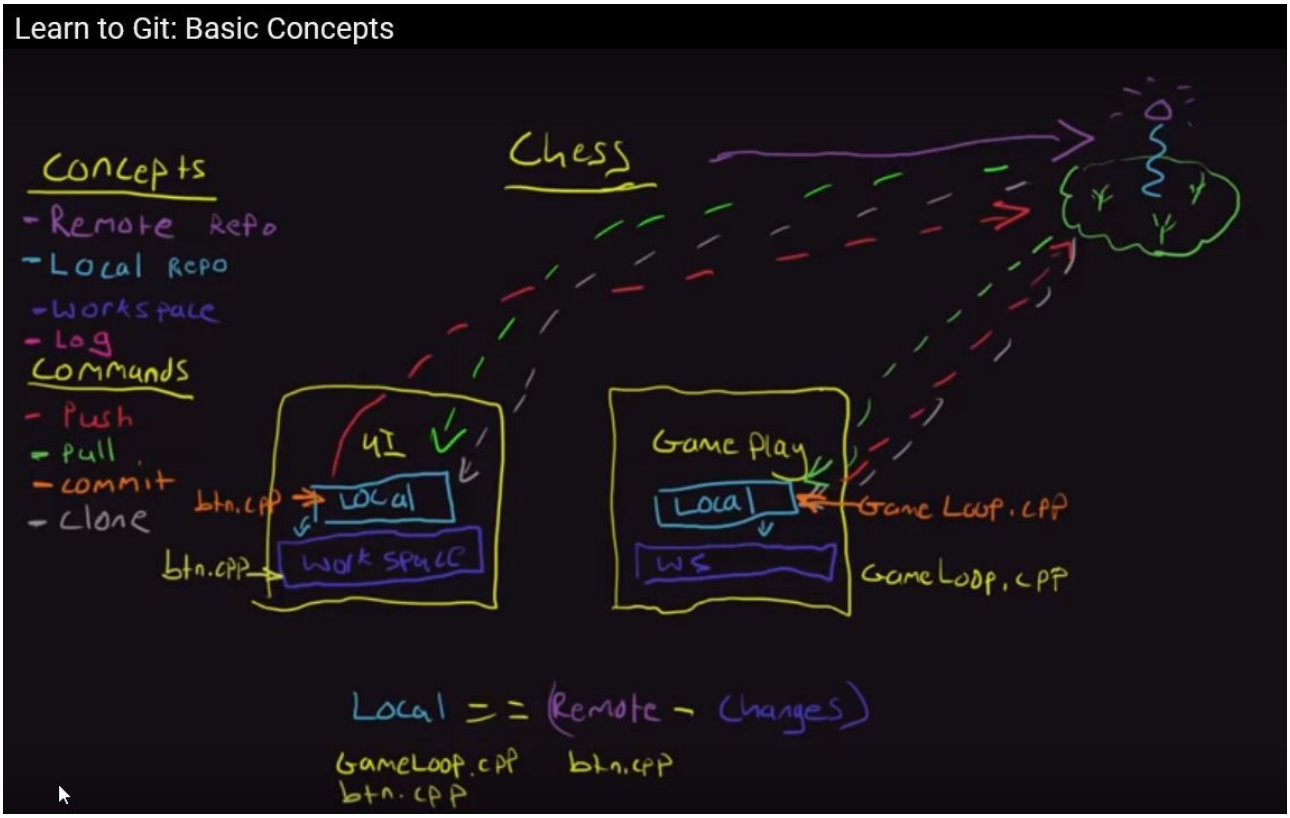
INETA – MedMan

Inhaltsverzeichnis

GIT – Grundlagen

Repository-Organisation und wichtige Befehle



- CLONE kopiert das Remote-Repository in ein lokales Repository
- COMMIT überträgt Änderungen vom lokalen Workspace in das lokale Repository
- PUSH überträgt das lokale Repository in das Remote-Repository
- PULL synchronisert das lokale Repository aus dem Remote-Repository

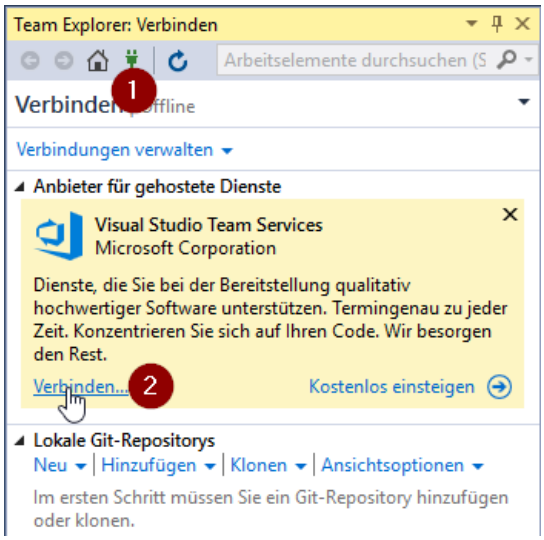
Visual Studio Team Services

Anmeldung

URL: https://devgroupgoeks.visualstudio.com/
Benutzer: <E-Mail-Adresse privat>
Passwort: *********

