# Retail Manager Development Manual

Michael Kaip

August 9, 2019

# Contents

# 1 Introduction

## 1.1 Project Summary

The goal of tkhis project is to build a dektop 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 lauyer 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:

- Git on Azure DevOps
- SQL Database (SSDT)
- WebAPI with Authentification
- WPF application that can log into the API

## 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 sulution isn't language specific.

# 4 Creating a WebAPI with Authentification

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:

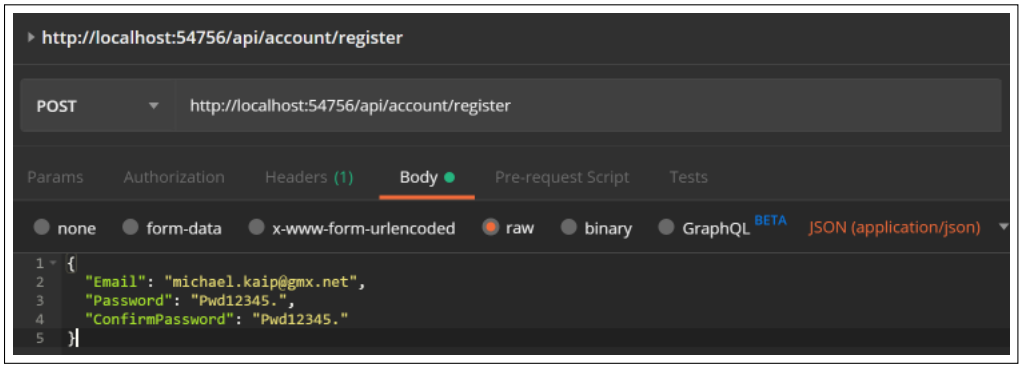
- UserValidator
- PasswordValidator

## 4.2 Getting authorized for development

### 4.2.1 Postman

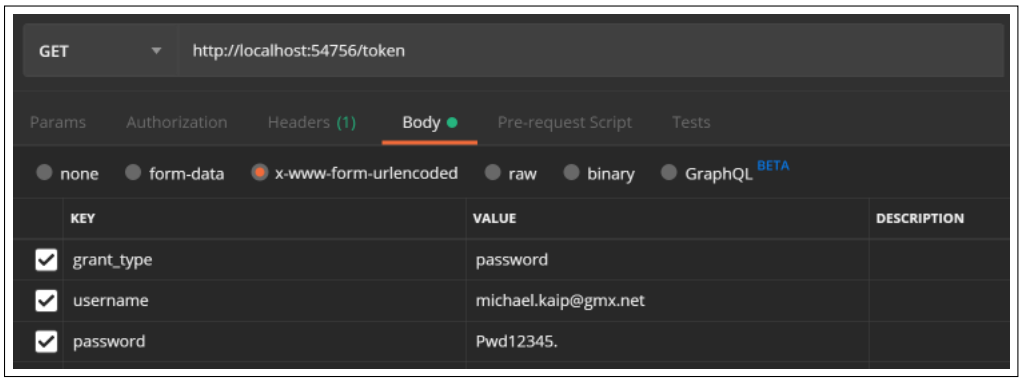
The following calls has to be applied in the given order:

1. **POST**



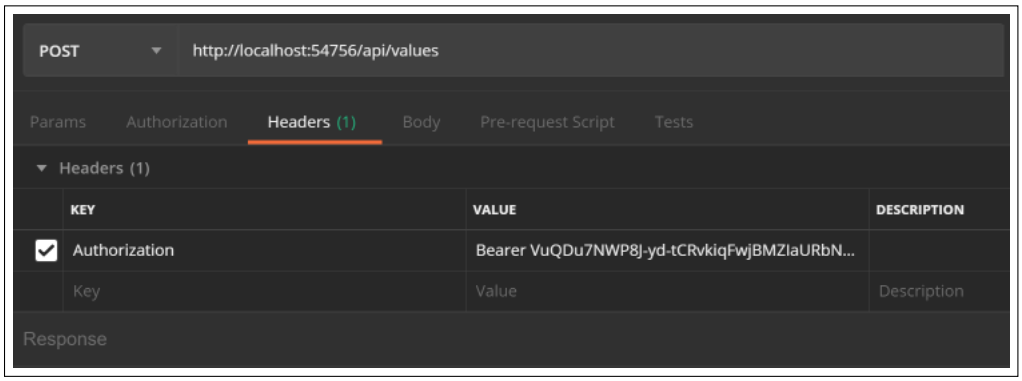

   Creates a new user account and stores this information into the user database.
   If **Status: 200 OK**, username and password has been succesfully created.

2. **GET**




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**




### 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 accesibility 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

