

## 3. Blockwoche 04.03.2024

In unserer dritten Blockwoche war die Aufgabe in C# eine MVC API zu erstellen mit Datenbank anbindung.

# Table of contents

Introduction .....	2
Aufgaben.....	3
Aufgabe 1 .....	4
Aufgabe 2 .....	11

# Introduction

## 3. Blockwoche 04.03.2024 - 07.03.2024

Programmieren einer MVC API in C# und .NET

# Aufgaben

## Aufgabe 1:

Am 04.03.2024 um 08:30 Uhr habe ich mit der ersten Aufgabe gestartet und bin dabei mr das erste Video anzuschauen. In dem ersten Video geht es um LINQ, was das ist und wie man damit in C# Arbeitet. *Video C# LINQ Tutorial Deutsch / German #1 - Was ist LINQ? - Programmieren Starten* ([https://www.youtube.com/watch?v=CVgzjkiXGg&list=PL\\_pqkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu\\_D](https://www.youtube.com/watch?v=CVgzjkiXGg&list=PL_pqkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu_D))

[https://www.youtube.com/watch?v=CVgzjkiXGg&list=PL\\_pqkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu\\_D](https://www.youtube.com/watch?v=CVgzjkiXGg&list=PL_pqkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu_D)

[Aufgabe 1](#)

## Aufgabe 2

Am 04.03.2024 um 09:15 Uhr bin ich mit Aufgabe 1 fertig geworden und beginnen nun mit Aufgabe 2. In Aufgabe 2 geht es darum das wir ein Video Schauen müssen wie man mit ASP.NET Core eine einfache Web API Programming kann. *Video: ASP.NET Core 5.0 Web API with C# [2021] made easy using a Project - Step by Step* (<https://www.youtube.com/watch?v=nG3yRTPY5wU>)

<https://www.youtube.com/watch?v=nG3yRTPY5wU>

[Aufgabe 2](#)

# Aufgabe 1

## Was ist LINQ?

- LINQ steht für "Language Integrated Query".
- LINQ ist in C# und Visual Basic integriert und ist eine Abfragesprache mit der man Daten aus unterschiedlichsten Datenquellen einfach abfragen kann.
- Mit LINQ kann man auf alle verschiedenen Datenquellen gleich zugreifen so lange die Datenquelle eine LING Integration hat.
- LINQ erleichtert das Zugreifen auf Datenstrukturen in .NET

## Einfache Praxis anwendung von LINQ | Video 1

Um LINQ nutzen zu können muss man mithilfe von `using System.Linq;`, LINQ erstmal in das Projekt einbinden um es nutzen zu können.

### LINQ Beispiel zum Abfragen nach einem bestimmten Zeichen

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace LINQTutorial {
    class Program {
        static void Main(string[] args) {
            string[] names = {"Peter", "Günter", "Manfred", "Uwe"};
            var linqTest = from name in names
                           where name.Contains('t')
                           select name;
            foreach(string name in linqTest) {
                Console.WriteLine(name);
            }
            Console.ReadKey();
        }
    }
}
```

```
}  
}
```

**⚠ Wichtig!** Die linqTest Abfrage wird erst dann ausgeführt, wenn man diese auch wo benutzt.

In diesem Beispiel wird ein Array mit dem namen `names` erstellen mit den werten `Peter`, `Günter`, `Manfred` und `Uwe`. Dann wird eine Variable namens `linqTest` erstellt und danach mit einer SQL ähnlichen Syntax mit `name.Contains('t')` nach Arrays gesucht die den Buchstaben `t` beinhalten und ausgewählt. Im anschluss wird dann eine `foreach` loop erstellt, wo dann die jeweiligen Arrays ausgegeben werden.

## Query- und Methoden Syntax von LINQ

In dem Video C# LINQ Tutorial Deutsch / German #2 - Die Query-Syntax und die Method-Syntax ([https://www.youtube.com/watch?v=htggFXYfvBw&list=PL\\_pqkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu\\_D&index=2](https://www.youtube.com/watch?v=htggFXYfvBw&list=PL_pqkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu_D&index=2)) wird erklärt wie die Query und der Methoden Syntax von LINQ funktioniert.

### Die Query Syntax

Die Query Syntax ist eines der beiden möglichen Kodierungs arten mit denen man mit LINQ Abfragen erstellen kann.

Wie schon oben erklärt kann man mit

```
var linqTest = from name in names  
               where name.Contains('t')  
               select name;
```

eine LINQ Abfrage machen. Diese LINQ Abfrage benutzt die Query Syntax von LINQ und ähnelt der SQL Sprache mehr. Und deswegen ist die Query Syntax von LINQ für die sehr zu Empfehlen die viel mit Datenbanken Arbeiten da es bei dieser Syntax wenige unterschiede, gibt.

Den Code kann man wie folgt interpretieren

Syntax	interpretation
from name	from ist soweit ich es verstanden habe einfach nur eine Variable die man dann benutzt zum Ausgeben oder nutzen von weiteren Anfragen
in names	in und die variable danach ist dann die Variable die man von linq verarbeitet haben will
where name.Contains('t')	where name.Contains('t') wird dann genutzt um die names durchzufiltern und zu schauen welches der Werte t beinhaltet
select name	select name ist dann da das wir das endresultat von der name variable auswählen

## Die Methode Syntax

Die Methode Syntax ist für die Personen nützlich die nicht viel mit SQL zu tun haben oder sich das nicht geben wollen und gerne mit C# Methoden Arbeiten wollen.

Die Obere Query Syntax sieht dann als Methoden Syntax wie folgt aus:

```
var linqTest = names.Where(name => name.Contains('t'));
```

Wenn man bei LINQ mit der Methoden Syntax arbeitet, muss man für Where dann Lambda anwenden. Was ist Lambda? Lambda ist in diesem fall das => in der name => name.Contains('t') abfrage. Das sagt nichts weiter aus als es am Anfang eine Variable erstellt namens name und danach name.Contains('t') ausgeführt wird mit den Daten von name. Einfach gesagt es überprüft, ob die Variable name ein t beinhaltet und wenn das der Fall ist gibt der true zurück und wenn nicht, dann false.

```
static void Main(string[] args)
{
    string[] names = { "Peter", "Günter", "Manfred", "Uwe" };

    // Query Syntax
    var linqTest = from name in names
                   where name.Contains('t')
                   select name;

    // Methode Syntax
    var linqTestMethod = names.Where(name => name.Contains('t'));

    foreach (string name in linqTest)
    {
        Console.WriteLine(name);
    }
    Console.ReadKey();
}
```

(Erweiterung) IEnumerable<string> IEnumerable<string>.Where<string>(Func<string, bool> predicate) (+ 1 Überladung)  
Filters a sequence of values based on a predicate.  
Rückgabewerte:  
An IEnumerable<out T> that contains elements from the input sequence that satisfy the condition.  
Ausnahmen:  
ArgumentNullException

Visual Studio Code Editor - Lambda LINQ Method Syntax implementation

## LINQ Filtern mit Where und OfType

In dem Video C# LINQ Tutorial Deutsch / German #3 - Filtern mit Where und OfType

([https://www.youtube.com/watch?](https://www.youtube.com/watch?v=IUgebK3MHKo&list=PL_pgkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu_D&index=3)

[v=IUgebK3MHKo&list=PL\\_pgkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu\\_D&index=3](https://www.youtube.com/watch?v=IUgebK3MHKo&list=PL_pgkvxZ6ho0FX6Ro3noCmwQ3ix5Cdu_D&index=3)) wird es wie im Titel beschrieben um das Filtern mit Where und OfType erklärt. Dafür hat der YouTuber folgende Vorlage angefertigt

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LINQTutorial
{
    class Video3
    {
        static void Main(string[] args)
        {
            List<Student> studentenListe = new List<Student>();
            studentenListe.Add(new Student(Studiengang.Biologie, 3,
            "Sabrina", "Vogel"));
            studentenListe.Add(new Student(Studiengang.Informatik, 5,
            "Patrick", "Müller"));
            studentenListe.Add(new Student(Studiengang.Informatik, 4,
```



```

    "Kurt", "Meier"));
        studentenListe.Add(new Student(Studiengang.Germanistik, 1,
    "Anna", "Kohl"));
        studentenListe.Add(new Student(Studiengang.Mathematik, 2,
    "Sebastian", "Müller"));

    }
}
enum Studiengang
{
    Informatik,
    Biologie,
    Germanistik,
    Mathematik
}
class Student
{
    public Studiengang Studiengang { get; set; }
    public int Semester { get; set; }
    public string Vorname { get; set; }
    public string Nachname { get; set; }

    public Student(Studiengang studiengang, int semester, string
vorname, string nachname)
    {
        Studiengang = studiengang;
        Semester = semester;
        Vorname = vorname;
        Nachname = nachname;
    }

    public override string ToString()
    {
        return Vorname + " " + Nachname + " " + Studiengang + " " +
Semester + " ";
    }
}

```

```
}
```

## Die Where Methode

Die Where Methode ist dafür da um Ergebnisse nach deren Booleschen Wert Filtern. Das heißt man kann eine abfragen machen mit Elementen und wenn diese Elemente sich gleichen dann ist der Wert auch `True` und wenn nicht, dann `False`. (Für mehr Informationen bezüglich der Booleschen Tabelle und wie die Funktioniert lese hierzu die Akami Solutions Dokumentation)

Jetzt wollen wir von der Liste nur die Schüler ausgeben die im Studiengang `Informatik` sind. hier müssen wir folgenden Code Schreiben:

```
[...]  
// Query Syntax einzelne where Abfrage  
var informatikStudenten = from Student  
                           in studentenListe  
                           where Student.Studiengang ==  
Studentengang.Informatik // Wenn der Student im Studiengang Informatik ist  
                           select Student; // Dann soll es ausgewählt werden  
[...]
```

In diesem Code Beispiel wird eine Variable Namens `informatikStudenten` erstellt. Um die Informatik Studenten zu bekommen, definieren wir mit `from` eine variable namens `Student` und sagen mithilfe von `in` das es die Daten aus der `studentenListe` Array nehmen soll. Und dann sagen wir mithilfe von `where` das der nur diese Arrays nimmt, bei denen der Studiengang Informatik ist.