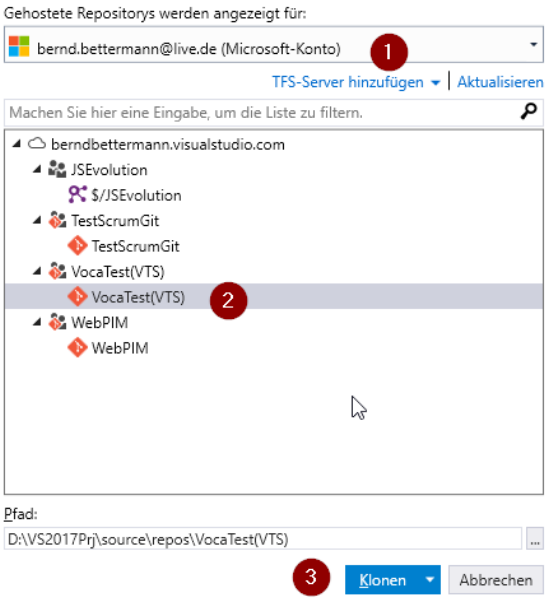
Verbindung aus Visual Studio heraus herstellen

Innerhalb des VS-Team Explorers das Icon „Verbindungen verwalten“ wählen und anschließend innerhalb von VS Team Services auf „Verbinden“ clicken:



Anschließend am VS Team Server anmelden, das Projekt auswählen und klonen:

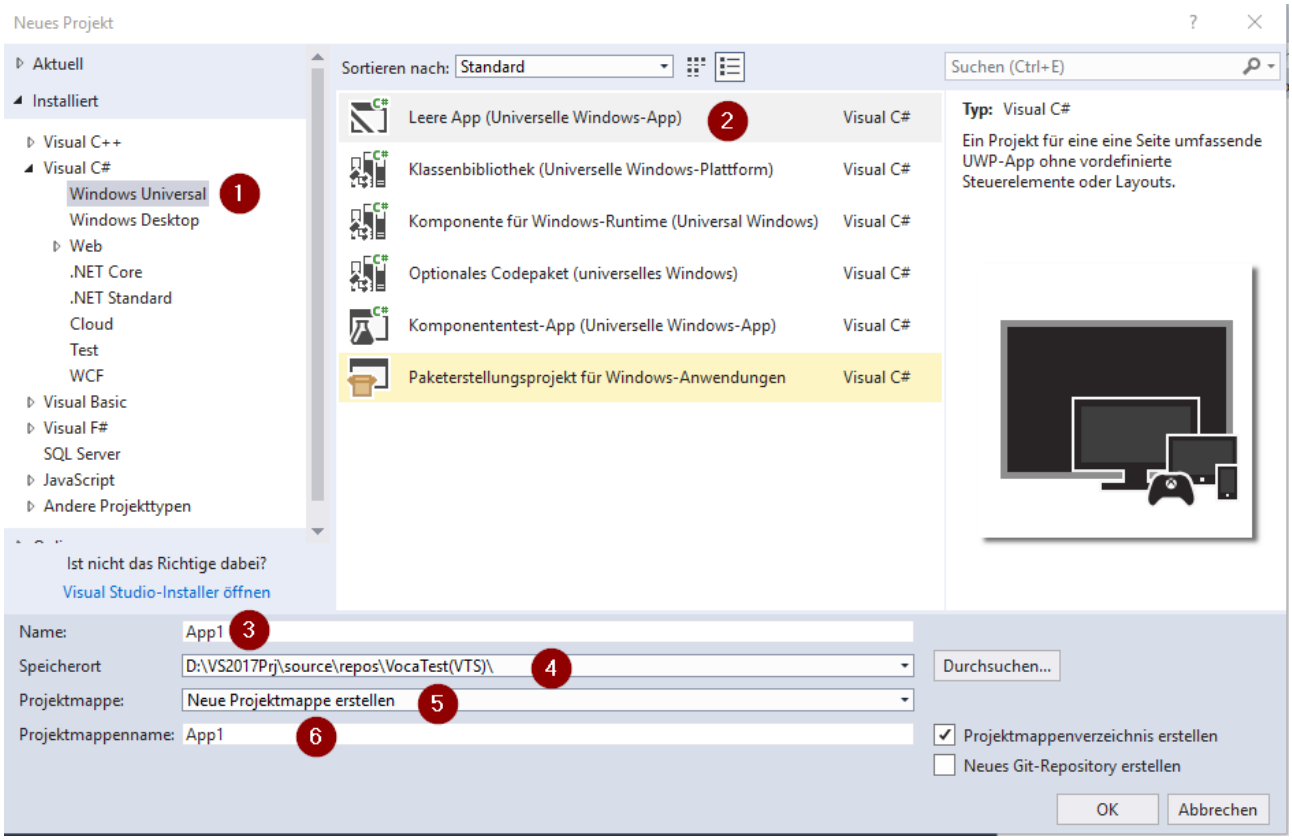
Verbindung mit einem Projekt herstellen



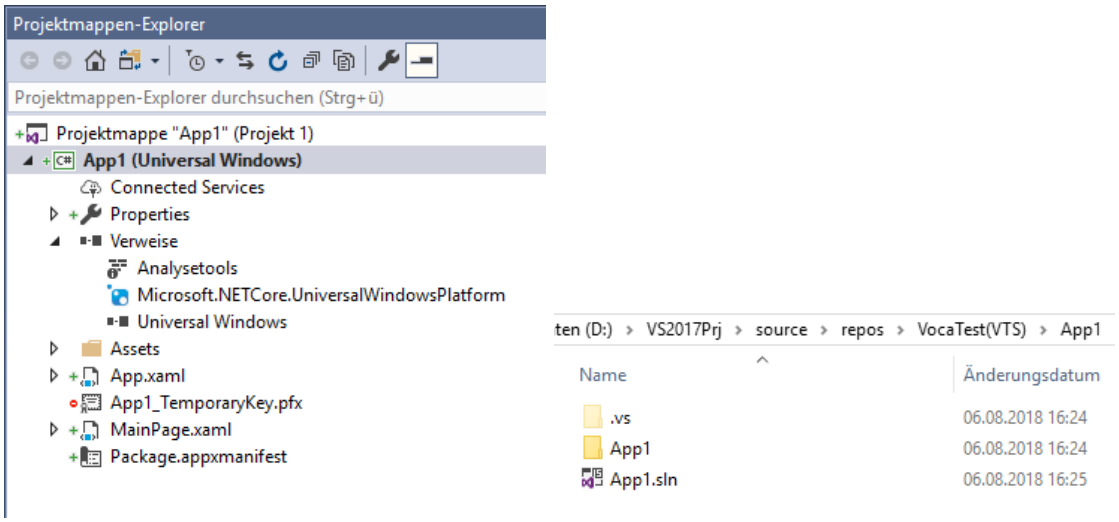
Projekt in VS erstellen und ins Remote-Repository übertragen