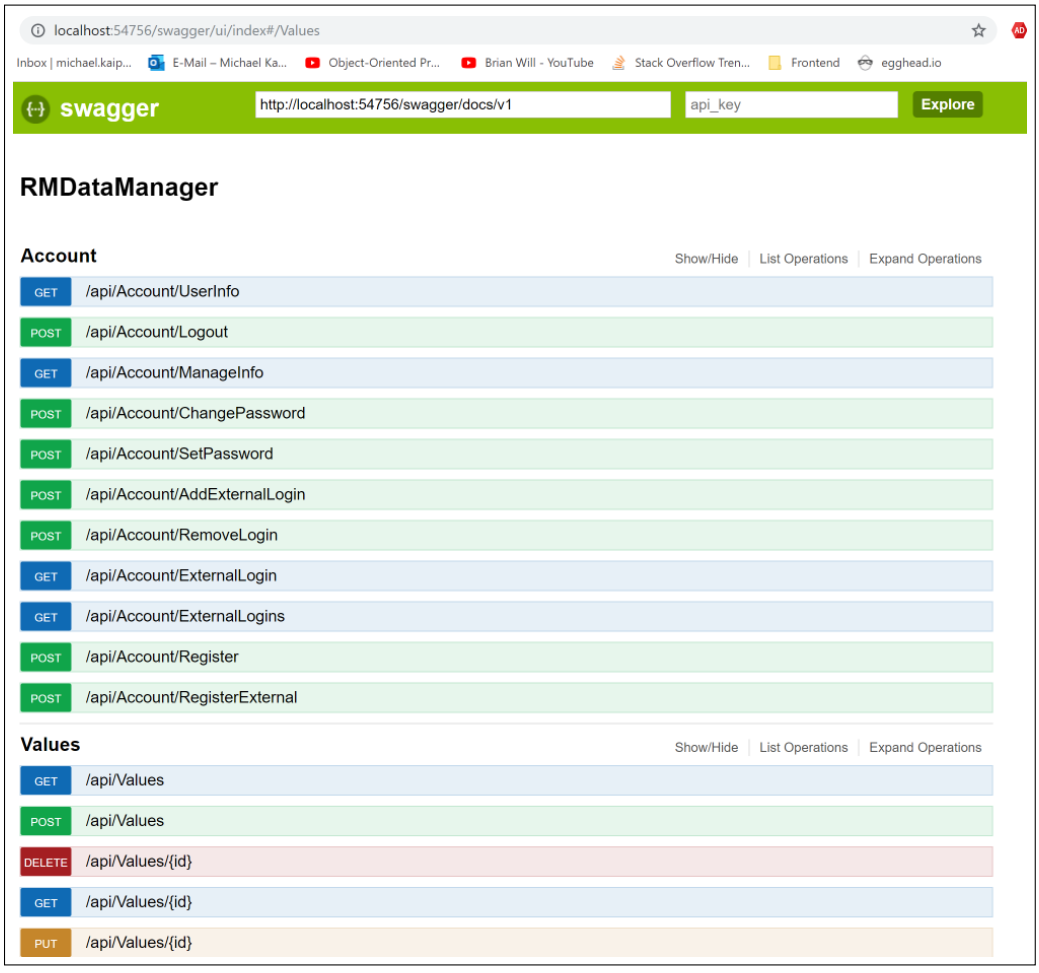
SWAGGER is an API documentation and demonstration tool.

## 5.1 Installing SWAGGER

1. NuGet-Manager


   Adds a SWAGGER to WebAPI-Projects.

2. Starting SWAGGER




## 5.2 Channging the configuration of SWAGGER

*RMDataManager.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 propper name
```

2. Enabling propper printing of ducuments

```
// 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
```

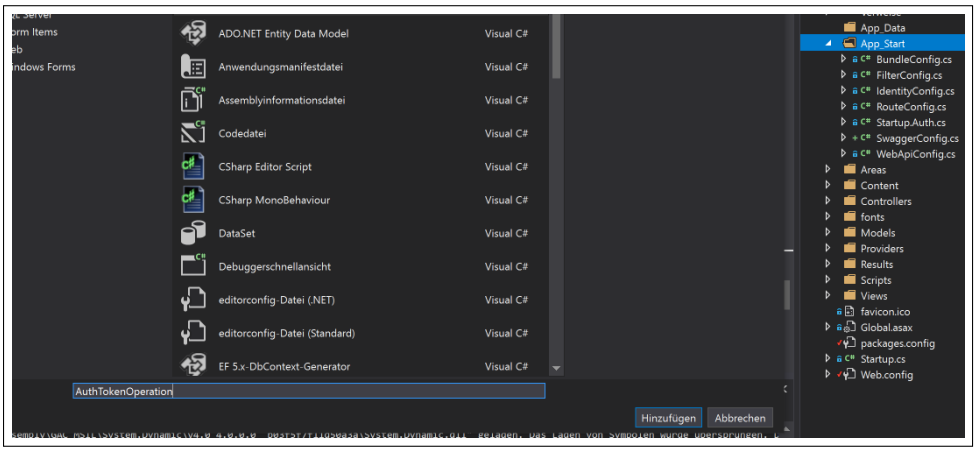
4. Changing document title

```
.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 $RMDataManager.App\_Start$



