

# Retail Manager Development Manual

Michael Kaip

August 23, 2019

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Project Summary . . . . .	1
<b>2</b>	<b>Initial Plan</b>	<b>2</b>
2.1	Outline . . . . .	2
2.2	Technologies . . . . .	2
<b>3</b>	<b>Initial Setup in Visual Studio</b>	<b>3</b>
<b>4</b>	<b>Creating a WebAPI with Authentication</b>	<b>4</b>
4.1	Indentity Configuration . . . . .	4
4.2	Getting authorized for development . . . . .	4
4.2.1	Postman . . . . .	4
4.2.2	Getting User Information . . . . .	5
<b>5</b>	<b>Installing and configuring SWAGGER</b>	<b>6</b>
5.1	Installing SWAGGER . . . . .	6
5.2	Channging the configuration of SWAGGER . . . . .	6
5.3	Adding OAuth ability . . . . .	7
<b>6</b>	<b>SQL Database Setup</b>	<b>11</b>
6.1	Adding new Database Project to the solution . . . . .	11
6.2	Adding several folders to the project . . . . .	11
6.3	Creating a profile and publishing the Database . . . . .	12
<b>7</b>	<b>WPF with MVVM Project Setup</b>	<b>13</b>
7.1	Adding the WPF Project to the solution . . . . .	13
7.2	Changing the Assamby Name to the name of the solution in Properties . . . . .	13
7.3	Adding Caliburn Micro MVVM-Framework . . . . .	14
7.3.1	Adding the folder structure for the MVVM-Framework . . . . .	14
7.3.2	Adding a new ShellViewModel class and a SchellView window . . . . .	14
7.3.3	Adding a Bootstrapper class to DesktopUI . . . . .	14
7.3.4	Removing StartUpURI from App.xaml and adding a new Ressource Dictionary . . . . .	15
<b>8</b>	<b>Dependency Injection in WPF</b>	<b>16</b>
8.1	SimpleContainer in Caliburn Micro . . . . .	16
8.1.1	Implementing SimpleContainer in Bootstrapper.cs . . . . .	16
8.2	Overriding Configure() Method for the container . . . . .	17
<b>9</b>	<b>Datamodel - planning and setup</b>	<b>18</b>
9.1	Planning the Register . . . . .	18
9.2	SQL Database Table Creation . . . . .	19
<b>10</b>	<b>WPF Login Form Creation</b>	<b>20</b>
10.1	Inheritance from the conductor class in Caliburn Micro . . . . .	20
10.2	Implementing the menue bar . . . . .	20
10.3	Adding a UserControl . . . . .	20
10.3.1	Adding a class LoginViewModel (public) and UserControl LoginView . . . . .	20
10.3.2	Designing the UserControl . . . . .	21
10.3.3	Activating the LoginView on startup in the ShellView . . . . .	21
10.3.4	Implementing LoginViewModel.cs . . . . .	22
10.3.5	Connecting the LoginViewModel to Caliburn.Micro . . . . .	23
<b>11</b>	<b>Wiring up the WPF Login Form</b>	<b>24</b>

11.1	Implementing a class <code>AuthenticatedUser.cs</code> . . . . .	24
11.2	Implementing a helper class to handle API call interactions . . . . .	25
11.3	Implementing a Interface <code>IAPIHelper.cs</code> . . . . .	26
11.4	Adding <code>&lt;appsettings&gt;</code> to <code>App.Config</code> . . . . .	26
11.5	Adding <code>APIHelper</code> and <code>IAPIHelper</code> to the container in <code>Bootstrapper.cs</code> . . . . .	26
11.6	Applying some changings to <code>LoginViewModel.cs</code> . . . . .	26
11.6.1	Adding a new private property as a backing field . . . . .	26
11.6.2	Adding a constructor and initializing the property from within . . . . .	27
11.6.3	Implementig the <code>Login()</code> method . . . . .	27
11.7	Enabling the solution to start multiple projects . . . . .	27
<b>12</b>	<b>Login Form Error Handling</b>	<b>29</b>
12.1	Displaying an login error message within the login form . . . . .	29
<b>13</b>	<b>Getting User Data</b>	<b>31</b>
13.1	Adding a class library for the API . . . . .	31
13.1.1	Installing <code>Dapper</code> . . . . .	31
13.1.2	The <code>SqlDataAccess</code> class . . . . .	32
13.1.3	The <code>UserData</code> class . . . . .	33
13.1.4	Adding a <code>UserController.cs</code> to <code>RMDDataManager.Controllers</code> . . . . .	34
13.2	Implementing data capturing from the UI side . . . . .	36
13.2.1	Adding a new class library . . . . .	36
13.2.2	The <code>LoggedInUserModel</code> class . . . . .	37
13.2.3	Moving helpers and Models . . . . .	37
13.2.4	Extracting an Interface and adding to <code>simpleContainer</code> . . . . .	37
13.2.5	Implementing a Method <code>GetLoggedInUserInfo</code> in <code>APIHelper.cs</code> . . . . .	38
<b>14</b>	<b>Sales Page Creation</b>	<b>39</b>
<b>15</b>	<b>Event Aggregation in WPF</b>	<b>39</b>
<b>16</b>	<b>Displaying Product data</b>	<b>39</b>
<b>17</b>	<b>Wiring up WPF Shopping Cart</b>	<b>39</b>
<b>18</b>	<b>Modifying SQL, the API and WPF to add Taxes</b>	<b>39</b>

# 1 Introduction

## 1.1 Project Summary

The goal of this project is to build a desktop app that runs a cash register, handles inventory and manages an entire retail store. Creating and implementing a **WebAPI layer**, will allow the whole project to grow. This layer will be able to serve each kind of application (desktop, mobile, web, ...).

## 2 Initial Plan

### 2.1 Outline

The App is going to be build as a MVP (Minimum Viable Product) that can be expanded to cover all of the features, which are needed over time - so it can grow into a full featured application. First step is getting all of the major pieces set up, including:

### 2.2 Technologies

- Unit Testing
- Dependency Injection
- WPF
- MVVM with Caliburn Micro
- ASP.NET MVC (Web Frontend)
- .NET Framework
- .NET Core 3.0
- SSDT - SQL Server Data Tools
- Git
- Azure DevOps
- Async
- Reporting
- WebAPI
- Logging
- Data Validation
- HTML
- CSS
- JavaScript
- Authentication

### 3 Initial Setup in Visual Studio

1. Setting up a Git-Repository, including README, GitIgnore (for VS) and License
2. Creating a **Blank Solution**: Other Project Types → Blank Solution  
Such type of solution isn't language specific.

## 4 Creating a WebAPI with Authentication

### 1. Adding new Project to the Solution:

Web → ASP.NET Web Application (.NET Framework) → WebAPI

Add folders and references for:

- MVC
- Web API

Change Authentication to

- Individual User Accounts

### 2. Upgrading all NuGet-Packages

### 4.1 Indentity Configuration

App\_Start → IdentityConfig.cs

In there are some settings for setting up the WebAPI, especially for authentication:

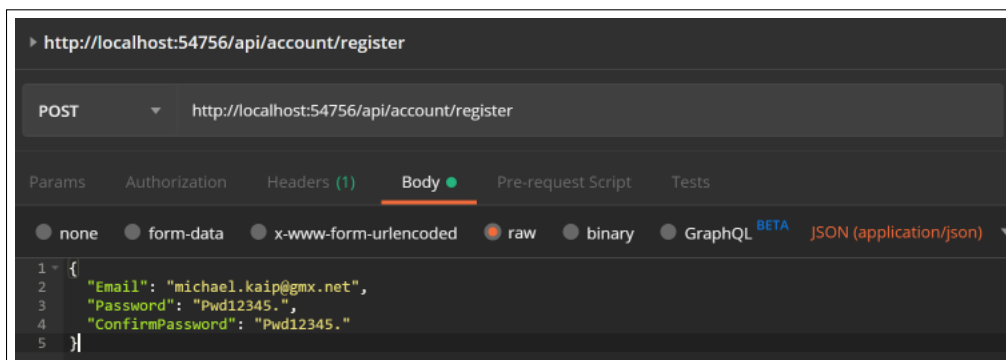
- UserValidator
- PasswordValidator

### 4.2 Getting authorized for development

#### 4.2.1 Postman

The following calls has to be applied in the given order:

#### 1. POST



Creates a new user account and stores this information into the user database.  
If **Status: 200 OK**, username and password has been succesfully created.

#### 2. GET

GET http://localhost:54756/token

Params Authorization Headers (1) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary GraphQL BETA

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	grant_type	password	
<input checked="" type="checkbox"/>	username	michael.kaip@gmx.net	
<input checked="" type="checkbox"/>	password	Pwd12345.	

It will return an **access\_token** which is, by default, valid for 14 days. Token is needed for all further interaction with the server. Can be also configured for shorter valid periods.

### 3. POST

POST http://localhost:54756/api/values

Params Authorization Headers (1) Body Pre-request Script Tests

Headers (1)

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	Authorization	Bearer VuQDu7NWP8J-yd-tCRvkiqFwjBMZlaURbN...	
	Key	Value	Description

Response

#### 4.2.2 Getting User Information

In order to get the Identity of users returned, some changes have to be implemented. Through this it's becomes possible to apply different accessibility rules, based on the user-group a certain user is part of.

##### 1. *RMDataManager.Controllers.ValuesController*

```
using System.Web.Http;
using Microsoft.AspNet.Identity; // Needed for getting information about the user

namespace RMDataManager.Controllers
{
    [Authorize]
    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class ValuesController : ApiController
    {
        // GET api/values
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public IEnumerable<string> Get()
        {
            // Stores the ID of each user
            var userId = RequestContext.Principal.Identity.GetUserId();
            return new string[] { "value1", "value2", userId };
        }
    }
}
```

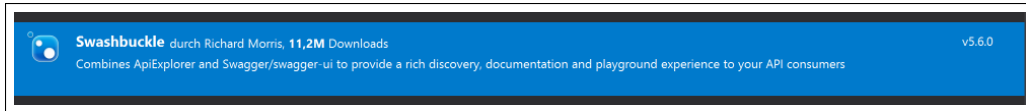


## 5 Installing and configuring SWAGGER

SWAGGER is an API documentation and demonstration tool.

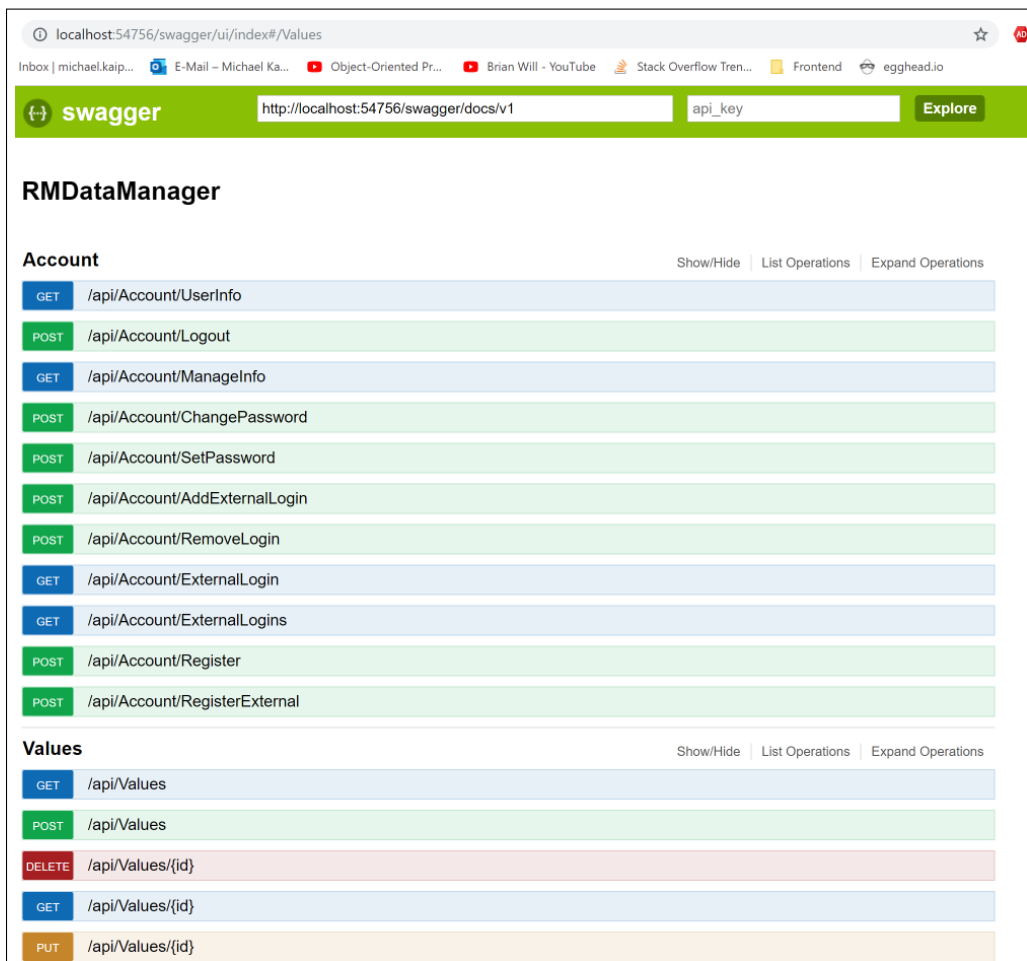
### 5.1 Installing SWAGGER

#### 1. NuGet-Manager



Adds a SWAGGER to WebAPI-Projects.

