diesen Link, erscheint links davon ein neuer Bereich namens **Felder**. Hier klicken Sie auf den Link **Feld hinzufügen**, um eine Liste aller Felder der unter **Datenquelle** angegebenen Tabelle anzuzeigen. Sie können diese dann bequem anhaken.

Klicken Sie dann auf die Schaltfläche **Hinzufügen**, fügt die Anwendung für jedes Feld ein paar Elemente zum Entwurf hinzu. Dass es sich um mehrere Elemente handelt, können Sie am besten an der Liste der Steuerelemente auf der rechten Seite erkennen (siehe Bild 16). Hier erscheint für jedes Feld ein Steuerelement des Typs **Karte**, das jeweils zwei Elemente des Typs

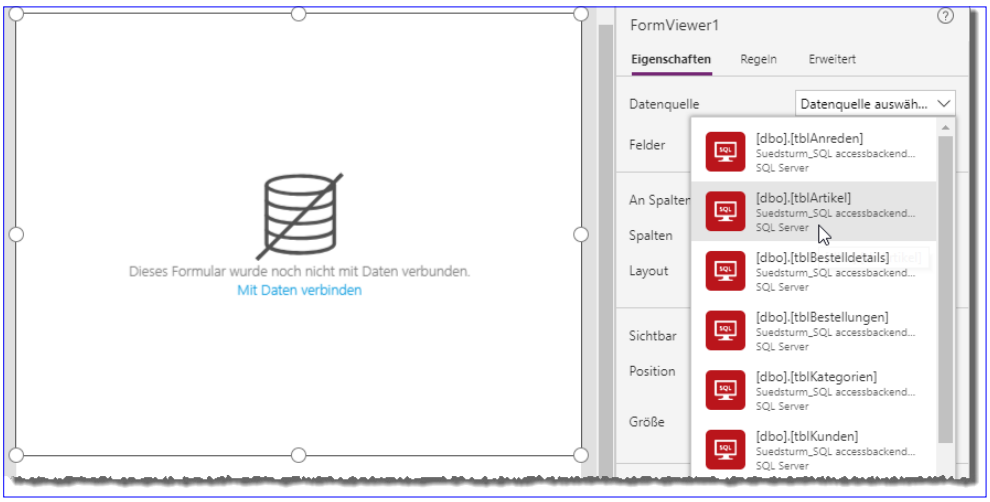


Bild 14: Hinzufügen der Daten zum Formular

Bezeichnung enthält. Das erste Bezeichnungs-Steuerelement zeigt dabei den Feldnamen an, das zweite den Inhalt des Feldes.