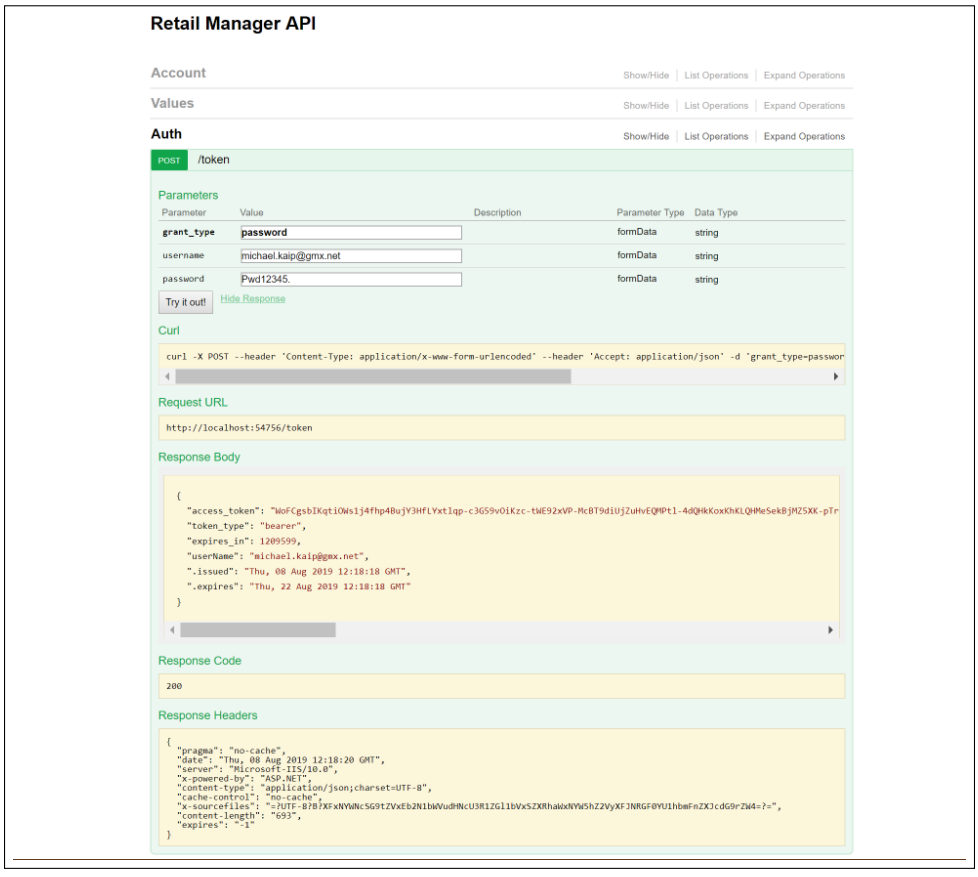
(b) Implementing the required Interface

```
public class AuthTokenOperation : IDocumentFilter
{
    0 Verweise | 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

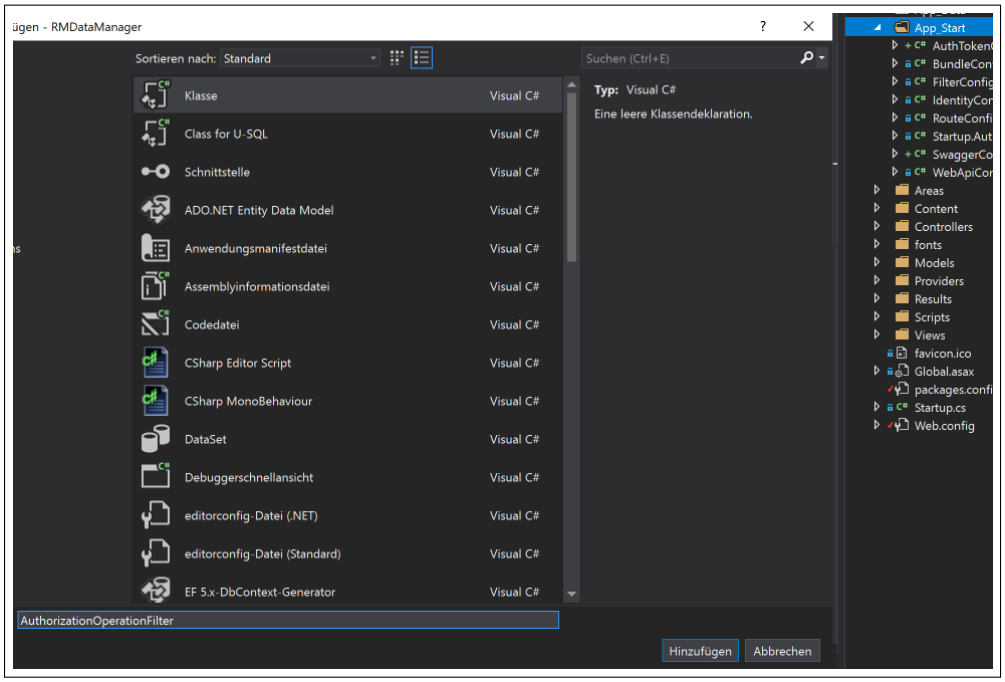
```
GlobalConfiguration.Configuration
    .EnableSwagger(configure: c =>
        {
            c.DocumentFilter<AuthTokenOperation>(); // adding the implemented document filter
```

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



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

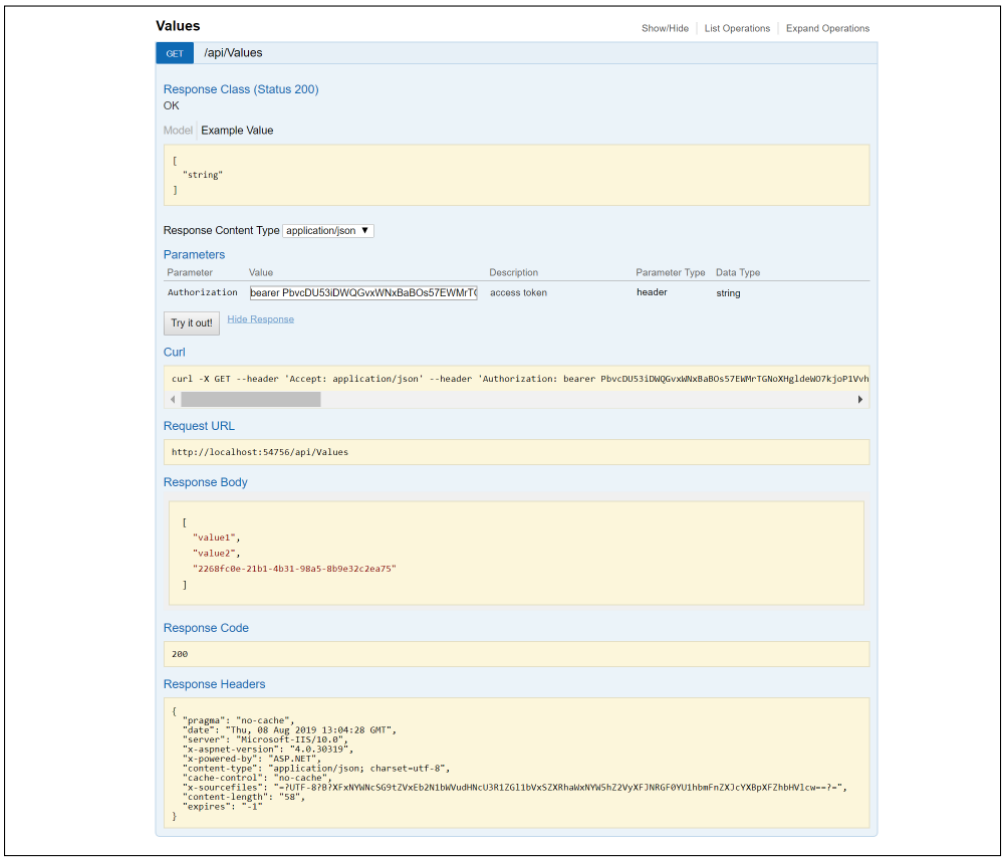
```
0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
public class AuthorizationOperationFilter : IOperationFilter
{
    0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
    public void Apply(Operation operation, SchemaRegistry schemaRegistry, ApiDescription apiDescription)
    {
        // Adding 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