#### 2. Starting SWAGGER



### 5.2 Changing the configuration of SWAGGER

*RMDDataManager.App\_Start.SwaggerConfig.cs*

#### 1. Changing title

```
// Use "SingleApiVersion" to describe a single version API. Swagger 2.0 includes an "Info" object to
// hold additional metadata for an API. Version and title are required but you can also provide
// additional fields by chaining methods off SingleApiVersion.
//
c.SingleApiVersion("v1", title: "Retail Manager API"); // Changed to a proper name
```

## 2. Enabling proper printing of documents

```
// If you want the output Swagger docs to be indented properly, enable the "PrettyPrint" option.
//
c.PrettyPrint(); // enabled
```

## 3. Treating Enums as Strings

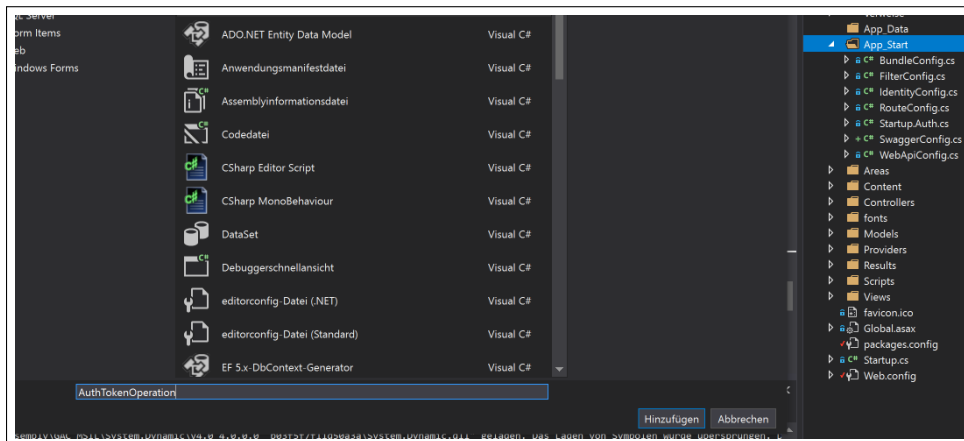
```
// In accordance with the built in JsonSerializer, Swashbuckle will, by default, describe enums as integers.
// You can change the serializer behavior by configuring the StringToEnumConverter globally or for a given
// enum type. Swashbuckle will honor this change out-of-the-box. However, if you use a different
// approach to serialize enums as strings, you can also force Swashbuckle to describe them as strings.
//
c.DescribeAllEnumsAsStrings(); // enabled
```

## 4. Changing document title

```
.EnableSwaggerUi(configure: c =>
{
    // Use the "DocumentTitle" option to change the Document title.
    // Very helpful when you have multiple Swagger pages open, to tell them apart.
    //
    c.DocumentTitle("RM API"); // changed the name
});
```

## 5.3 Adding OAuth ability

## 1. Enabling token endpoint allowance in the SWAGGER documentation

(a) Adding a new Class to `RMDataManager.App_Start`

## (b) Implementing the required Interface

```

public class AuthTokenOperation : IDocumentFilter
{
    0 Version: 1.0 | 0 Änderungen | 0 Autoren, 0 Änderungen
    public void Apply(SwaggerDocument swaggerDoc, SchemaRegistry schemaRegistry, IApiExplorer apiExplorer)
    {
        swaggerDoc.paths.Add("/token", new PathItem
        {
            post = new Operation
            {
                tags = new List<string> { "Auth" },
                consumes = new List<string>
                {
                    "application/x-www-form-urlencoded"
                },
                parameters = new List<Parameter>
                {
                    new Parameter
                    {
                        type = "string",
                        name = "grant_type",
                        required = true,
                        @in = "formData",
                        @default = "password"
                    },
                    new Parameter
                    {
                        type = "string",
                        name = "username",
                        required = false,
                        @in = "formData"
                    },
                    new Parameter
                    {
                        type = "string",
                        name = "password",
                        required = false,
                        @in = "formData"
                    }
                }
            }
        });
    }
}

```

## (c) Applying it to SwaggerConfig.cs

```

GlobalConfiguration.Configuration
    .EnableSwagger(c =>
    {
        c.DocumentFilter<AuthTokenOperation>(); // adding the implemented document filter
    }
)

```

## (d) Logging into the application using SWAGGER and get the token

**Retail Manager API**

Account Show/Hide List Operations Expand Operations

Values Show/Hide List Operations Expand Operations

**Auth** Show/Hide List Operations Expand Operations

**post** /token

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
grant_type	password		formData	string
username	michael.kaip@gmx.net		formData	string
password	Pwd12345.		formData	string

[Try it out!](#) [Hide Response](#)

**Curl**

```
curl -X POST --header 'Content-Type: application/x-www-form-urlencoded' --header 'Accept: application/json' -d 'grant_type=password&username=michael.kaip@gmx.net&password=Pwd12345.'
```

**Request URL**

```
http://localhost:54756/token
```

**Response Body**

```
{
  "access_token": "MofCgsbIKqt10Nn1j4fh4BuY3HfLYxt1qp-c3659v0LKzc-tME92xVP-McBT9d1UjZuHvEQPt1-4dQkKoxhKlQhMeSek8jM2SXX-pTr",
  "token_type": "bearer",
  "expires_in": 1209599,
  "userName": "michael.kaip@gmx.net",
  ".issued": "Thu, 08 Aug 2019 12:18:18 GMT",
  ".expires": "Thu, 22 Aug 2019 12:18:18 GMT"
}
```

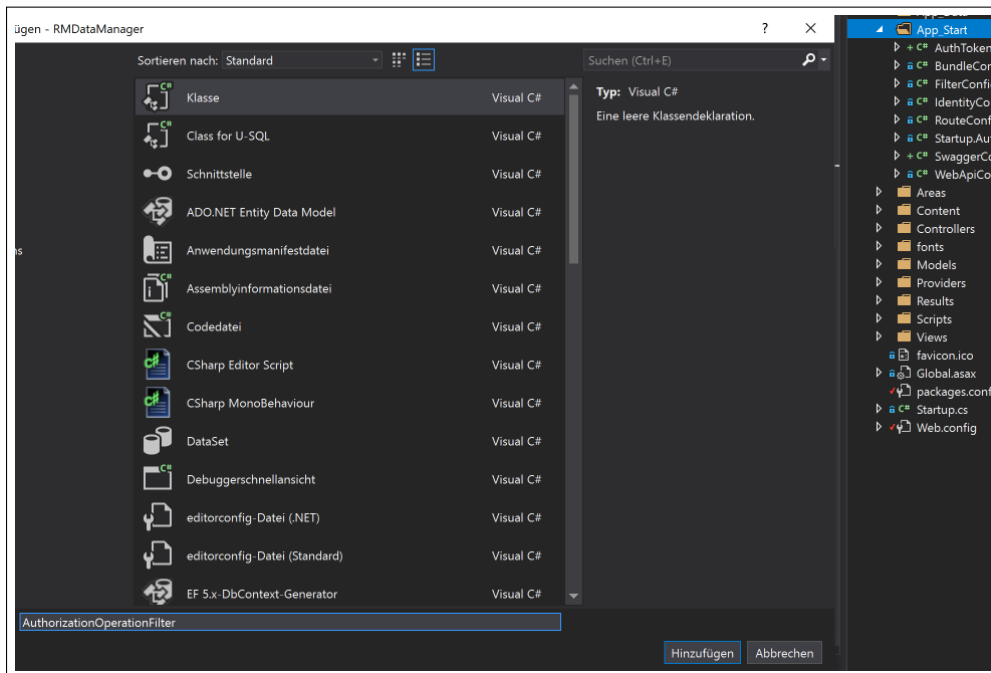
**Response Code**

```
200
```

**Response Headers**

```
{
  "pragma": "no-cache",
  "date": "Thu, 08 Aug 2019 12:18:20 GMT",
  "server": "Microsoft-IIS/10.0",
  "x-powered-by": "ASP.NET",
  "content-type": "application/json; charset=UTF-8",
  "cache-control": "no-cache",
  "x-sourcefiles": "JUTf-87B7FXxNYWnc5G9tZVxeb2N1bVudhMcUSR12G1bVx5ZXRhawxYVW5hZ2VzYXJNRG90YU1ibG9nZXJ0c3d0dW4=+",
  "content-length": "693",
  "expires": "-1"
}
```

## 2. Enabling to paste in the bearer token in order to authorize restricted commands

(a) Adding a new Class to `RMDataManager.App_Start`

(b) Implementing the required Interface

```

0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public class AuthorizationOperationFilter : IOperationFilter
{
    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public void Apply(Operation operation, SchemaRegistry schemaRegistry, ApiDescription apiDescription)
    {
        // Adding a parameter to each operation.
        if (operation.parameters == null)
        {
            operation.parameters = new List<Parameter>();
        }

        operation.parameters.Add(new Parameter
        {
            name = "Authorization",
            @in = "header",
            description = "access token",
            type = "string"
        });
    }
}

```

(c) Applying it to SwaggerConfig.cs

```

.EnableSwagger(configure: c =>
{
    c.DocumentFilter<AuthTokenOperation>(); // adding the implemented document filter
    c.OperationFilter<AuthorizationOperationFilter>(); // adding the implmented operation filter
}

```

(d) Get user information from the application via SWAGGER using the token

**Values** [Show/Hide](#) [List Operations](#) [Expand Operations](#)

**GET** /api/Values

Response Class (Status 200)  
OK

Model: **Example Value**

```
[
  "string"
]
```

Response Content Type: application/json ▼

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
Authorization	bearer PbvcDU53IDWQvxxWnxBaBOs57EWMrT6	access token	header	string

[Try it out!](#) [Hide Response](#)

**Curl**

```
curl -X GET --header 'Accept: application/json' --header 'Authorization: bearer PbvcDU53IDWQvxxWnxBaBOs57EWMrT6NoXhgIdem07kJoP1Vvh' http://localhost:54756/api/Values
```

**Request URL**

```
http://localhost:54756/api/Values
```

**Response Body**

```
[
  "value1",
  "value2",
  "2268F0be-21b1-4b31-98a5-8b9e32c2ea75"
]
```

**Response Code**

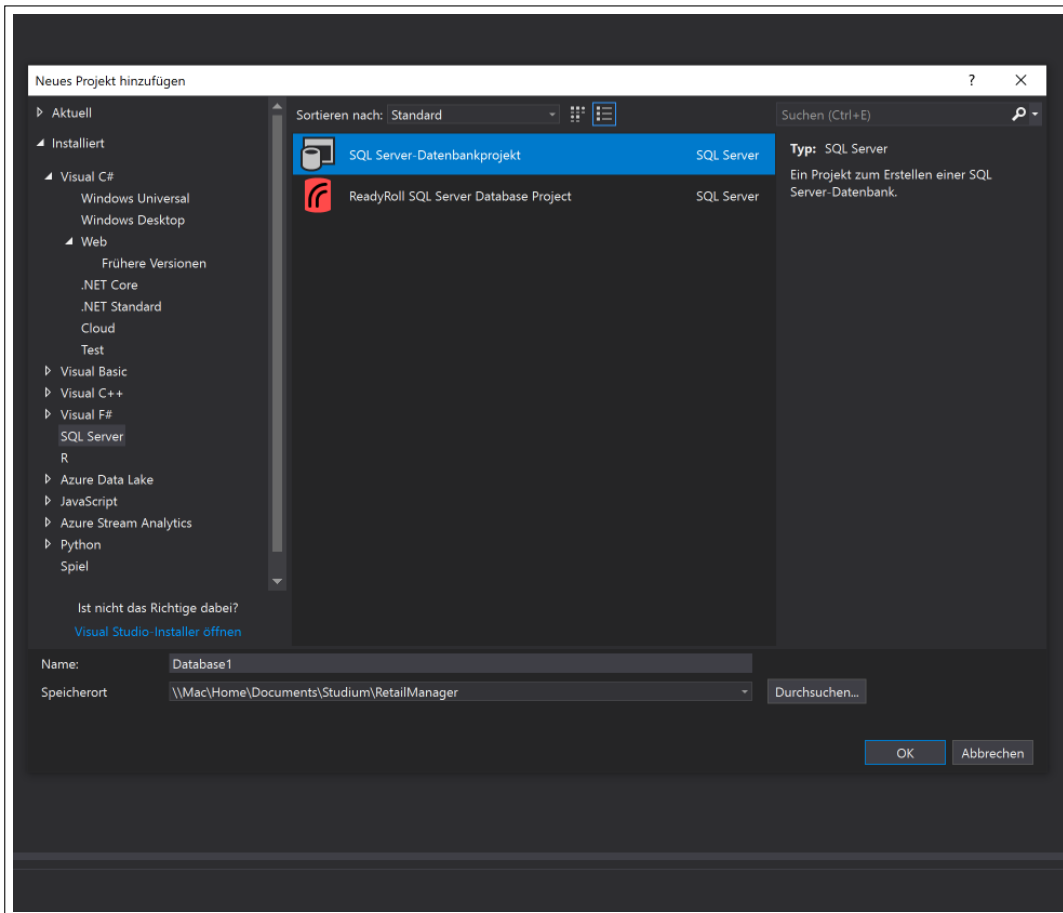
```
200
```

**Response Headers**

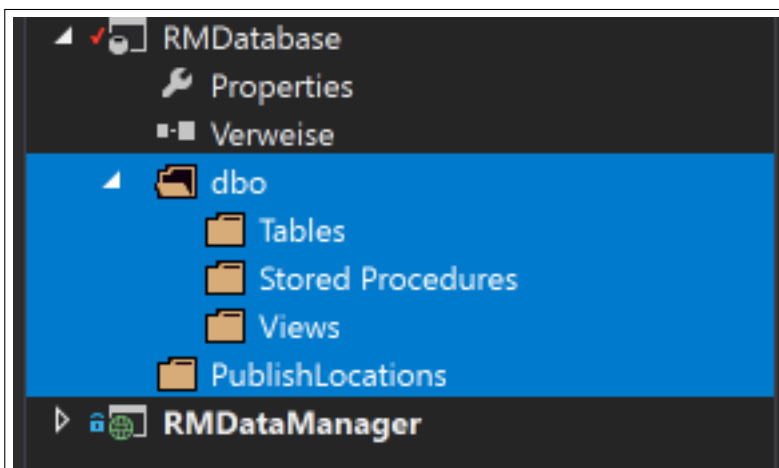
```
{
  "pragma": "no-cache",
  "date": "Thu, 08 Aug 2019 13:04:28 GMT",
  "server": "Microsoft-IIS/10.0",
  "x-aspnet-version": "4.0.30319",
  "x-powered-by": "ASP.NET",
  "content-type": "application/json; charset=utf-8",
  "cache-control": "no-cache",
  "x-sourcefiles": "~7UTF-878fXfxNvMhc569t2VxEb2N1bMvudHncU3R1ZG11bVxSZXRhbmRvYXhZVzVvYXZJNRGF0YU1hbmFnZXIjCvXBPfZhhbW1cw==7-",
  "content-length": "58",
  "expires": "-1"
}
```