Projekt erstellen

Im VS-Dialog den entsprechenden Projekttyp wählen und die Option „Hinzufügen“ aktivieren:



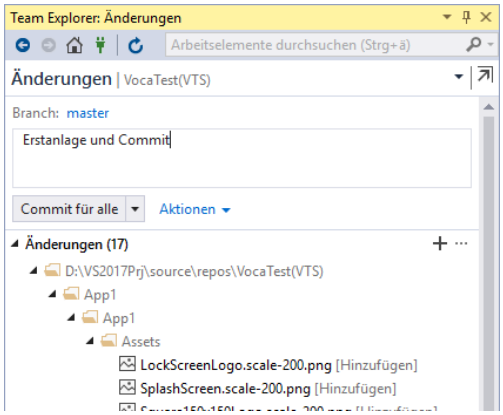
Projekt- und Verzeichnisstruktur:



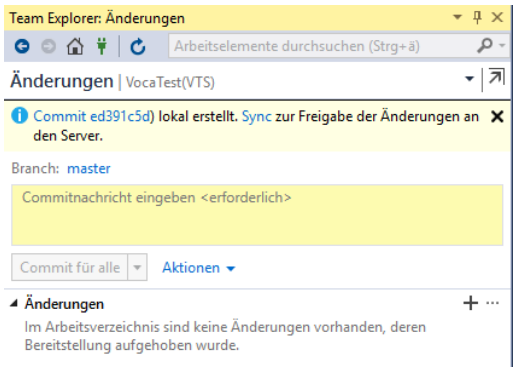
Projekt mit dem Remote-Repository synchronisieren

Die TeamExplorer-Startseite bietet verschiedene Optionen:

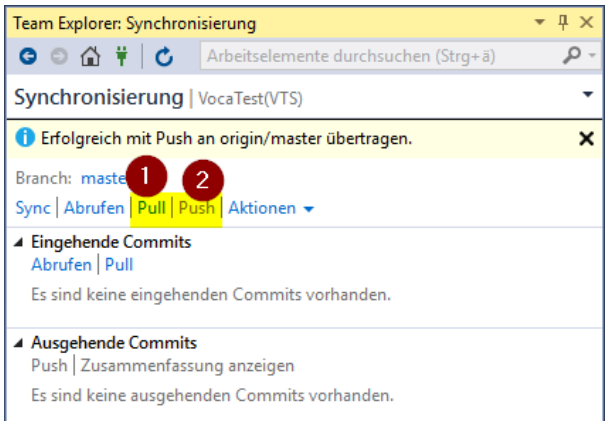
- Änderungen zeigt die ausstehenden Änderungen (im Workspace) an



Nach Eingabe des Kommentars über „Commit für alle“ Änderungen in das lokale (!) Repository übernehmen.



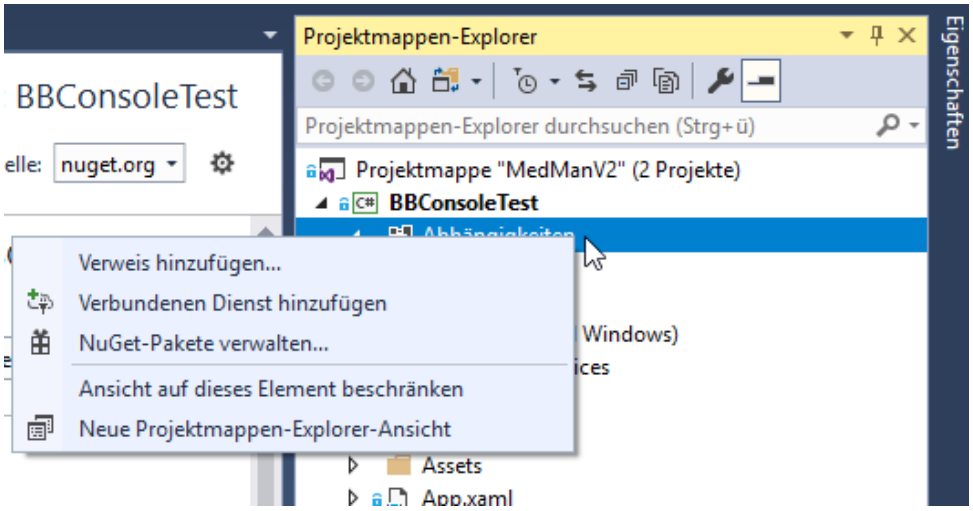
Anschließend können die Änderungen per „Sync“ (Pull + Push) in das Remote-Repository übertragen werden.