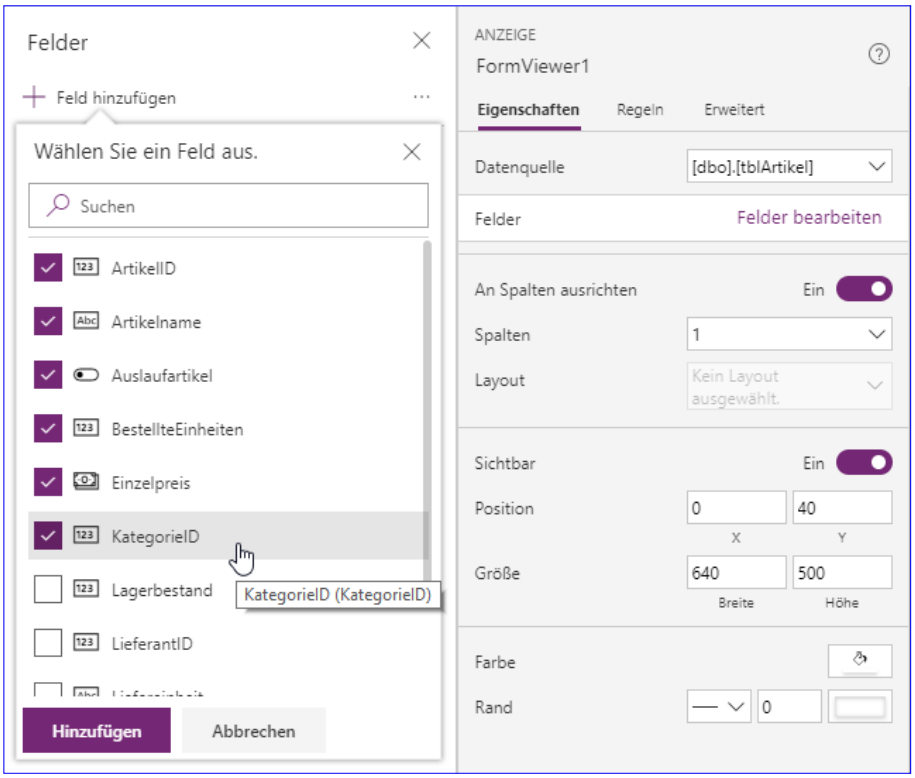


Bild 15: Auswählen der Felder

Damit die Felder wissen, was sie anzeigen müssen, enthält das **Karte**-Element ein paar passende Eigenschaften (siehe Bild 17):

- **DataField**: Feld der Tabelle, an welches das Steuerelement gebunden ist
- **DisplayName**: Anzuzeigender Text des Bezeichnungsfeldes
- **Required**: Gibt an, ob die Eingabe in das Feld erforderlich ist.
- **Default**: Standardwert, der aus dem tatsächlichen Wert des Feldes für den angezeigten Datensatz gewonnen wird

Nun soll die Detailansicht eines Artikels natürlich auch einen Artikel anzeigen, was aber noch nicht geschieht – aktuell erscheinen nur die Bezeichnungsfelder mit den Feldnamen.

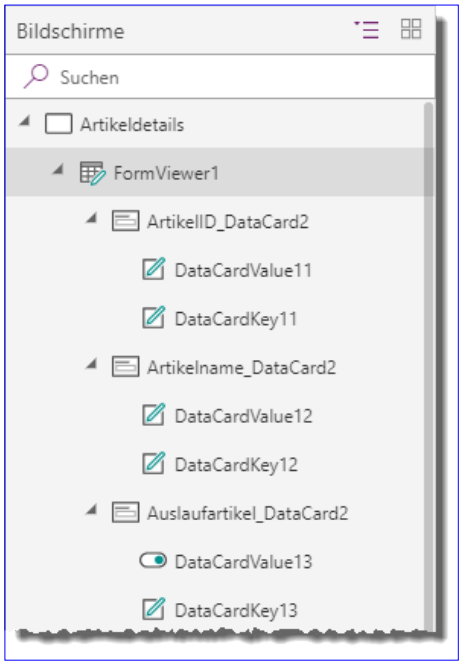


Bild 16: Steuerelemente zur Anzeige von Daten

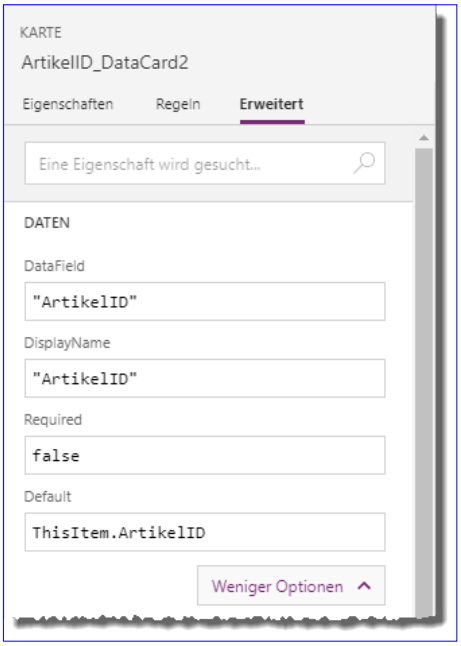


Bild 17: Eigenschaften dieser Steuerelemente

Um Daten anzuzeigen, müssen wir dem Detailformular noch mitteilen, welchen Datensatz der Datenquelle es anzeigen soll. Dies stellen wir mit der Eigenschaft **Item** ein, die wir im **Eigenschaften**-Bereich **Erweitert** des **Anzeige**-Elements finden. Hier tragen wir den Wert **First('[dbo].[tblArtikel]')** ein (siehe Bild 18). Der Ausdruck

verwendet die Funktion **First**, um den ersten Datensatz der Tabelle **tblArtikel** zu ermitteln.