## 6 SQL Database Setup

### 6.1 Adding new Database Project to the solution

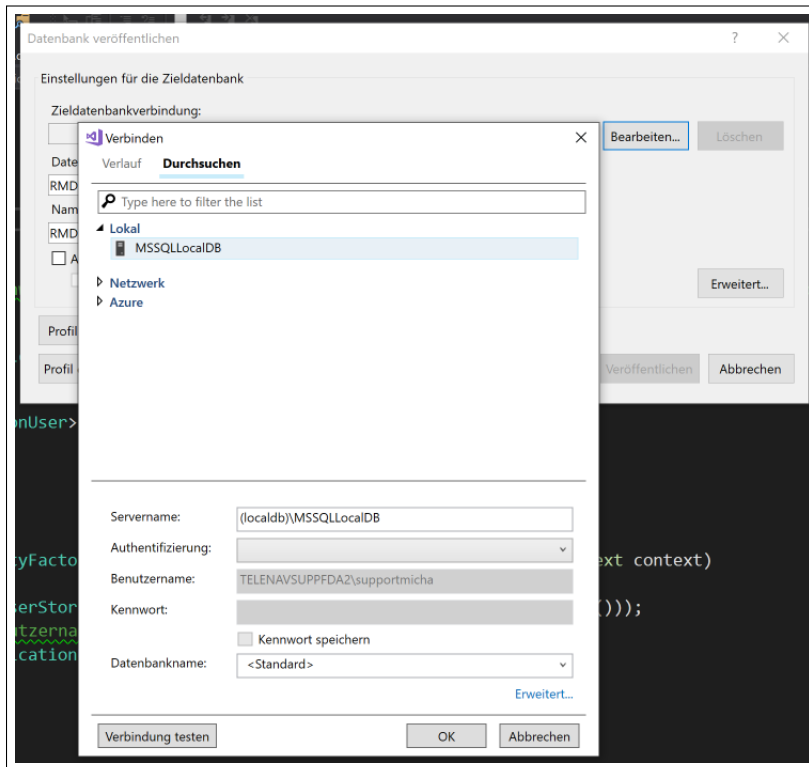


### 6.2 Adding several folders to the project

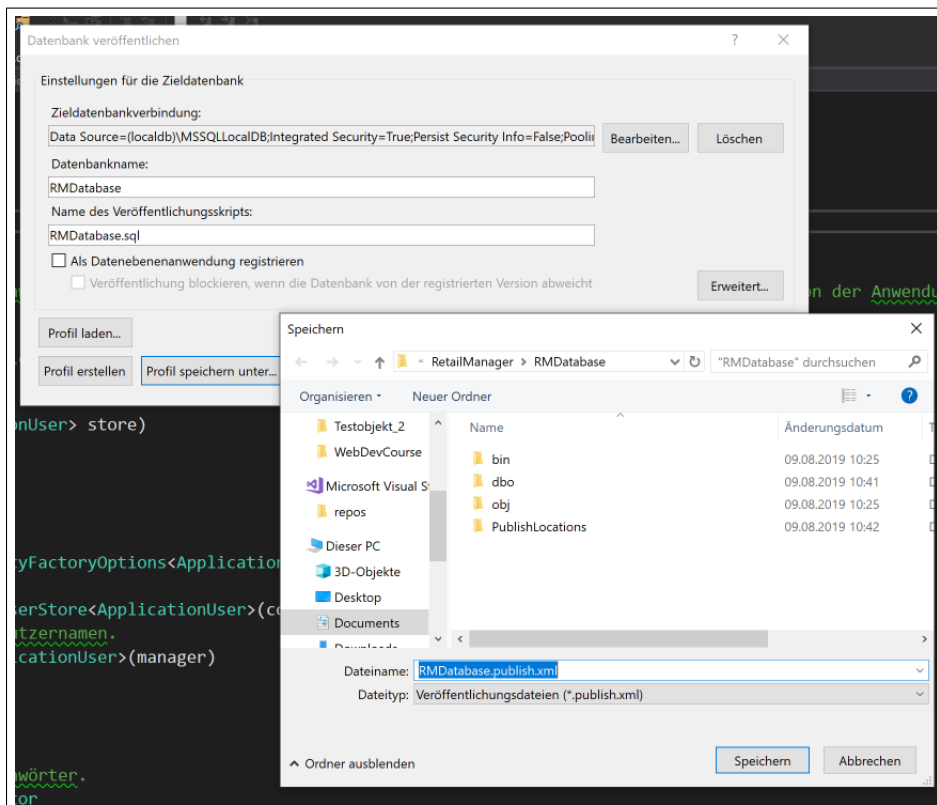


## 6.3 Creating a profile and publishing the Database

1. *RightClick on RMDatabase → Publish → Edit → Browse*

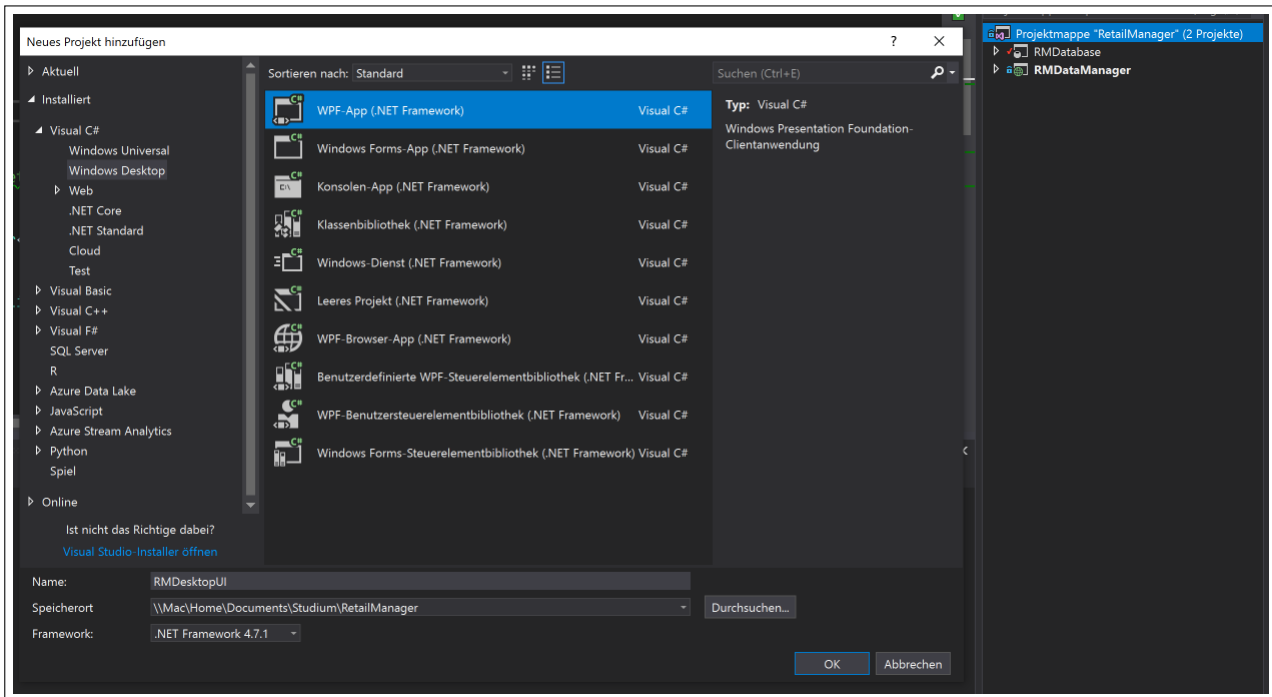


2. Naming and saving profile to PublishLocations

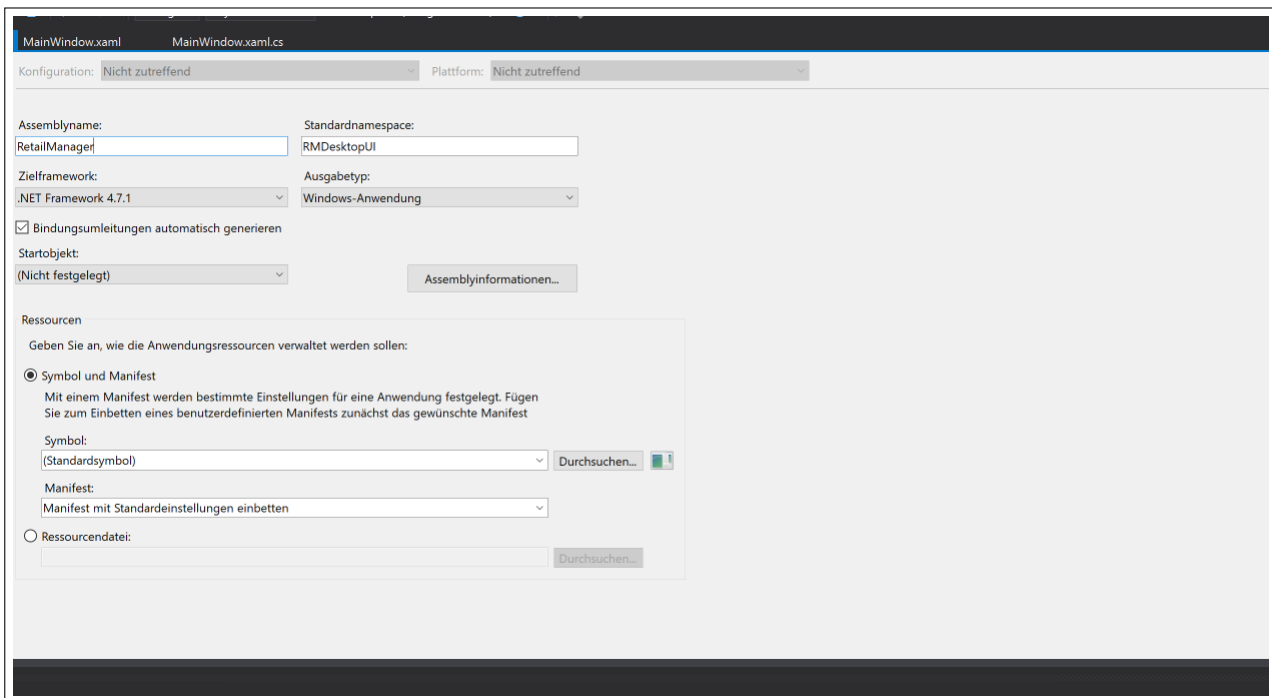


## 7 WPF with MVVM Project Setup

### 7.1 Adding the WPF Project to the solution



### 7.2 Changing the Assembly Name to the name of the solution in Properties



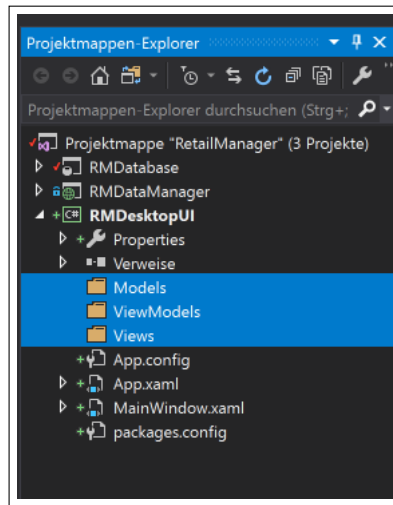
Also set project as the default startup-project.



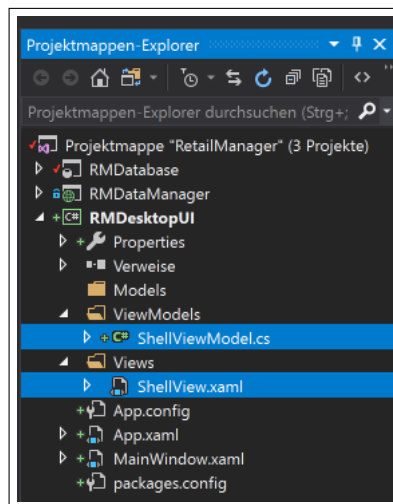
## 7.3 Adding Caliburn Micro MVVM-Framework

Add NuGet-Package to references.