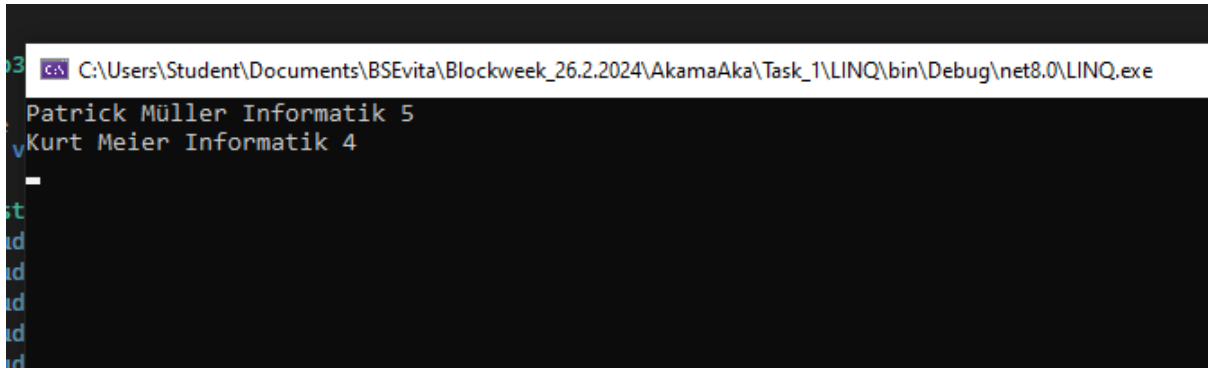
Aufgrund dessen das diese Variable nicht direkt ausgeführt wird müssen wir noch folgenden Code unten dran geben:

```
[...]  
foreach (Student student in informatikStudenten) {  
    Console.WriteLine(student.ToString());  
}  
[...]
```

in diesem Code Beispiel gehen wir mit `foreach` durch alle Arrays aus der `informatikStudenten` Variable durch und die einzelnen Arrays bekommen den Namen `student` und dann wird mit

`Console.WriteLine(student.ToString())` die Student daten ausgegeben.

Wenn das Programm ausgeführt wird, wird dann folgendes ausgegeben:



```
C:\Users\Student\Documents\BSEvita\Blockweek_26.2.2024\AkamaAka\Task_1\LINQ\bin\Debug\net8.0\LINQ.exe
Patrick Müller Informatik 5
Kurt Meier Informatik 4
```

img\_23

Wenn man mehrere sachen Abfragen will wie z.B. nur die Schüler die über dem 4. Studiengang sind dann kann man das wie folgt machen:

```
[...]  
  
var informatikStudenten = from Student  
                           in studentenListe  
                           where (Student.Studiengang ==  
Student.Studiengang.Informatik) && (Student.Semester > 4) // Wenn der Student im  
Studiengang Informatik ist  
                           select Student; // Dann soll es ausgewählt  
werden*/  
[...]
```

Wie man sieht, muss man die einzelnen `where` abfragen dann in einer `()` geben und mit `&&` trennen. Sollte man ein oder gebrauchen dann muss man anstatt `&&`, `||` machen.

# Aufgabe 2

In Aufgabe zwei geht es schon um das Programmieren und Installieren von Visual Studio. Allerdings gilt es hier auch darauf zu achten das man auch **Visual Studio** und nicht Visual Studio Code sich Installiert, weil beides etwas anderes ist. Visual Studio ist eine IDE. Eine IDE beinhaltet sehr viele Features, Funktionen und Vorlagen, die von dem Entwickler in dem fall Microsoft direkt beim Download auch mitenthalten ist und ist meistens für eine bestimmte Sprache oder Sprach-Gruppe ausgelegt. Visual Studio Code hingegen ist ein Code Editor. Code Editoren kann man für alles mögliche Verwenden allerdings gibt es da auch sehr wenige Funktionen und Vorlagen da das meiste von der Community dann erstellt wird und man die Funktionen extra sich noch heraussuchen muss und Installieren muss.

## Installation von Visual Studio

Um Visual Studio zu installieren und einzurichten habe ich mir Visual Studio von der Webseite <https://visualstudio.microsoft.com/de/free-developer-offers/> heruntergeladen. Hierbei habe ich beachtet, dass ich das Rosane herunterlade da das auch die IDE ist die ich Brauche.

Sobald ich die Visual Studio.exe Datei ausgeführt habe, hat sich ein Fenster geöffnet, wo ich Auswählen muss was ich benötige. Hier habe ich dann ASP.NET und Webentwicklung, .NET-Desktopentwicklung ausgewählt und installiert.

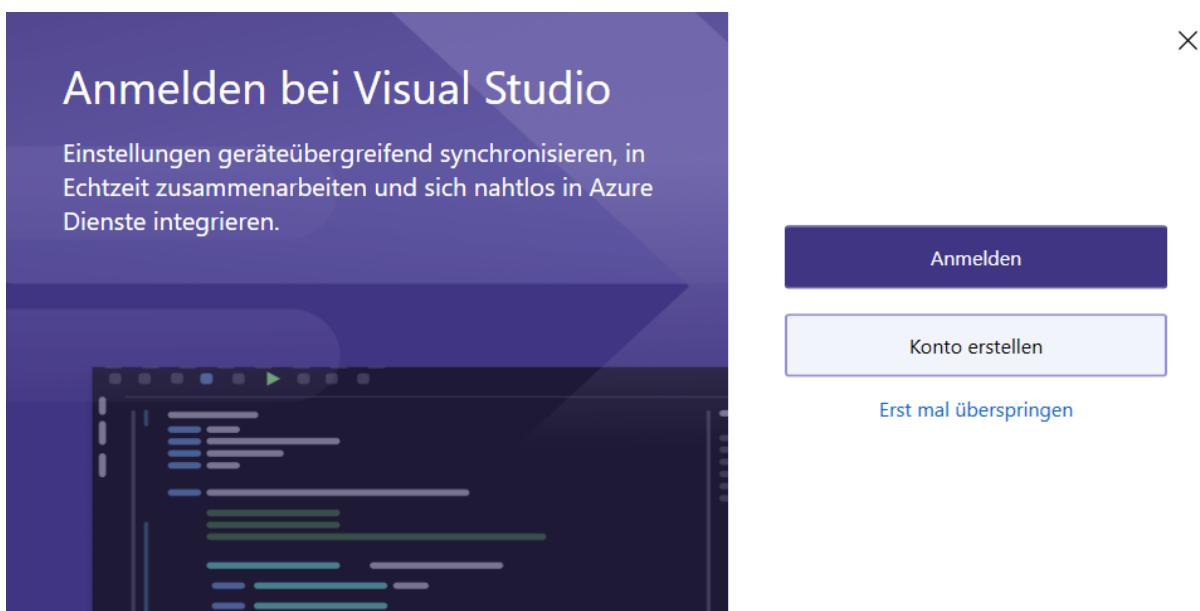
Nach ein paar Minuten war die Installation auch schon fertig und dann sollte das endresultat wie folgt aussehen:



Visual Studio Installer nach der erfolgreichen Installation

Sollte man Änderungen vornehmen wollen an der Installation oder Funktionen hinzufügen, kann man diese dann hinzufügen, indem man im Installer dann auf **Ändern** klickt und die jeweiligen Kästchen auswählt.

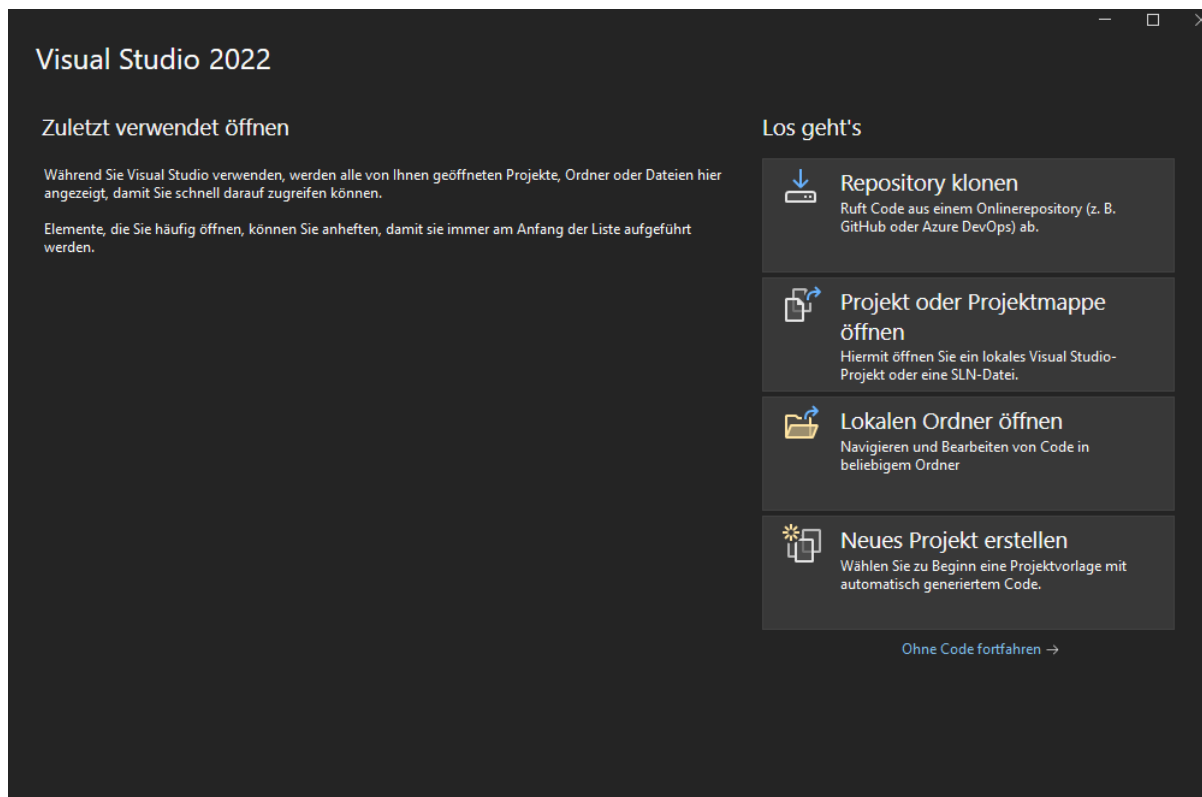
Sollte man nun auf **Starten** klicken, lädt die Applikation kurz und dann sollte man folgendes Fenster sehen:



Anmelden bei Visual Studio Fenster

Da es fürs Erste nicht nötig ist, werde ich mich in dem fall nicht Anmelden und somit erstmal auf **erst mal überspringen** klicken.

Nach dem man die IDE für sich Personalisiert hat findet man sich in der Projektübersicht wieder.