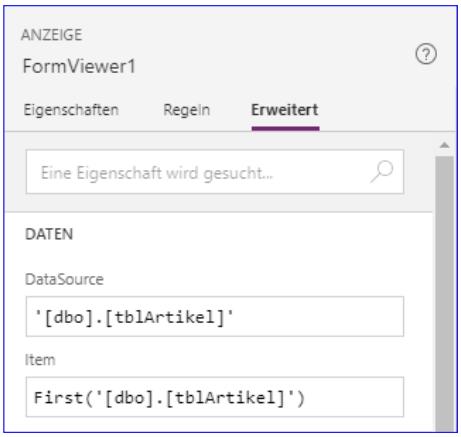


Bild 18: Angabe des zu ladenden Datensatzes

In der Entwurfsansicht sehen die Steuerelemente so aus, als ob sie zu teilweise nicht hoch genug wären, um den Inhalt anzuzeigen. Wenn Sie allerdings die Seite an die oberste Position verschieben und dann die PowerApp speichern, veröffentlichen und im Smartphone starten, erscheint diese dort wie gewünscht (siehe Bild 19).

Hier gibt es noch einige unschöne Dinge wie etwa die Anzeige der Fremdschlüsselwerte bei den Fremdschlüsselfeldern statt der Werte aus den Lookup-Tabellen und dem Wert **Aus** für das Feld **Auslaufartikel**, aber darauf kommen wir später zurück, wenn wir ein Formular zum Bearbeiten eines Datensatzes erstellen. Um etwa die Daten aus den Tabellen **tblLieferanten** und **tblKategorien** hier anzuzeigen, könnten wir eine SQL Server-View erstellen, welche die Daten der drei Tabellen zusammenführt. Wenn wir die Daten jedoch bearbeiten wollen, was wir weiter unten demonstrieren, müssen wir ohnehin mit Auswahlfeldern arbeiten.

Details vom Katalog aus aufrufen

Die nächste Aufgabe ist, die Details zu einem Datensatz vom Katalog aus aufzurufen. Dazu fügen wir dem obersten Element des Katalogs aus der Liste der Symbole den Pfeil nach rechts hinzu (siehe Bild 20). Ob Sie ein Element immer direkt im Katalog-Element angelegt haben, erkennen Sie daran, dass es direkt in allen Kopien dieses Katalogelements für die übrigen Datensätze erscheint.