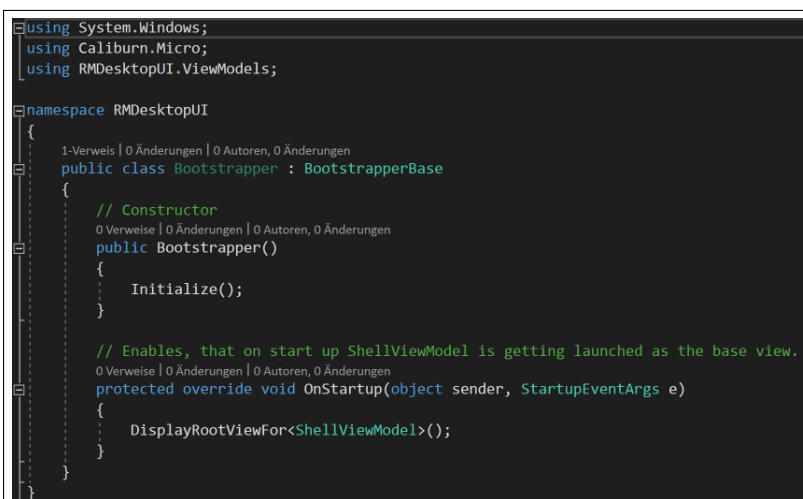
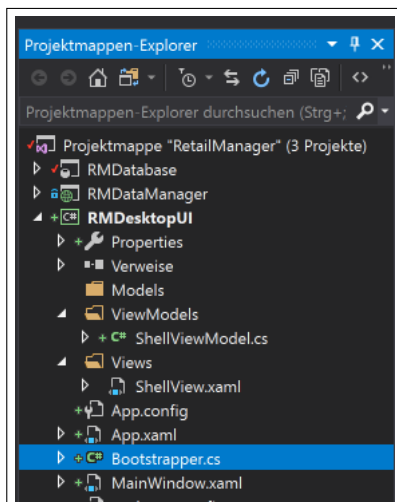
### 7.3.1 Adding the folder structure for the MVVM-Framework



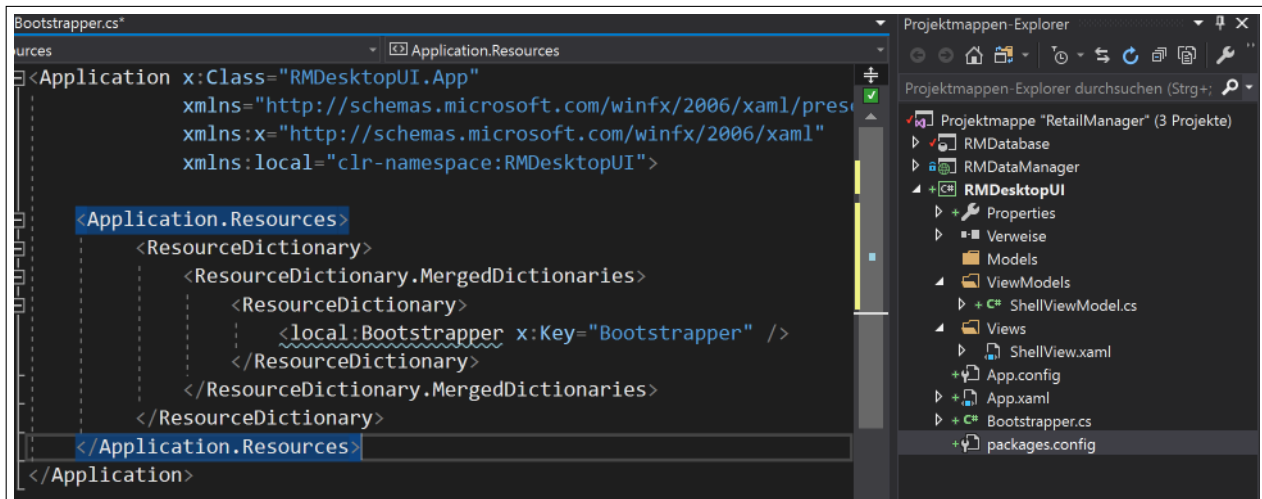
### 7.3.2 Adding a new ShellViewModel class and a SchellView window



### 7.3.3 Adding a Bootstrapper class to DesktopUI



### 7.3.4 Removing StartUpURI from App.xaml and adding a new Ressource Dictionary



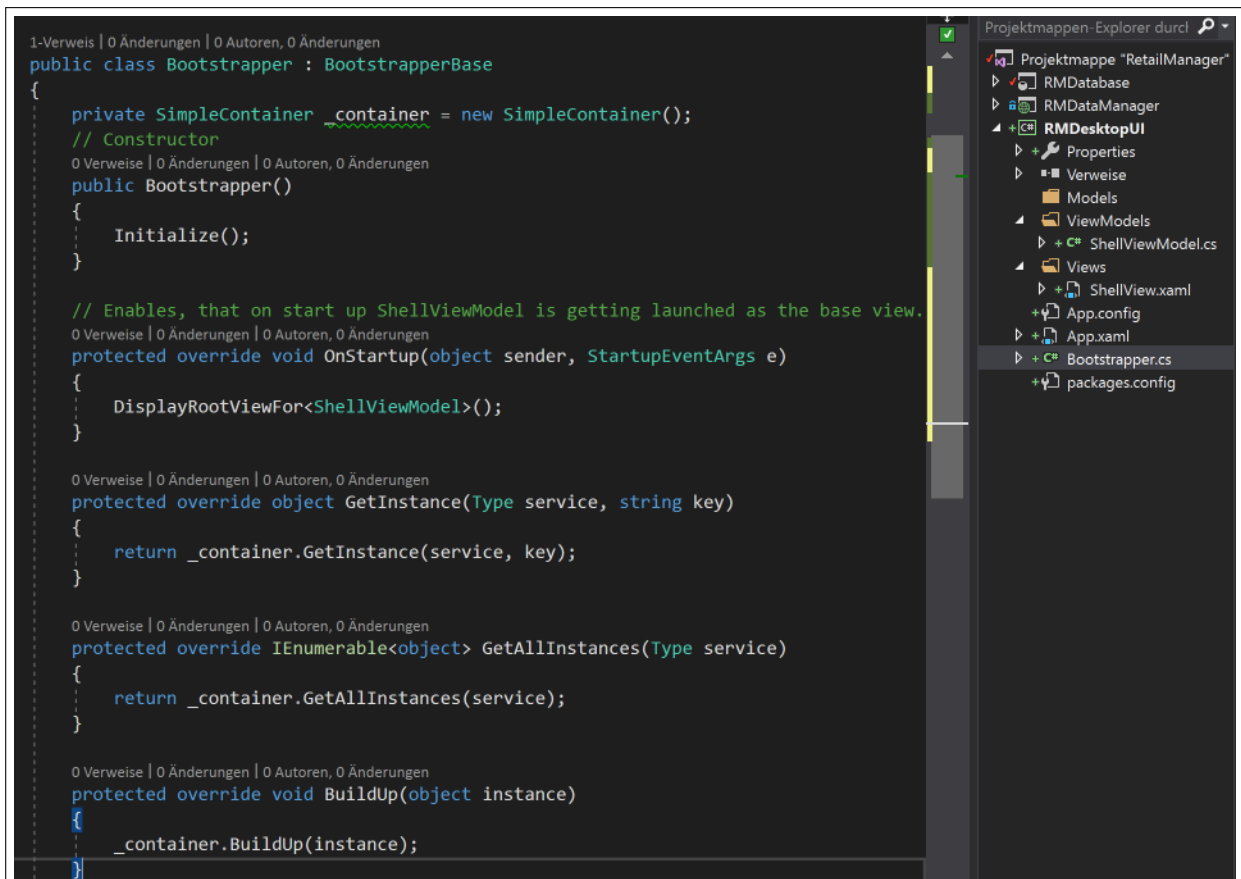
MainWindow.xaml can be deleted afterwards!!!

## 8 Dependency Injection in WPF

### 8.1 SimpleContainer in Caliburn Micro

Caliburn.Micro comes pre-bundled with a Dependency Injection container called SimpleContainer. A dependency injection container is an object that is used to hold dependency mappings for use later in an app via Dependency Injection. Dependency Injection is actually a pattern typically using the container element instead of manual service mapping.

#### 8.1.1 Implementing SimpleContainer in Bootstrapper.cs



## 8.2 Overriding Configure() Method for the container

```
1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
public class Bootstrapper : BootstrapperBase
{
    private SimpleContainer _container = new SimpleContainer();
    // Constructor
    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public Bootstrapper()
    {
        Initialize();
    }

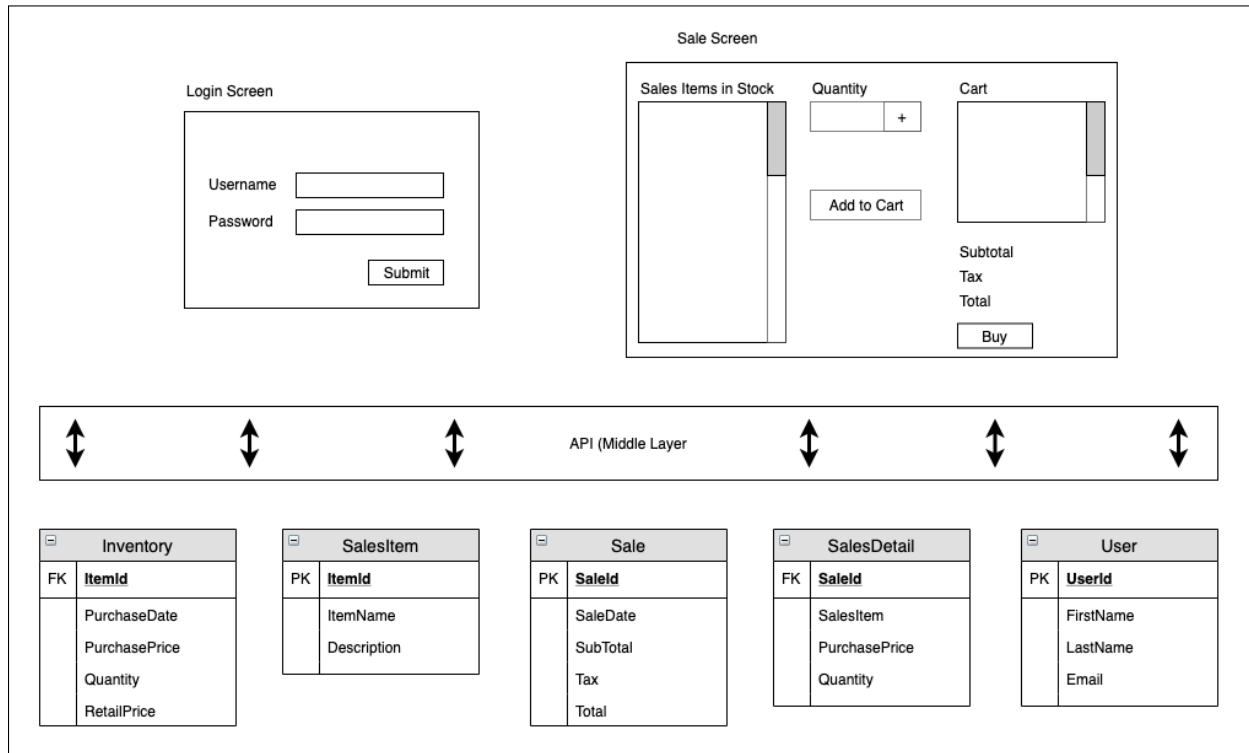
    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    protected override void Configure()
    {
        _container.Instance(_container);

        _container
            .Singleton<IWindowManager, WindowManager>()
            .Singleton<EventAggregator, EventAggregator>();

        // Connecting the ViewModel to the Views using reflection
        GetType().Assembly.GetTypes()
            .Where(type => type.IsClass)
            .Where(type => type.Name.EndsWith(value: "ViewModel"))
            .ToList()
            .ForEach(action: viewModelType => _container.RegisterPerRequest(
                service: viewModelType, viewModelType.ToString(), implementation: viewModelType));
    }
}
```

## 9 Datamodel - planning and setup

### 9.1 Planning the Register



## 9.2 SQL Database Table Creation

## User.sql

Name	Datentyp	NULL-Werte zulassen	Standard
Id	nvarchar(128)	<input type="checkbox"/>	
FirstName	nvarchar(50)	<input type="checkbox"/>	
LastName	nvarchar(50)	<input type="checkbox"/>	
EmailAddress	nvarchar(256)	<input type="checkbox"/>	
CreatedDate	datetime2(7)	<input type="checkbox"/>	getutcdate()

```

1 CREATE TABLE [dbo].[User]
2 (
3     [Id] NVARCHAR(128) NOT NULL,
4     [FirstName] NVARCHAR(50) NOT NULL,
5     [LastName] NVARCHAR(50) NOT NULL,
6     [EmailAddress] NVARCHAR(256) NOT NULL,
7     [CreatedDate] DATETIME2 NOT NULL DEFAULT getutcdate()
8 )
9

```

## Product.sql

Name	Datentyp	NULL-Werte zulassen	Standard
Id	int	<input type="checkbox"/>	
ProductName	nvarchar(100)	<input type="checkbox"/>	
Description	nvarchar(MAX)	<input type="checkbox"/>	
RetailPrice	money	<input type="checkbox"/>	
CreateDate	datetime2(7)	<input type="checkbox"/>	getutcdate()
LastModified	datetime2(7)	<input type="checkbox"/>	getutcdate()

```

1 CREATE TABLE [dbo].[Product]
2 (
3     [Id] INT NOT NULL PRIMARY KEY IDENTITY,
4     [ProductName] NVARCHAR(100) NOT NULL,
5     [Description] NVARCHAR(MAX) NOT NULL,
6     [RetailPrice] MONEY NOT NULL,
7     [CreateDate] DATETIME2 NOT NULL DEFAULT getutcdate(),
8     /* Has to be modified manually everytime the entry gets modified.*/
9     [LastModified] DATETIME2 NOT NULL DEFAULT getutcdate()
10 )

```

## Sale.sql

Name	Datentyp	NULL-Werte zulassen	Standard
Id	int	<input type="checkbox"/>	
CashierId	nvarchar(128)	<input type="checkbox"/>	
SaleDate	datetime2(7)	<input type="checkbox"/>	
SubTotal	money	<input type="checkbox"/>	
Tax	money	<input type="checkbox"/>	
Total	money	<input type="checkbox"/>	

```

1 CREATE TABLE [dbo].[Sale]
2 (
3     [Id] INT NOT NULL PRIMARY KEY IDENTITY, /* IDENTITY Makes the Id auto increment */
4     [CashierId] NVARCHAR(128) NOT NULL,
5     [SaleDate] DATETIME2 NOT NULL,
6     [SubTotal] MONEY NOT NULL,
7     [Tax] MONEY NOT NULL,
8     [Total] MONEY NOT NULL
9 )