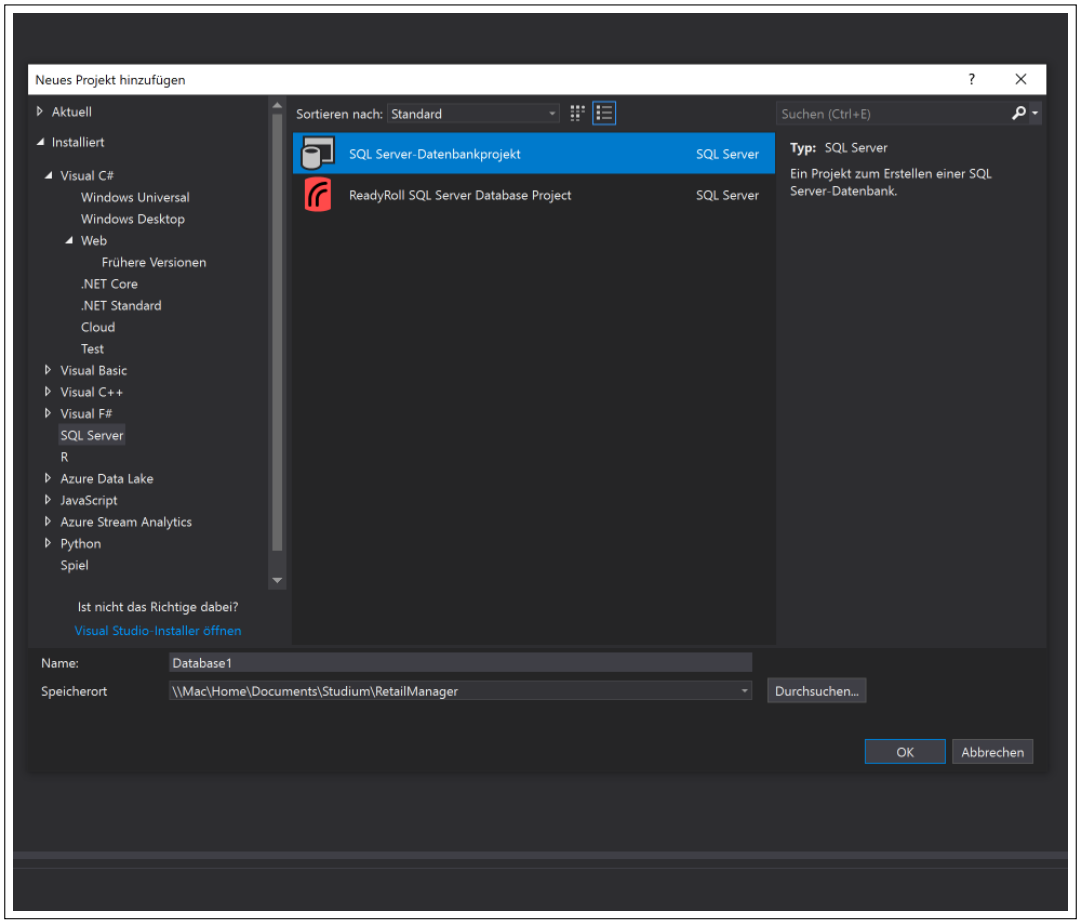
```
.EnableSwagger( configure: c =>
    {
        c.DocumentFilter<AuthTokenOperation>(); // adding the implemented document filter
        c.OperationFilter<AuthorizationOperationFilter>(); // adding the implented operation filter
```

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

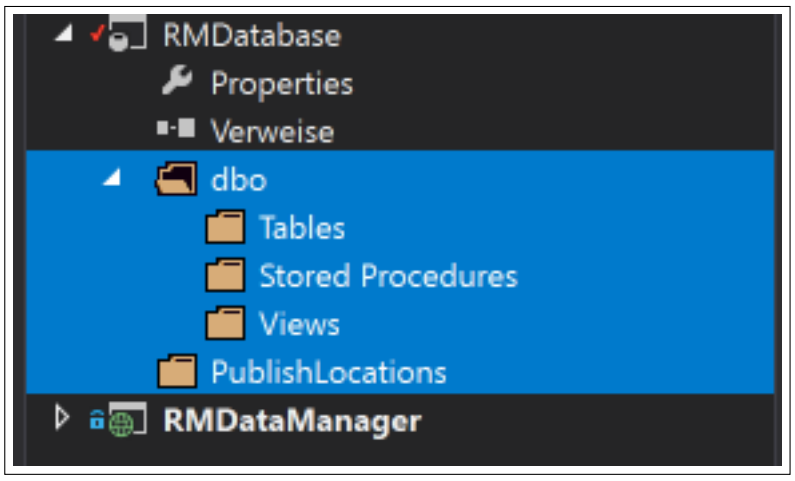

Values
Show/Hide | List Operations | Expand Operations
GET /api/Values
Response Class (Status 200)
OK
Model Example Value
[
"string"
]
Response Content Type application/json
Parameters
Parameter Value Description Parameter Type Data Type
Authorization bearer PbvcDU53iDWQGvxWNxBaBOs57EWMrT( access token header string
Try it out! Hide Response
Curl
curl -X GET --header 'Accept: application/json' --header 'Authorization: bearer PbvcDU53iDWQGvxWNxBaBOs57EWMrTGNoXHgldeWO7kjoP1Vvh
Request URL
http://localhost:54756/api/Values
Response Body
[
"value1",
"value2",
"2268fc0e-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": "=?UTF-8?B?XFxNYWNcSG9tZVxEb2N1bWVudHNcU3R1ZGl1bVxSZXRhaWxNYW5hZ2VyXFJNRGF0YU1hbmFnZXJcYXBpXFZhbHV1cw==?=",
"content-length": "58",
"expires": "-1"
}