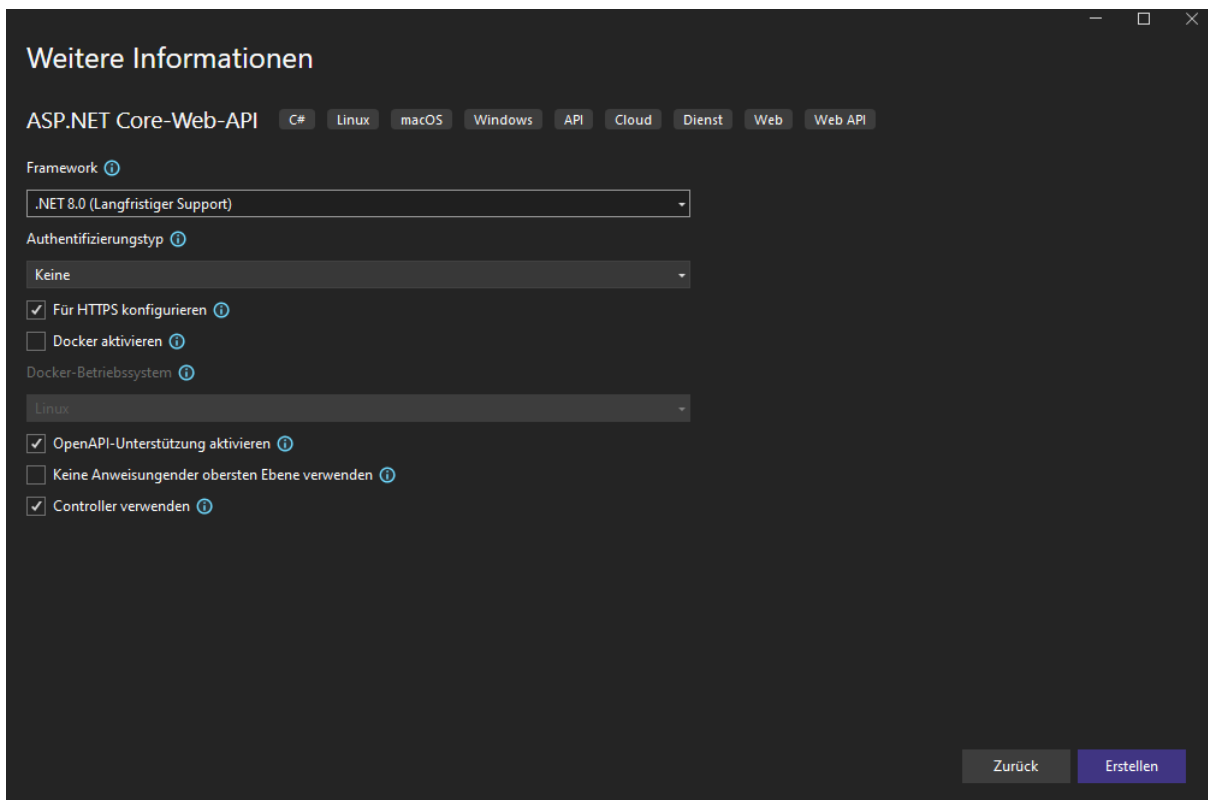


Visual Studio Projektübersicht

## Erstellen eines neuen Projektes

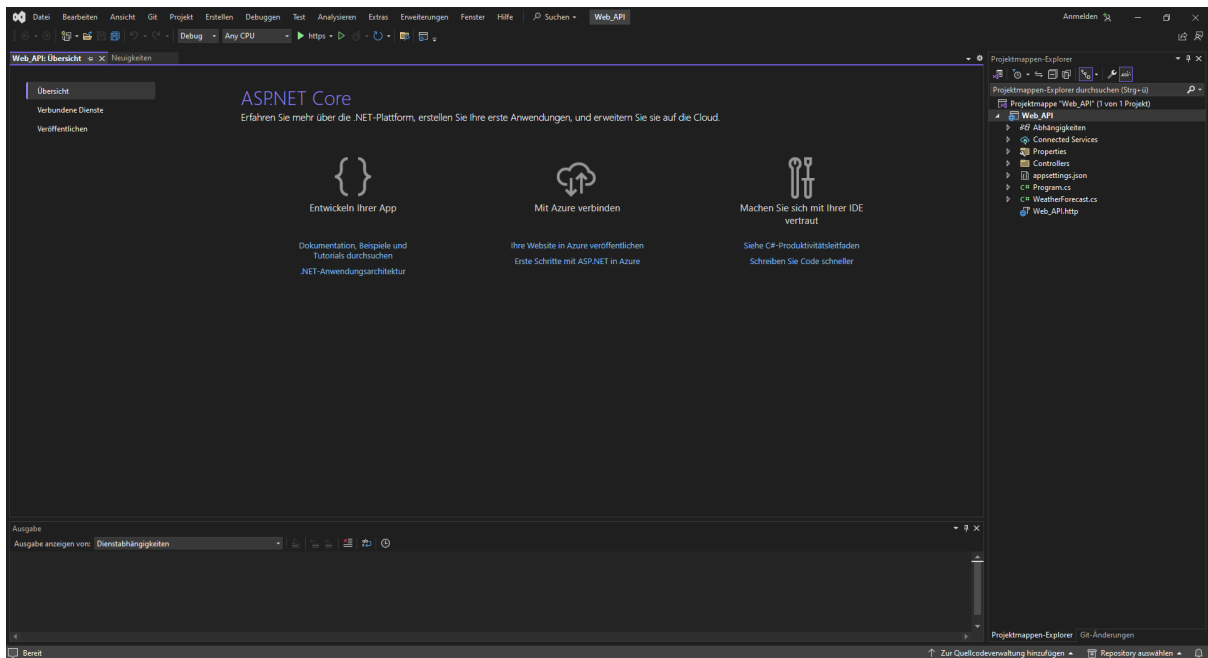
In der Projektübersicht angekommen müssen wir ein neues ASP.NET Core Web-App erstellen. Dafür klickt man als Erstes auf **Neues Projekt erstellen**. Hier wählt man dann bei **Alle Sprachen**, **C#** aus, bei **Alle Plattformen**, **Windows** und bei **Alle Projekttypen** dann **API**.

Dann sollte ganz oben **ASP.NET Core-Web-API** auftauchen. Nun klickt man nur noch auf **Weiter** und dann unter **Projektname** dann den Namen deines Projektes. Wenn man jetzt wieder auf **Weiter** klickt, sollte man dann zu weiteren Einstellungen kommen. hier habe ich die Standarteinstellungen so gelassen.



Visual Studio weitere Informationen Tab beim Erstellen eines neuen Projektes.

Nachdem man auf **Erstellen** geklickt hat sollte das Projekt für dich Generiert werden und dann sollte es wie folgt aussehen:



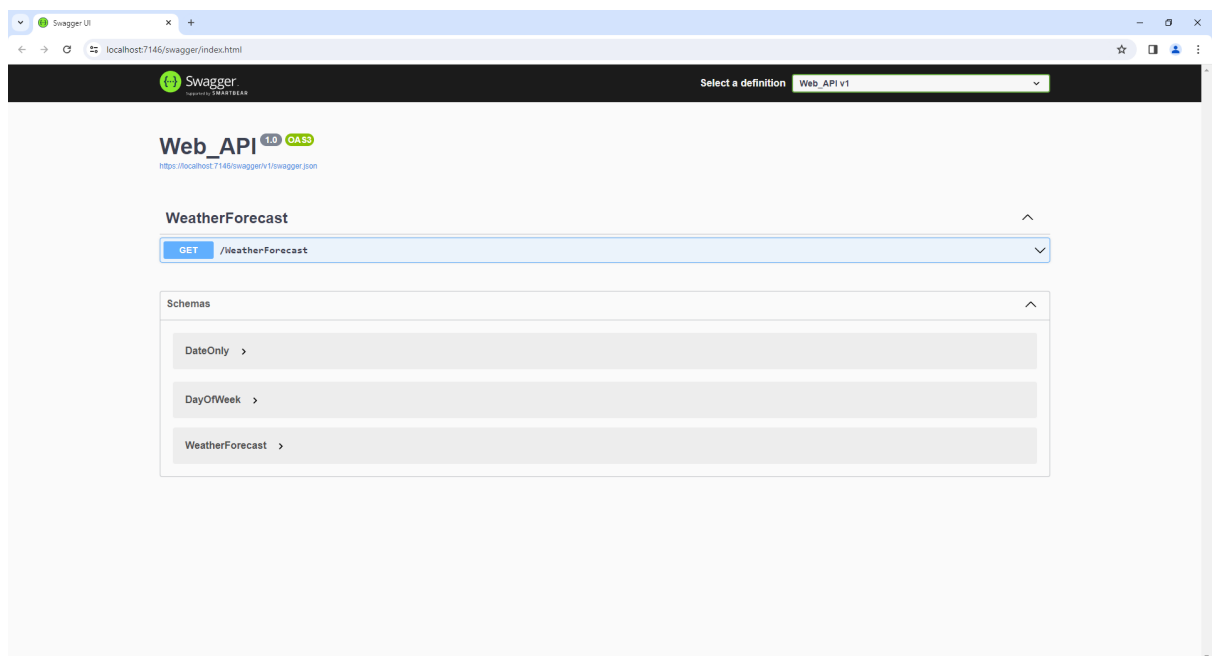
Visual Studio Editor mit der ASP.NET Core Web-API Vorlage

## Der erste Start

Sobald das Projekt erstellt wurde mit der ASP.NET Core Web-API Vorlage sollte man nun in der Lage sein das Projekt zu starten um sicher gehen zu können, ob das projekt erfolgreich und ohne Fehler wirklich erstellt wurde.

Um das Projekt zu starten, muss man dann nur oben auf den Grünen Pfeil klicken, wo `https` steht. Dann sollte der Google Chrome Browser sich öffnen mit der generierten Standard Seite. Solltest du einen anderen Browser bevorzugen kannst du diesen auch ändern, indem du auf den weißen Pfeil rechts neben dem `https` Text klickst und dann bei `Webbrowser` deinen Browser nach Wahl aussuchst. Beachte allerdings das Google Chrome am besten funktionieren sollte.

Sobald der Webserver gestartet ist, sollte sich dann ein Google Chrome Fenster öffnen und folgendes anzeigen