```

## SaleDetail.sql

Name	Datentyp	NULL-Werte zulassen	Standard
Id	int	<input type="checkbox"/>	
SaleId	int	<input type="checkbox"/>	
ProductId	int	<input type="checkbox"/>	
Quantity	nchar(10)	<input checked="" type="checkbox"/>	1
PurchasePrice	money	<input type="checkbox"/>	
Tax	money	<input type="checkbox"/>	0

```

1 CREATE TABLE [dbo].[SaleDetail]
2 (
3     [Id] INT NOT NULL PRIMARY KEY IDENTITY,
4     [SaleId] INT NOT NULL,
5     [ProductId] INT NOT NULL,
6     [Quantity] NCHAR(10) NOT NULL DEFAULT 1,
7     [PurchasePrice] MONEY NOT NULL,
8     [Tax] MONEY NOT NULL DEFAULT 0,
9 )

```

## Inventory.sql

Name	Datentyp	NULL-Werte zulassen	Standard
Id	int	<input type="checkbox"/>	
ProductId	int	<input type="checkbox"/>	
Quantity	nchar(10)	<input type="checkbox"/>	1
PurchasePrice	money	<input type="checkbox"/>	
PurchaseDate	datetime2(7)	<input checked="" type="checkbox"/>	getutcdate()

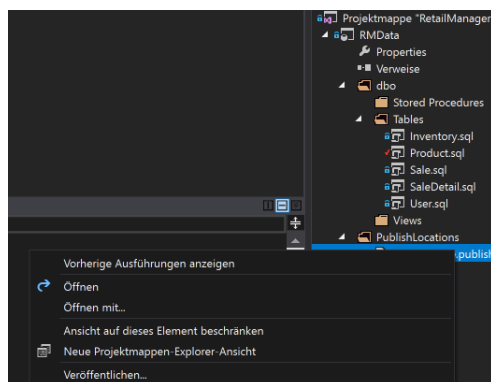
  

```

1 CREATE TABLE [dbo].[Inventory]
2 (
3     [Id] INT NOT NULL PRIMARY KEY IDENTITY,
4     [ProductId] INT NOT NULL,
5     [Quantity] NCHAR(10) NOT NULL DEFAULT 1,
6     [PurchasePrice] MONEY NOT NULL,
7     [PurchaseDate] DATETIME2 NOT NULL DEFAULT getutcdate()
8 )
9

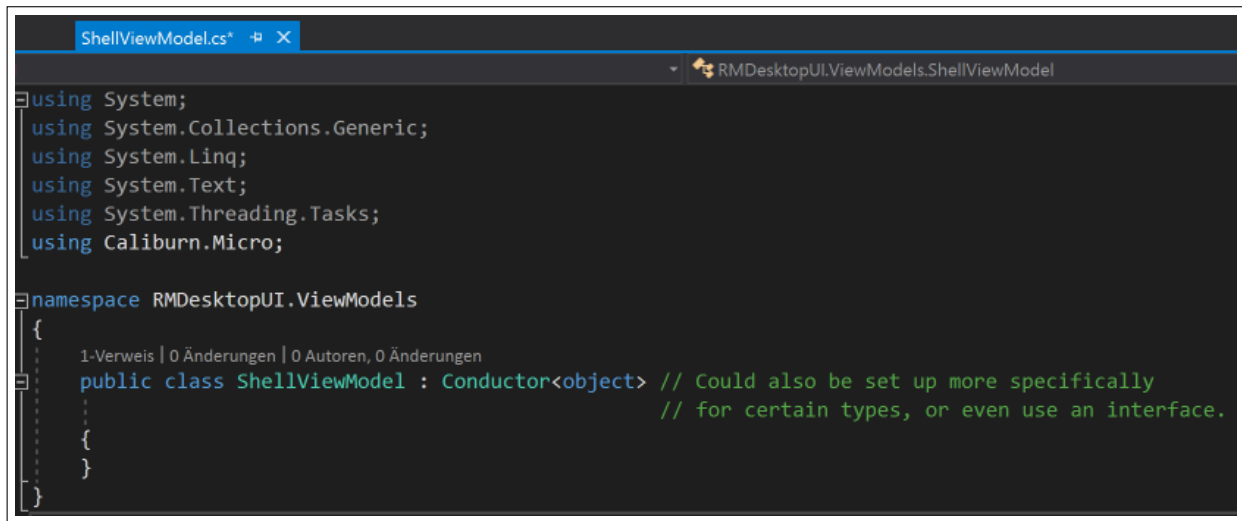
```

## !!! Publishing Tables !!!



## 10 WPF Login Form Creation

### 10.1 Inheritance from the conductor class in Caliburn Micro



```

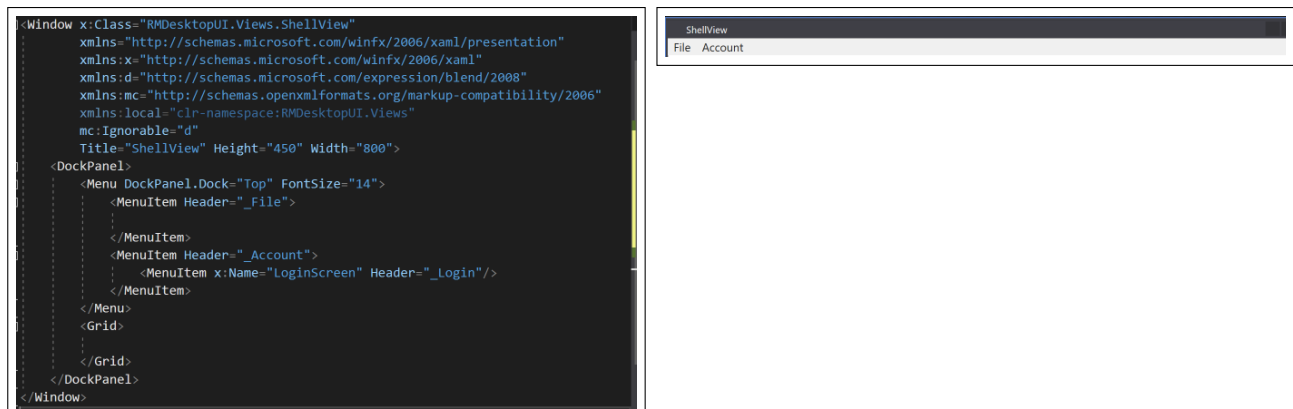
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Caliburn.Micro;

namespace RMDesktopUI.ViewModels
{
    1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class ShellViewModel : Conductor<object> // Could also be set up more specifically
                                                // for certain types, or even use an interface.
    {
    }
}

```

Conductor is a base class which inherits from Screen. Its responsibility is to conduct other objects by managing an active item and maintain a strict lifecycle of this conducted item. The conductor exists in multiple variants such as the one item conductor simple called Conductor, the multiple item conductors such as *Conductor.Collection.OneActive* and *Conductor.Collection.AllActive*.

### 10.2 Implementing the menu bar



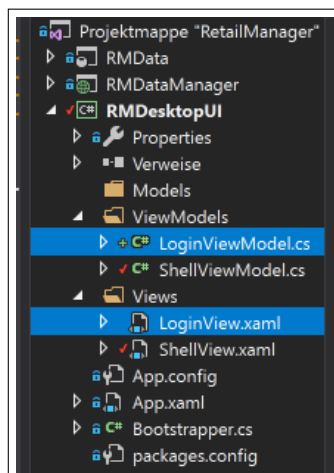
```

<Window x:Class="RMDesktopUI.Views.ShellView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:RMDesktopUI.Views"
        mc:Ignorable="d"
        Title="ShellView" Height="450" Width="800">
    <DockPanel>
        <Menu DockPanel.Dock="Top" FontSize="14">
            <MenuItem Header="_File">
                </MenuItem>
            <MenuItem Header="_Account">
                <MenuItem x:Name="LoginScreen" Header="_Login"/>
            </MenuItem>
        </Menu>
        <Grid>
            </Grid>
        </DockPanel>
    </Window>

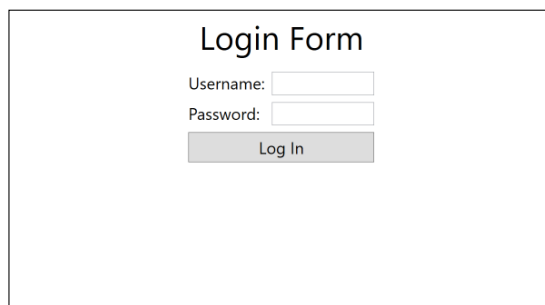
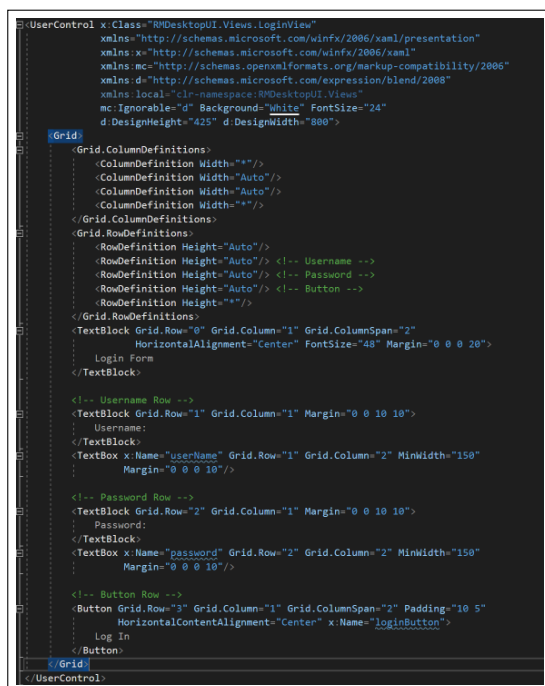
```

### 10.3 Adding a UserControl

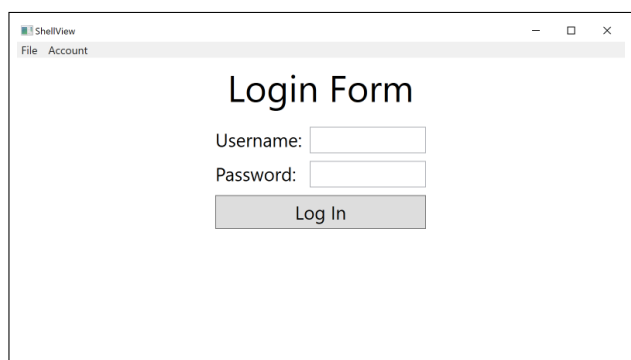
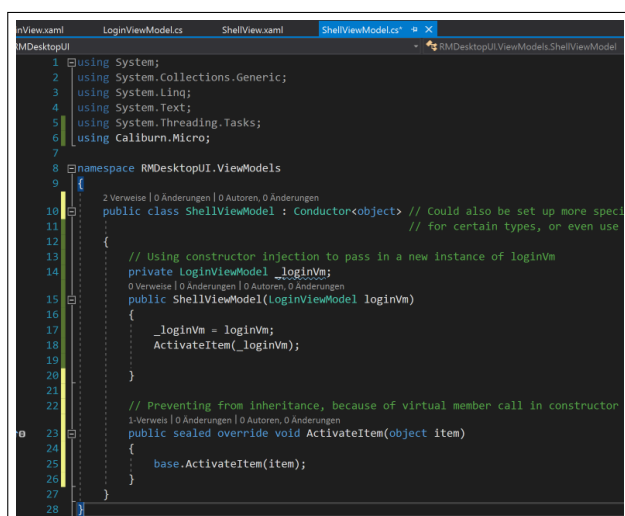
#### 10.3.1 Adding a class LoginViewModel (public) and UserControl LoginView



## 10.3.2 Designing the UserControl



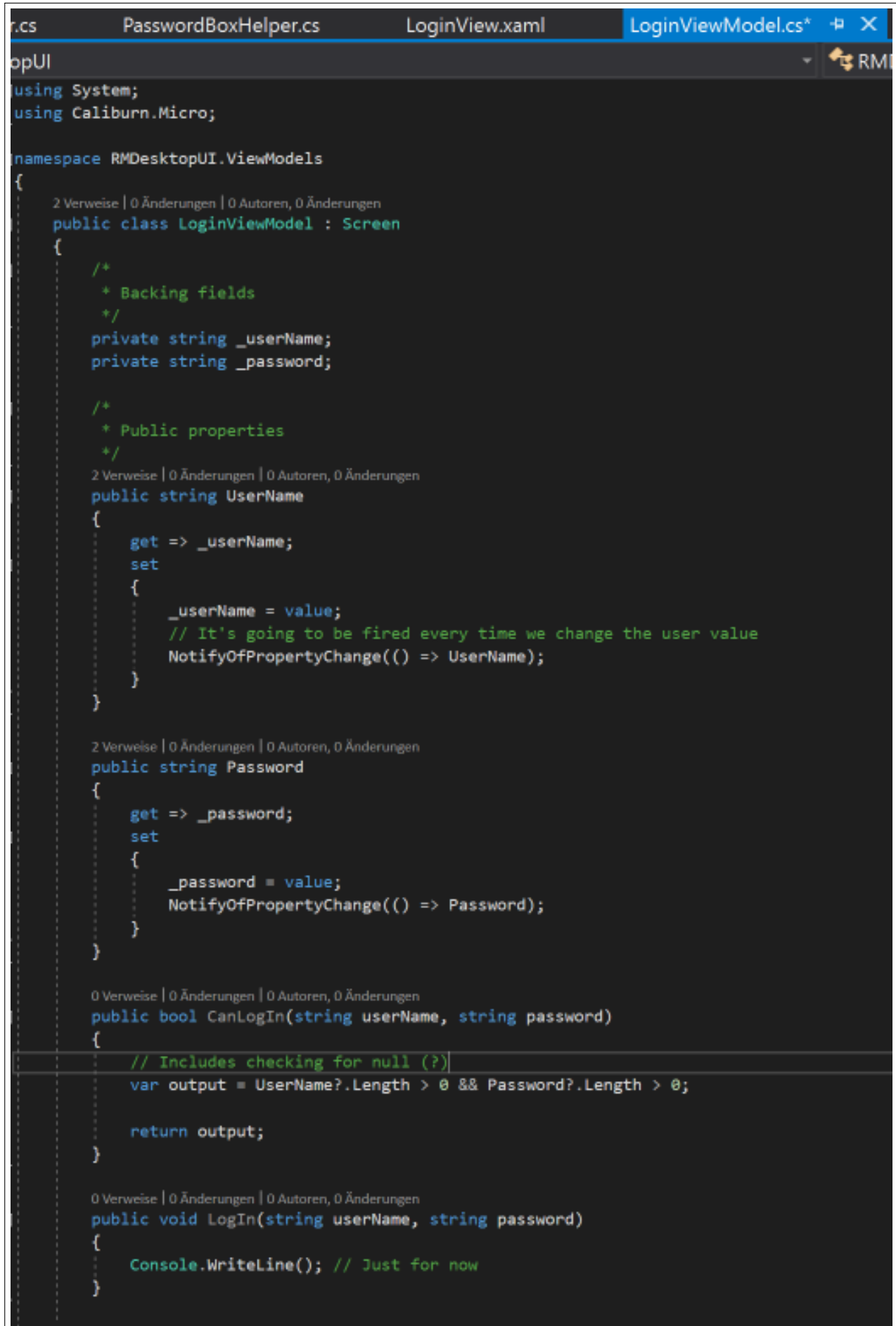
## 10.3.3 Activating the LoginView on startup in the ShellView



Sealed is used to restrict the users from inheriting. A class can be sealed by using the sealed keyword or a single method. The keyword tells the compiler that class or method cannot be extended. No class can be derived from a sealed class.