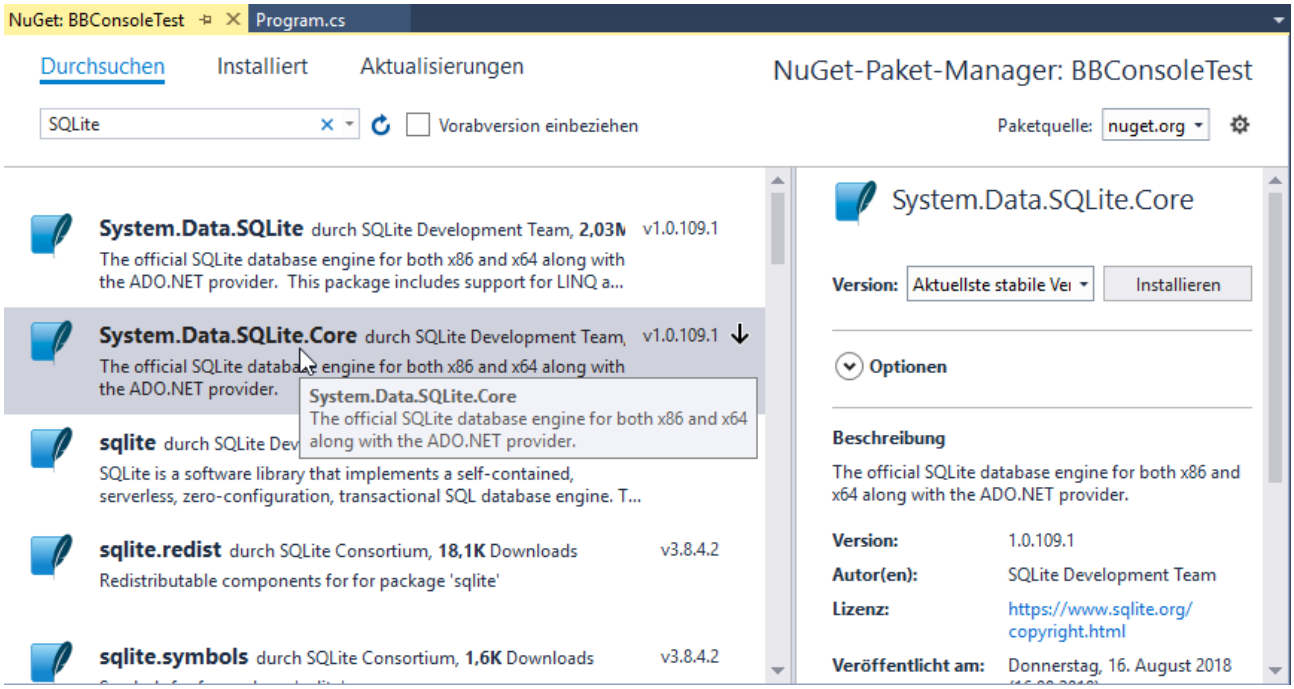
SQLite

NuGet-Pakete laden

Im Projektmappen-Explorer rechtsclick auf die Abhängigkeiten und „NuGet-Pakete verwalten...“ aufrufen:



Danach nach „SQLite“ suchen und das gewünschte Paket installieren:



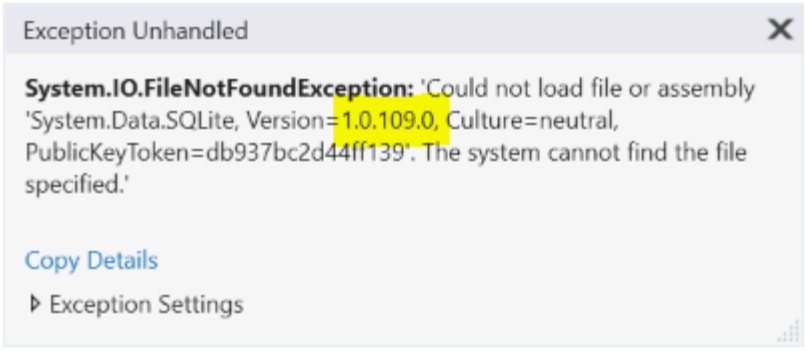
SQLite verwenden (Konsolenanwendung)

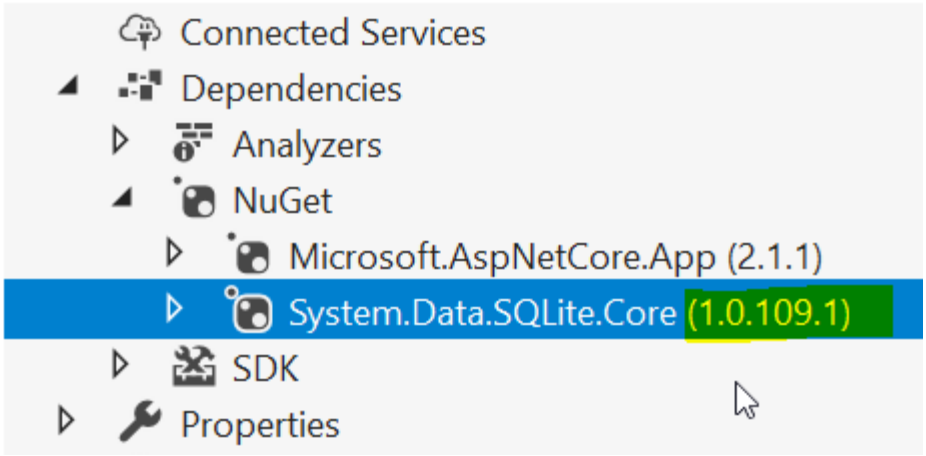
Einfach „using System.Data.SQLite“ hinzufügen und dann:

```
SQLiteConnection.CreateFile("SQLTutorial.sqlite");

SQLiteConnection dbConnection = new SQLiteConnection("Data Source =
                                    SQLTutorial.sqlite; Version = 3;");
dbConnection.Open();

string sql = "CREATE TABLE personen(vorname TEXT, nachname TEXT)";
SQLiteCommand Command = new SQLiteCommand(sql, dbConnection);
Command.ExecuteNonQuery();

sql = "INSERT INTO personen(vorname, nachname) VALUES ('Frank', 'Röger')";
Command = new SQLiteCommand(sql, dbConnection);
Command.ExecuteNonQuery();

sql = "INSERT INTO personen(vorname, nachname) VALUES(@vorname, @nachname)";
Command = new SQLiteCommand(sql, dbConnection);
Command.Parameters.Add("@vorname", System.Data.DbType.String).Value =
                                    "Luna’Chiwa";
Command.Parameters.Add("@nachname", System.Data.DbType.String).Value =
                                    "Musterfrau";
Command.ExecuteNonQuery();
Command.Parameters.Clear();

sql = "SELECT * FROM personen";
Command = new SQLiteCommand(sql, dbConnection);
SQLiteDataReader reader = Command.ExecuteReader();
while (reader.Read())
    Console.WriteLine("Name: " + reader["vorname"] + " " + reader["nachname"]);

dbConnection.Close();
```