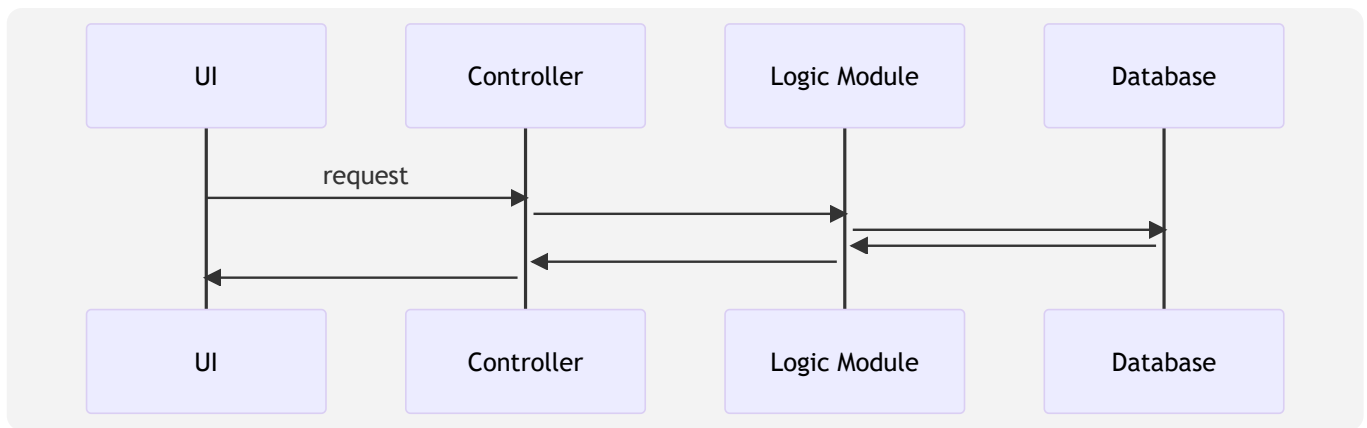


Google Chrome Fenster mit Swagger Open API Vorlage

## Erstellung eines neuen Controllers

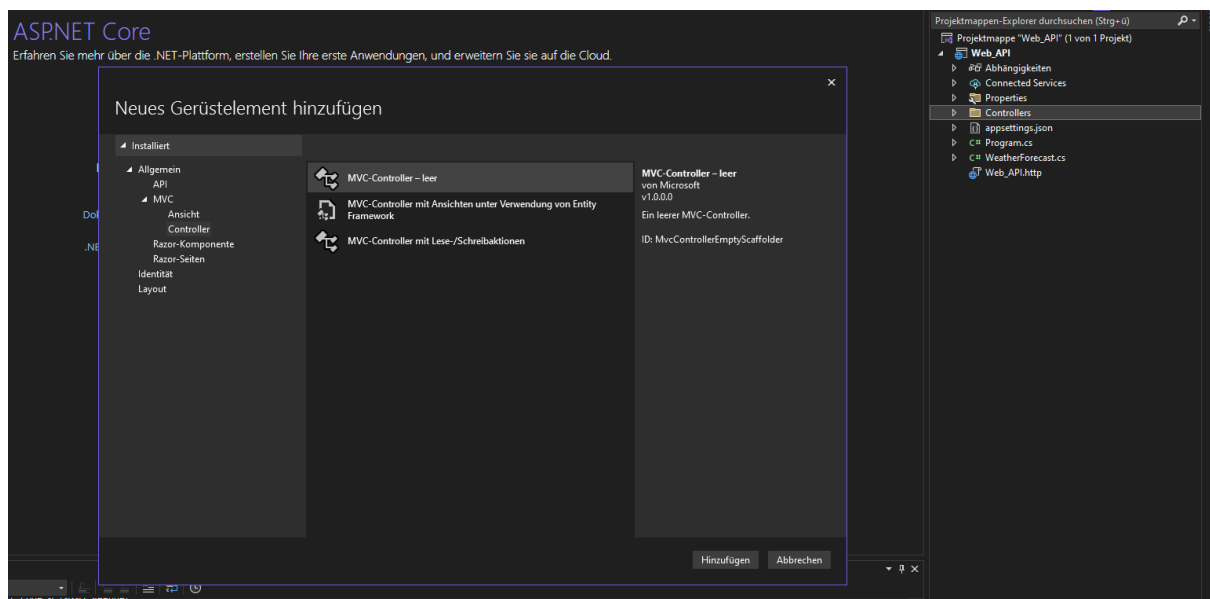
Bevor wir ein Controller erstellen müssen, wir erstmal wissen, wofür ein Controller ist und was der macht.





Alle Interaktionen und Anfragen gehen von der UI als Erstes zum Controller und vom Controller dann zum Logic Module und dann zur Datenbank

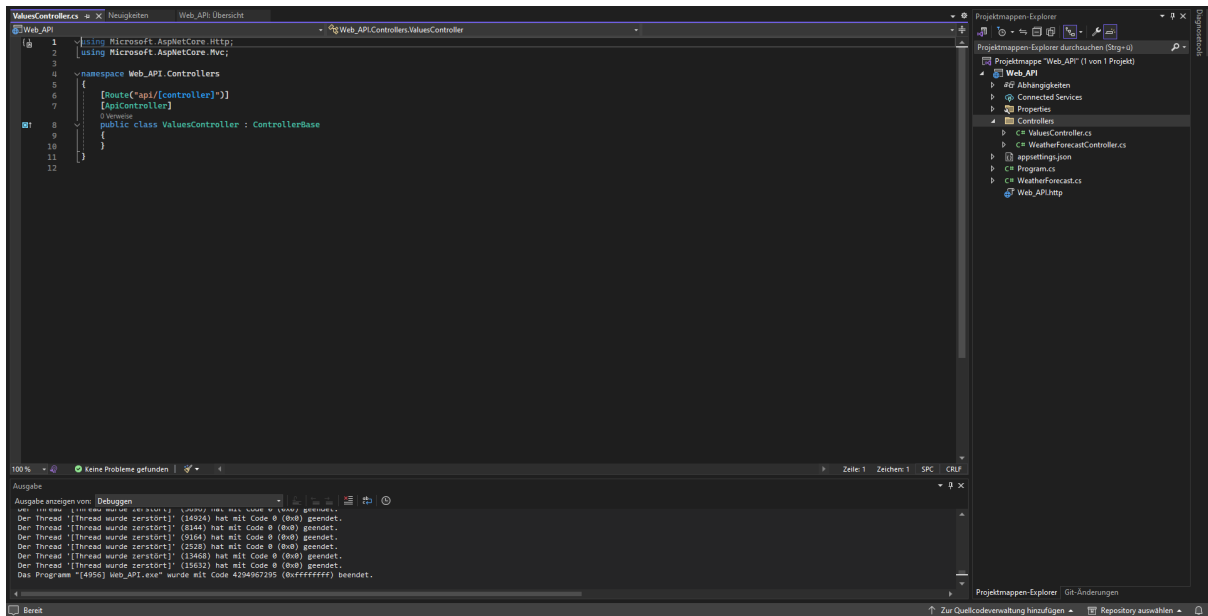
Nun können wir einen Controller erstellen, indem wir bei dem Controllers Ordner Rechtsklick machen und dann auf Hinzufügen und dann auf Controller. Jetzt sollte sich ein Fenster öffnen mit sachen die man Auswählen kann. Dieses Fenster sieht dann wie folgt aus:



Visual Studio mit dem "Neues Gerüstelement hinzufügen" Fenster

Sobald das Fenster dann offen ist wählen wir Links dann API aus und dann API Controller - Leer. Als Nächstes kannst du dann einen Namen für den Controller vergeben allerdings ist es hier besser den Standard Namen bei-belassen. In meinem Fall ist es ValuesController.cs. Danach kann man dann auf Hinzufügen klicken.

Jetzt sollte sich ein neues Fenster geöffnet haben dies sieht wie folgt aus:



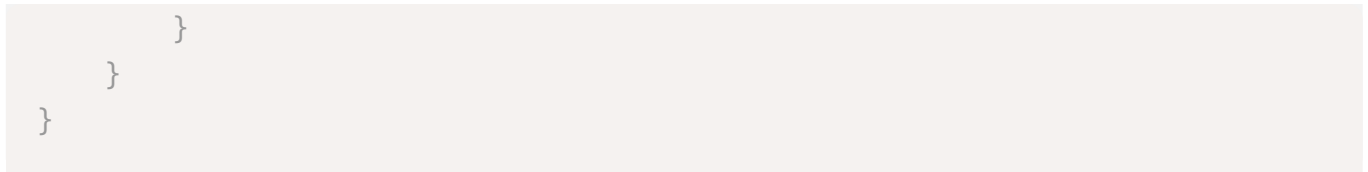
Visual Studio Code Editor - ValuesController.cs Standard Datei

In dem Tutorial wird erklärt das man etwas von ihm hinein Kopieren sollte. Allerdings gibt es dazu keine Referenz oder code zum Kopieren somit Tippe ich das von selbst ab.

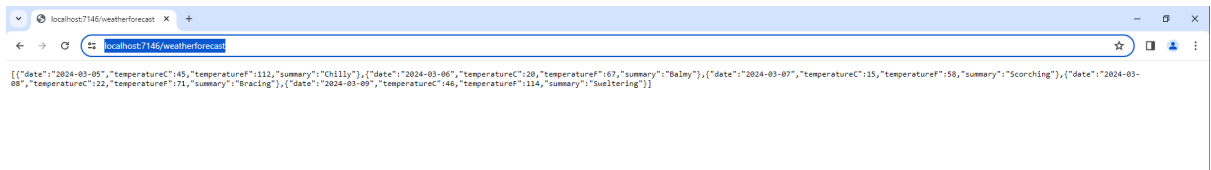
```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace Web_API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ValuesController : ControllerBase
    {
        [HttpGet]
        public IEnumerable<string> Get()
        {
            return new string[] { "Values 1", "Values 2", "Values 3",
"values4"};
        }

        [HttpGet("{id}")]
        public string Get(int id)
        {
            return "The value is "+id;
        }
    }
}
```



nun kann man das Program noch einmal starten und dann im Link `weatherforecast` hinzufügen so das der Link so ähnlich wie `https://localhost:7146/weatherforecast` ist. Wenn es geklappt hat, dann sieht die Seite wie folgt aus:



Google Chrome - weatherforecast API Page

und wenn man `/api/values` aufruft dann sollte die Seite wie folgt aussehen:

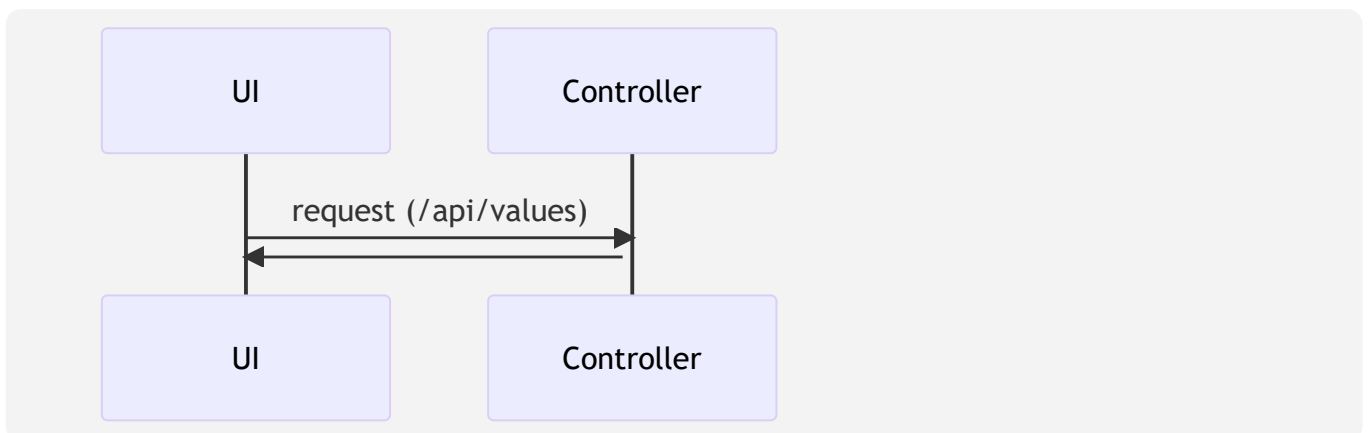


Google Chrome - API Values Page

Sobald man diese Seite aufruft und das auch Wiederbekommt dann hat es geklappt.