## 10.3.4 Implementing LoginViewModel.cs



```

LoginViewModel.cs
PasswordBoxHelper.cs
LoginView.xaml
LoginViewModel.cs*
RMDesktopUI
using System;
using Caliburn.Micro;

namespace RMDesktopUI.ViewModels
{
    2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class LoginViewModel : Screen
    {
        /*
         * Backing fields
         */
        private string _userName;
        private string _password;

        /*
         * Public properties
         */
        2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string UserName
        {
            get => _userName;
            set
            {
                _userName = value;
                // It's going to be fired every time we change the user value
                NotifyOfPropertyChange(() => UserName);
            }
        }

        2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string Password
        {
            get => _password;
            set
            {
                _password = value;
                NotifyOfPropertyChange(() => Password);
            }
        }

        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public bool CanLogIn(string userName, string password)
        {
            // Includes checking for null (?)
            var output = UserName?.Length > 0 && Password?.Length > 0;

            return output;
        }

        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public void LogIn(string userName, string password)
        {
            Console.WriteLine(); // Just for now
        }
    }
}

```

### 10.3.5 Connecting the LoginViewModel to Caliburn.Micro

1. Adding a helper class (PasswordBoxHelper.cs) to RMDesktopUI

```
using System.Reflection;
using System.Windows;
using System.Windows.Controls;

namespace RMDesktopUI.Helpers
{
    /*
     * The aim of this class is to include a binding convention so that binding in Caliburn.Micro
     * Source: https://stackoverflow.com/questions/30631522/caliburn-micro-support-for-password
     */
    1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
    public static class PasswordBoxHelper
    {
        public static readonly DependencyProperty BoundPasswordProperty =
            DependencyProperty.RegisterAttached(name: "BoundPassword",
                propertyType: typeof(string),
                ownerType: typeof(PasswordBoxHelper),
                new FrameworkPropertyMetadata(defaultValue: string.Empty, OnBoundPasswordChanged));

        2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public static string GetBoundPassword(DependencyObject d)
        {
            if (!(d is PasswordBox box)) return (string) d.GetValue(dp: BoundPasswordProperty);

            // this funny little dance here ensures that we've hooked the
            // PasswordChanged event once, and only once.
            box.PasswordChanged -= PasswordChanged;
            box.PasswordChanged += PasswordChanged;

            return (string)d.GetValue(dp: BoundPasswordProperty);
        }

        1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
        public static void SetBoundPassword(DependencyObject d, string value)
        {
            ...
        }
    }
}
```

2. Adding some lines to the constructor of Bootstrapper.cs

```
1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
public class Bootstrapper : BootstrapperBase
{
    private readonly SimpleContainer _container = new SimpleContainer();
    // Constructor
    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public Bootstrapper()
    {
        Initialize();

        // Source: https://stackoverflow.com/questions/30631522/caliburn-micro-support-for-passwordbox
        ConventionManager.AddElementConvention<PasswordBox>(
            bindableProperty: PasswordBoxHelper.BoundPasswordProperty,
            parameterProperty: "Password",
            eventName: "PasswordChanged");
    }
}
```

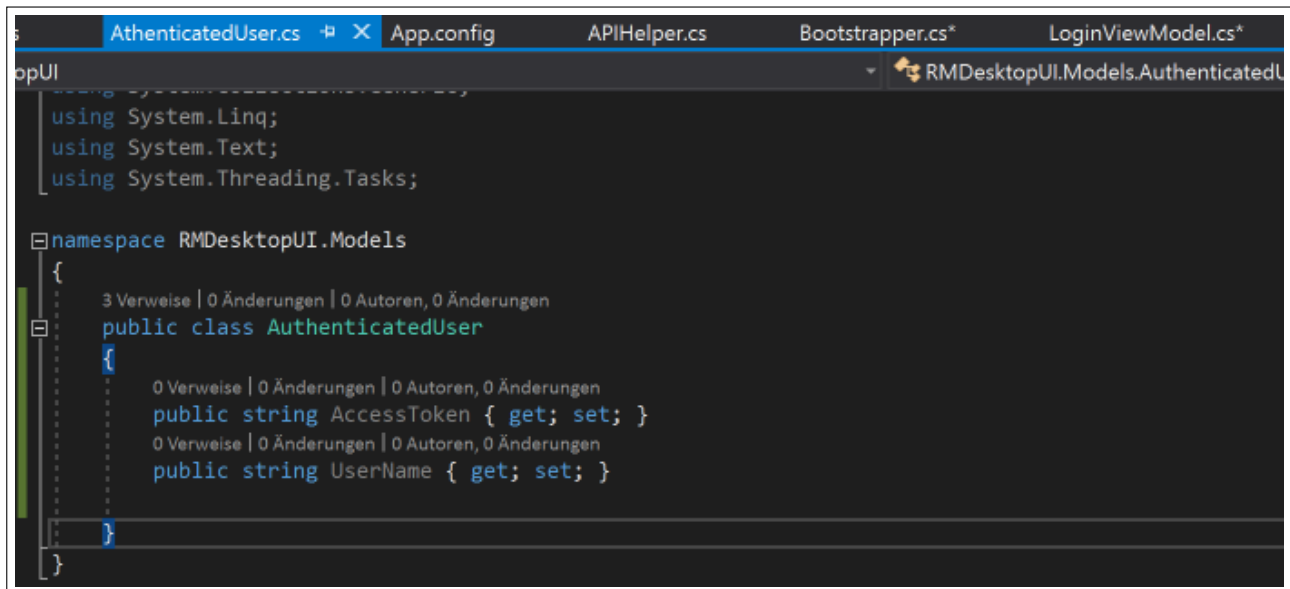
See the example on [stackoverflow.com](https://stackoverflow.com)...

## 11 Wiring up the WPF Login Form

Connecting the login form button to the authentication API endpoint (/token). Gets back the bearer token or an exception if failed.

### 11.1 Implementing a class `AuthenticatedUser.cs`

Holds the information for already authenticated users.

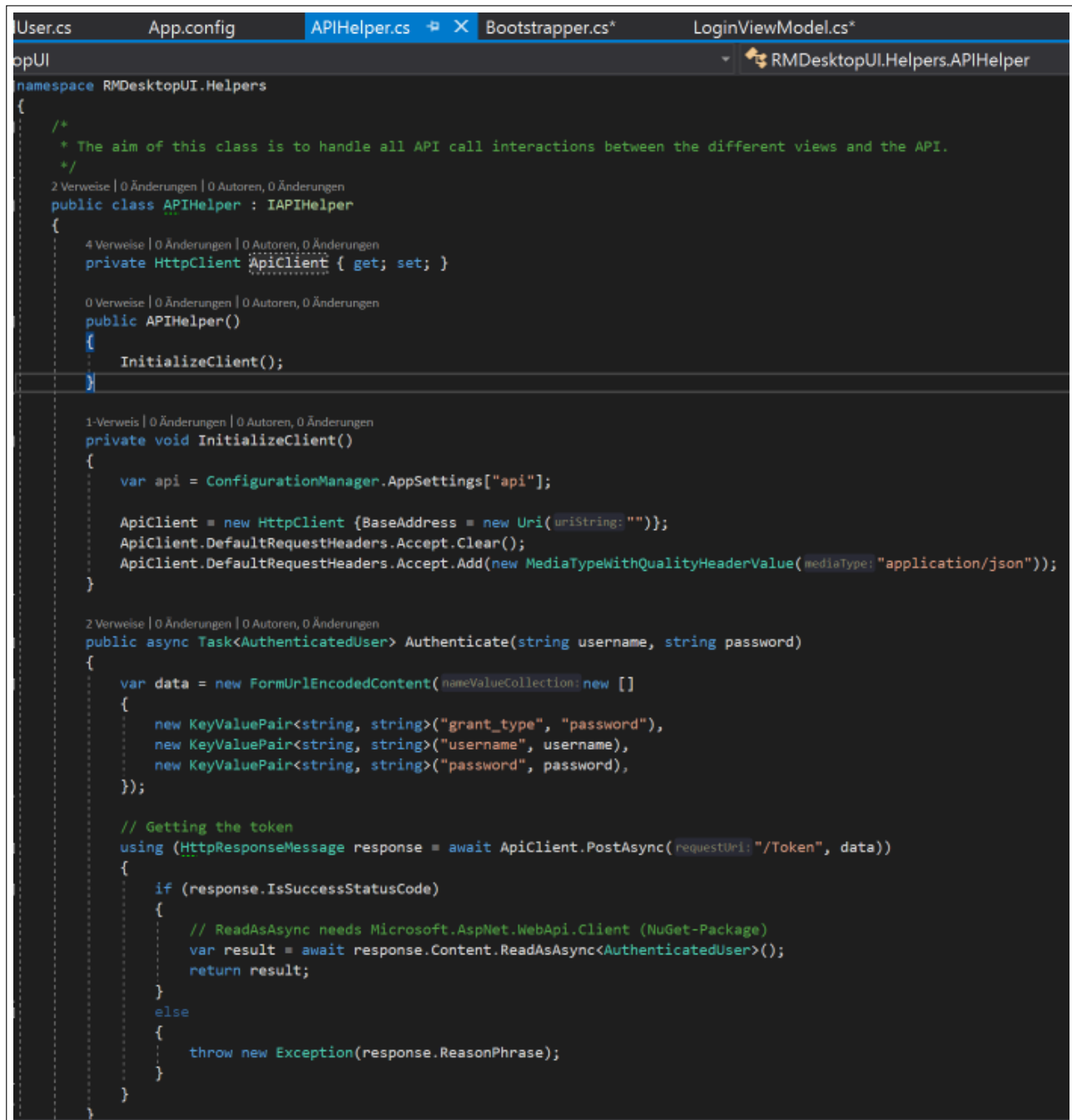


```
AuthenticatedUser.cs App.config APIHelper.cs Bootstrapper.cs* LoginViewModel.cs*
RMDesktopUI.Models.AuthenticatedU

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RMDesktopUI.Models
{
    3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class AuthenticatedUser
    {
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string AccessToken { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string UserName { get; set; }
    }
}
```

## 11.2 Implementing a helper class to handle API call interactions



```
User.cs    App.config    APIHelper.cs    Bootstrapper.cs*    LoginViewModel.cs*
opUI
namespace RMDesktopUI.Helpers
{
    /*
     * The aim of this class is to handle all API call interactions between the different views and the API.
     */
    2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class APIHelper : IAPIHelper
    {
        4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        private HttpClient ApiClient { get; set; }

        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public APIHelper()
        {
            InitializeClient();
        }

        1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
        private void InitializeClient()
        {
            var api = ConfigurationManager.AppSettings["api"];

            ApiClient = new HttpClient {BaseAddress = new Uri(uriString: "")};
            ApiClient.DefaultRequestHeaders.Accept.Clear();
            ApiClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue(mediaType: "application/json"));
        }

        2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public async Task<AuthenticatedUser> Authenticate(string username, string password)
        {
            var data = new FormUrlEncodedContent(nameValueCollection: new []
            {
                new KeyValuePair<string, string>("grant_type", "password"),
                new KeyValuePair<string, string>("username", username),
                new KeyValuePair<string, string>("password", password),
            });

            // Getting the token
            using (HttpResponseMessage response = await ApiClient.PostAsync(requestUri: "/Token", data))
            {
                if (response.IsSuccessStatusCode)
                {
                    // ReadAsStringAsync needs Microsoft.AspNet.WebApi.Client (NuGet-Package)
                    var result = await response.Content.ReadAsStringAsync<AuthenticatedUser>();
                    return result;
                }
                else
                {
                    throw new Exception(response.ReasonPhrase);
                }
            }
        }
    }
}
```

### 11.3 Implementing a Interface IAPIHelper.cs

Needed for dependency injection, in order to add it to the Configure() method in Bootstrapper.cs

```

AuthenticatedUser.cs  App.config  APIHelper.cs  Bootstrapper.cs*  LoginViewModel.cs*
opUI
using System.Threading.Tasks;
using RMDesktopUI.Models;

namespace RMDesktopUI.Helpers
{
    4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public interface IAPIHelper
    {
        2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        Task<AuthenticatedUser> Authenticate(string username, string password);
    }
}

```

### 11.4 Adding <appsettings> to App.Config

```

<configuration>
  <appSettings>
    <!--The value here is the project url, which can be found in properties/web.
         Can be changed at any time on runtime. It's possible to create several App.Config overrides.-->
    <add key="api" value="http://localhost:54756/" />
  </appSettings>
  <startup>