# 6 SQL Database Setup

## 6.1 Adding new Database Project to the solution

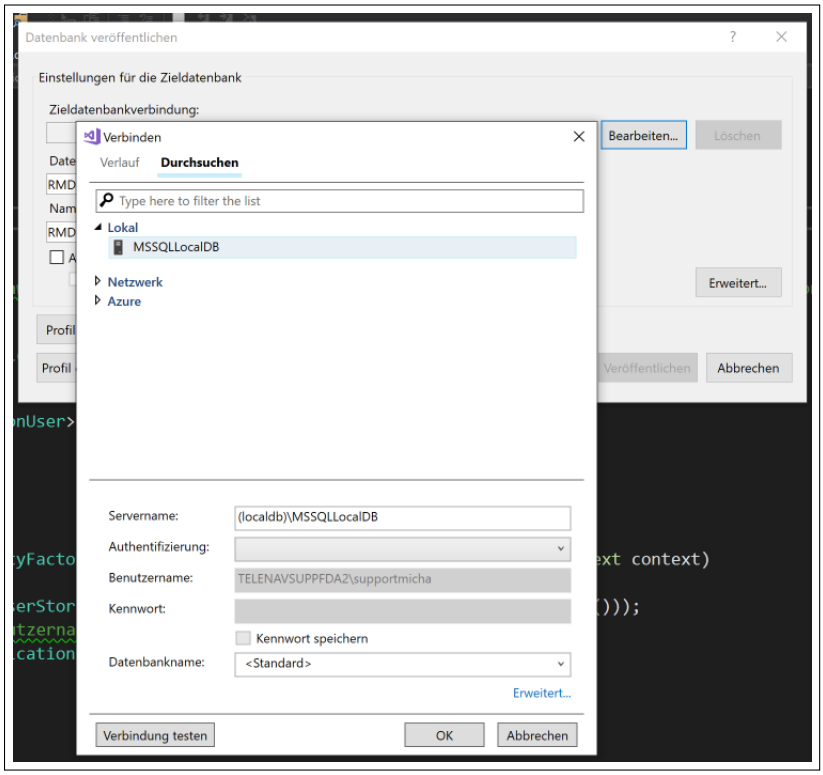



## 6.2 Adding several folders to the project

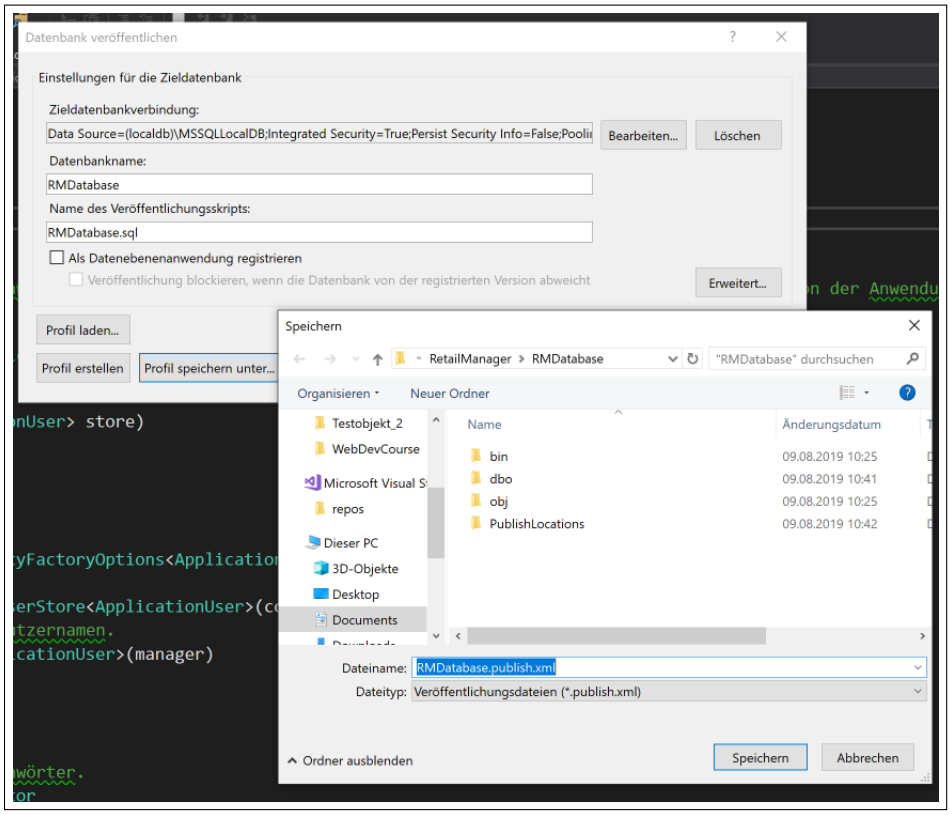


## 6.3 Creating a profile and publishing the Database

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

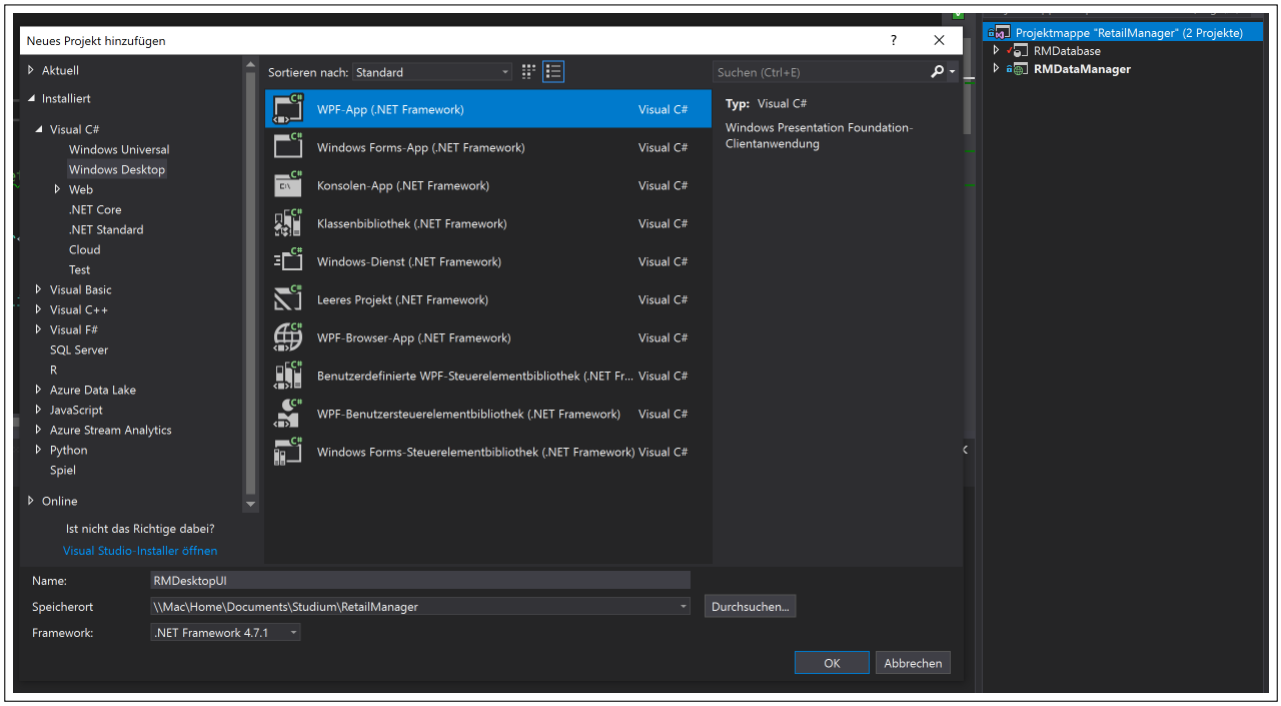