## Was macht der Code und wie funktioniert der?

zum aktuellen Zeitpunkt verläuft das so:



Der Nutzer sendet eine Anfrage an die UI. Die UI übergibt die Anfrage an dem Controller und der Controller sendet dann die jeweiligen Daten wieder zurück. In dem Fall sind es die folgenden Daten:

```
["Values 1","Values 2","Values 3","values4"]
```

Nun schauen wir uns den Code etwas genauer an.

Methode	Beschreibung
<code>[Route("api/[controller]")]</code>	Ist dafür da um dem Controller bzw der Web API zu sagen das der Link für diesen Controller <code>/api/values</code> ist und das man den darüber aufrufen kann. Wenn man nicht will das man <code>/api/values</code> Schreiben muss dann kann man <code>api/</code> auch einfach weg lassen.
<code>[HttpGet]</code>	Sagt dem Program dass das eine <code>GET</code> Anfrage ist.
<code>return new string[] { "Values 1", "Values 2", "Values 3", "values4" };</code>	Sagt das <code>["Values 1","Values 2","Values 3","values4"]</code> Zurückgegeben werden soll.
<code>[HttpGet("{id}")]</code>	Funktioniert wie <code>[HttpGet]</code> allerdings wird hier mit <code>[HttpGet("{id}")]</code> gesagt das es sich hier um etwas handelt was man nach dem <code>values</code> Schreibt. Also z.B. <code>/api/values/1</code>
<code>public string Get(int id)</code>	Ist auch eine Klasse für den <code>values</code> Endpunkt allerdings tritt das erst dann in Kraft wenn da auch eine zahl Definiert ist. Also z.B. <code>/api/values/1</code>

auf die andere Klasse kommen wir noch in den nächsten Schritten.

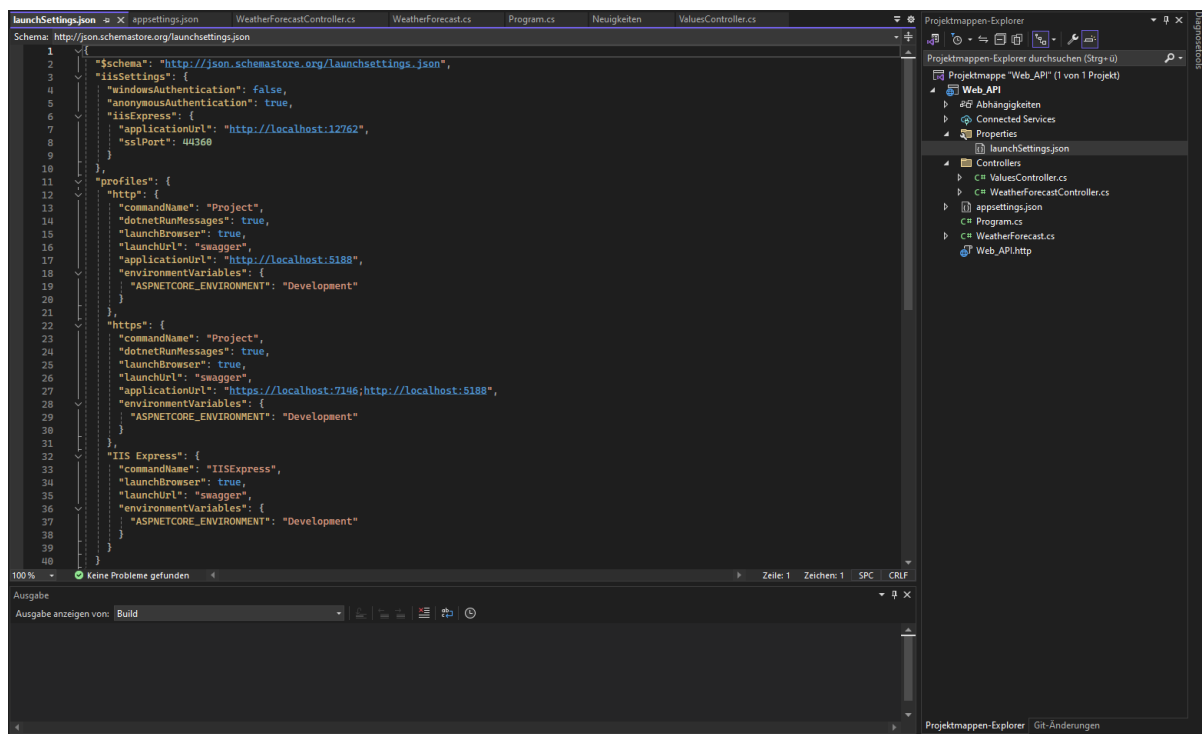
Jetzt fragt man sich sicher woher, das Program weiß, dass es bei diesem Controller um den `ValuesController.cs` handelt. Ganz Einfach. So wie ich es verstanden habe nimmt sich das dein Controller aus dem Namen. Das heißt wenn deine Controller Datei `AirController.cs` heißt dann handelt es sich bei `[controller]` dann um das Wort `air`

## Ändern des Startpfades

Wir haben sicher schon gemerkt das, wenn jedes Mal, wenn wir den Webserver Starten das wir auf eine bestimmte Seite immer weitergeleitet werden. Diesen Standartpfad werden wir

hier jetzt mal abändern zu unserer Values API Endpunkt.

Als Erstes suchen wir nach dem Ordner `Properties`. In diesem Ordner sollte dann eine Datei namens `launchSettings.json` zu finden sein. Diese sieht dann wie folgt aus:



The screenshot shows the Visual Studio Code editor with the `launchSettings.json` file open. The file content is as follows:

```
1 {
2   "$schema": "http://json.schemastore.org/launchsettings.json",
3   "iisSettings": {
4     "windowsAuthentication": false,
5     "anonymousAuthentication": true,
6     "iisExpress": {
7       "applicationUrl": "http://localhost:12762",
8       "sslPort": 44360
9     }
10  },
11  "profiles": {
12    "http": {
13      "commandName": "Project",
14      "dotnetRunMessages": true,
15      "launchBrowser": true,
16      "launchUrl": "swagger",
17      "applicationUrl": "http://localhost:5188",
18      "environmentVariables": {
19        "ASPNETCORE_ENVIRONMENT": "Development"
20      }
21    },
22    "https": {
23      "commandName": "Project",
24      "dotnetRunMessages": true,
25      "launchBrowser": true,
26      "launchUrl": "swagger",
27      "applicationUrl": "https://localhost:7146;http://localhost:5188",
28      "environmentVariables": {
29        "ASPNETCORE_ENVIRONMENT": "Development"
30      }
31    },
32    "IIS Express": {
33      "commandName": "IISExpress",
34      "launchBrowser": true,
35      "launchUrl": "swagger",
36      "environmentVariables": {
37        "ASPNETCORE_ENVIRONMENT": "Development"
38      }
39    }
40  }
41 }
```

Visual Studio Code Editor - launchSettings.json Datei Inhalt

In dieser Datei müssen wir bei `profiles` folgendes ändern:

```
"profiles": {
  "http": {
    "commandName": "Project",
    "dotnetRunMessages": true,
    "launchBrowser": true,
    "launchUrl": "api/values",
    "applicationUrl": "http://localhost:5188",
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    }
  },
}
```

also bei `launchUrl` von `swagger` zu `api/values` ändern.



Sollte das bei dir nicht funktioniert haben stelle einfach überall wo swagger steht zu `api/values` um.

## Erstellen von Models

### Wofür sind Models?

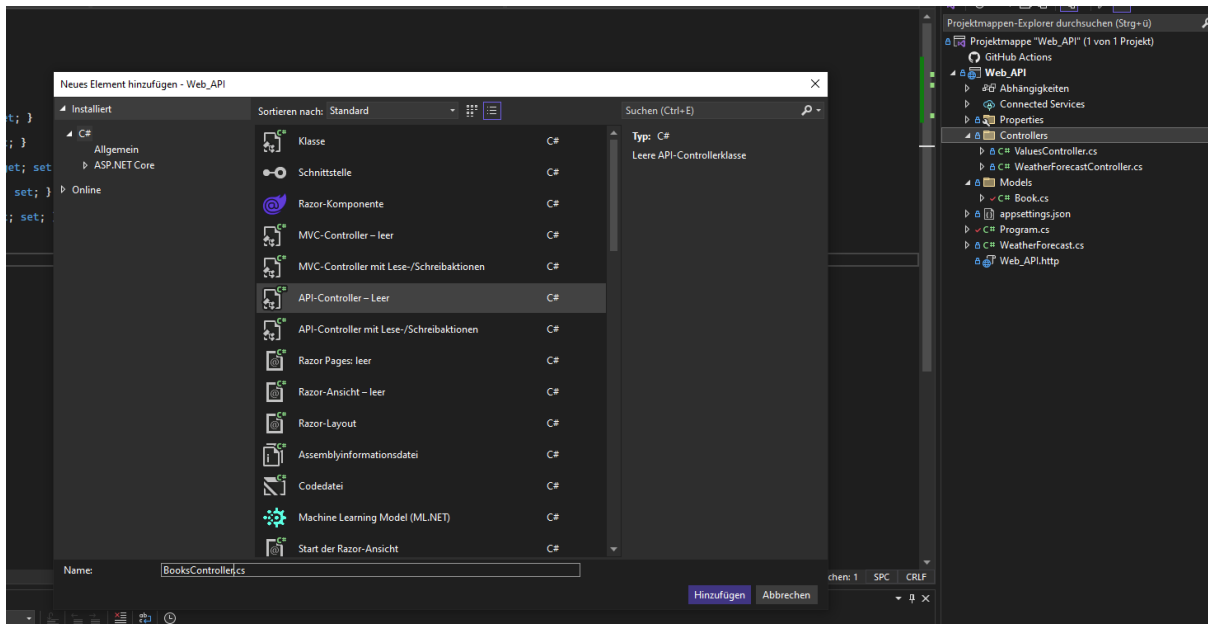
### Wie erstellt man ein Model?