Versionsprobleme beim ausführen beheben

Sollte beim Start der Applikation ein Fehler der folgenden Art erscheinen:





so hilft folgendes Vorgehen:

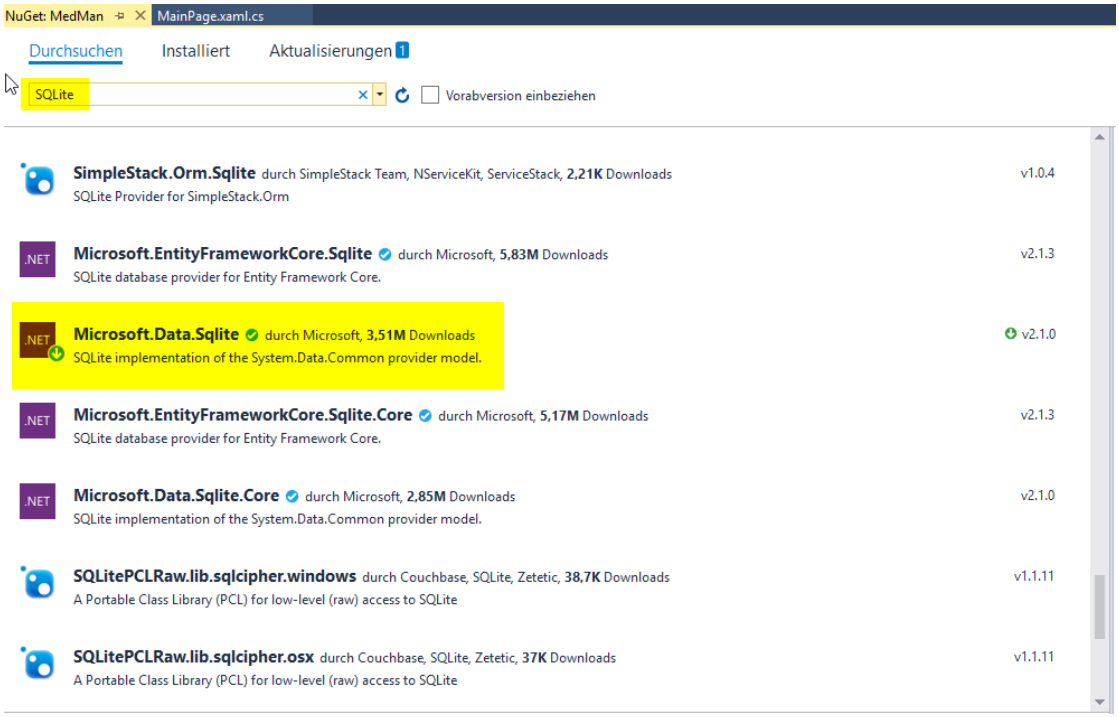
In Windows Explorer, navigate to `%UserProfile%\.nuget\packages`

1. Delete the directories for `system.data.sqlite.core` and `system.data.sqlite.core`
2. In your `.csproj`, change your sqlite package reference to
   `` `<PackageReference Include="System.Data.SQLite.Core" Version="1.0.109.0"/>` ``
3. Rebuild the project to restore the package

MedMan und SQLite

Zunächst über NuGet das Paket „Microsoft.Data.Sqlite“ in das Projekt laden:

Projekt „MedMan“ - Verweise – Rechtsclick – NuGet-Pakete verwalten



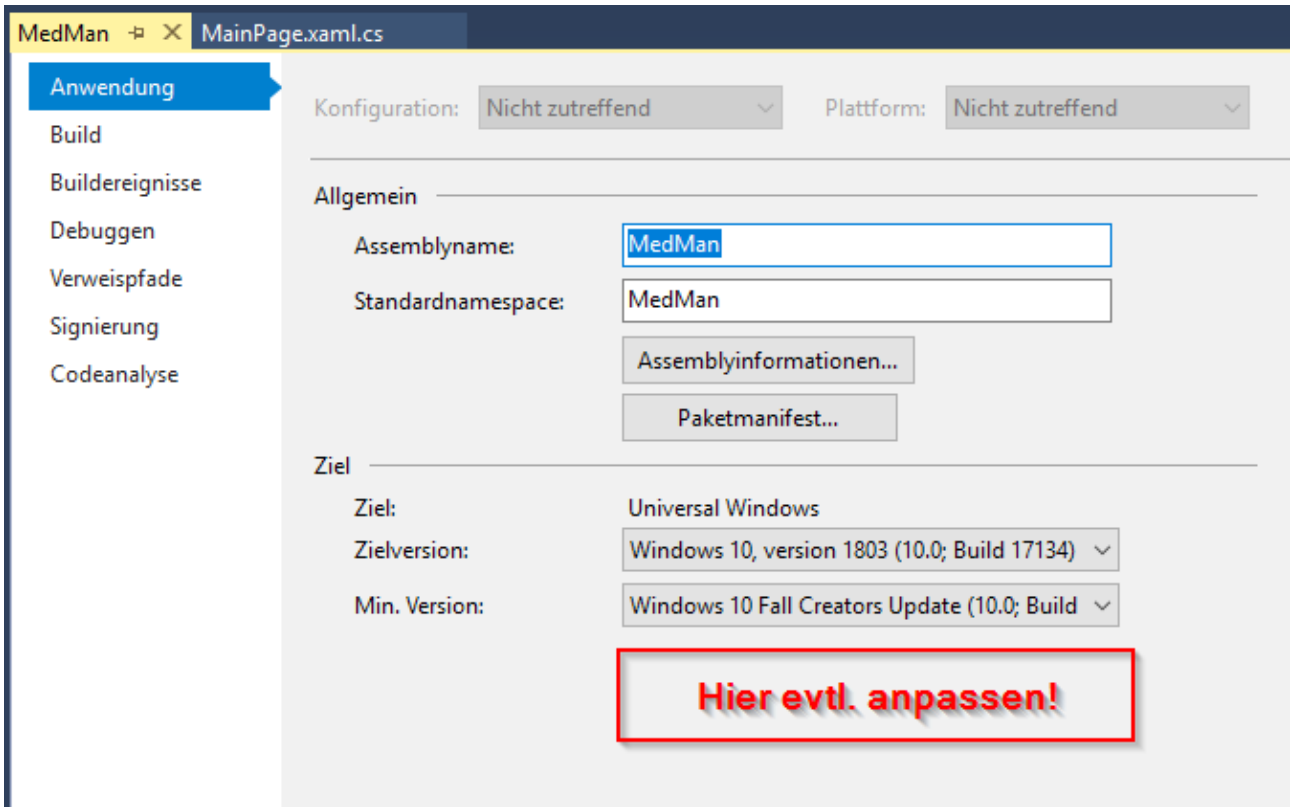
Über „using Microsoft.Data.Sqlite“ einbinden und z.B. wie folgt benutzen:

```
public sealed partial class MainPage : Page
{
        public MainPage()
        {
            this.InitializeComponent();

            InitializeDatabase();
        }

        public static void InitializeDatabase()
        {
            using (SqliteConnection db =
                new SqliteConnection("Filename=MedMan.sqlite"))
            {
                db.Open();
                String tableCommand = "CREATE TABLE IF NOT " +
                    "EXISTS Medikament (ID INTEGER PRIMARY KEY, " +
                    "Name NVARCHAR(200) NULL)";
                SqliteCommand createTable = new SqliteCommand(tableCommand, db);
                createTable.ExecuteReader();
            }
        }
}
```

Versionsprobleme beim erzeugen der Anwendung

Sollte es beim compilieren der Anwendung zu Fehlermeldungen kommen, kann dies daran liegen, dass die Zielversion der Applikation entsprechend angepasst werden muss.

Dazu Rechtsclick auf das Projekt und „Eigenschaften“ wählen:

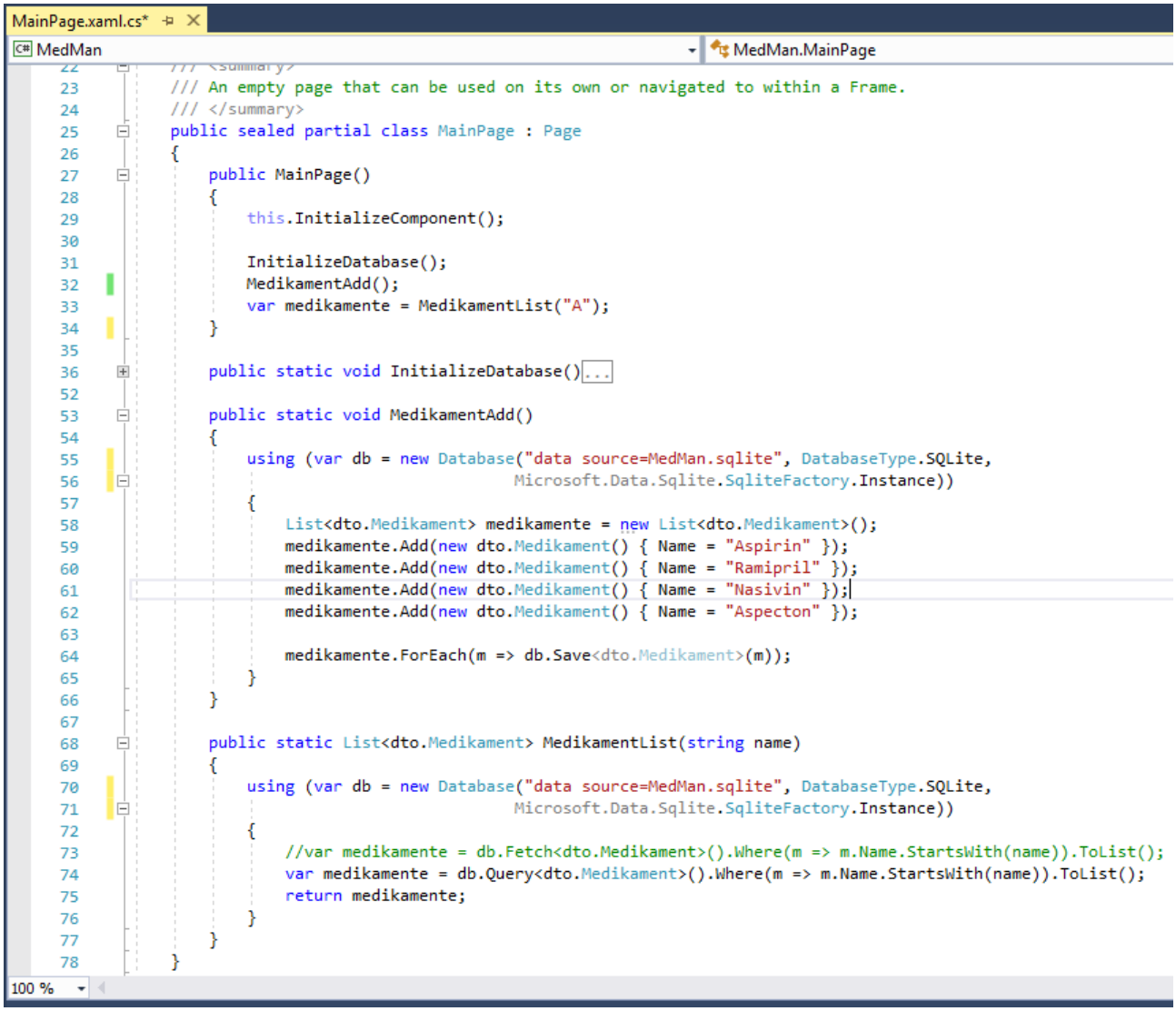


NPoco

Grundlagen

NPoco ist ein „leichter“ Datenbank-Wrapper, der auch für SQLite genutzt werden kann

Dazu über NuGet das Paket „NPoco“ laden und per „using NPoco“ im Code verwenden:



Über die Database-Variable (im Beispiel „db“) lässt sich über die Eigenschaften „LastCommand“ bzw. „LastSQL“ erkennen, welche SQL-Befehle NPoco an die Datenbank gesendet hat.

Data Transfer Objekte (dto's) definieren

Npoco kann Datenklassen (DTOs) automatisch in der Datenbank ablegen bzw. aus dieser gezielt laden.

Ein DTO wird z.B. wie folgt definiert:

```
using NPoco;

namespace MedMan.dto
{
    [TableName("Medikament"), PrimaryKey("ID", AutoIncrement =true)]
    public class Medikament
    {
        public int ID { get; set; }
        public string Name { get; set; }
    }
}
```

Es können verschiedene Attribute für eine Datenklasse definiert werden.

Im obigen Beispiel wird der Datenbank-Tabellenname über „TableName“ definiert.

Der Primary Key kann auch aus mehreren Feldern bestehen, dann sind diese, per Komma getrennt anzugeben.
Handelt es sich beim Primary Key um ein einzelnes Integer-Feld, so kann die Verwaltung per Attribut „AutoIncrement=true“ von NPoco erledigt werden.

Beim erzeugen der Datenobjekte darf der Primary Key dann nicht belegt werden; nur so greift die automatische Verwaltung!

Datenabfrage (Query vs. Fetch)

Zur Datenabfrage gibt es in NPoco die alternativen Methoden „Query“ und „Fetch“:

```
var medikamente = db.Fetch<dto.Medikament>().Where(m => m.Name.StartsWith(name)).ToList();

var medikamente = db.Query<dto.Medikament>().Where(m => m.Name.StartsWith(name)).ToList();
```

Der Unterschied lässt sich am erzeugten SQL-Statement erkennen (db.LastSQL).

Fetch erzeugt aus der obigen Abfrage das folgende Statement für die Datenbank:

SELECT [ID] AS [ID], [Name] AS [Name] FROM [Medikament]

Query erzeugt aus der obigen Abfrage hingegen folgendes Statement:

SELECT [M].[ID] as [ID], [M].[Name] as [Name] FROM [Medikament] [M]
WHERE upper([M].[Name]) like @0 escape '\'

D.h., Fetch liest immer die komplette Tabelle ein und führt auf den eingelesenen Daten eine Abfrage aus während Query die Abfrage bereits in das SQL-Statement integriert.

Fetch hat seine Berechtigung, wenn es NPoco nicht möglich ist, aus der Abfrage ein entsprechendes SQL-Statement zu erzeugen.

Auszug aus stackoverflow:

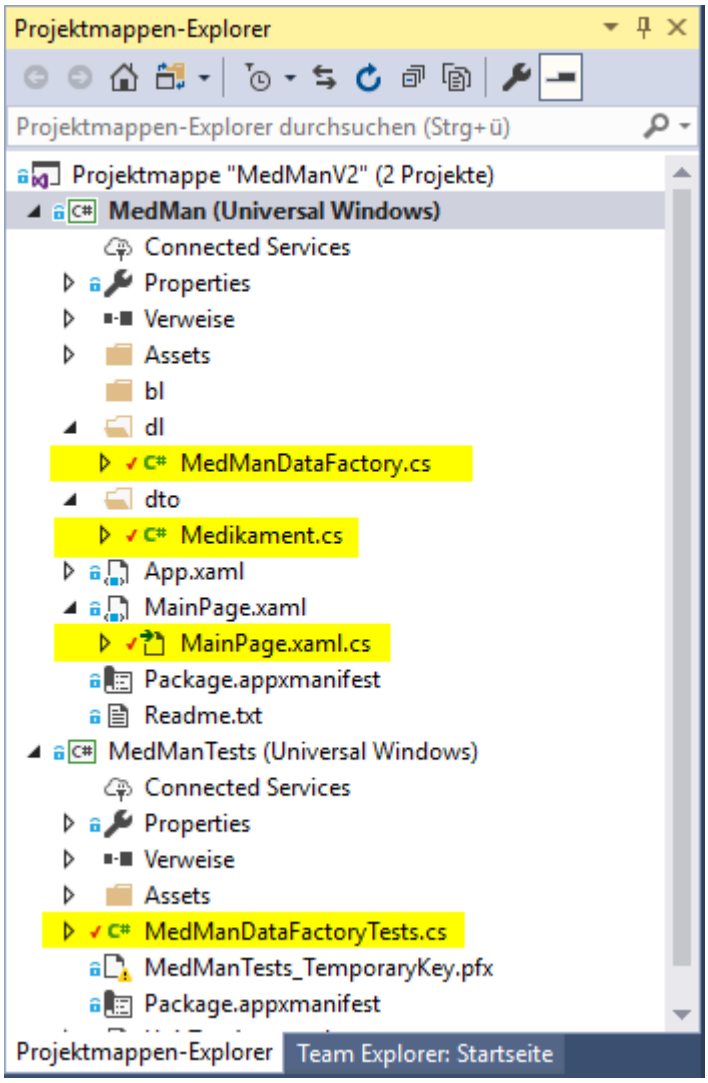
> **Query vs Fetch**
>
> The Database class has two methods for retrieving records Query and Fetch. These are pretty much identical except Fetch returns a List<> of POCO's whereas Query uses yield return to iterate over the results without loading the whole set into memory.