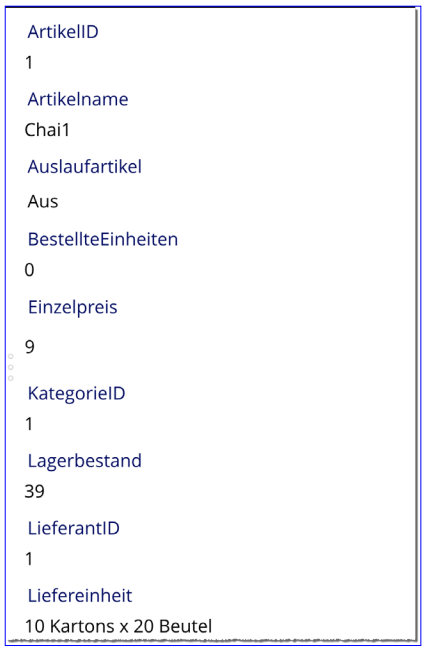


Bild 19: Anzeige des ersten Datensatzes im Smartphone

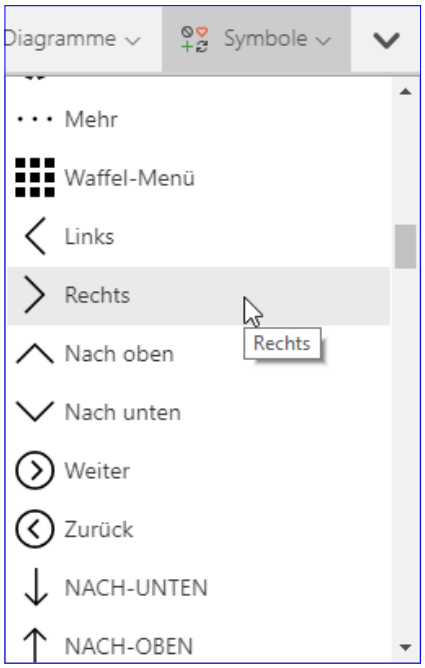


Bild 20: Liste der Symbole

Bild 21: Vorschau des Katalogs

Powerapps: Datensatz per DropDown auswählen

Eine oft verwendete Darstellung ist die, bei der sich oben im Fenster ein Kombinationsfeld befindet, mit dem der Benutzer dann den Datensatz auswählt, dessen Detaildaten in den übrigen Steuerelementen im unteren Bereich angezeigt werden sollen. Dieser Artikel zeigt, aufbauend auf den vorherigen Artikeln, wie Sie einen Bildschirm erzeugen, der im oberen Bereich ein solches Kombinationsfeld liefert und im unteren Bereich die Daten des gewählten Datensatzes anzeigt. Die Daten stammen dabei aus der Kunden-Tabelle der Beispieldatenbank Suedsturm_SQL.

Wir setzen auf unserem PowerApps-Projekt auf, das wir im vorherigen Artikel bereits begonnen haben. Wir verwenden also die Datenbank **Suedsturm_SQL** als Datenquelle.

Um nicht immer wieder die neu hinzufügten Bildschirmseiten nach oben im Projekt verschieben zu müssen, um diese direkt beim Öffnen der PowerApp testen zu können, haben wir nun einen Startbildschirm erstellt. Dieser sieht wie in Bild 1 aus. Für die Schaltfläche **Kundenauswahl** haben wir für die Eigenschaft **OnSelect** den folgenden Ausdruck festgelegt:

```
Navigate(Kundenauswahl;
ScreenTransition.Fade)
```

Damit öffnen wir per Mausklick auf die Schaltfläche einen weiteren Bildschirm namens **Kundenauswahl**. Den fügen wir mit dem Befehl **Neuer Bildschirm\|Leer** zum Projekt hinzu.