Um ein Model zu erstellen ist es wichtig ein ungeschriebenes Gesetz zu befolgen. Und zwar bekommen Models auch immer ihren eigenen `Models` Ordner. Als Nächstes erstellen wir eine Klasse in diesem `Models` ordner. In diesem Fall haben wir diese Klasse `Book` wie im Tutorial genannt und fügen folgende Zeilen hinzu:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
namespace Web_API.Models
{
    public class Book
    {
        public int Id { get; set; }
        public string Author { get; set; }
        public string Title { get; set; }
        public int PublicationYear { get; set; }
        public bool IsAvailable { get; set; }
        public string CallNumber { get; set; }
    }
}
```

### Erstellung des Books Controllers

Um den Controller `Books` zu erstellen, muss man hier wieder bei dem `Controllers` Ordner rechtsklick machen und dann auf Hinzufügen > Controller. Dann öffnet sich wieder ein Fenster. In diesem Fenster wählen wir dann links wieder `API` aus und nehmen `API Controller - Empty`, klicken auf Hinzufügen und nun müssen wir es unten bei Name `BooksController` nennen.



Visual Studio - New Element Window

Nun sollte sich eine wieder eine neue Datei Öffnen und drinnen steht folgendes:

```
[...]

namespace Web_API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class BooksController : ControllerBase
    {

    }
}
```

### **i** Info

[...] bedeutet nichts weiter als einfach die Standarddaten wie using Microsoft.AspNetCore.Http; usw. Es ist einfach nur ein Platzhalter und bedeutet das hier Daten sind.

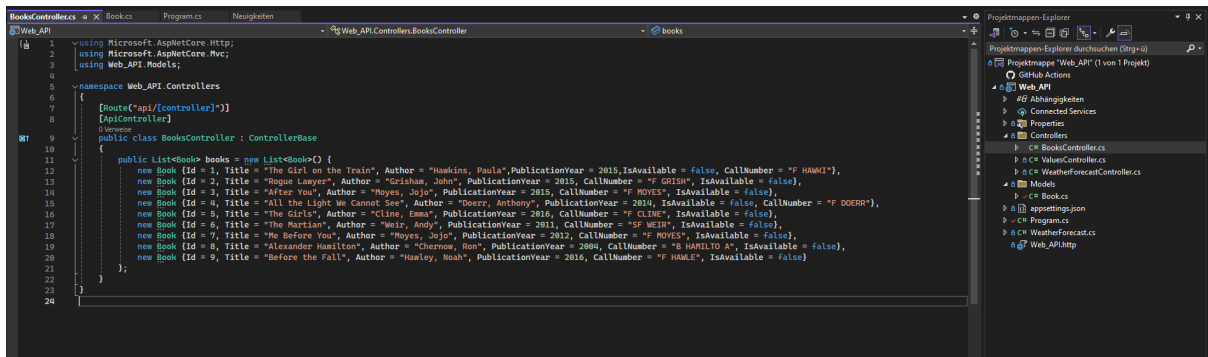
Sobald wir sichergestellt haben das die Datei erfolgreich erstellt wurde fügen wir nun die folgenden Testdaten ein.

## Info

In dem Tutorial Video wird ständig gesagt, das man es von der verlinkten Ressource sich rauskopieren soll, allerdings ist weder in der Beschreibung, noch in den Kommentaren ein Link wo man das Raus kopieren kann. Somit habe ich das abgetippt.

```
[...]  
public class BooksController : ControllerBase  
{  
    public List<Book> books = new List<Book>() {  
        new Book {Id = 1, Title = "The Girl on the Train", Author =  
            "Hawkins, Paula", PublicationYear = 2015, isAvailable = false, CallNumber = "F  
HAWKI}  
        new Book {Id = 2, Title = "Rogue Lawyer", Author = "Grisham, John",  
            PublicationYear = 2015, CallNumber = "F GRISH", IsAvailable = false}  
        new Book {Id = 3, Title = "After You", Author = "Moyes, Jojo",  
            PublicationYear = 2015, CallNumber = "F MOYES", IsAvailable = false}  
        new Book {Id = 4, Title = "All the Light We Cannot See", Author =  
            "Doerr, Anthony", PublicationYear = 2014, IsAvailable = false, CallNumber =  
            "F DOERR"}  
        new Book {Id = 5, Title = "The Girls", Author = "Cline, Emma",  
            PublicationYear = 2016, CallNumber = "F CLINE", IsAvailable = false}  
        new Book {Id = 6, Title = "The Martian", Author = "Weir, Andy",  
            PublicationYear = 2011, CallNumber = "SF WEIR", IsAvailable = false}  
        new Book {Id = 7, Title = "Me Before You", Author = "Moyes, Jojo",  
            PublicationYear = 2012, CallNumber = "F MOYES", IsAvailable = false}  
        new Book {Id = 8, Title = "Alexander Hamilton", Author = "Chernow,  
            Ron", PublicationYear = 2004, CallNumber = "B HAMILTO A", IsAvailable =  
            false}  
        new Book {Id = 9, Title = "Before the Fall", Author = "Hawley,  
            Noah", PublicationYear = 2016, CallNumber = "F HAWLE", IsAvailable = false}  
    };  
}  
[...]
```





Visual Studio Code Editor - BooksController Static Template Data

## Info

Solltest du einen Fehler bekommen bei Book das es nicht existiert dann Hovere mit der Maus über den Book Text und dann wähle bei dem Menü aus das du die Books.Models importierst.

Nun erstellen wir wieder in der Datei und unter dem Code von den Test Daten zwei Methoden die wieder HTTP GET Methoden Verarbeiten sollen.

Diese sieht dann so aus:

[...]

```
[HttpGet]
public ActionResult<IEnumerable<Book>> GetAllBooks()
{
    return books;
}

[HttpGet("{id}")]
public ActionResult<Book> GetBook(int id)
{
    var book = books.FirstOrDefault(x => x.Id == id);
    if(book == null)
    {
        return NotFound();
    }
}
```

[...]

Das was, der Code macht, ist wie folgt. Sobald man im Link `/api/books` eingibt, wird der folgende Code ausgeführt:

```
[HttpGet]
public ActionResult<IEnumerable<Book>> GetAllBooks()
{
    return books;
}
```

`IEnumerable<Book>` sagt einfach nur aus das eine Liste an Arrays von der Klasse `Book` ausgegeben wird und mit `return books;` werden die Daten dann auch ausgegeben.

allerdings wenn man `/api/books/1` zum Beispiel in die URL eingibt, dann wird der Folgende Code ausgegeben:

```
[HttpGet("{id}")]
public ActionResult<Book> GetBook(int id)
{
    var book = books.FirstOrDefault(x => x.Id == id);
    if(book == null)
    {
        return NotFound();
    }
    return book;
}
```

aufgrund dessen das `("{id}")` in `[HttpGet("{id}")]` steht weiß das Programm das nach dem `books/` in der URL noch etwas folgt. In diesem Fall ist es eine ID. `GetBook(int id)` sagt einfach nur das die Klasse einen Integer braucht und diese Integer Nummer, die dann da übergeben wird, ist dann in der Variable `id` gespeichert. Aufgrund dessen das darüber `[HttpGet("{id}")]` steht, wird Automatisch der Klasse die `id` übergeben von dem Link. In der Klasse wird dann in der `books` Klasse abgefragt ob, es so einen Eintrag in der `Id` existiert und wenn nicht, soll ein `404 Not Found` fehler zurückgegeben werden.

## Warum ActionResult<> ?

In dem Code was wir bei Erstellung eines neuen Controllers hatten, haben wir kein `ActionResult<>` genutzt, sondern `IEnumerable`. Das hat den Grund das man bei nicht so komplexen Aufgaben das mit `ActionResult<>` nicht braucht. Allerdings bei etwas mehr komplexeren sachen wie in dem fall, was wir jetzt gerade haben ist, `ActionResult<>` recht nützlich da wir damit dann eine überprüfung machen können und dann, auch wenn da etwas nicht klappt oder nicht gefunden wird das dann auch dementsprechend unsere Fehlermeldung zurückkommt.


Wenn das ganze funktioniert hat, dann können wir die API weider Starten und dann das, was wir gerade programmiert haben ausprobieren. Wie schon gesagt, wenn wir `/api/books/` eingeben dann bekommen wir alle Daten wir angegeben haben zurück. Allerdings, wenn wir Beispiel `/api/books/4` eingeben dann bekommen wir die Daten von nur einem Buch, und zwar den wessen ID wir angegeben haben.


## Dynamisieren des Codes

### Als Erstes was bedeutet Dynamisch?

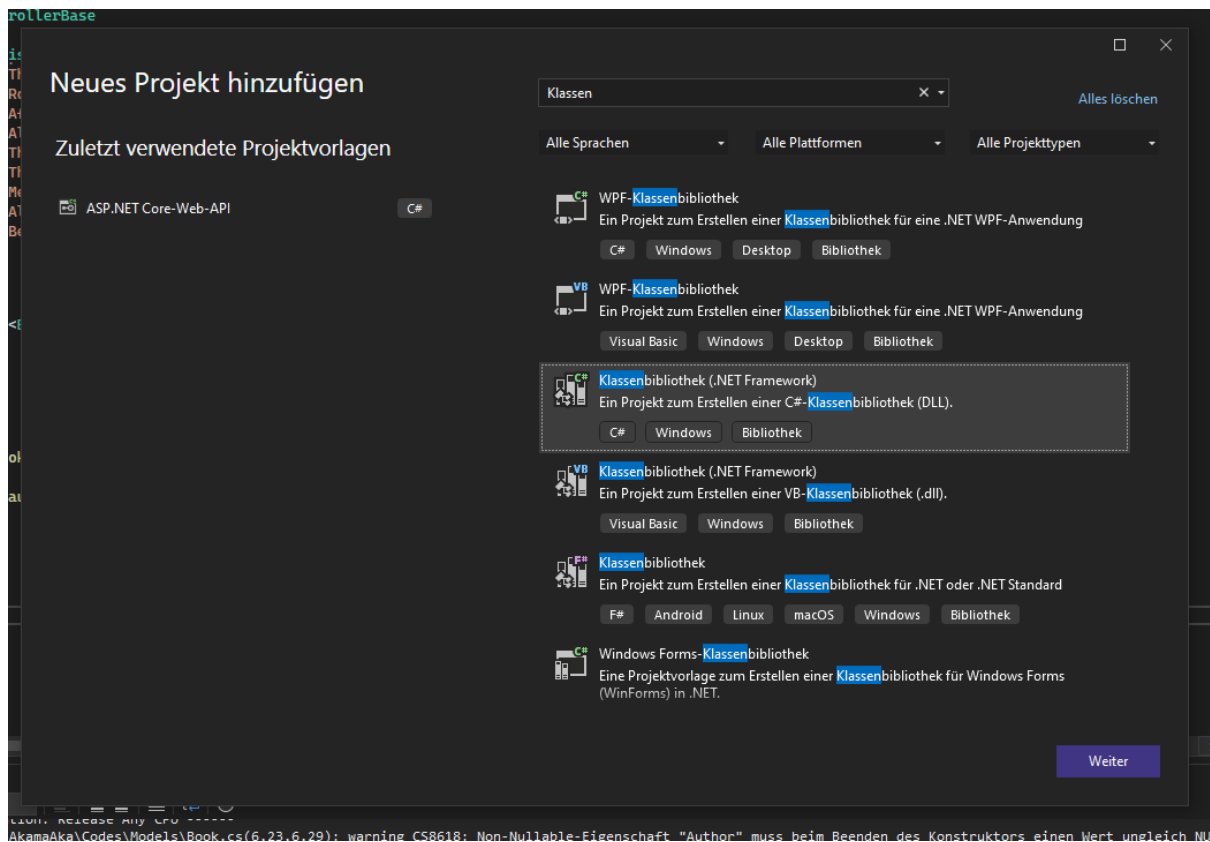
Dynamisch ist immer etwas, was sich jederzeit ändert ohne etwas am Quellcode z.B. zu ändern. Ab hier implementieren wir eine Struktur das die Bücher aus der Datenbank herausliest und dann verarbeitet und anzeigt.

Ab hier erstellen wir ein neues Projekt. Man wird sich sicher fragen, warum ein neues Projekt? Es ist so das es immer eine gute Idee ist die UI von der gesamten Logik zu trennen laut dem YouTuber wo er bei einem Kommentar das beantwortet.

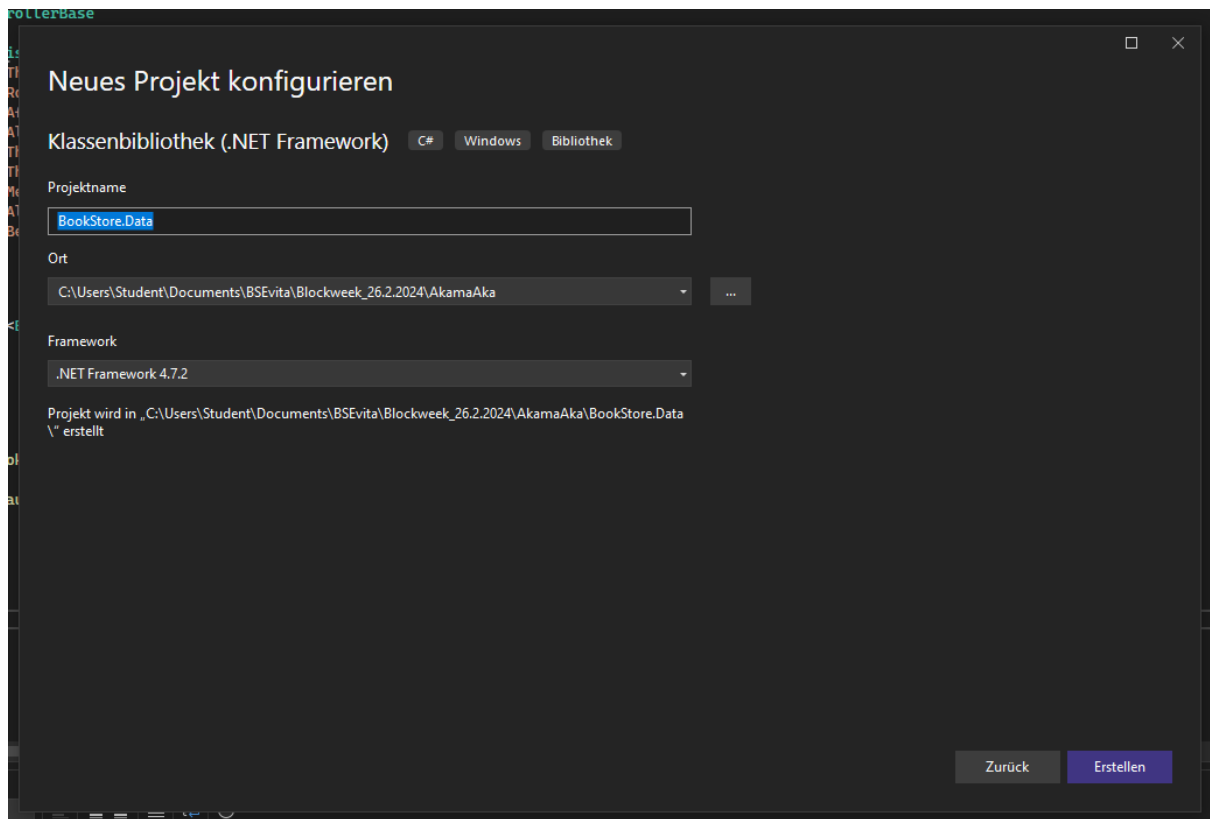
 [...] before creating the BookStore.Data project with its interfaces,[...] the API was already working... why did you have to create it then? [...]

 Thank you for your comment. It's good to separate the UI from the business logic. Data and business logic should be in its own layer

Um ein neues Projekt zu erstellen in dem aktuellen Projekt, was wir gerade haben, machen wir, ganz oben bei dem Projektmappen-Explorer wo am anfang steht Projektmappe ein Rechtsklick und dann auf Hinzufügen > Neues Projekt. Dann soll sich ein neues Fenster Auswählen und hier suchst du nach class und wählst dann das aus wo als Titel Klassenbibliothek (.NET Framework) steht. Dann klickst du auf Weiter und nennst die Klasse BookStore.Data und klickst dann auf Erstellen

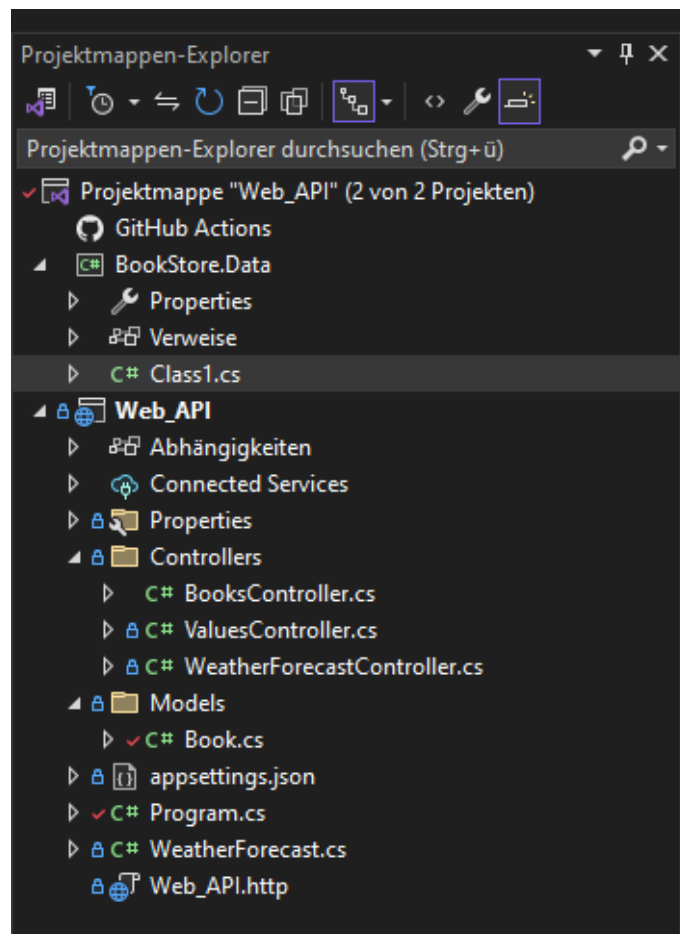


Visual Studio - New Project Window Class Library .NET Core



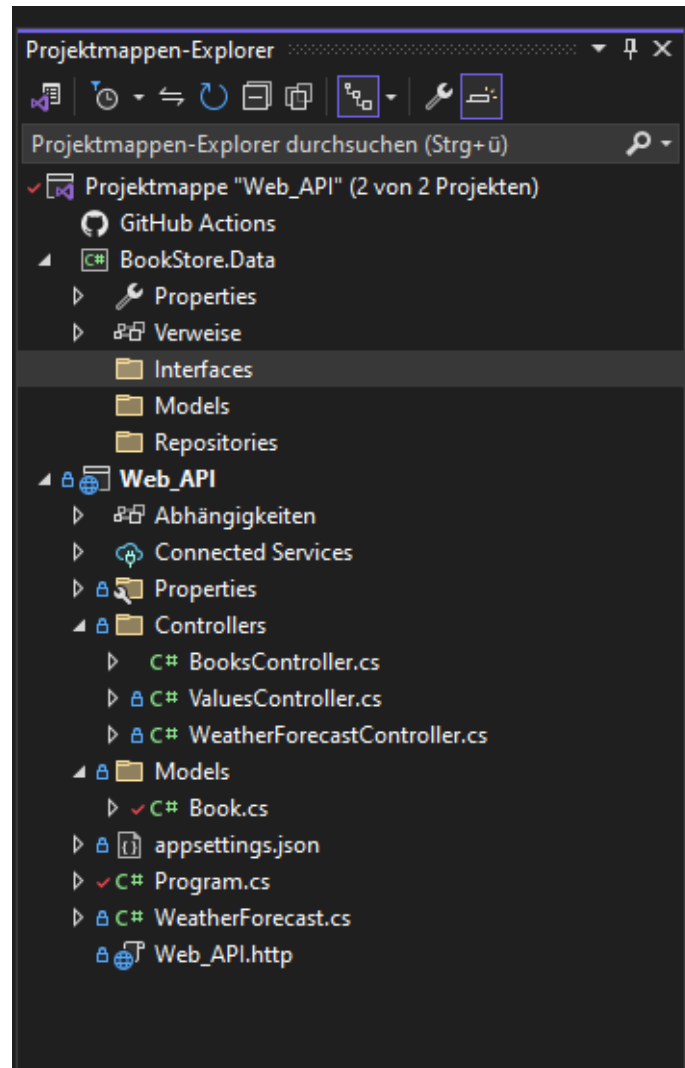
Visual Studio - New Project Window Class Library .NET Core Naming

Jetzt solltest du ein neues Projekt sehen in deinem Solutions Explorer mit dem Namen `BookStore.Data`. Nun öffnest du dieses Projekt und löschst erstmal die Standardklasse namens `Class1.css`



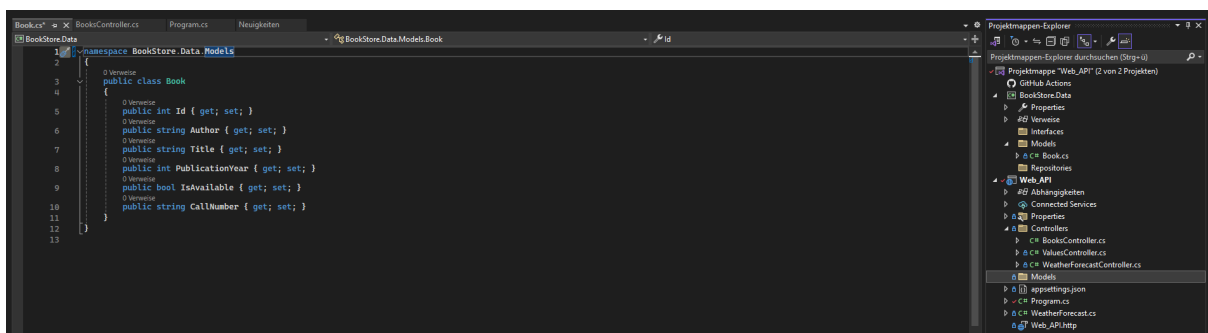
Visual Studio - Project Explorer Window BookStore.Data

Nun erstellen wir drei neue Verzeichnisse. Einmal Interfaces, Models und Repositories.



Visual Studio - Project Explorer Window BookStore.Data default folder creation

Jetzt können wir bei dem Projekt den wir vorher hatten (ich nenne es ab hier mal Projekt 1) im **Models** Verzeichnis die Klasse **Book.cs** Kopieren und in das neue Projekt (Ich nenne das ab hier jetzt Projekt 2) in den **Models** Ordner rein.



Visual Studio - Book.cs File in Project 2

### ⚠ Wichtig

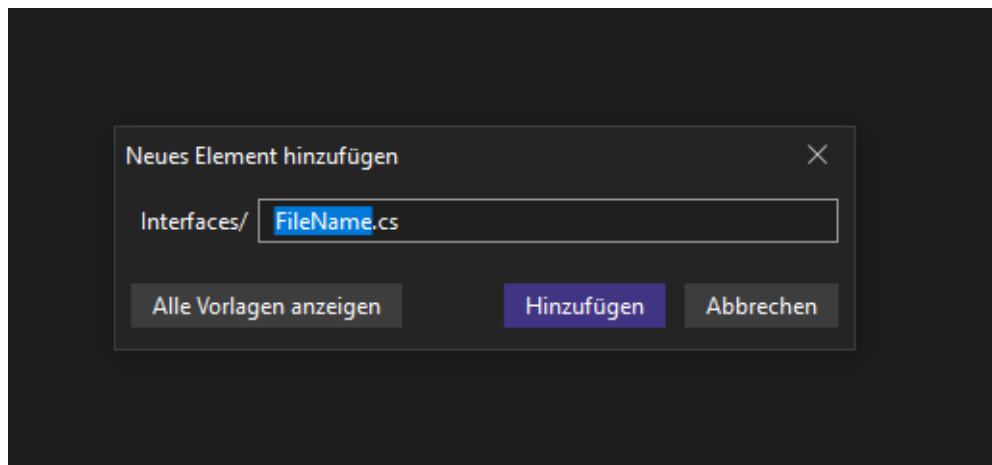
Beachte das du in der `Book.cs` Klasse in Projekt 2 auch den Namespace von `Web_API.Models` zu `BookStore.Data.Models` umbenennst.

Nachdem du das gemacht hast, kannst du in Projekt 1 auch dann in der `Book.cs` Datei die Klasse `Book` auch Auskommentieren oder löschen.

### Erstellung des Interfaces

Jetzt erstellen wir in Projekt 2 ein Interface. Dies kann man machen, indem man bei Projekt 2 bei dem interface Ordner Rechtsklick drückt und dann auf `Neues Element hinzufügen` klickt. Dann sollte sich wieder ein neues Fenster öffnen und da wählst du dann `Schnittstelle` aus und benennst es nach `IBookRepository.cs`. Sollte das erfolgreich gewesen sein sollte sich wieder eine neue Datei geöffnet haben mit dem folgenden Inhalt:

Solltest du nur folgendes sehen dann Klicke unten auf `Alle Vorlagen anzeigen`.



Img 18