```

### 11.5 Adding APIHelper and IAPIHelper to the container in Bootstrapper.cs

```

        _container
            .Singleton<IWindowManager, WindowManager>()
            .Singleton<IEventAggregator, EventAggregator>()
            .Singleton<IAPIHelper, APIHelper>(); // Enables keeping the http-client open
                                                // until the application gets closed

```

### 11.6 Applying some changings to LoginViewModel.cs

#### 11.6.1 Adding a new private property as a backing field

```

3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public class LoginViewModel : Screen
{
    /*
     * Backing fields
     */
    private string _userName;
    private string _password;

    private readonly IAPIHelper _apiHelper;
}

```

### 11.6.2 Adding a constructor and initializing the property from within

```

/*
 * Constructor
 */
0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public LoginViewModel(IAPIHelper apiHelper)
{
    _apiHelper = apiHelper;
}

```

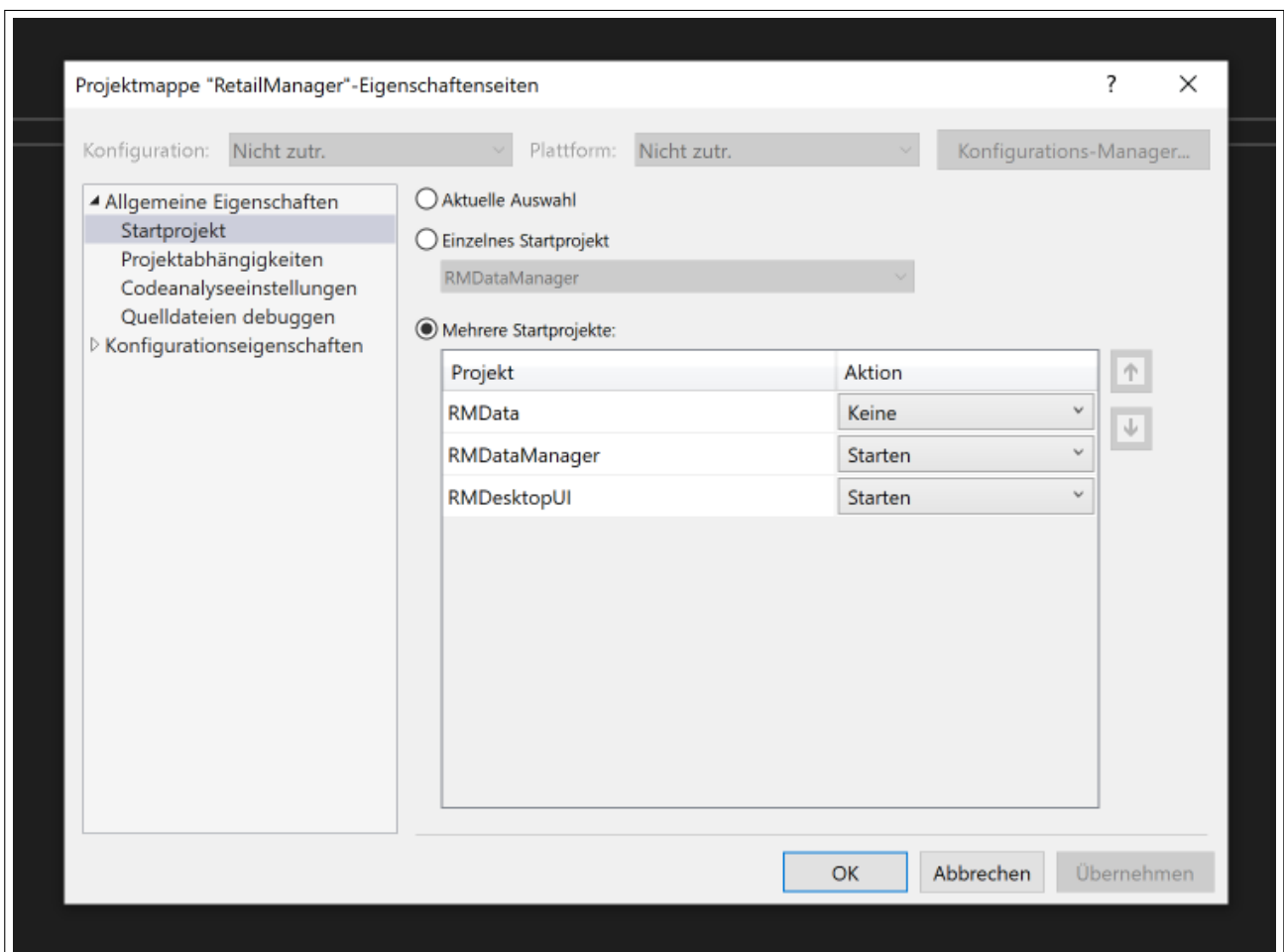
### 11.6.3 Implementing the Login() method

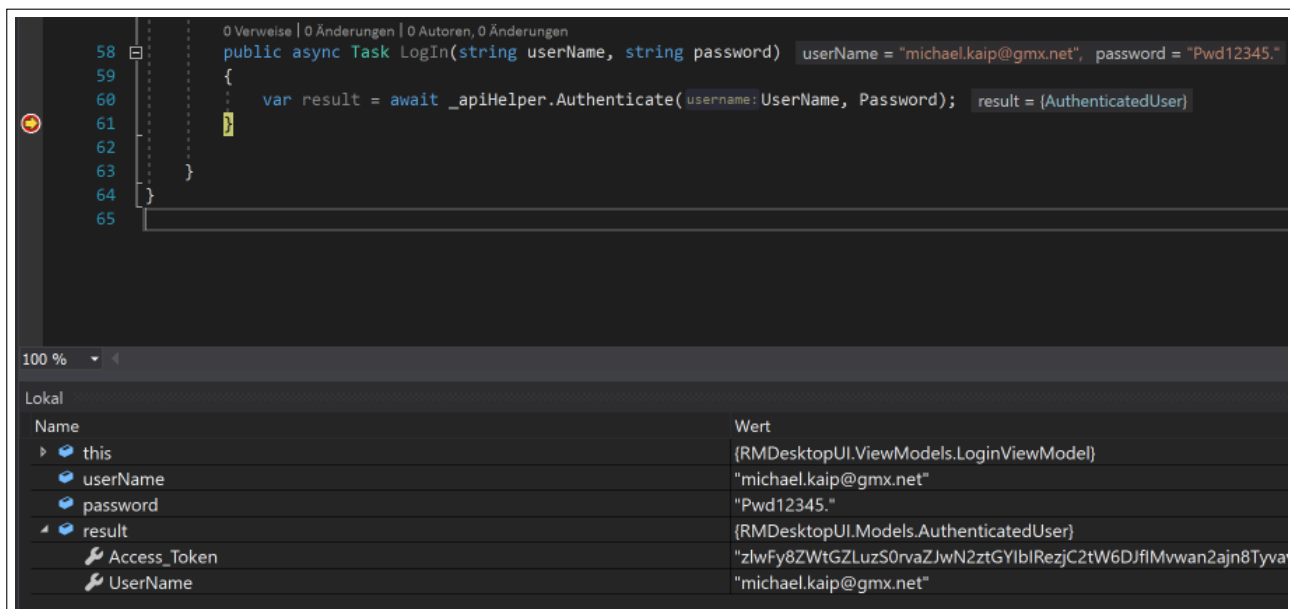
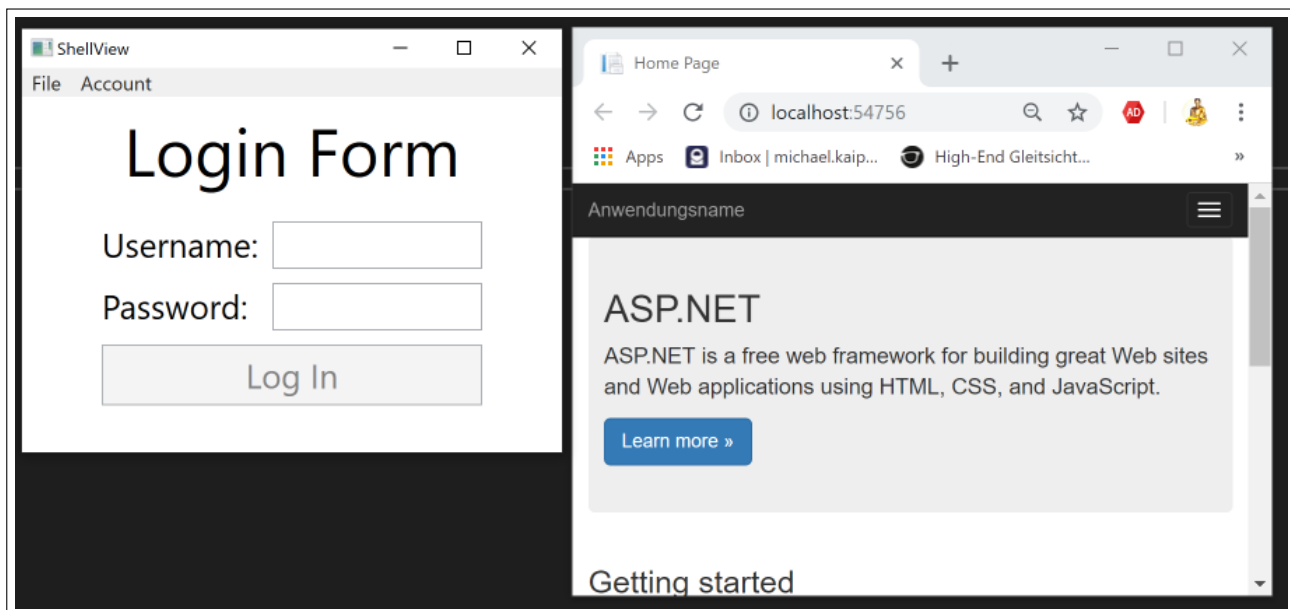
```

0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public async Task LogIn(string userName, string password)
{
    try
    {
        var result = await _apiHelper.Authenticate(username: UserName, Password);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

## 11.7 Enabling the solution to start multiple projects





## 12 Login Form Error Handling

### 12.1 Displaying an login error message within the login form

```

Grid.RowDefinitions>
  <RowDefinition Height="Auto"/>
  <RowDefinition Height="Auto"/> <!-- Error Message -->
  <RowDefinition Height="Auto"/> <!-- Username -->
  <RowDefinition Height="Auto"/> <!-- Password -->
  <RowDefinition Height="Auto"/> <!-- Button -->
  <RowDefinition Height="*/>
</Grid.RowDefinitions>
<TextBlock Grid.Row="0" Grid.Column="1" Grid.ColumnSpan="2"
  HorizontalAlignment="Center" FontSize="48" Margin="0 0 0 20">
  Login
</TextBlock>

<!-- Error Message Row -->
<TextBlock x:Name="ErrorMessage"
  Grid.Row="1" Grid.Column="1" Grid.ColumnSpan="2"
  Margin="0 0 0 20" Foreground="Red" MaxWidth="270"
  Visibility="{Binding IsErrorVisible, Converter={StaticResource BooleanToVisibilityConverter}}"
  FontSize="12" TextWrapping="Wrap"/>

<!-- Username Row -->

```

Through `Visibility` the functionality of collapsing the error message space in case of no error is going to be displayed is added. In case of an error the field expands and the error is going to be displayed. To make it work, first the `BooleanToVisibilityConverter` has to be added to the ResourceDictionary in App.xaml:

```

<local:Bootstrapper x:Key="Bootstrapper" />
</ResourceDictionary>
</ResourceDictionary.MergedDictionaries>
<BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>
</ResourceDictionary>
</Application.Resources>
</Application>

```

Afterwards properties for `ErrorMessage` and `IsErrorVisible` has to be implemented in Login-ViewModel.cs:



```
private string _errorMessage;

3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public string ErrorMessage
{
    get => _errorMessage;
    set
    {
        NotifyOfPropertyChanged(() => IsErrorVisible);
        NotifyOfPropertyChanged(() => ErrorMessage);
        _errorMessage = value;
    }
}

/*
 * Properties
 */
1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
public bool IsErrorVisible
{
    get
    {
        var output = ErrorMessage?.Length > 0;

        return output;
    }
}
```

And finally the Login() method has to be changed in the way, that in case of an exception, the exception message is stored into the `ErrorMessage` property:

```
0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public async Task Login(string userName, string password)
{
    try
    {
        ErrorMessage = "";

        var result = await _apiHelper.Authenticate(username: UserName, Password);
    }
    catch (Exception ex)
    {
        ErrorMessage = ex.Message;
    }
}
```

## 13 Getting User Data

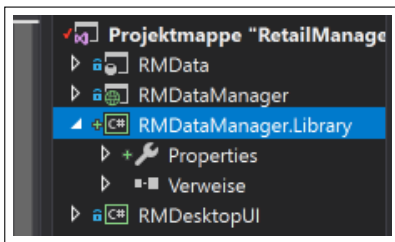
Getting all information about a logged in user from the database. This information is going to be stored in an singleton object and therefore can be used as long as the user is logged in. Once he's logged out, the corresponding object which holds the user information is going to be destroyed.

In order to make it work, following things has to be done:

- implementing a stored procedure
- implementing a model to store the data in the API
- creating an API endpoint
- implementing a method that calls the endpoint from the WPF application
- implementing a model that hold data in the WPF application

### 13.1 Adding a class library for the API

The aim of this library is to provide data access. It is separate from the WPF application and so it knows nothing about the database, nor have access to it. Putting code not directly into the API enables re-usability.



#### 13.1.1 Installing Dapper

Dapper is an object-relational mapping (ORM) product for the Microsoft .NET platform: it provides a framework for mapping an object-oriented domain model to a traditional relational database. Its purpose is to relieve the developer from a significant portion of relational data persistence-related programming tasks.

Add NuGet-Package to references...

## 13.1.2 The SqlDataAccess class