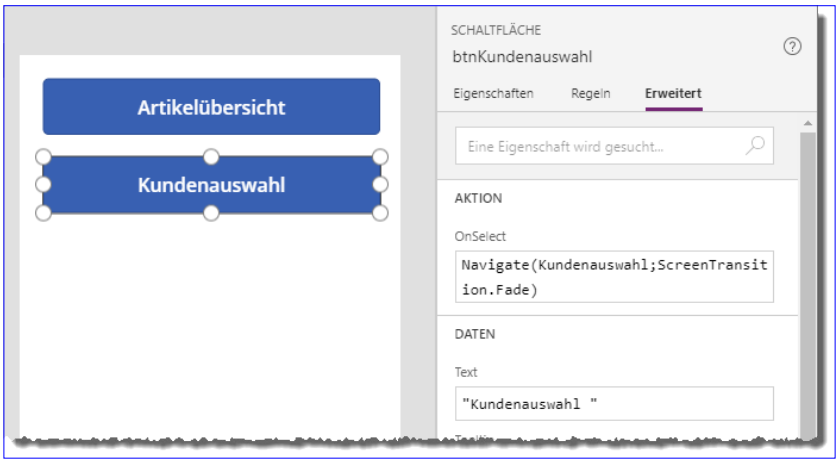


Bild 1: Hinzufügen einer Schaltfläche zum Öffnen einer weiteren Bildschirmseite

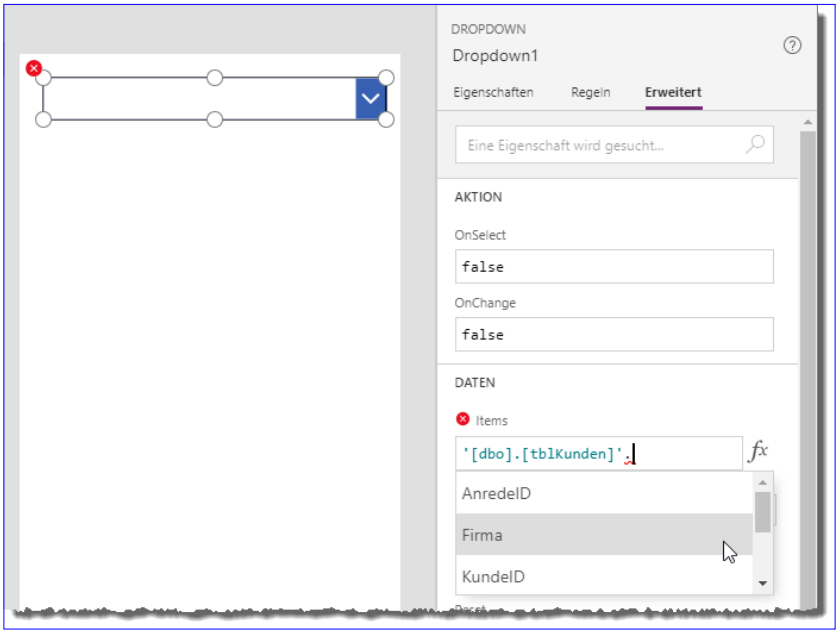


Bild 2: Anlegen des Dropdown-Steuerelements

Dropdown hinzufügen

Für die Auswahl der anzuzeigenden Daten verwenden wir das **Dropdown**-Steuerelement, das wir mit dem Befehl **Steuerelemente\|Dropdown** zur Bild-

schirmseite **Kundenauswahl** hinzufügen. Damit dieses beispielsweise die Werte des Feldes Firma der Tabelle **tblKunden** anzeigt, tragen Sie den Wert **'[dbo].[tblKunden]'** in die Eigenschaft **Items** ein und fügen noch einen Punkt hinzu.

Danach bietet die Benutzeroberfläche eine Liste der enthaltenen Felder an, die Sie so einfach auswählen und festlegen können (siehe Bild 2). Außerdem ändern wir noch den Namen des Dropdown-Steuerelements in **drpKunden**.

Interessanterweise werden die Daten hier nicht direkt zur Entwurfszeit angezeigt, wie es in den anderen Artikeln schon der Fall war. Also probieren wir das Dropdown-Feld im Smartphone aus oder im Webplayer. Dazu speichern wir die aktuelle Version der App mit dem Befehl **DateiSpeichern**.

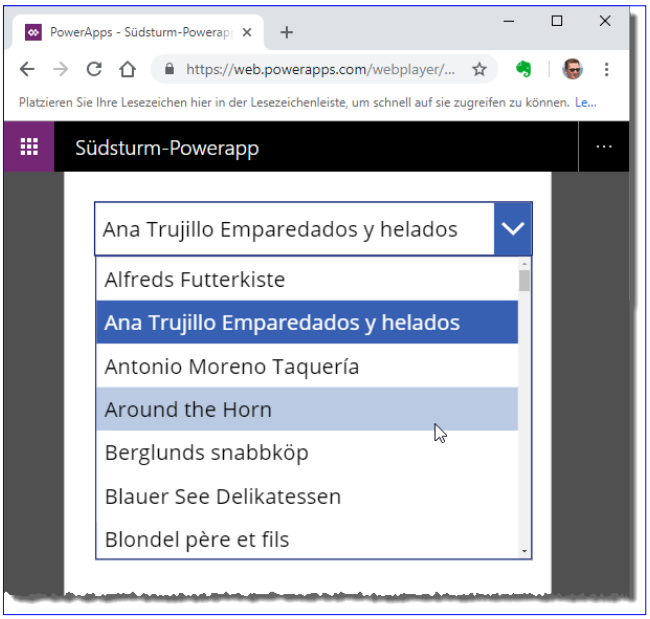


Bild 3: Testen der App im Webplayer

Anschließend müssen Sie diese über den im gleichen Bereich erscheinenden Befehl **Veröffentlichen** noch publizieren. Danach liefert diese Bildschirmseite im Webplayer die Ansicht aus Bild 3.