2. Naming and saving profile to PublishLocations

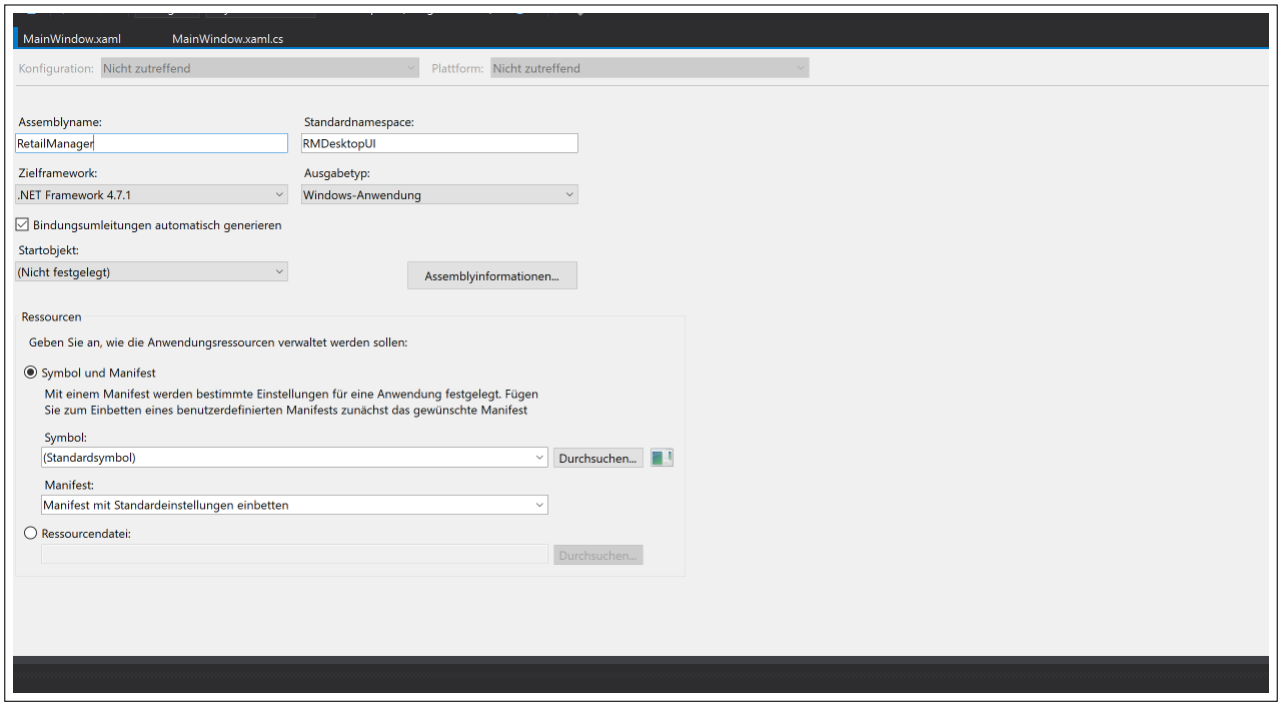


# 7 WPF with MVVM Project Setup

## 7.1 Adding the WPF Project to the solution




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

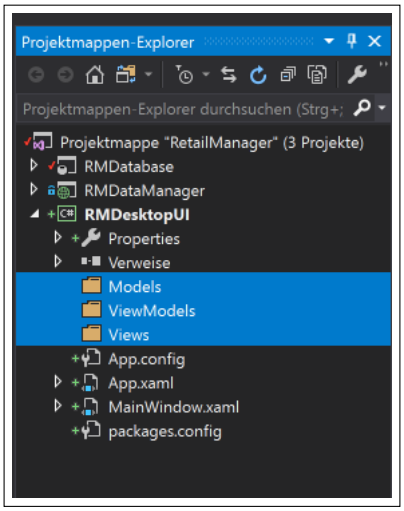



Also set project as the default startup-project.