```

namespace RMDDataManager.Library.Internal.DataAccess
{
    /**
     * "Internal" makes sure that the database can't be accessed directly from outside this class library.
     *
     * The aim of this class is to provide the necessary data for the methods in UserData.cs, which talks
     * to the database in order to get back the information about users.
     */
    2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    internal class SqlDataAccess
    {
        // Gets the connection string with the matching name from WebConfig
        // (DefaultConnection) and returns it back.
        2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string GetConnectionString(string name)
        {
            return ConfigurationManager.ConnectionStrings[name].ConnectionString;
        }

        // Loading data from the database using GetConnectionString().
        1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
        public List<T> LoadData<T, U>(string storedProcedure, U parameters, string connectionStringName)
        {
            var connectionString = GetConnectionString(connectionStringName);

            using (IDbConnection connection = new SqlConnection(connectionString))
            {
                // Connects the database, makes a query and returns back a set of rows.
                var rows = connection
                    .Query<T>(storedProcedure, parameters, commandType: CommandType.StoredProcedure).ToList();

                return rows;
            }
        }

        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public void SaveData<T>(string storedProcedure, T parameters, string connectionStringName)
        {
            var connectionString = GetConnectionString(connectionStringName);

            using (IDbConnection connection = new SqlConnection(connectionString))
            {
                // Stores data into the database
                connection.Execute(storedProcedure, parameters, commandType: CommandType.StoredProcedure);
            }
        }
    }
}

```

### 13.1.3 The UserData class

This class contains methods and the logic on **how** to get data.

1. Adding a stored Procedure to `RMData.dbo.StoredProcedures`

```
CREATE PROCEDURE [dbo].[spUserLookup]
    @Id NVARCHAR(128)
AS
BEGIN
    SET NOCOUNT ON;

    SELECT Id, FirstName, LastName, EmailAddress, CreatedDate
    FROM [dbo].[User]
    WHERE Id = @Id;
end
```

2. Adding a UserModel class

```
using System.Threading.Tasks;

namespace RMDataManager.Library.Models
{
    3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class UserModel
    {
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string Id { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string FirstName { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string LastName { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string EmailAddress { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public DateTime CreatedDate { get; set; }
    }
}
```

## 3. Adding the UserData class

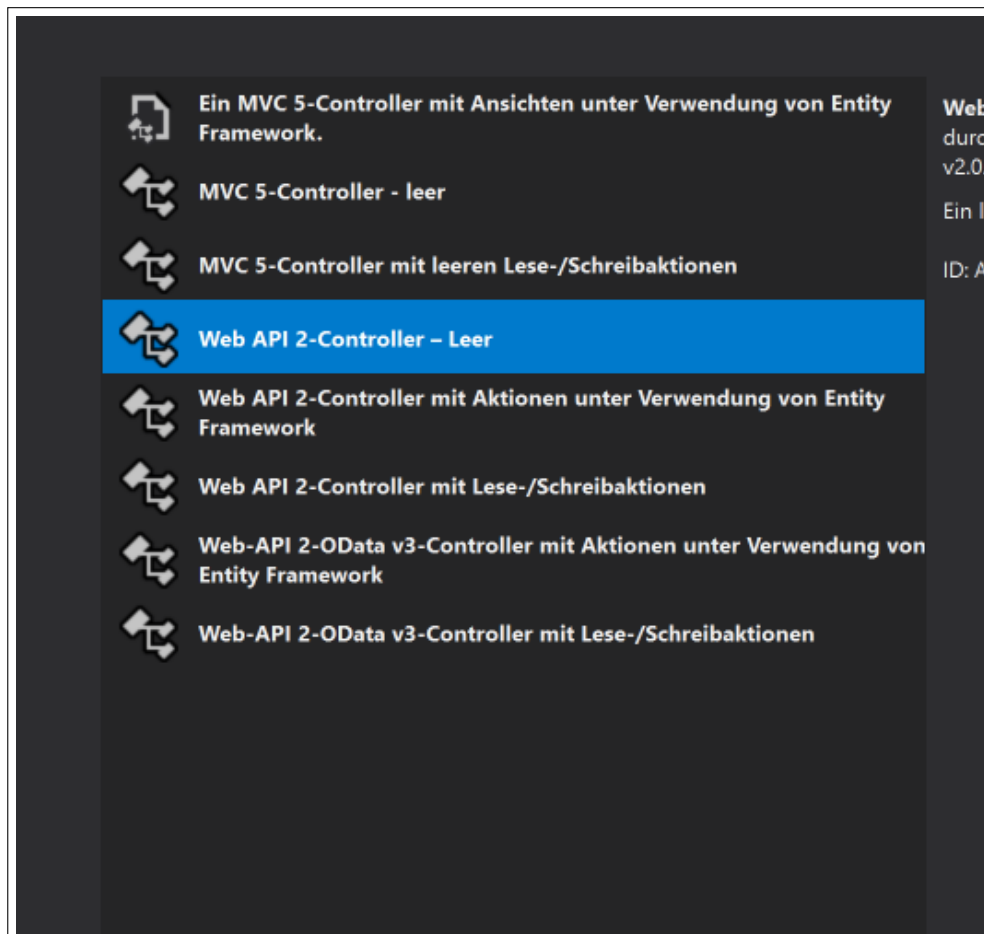
```
namespace RMDDataManager.Library.DataAccess
{
    2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class UserData
    {
        1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
        public List<UserModel> GetUserById(string Id)
        {
            SqlDataAccess sql = new SqlDataAccess();

            var p = new { Id };

            var output = sql.LoadData<UserModel, dynamic>(storedProcedure: "dbo.spUserLookup",
                p, connectionStringName: "DefaultConnection");

            return output;
        }
    }
}
```

## 13.1.4 Adding a UserController.cs to RMDDataManager.Controllers



Implementing the UserController:

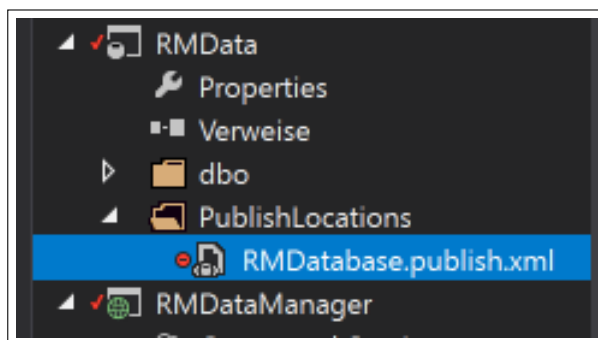
```

namespace RMDDataManager.Controllers
{
    [Authorize]
    [RoutePrefix("api/User")]
    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class UserController : ApiController
    {
        [HttpGet]
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public UserModel GetById()
        {
            var userId = RequestContext.Principal.Identity.GetUserId();
            var data = new UserData();

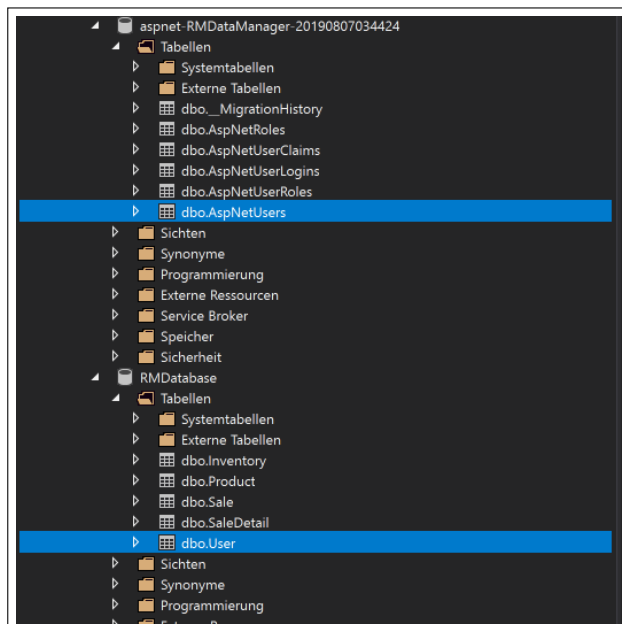
            return data.GetUserById(userId).First();
        }
    }
}

```

Publishing the stored procedure to the database:

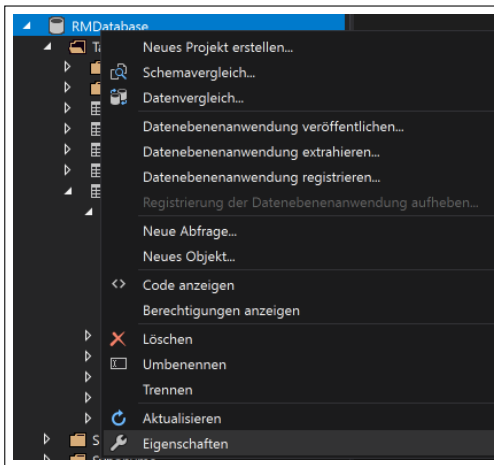


Adding a user for testing:



dbo.AspNetUsers [Daten]				
dbo.User [Daten]				
ValuesController.cs				
Account				
Max. Zeilen: 1000				
Id	FirstName	LastName	EmailAddress	Creat...
8b9e32c2ea75	Michael	Kaip	michael.kaip@gmx.net	NULL
NULL	NULL	NULL	NULL	NULL

Getting the connection string from the database and adding it to Web.config:



```
<configSections>
  <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
  <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework" />
</configSections>
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=[DataDirectory]
  <add name="RMDatabase" connectionString=
    "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=RMDatabase;Integrated Security=True;Connect
    Timeout=60;Encrypt=False;TrustServerCertificate=True;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"/>
</connectionStrings>
</appSettings></appSettings>
```

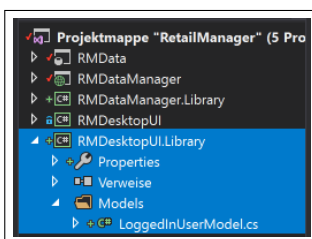
Now that there's an entry in the user database, the application can be started and use SWAGGER to run and test the GET command in `Values/GET`

Next step is to implement the functionality to get back the information about the user automatically, once he logged in successfully.

## 13.2 Implementing data capturing from the UI side

### 13.2.1 Adding a new class library

The aim of this library is for user inface support.



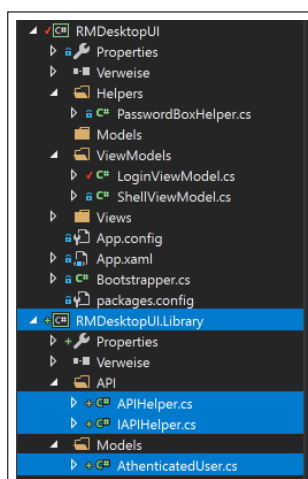
### 13.2.2 The LoggedInUserModel class

```

/*
 * The aim of this class is to catch the data, which the API sends out.
 */
namespace RMDesktopUI.Library.Models
{
    1-Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class LoggedInUserModel
    {
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string Token { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string Id { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string FirstName { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string LastName { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public string EmailAddress { get; set; }
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public DateTime CreatedDate { get; set; }
    }
}

```

### 13.2.3 Moving helpers and Models



Namespaces has to be adjusted and missing references added afterwards.

### 13.2.4 Extracting an Interface and adding to simpleContainer

```

namespace RMDesktopUI.Library.Models
{
    4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public interface ILoggedInUserModel
    {
        2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        string Token { get; set; }
        3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        string Id { get; set; }
        3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        string FirstName { get; set; }
        3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        string LastName { get; set; }
        3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        string EmailAddress { get; set; }
        3 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        DateTime CreatedDate { get; set; }
    }
}

```

```

.Singleton<ILoggedInUserModel, LoggedInUserModel>()

```



## 13.2.5 Implementing a Method GetLoggedInUserInfo in APIHelper.cs

```

public async Task<LoggedInUserModel> GetLoggedInUserInfo(string token)
{
    ApiClient.DefaultRequestHeaders.Clear();
    ApiClient.DefaultRequestHeaders.Accept.Clear();
    ApiClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue(mediaType: "application/json"));
    // Adding the "Authorization Bearer Token" to the DefaultRequestHeader for every call.
    ApiClient.DefaultRequestHeaders.Add(name: "Authorization", value: $"Bearer { token }");

    using (HttpResponseMessage response = await ApiClient.GetAsync(requestUri: "/user"))
    {
        if (response.IsSuccessStatusCode)
        {
            var result = await response.Content.ReadAsAsync<LoggedInUserModel>();

            // data mapping
            _loggedInUser.CreatedDate = result.CreatedDate;
            _loggedInUser.EmailAddress = result.EmailAddress;
            _loggedInUser.FirstName = result.FirstName;
            _loggedInUser.Id = result.Id;
            _loggedInUser.LastName = result.LastName;
            _loggedInUser.Token = token;

            return result;
        }
        else
        {
            throw new Exception(response.ReasonPhrase);
        }
    }
}

```

In order to make it work, some more changes has to be done in the same class:

```

2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public class APIHelper : IAPIHelper
{
    9 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    private HttpClient ApiClient { get; set; }
    private readonly ILoggedInUserModel _loggedInUser;

    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public APIHelper(ILoggedInUserModel loggedInUser)
    {
        InitializeClient();
        _loggedInUser = loggedInUser;
    }
}

```

Finally the LogIn() method in the LoginViewModel has to be extended:

```

public async Task LogIn(string userName, string password)
{
    try
    {
        ErrorMessage = "";
        var result = await _apiHelper.Authenticate(username: UserName, Password);

        // Capture more information about the user
        await _apiHelper.GetLoggedInUserInfo(result.Access_Token);
    }
    catch (Exception ex)
    {
        ErrorMessage = ex.Message;
    }
}

```

- 14 Sales Page Creation
- 15 Event Aggregation in WPF
- 16 Displaying Product data
- 17 Wiring up WPF Shopping Cart
- 18 Modifying SQL, the API and WPF to add Taxes