Bereich zur Anzeige der Daten hinzufügen

Nun benötigen wir unterhalb des Dropdown-Elements weitere Steuerelemente, um die Felder des gewählten Datensatzes anzuzeigen. Dazu fügen wir der Bildschirmseite zunächst ein übergeordnetes Element hinzu, und zwar über den Befehl **FormularelBearbeiten** des Bereichs **Einfügen**.

Das so hinzugefügte Element platzieren wir so auf der Bildschirmseite, dass es sich unterhalb des Dropdown-Steuerelements befindet und den verbleibenden Platz einnimmt (siehe Bild 4). Außerdem stellen wir seinen Namen auf **frmKundenuebersicht** ein.

Dann wählen Sie für die Eigenschaft **Datenquelle** im Bereich **Eigenschaften** den Wert **[dbo].[tblKunden]** aus. Dies stellt automatisch die Eigenschaft **DataSource** im Bereich **Erweitert** auf den Wert **'[dbo].[tblKunden]'** ein.

Darunter finden Sie die Eigenschaft **Item**, der Sie folgenden Ausdruck zuweisen:

```
First(Filter('[dbo].[tblKunden]';Firma=drpKunden.Selected.Value))
```

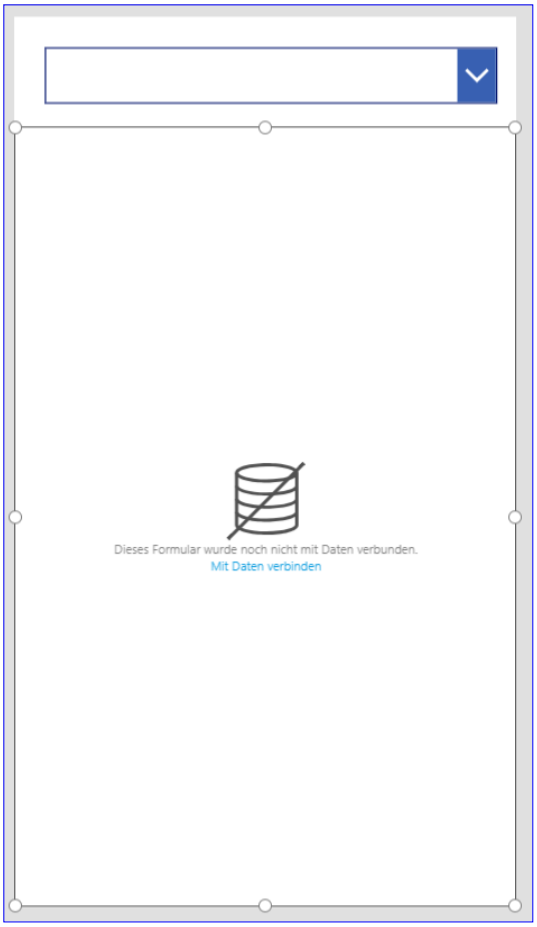


Bild 4: Das neue Formular ist noch nicht mit Daten verbunden.

Mittlerweile hat sich auch der Inhalt des Formulars zum Bearbeiten der Daten geändert. Dieses zeigt nun den Text aus Bild 5 an, der darauf hinweist, dass noch die Felder für Anzeige ausgewählt werden müssen.

Bild 5: Es fehlen noch die Felder zum Anzeigen der Daten.

Klicken wir auf diesen Link, erscheint der Bereich **Felder**, in dem wir mit einem Klick auf **Feld hinzufügen** eine Liste der Felder der Datenherkunft anzeigen (siehe Bild 6).