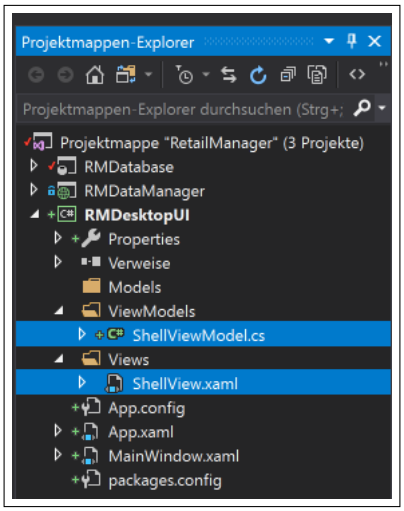
## 7.3 Adding Caliburn Micro MVVM-Framework

Add NuGet-Package to references.

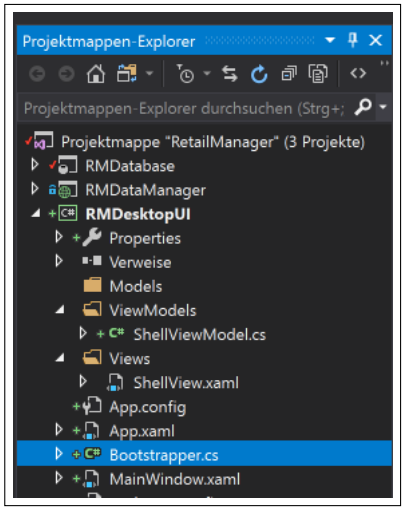
### 7.3.1 Adding the folder structure for the MVVM-Framework




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




### 7.3.3 Adding a Bootstrapper class to DesktopUI

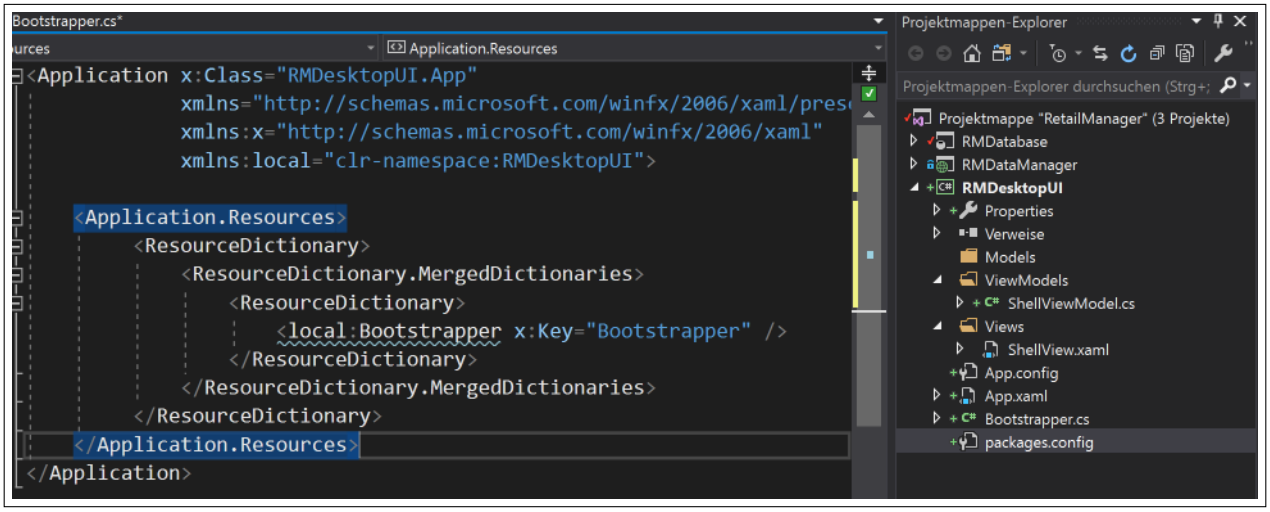



```
using System.Windows;
using Caliburn.Micro;
using RMDesktopUI.ViewModels;

namespace RMDesktopUI
{
    1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen
    public class Bootstrapper : BootstrapperBase
    {
        // Constructor
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        public Bootstrapper()
        {
            Initialize();
        }

        // Enables, that on start up ShellViewModel is getting launched as the base view.
        0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen
        protected override void OnStartup(object sender, StartupEventArgs e)
        {
            DisplayRootViewFor<ShellViewModel>();
        }
    }
}
```