Weitere Dokumentation zu NPoco:

- https://github.com/schotime/Npoco/wiki
- https://github.com/CollaboratingPlatypus/PetaPoco/wiki

Verwenden einer DataFactory

Projektstruktur:



Source „Medikament.cs “:

```
using NPoco;

namespace MedMan.dto
{
    [TableName("Medikament"), PrimaryKey("ID", AutoIncrement =true)]
    public class Medikament
    {
        public int ID { get; set; }
        public string Name { get; set; }
    }
}
```

Source „MainPage.xaml.cs “:

```
using Windows.UI.Xaml.Controls;
using MedMan.dl;

// The Blank Page item template is documented at https://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace MedMan
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();

            var factory = new MedManDataFactory();
            factory.InitializeDatabase();
            factory.MedikamenteAdd();

            var newMedikament = new dto.Medikament() { Name = "AspirinBB" };
            factory.SaveData(newMedikament);

            var medikamenteAlle = factory.GetData<dto.Medikament>(); //5
            var medikamenteA = factory.GetData<dto.Medikament>(m => m.Name.StartsWith('A')); //3
            var medikamenteX = factory.GetData<dto.Medikament>(m => m.Name.StartsWith('X')); //0
            medikamenteA = null;
        }
    }
}
```

Source „MedManDataFactory.cs “:

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using Microsoft.Data.Sqlite;
using NPoco;

namespace MedMan.dl
{
    public class MedManDataFactory
    {
        public static Func<IDatabase> MedManDatabase = () => new Database("data source=MedMan.sqlite", DatabaseType.SQLite,
                                                                Microsoft.Data.Sqlite.SqliteFactory.Instance);
        public virtual List<T> GetData<T>()
        {
            using (var db = MedManDatabase())
            {
                try
                {
                    return db.Fetch<T>();
                }
                catch (Exception)
                {

                }
                return new List<T>();
            }
        }
        public virtual List<T> GetData<T>(Expression<Func<T,bool>> where)
        {
            using (var db = MedManDatabase())
            {
                try
                {
                    var erg= db.Query<T>().Where(where).ToList();
                    return erg;
                }
                catch (Exception)
                {

                }
```

```
                return new List<T>();
            }
        }
        public virtual void SaveData<T>(T poco)
        {
            using (var db = MedManDatabase())
            {
                try
                {
                    db.Save<T>(poco);
                    // auch möglich: db.Save(poco);
                }
                catch (Exception)
                {

                }
            }
        }
        public virtual void InitializeDatabase()
        {
            using (var db = MedManDatabase())
            {
                db.Execute("DROP TABLE IF EXISTS Medikament");
                db.Execute("CREATE TABLE Medikament (ID INTEGER PRIMARY KEY, Name NVARCHAR(200) NULL)");
            }
        }
        public virtual void MedikamenteAdd()
        {
            using (var db = MedManDatabase())
            {
                List<dto.Medikament> medikamente = new List<dto.Medikament>();
                medikamente.Add(new dto.Medikament() { Name = "Aspirin" });
                medikamente.Add(new dto.Medikament() { Name = "Ramipril" });
                medikamente.Add(new dto.Medikament() { Name = "Nasivin" });
                medikamente.Add(new dto.Medikament() { Name = "Aspecton" });

                medikamente.ForEach(m => db.Save<dto.Medikament>(m));
                // auch möglich: medikamente.ForEach(m => db.Save(m));
            }
        }
    }
}
```

DataFactory per UnitTests prüfen

Neues Projekt anlegen:

Projektmappe – Hinzufügen – Neues Projekt – Windows Univeral – Komponententest-App
Wichtig: MinVersion (wegen Npoco) auf „Window 10 Fall Creators Update“ setzen!

Name „MedManTests“

Verweise hinzufügen:

- Projekt „MedMan“
- NuGet „Moq“
- NuGet „NPoco“

Neue Klasse „MedManDataFactoryTests“ anlegen

Source „MedManDataFactoryTests.cs “:

```
using System.Collections.Generic;
using MedMan.dl;
using MedMan.dto;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using NPoco;

namespace MedManTests
{
    [TestClass]
    public class MedManDataFactoryTests
    {
        [TestMethod]
        public void Fetch_wurde_einmal_aufgerufen()
        {
            var mock = new Mock<IDatabase>();
            mock.Setup(m => m.Fetch<Medikament>()).Returns(new List<Medikament>());
            MedManDataFactory.MedManDatabase = () => mock.Object;
            var sut = new MedManDataFactory();
            var erg = sut.GetData<Medikament>();
            mock.Verify(m => m.Fetch<Medikament>(), Times.Once);
        }
    }
}
```