Hier wählen wir nun die gewünschten Felder aus und klicken auf **Hinzufügen**. Wenn Sie die Felder in einer bestimmten Reihenfolge hinzufügen wollen, können Sie das nacheinander erledigen, indem Sie zunächst den obersten Eintrag auswählen und mit **Hinzufügen** im Formular platzieren, dann wieder auf **Feld hinzufügen** klicken und das nächste Feld hinzufügen. Wenn Sie zwischendurch den Dialog **Felder** schließen, können Sie diesen wieder öffnen, indem Sie im Eigenschaftenblatt auf den Link **Felder bearbeiten** rechts neben der Eigenschaft **Felder** klicken (siehe Bild 7).

Auf diese Weise fügen wir ein Feld nach dem anderen zum Formular hinzu. Sie können allerdings auch die Felder, die Sie hinzufügen wollen, einfach in der richtigen Reihenfolge in der Feldliste anklicken – diese werden dann in dieser Reihenfolge zum Formular hinzugefügt.

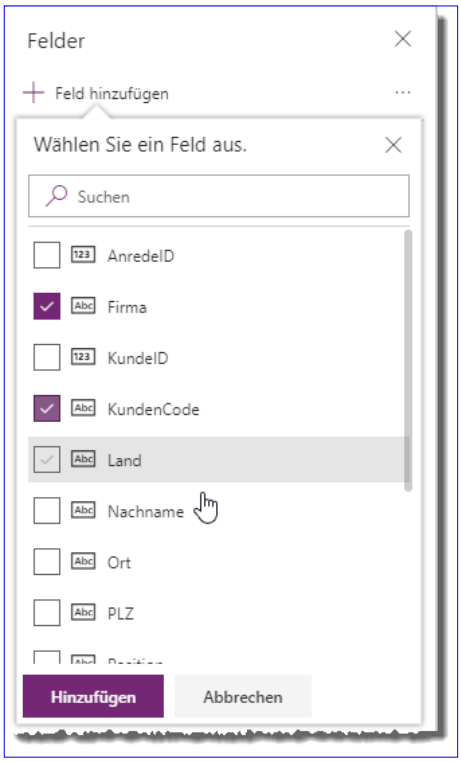


Bild 6: Auswahl der anzuzeigenden Felder

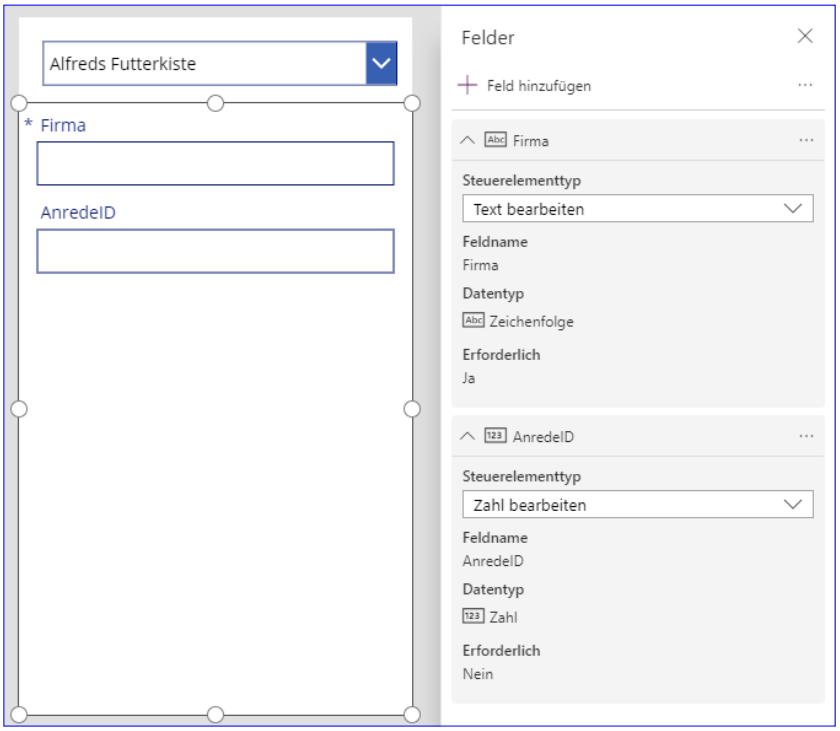


Bild 7: Auswahl weiterer Felder