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



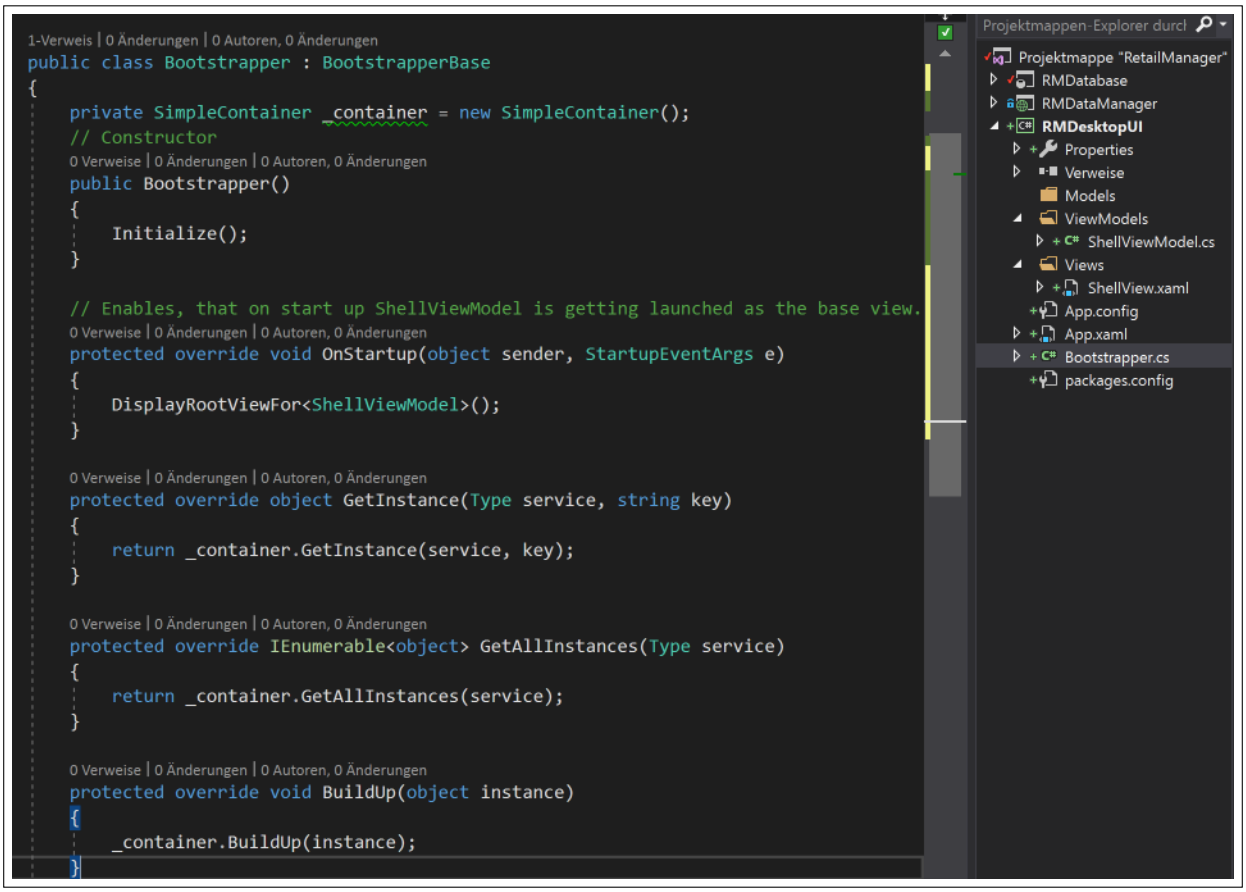

**MainWindow.xaml can be deleted afterwards!!!**

# 8 Dependency Injection in WPF

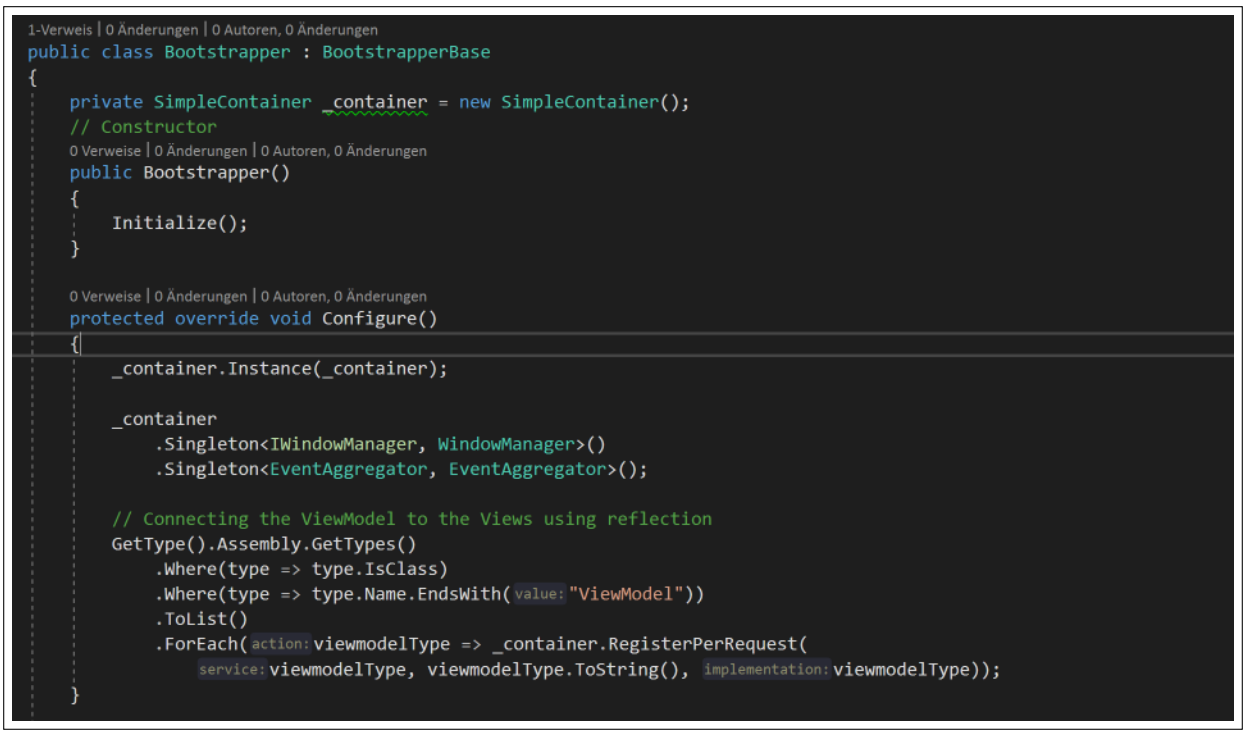
## 8.1 SimpleContainer in Caliburn Micro

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));
    }
```