```
[...]
```

```
namespace BookStore.Data.Interfaces
{
    interface IBookRepository
    {
    }
}
```



Damit das Interface auch von überall aus Zugreifbar ist, müssen wir die Datei Public machen. Das sieht dann so aus:

```
[...]  
public interface IBookRepository  
[...]
```

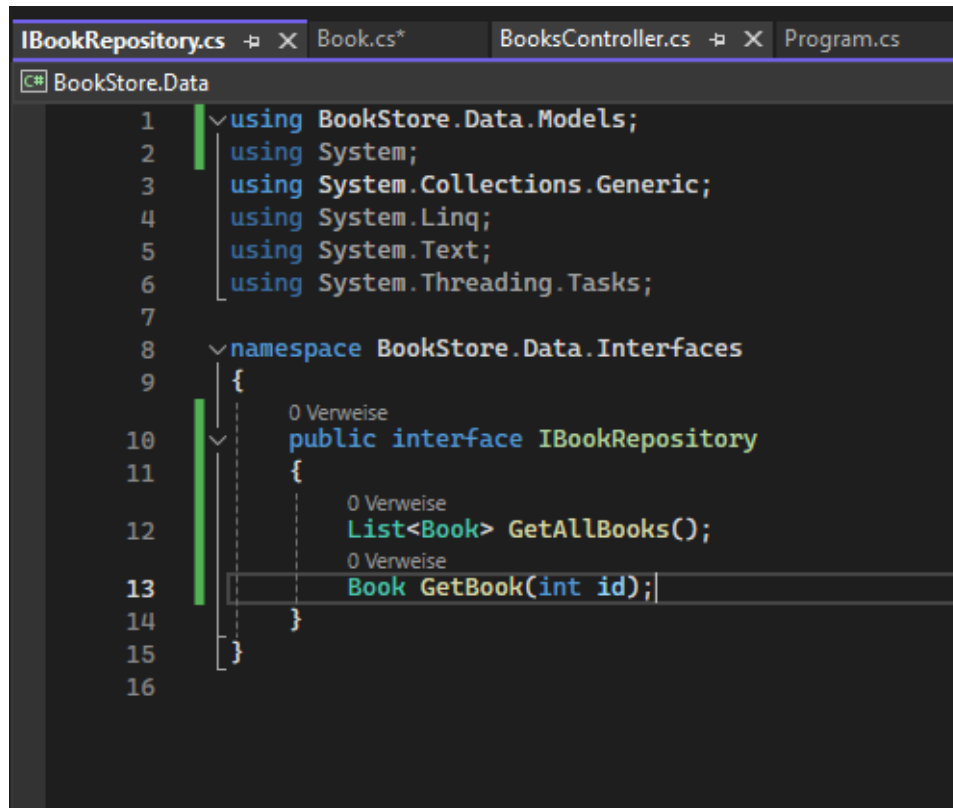
Damit die API mit der Klasse auch Interagieren kann und auch die Bücher abrufen kann ändern wir den Code noch wie folgt:

```
public interface IBookRepository  
{  
    List<Book> GetAllBooks();  
    Book GetBook(int id);  
}
```

#### **Warnung**

Solltest du Fehlernachrichten bekommen bei `books` dann Ignoriere die. Die werden im Laufe der Dokumentation dann gefixt.

Damit auch auf die Models zugegriffen werden kann, müssen wir das Model dann auch mit einbinden. Dazu Schreibt man einfach ganz oben in der Datei `using BookStore.Data.Models;`.



```
1 using BookStore.Data.Models;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace BookStore.Data.Interfaces
9 {
10     public interface IBookRepository
11     {
12         List<Book> GetAllBooks();
13         Book GetBook(int id);
14     }
15 }
16
```

Visual Studio Code Editor - BookStore.Data.Interfaces IBookRepository.cs File

Als Nächstes erstellen, wir in dem Repositories Ordner eine Klasse mit dem Namen `BookRepository.cs` die dann mit dem Interface interagiert.

Der plan den wir gerade verfolgen ist diese. Wir haben einmal Projekt 1. Projekt 1 ist einfach nur die API was die ganzen Daten von Projekt 2 empfängt und dann auch ausgibt. Einfach gesagt der Empfänger und Ausgeber.

Projekt 2 hingegen verarbeitet die Daten, die es erhält und sendet diese dann an Projekt 1 weiter und Projekt 1 gibt diese dann dem UI also dem Endbenutzer weiter.

Sobald die Klasse erstellt wurde, werden wir wieder mit der Standard Datei begrüßt. Hier müssen wir dann einfach wieder die Klasse `public` machen wie schon bei der `IBookrepositorie.cs` Datei gemacht wurde.

```
[...]  
namespace BookStore.Data.Repositories {  
    public class BookRepository : IBookRepository  
    {  
          
    }  
}
```

```
}
}
```

Sobald das gemacht wurde geben wir der Klasse auch unseren IBookRepository interface mit. Als Nächstes müssen wir dann auch noch die Methoden hinzufügen, die auch wichtig sind damit wir mit dieser Datei Abschließen können. hier können wir dann einfach den Code aus der BookController.cs Datei rein Kopieren. Das ganze sieht dann ungefähr so aus:

Jetzt können wir bei dem rot unterstrichenen Text mit der Maus darüber gehen und dann sollte dir Angezeigt werden unten mögliche Korrekturen Anzeigen. Hier klickst du drauf und dann noch auf Schnittstelle Implementieren. So werden die Klassen dann Automatisch für dich erstellt.

Am Ende sieht das dann so aus:

```

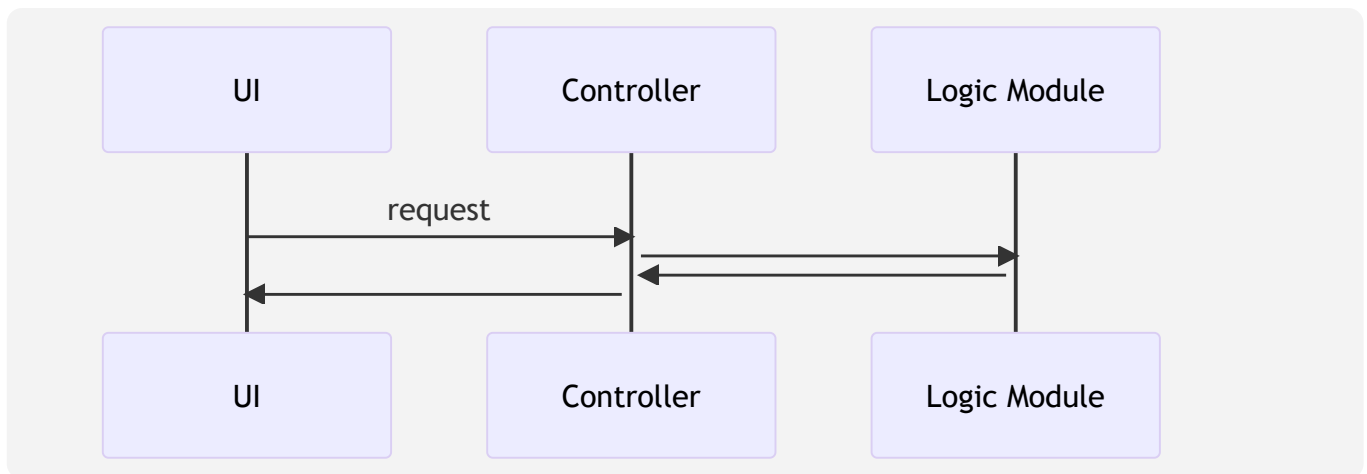
8
9
10 namespace BookStore.Data.Repositories
11 {
12     2 Verweise
13     public class BookRepository : IBookRepository
14     {
15         public List<Book> books = new List<Book>() {
16             new Book {Id = 1, Title = "The Girl on the Train", Author = "Hawkins, Paula", PublicationYear = 2015, IsAvailable = false, CallNumber = "F HAWKI"},
17             new Book {Id = 2, Title = "Rogue Lawyer", Author = "Grisham, John", PublicationYear = 2015, CallNumber = "F GRISH", IsAvailable = false},
18             new Book {Id = 3, Title = "After You", Author = "Moyes, Jojo", PublicationYear = 2015, CallNumber = "F MOYES", IsAvailable = false},
19             new Book {Id = 4, Title = "All the Light We Cannot See", Author = "Doerr, Anthony", PublicationYear = 2014, IsAvailable = false, CallNumber = "F DOERR"},
20             new Book {Id = 5, Title = "The Girls", Author = "Cline, Emma", PublicationYear = 2016, CallNumber = "F CLINE", IsAvailable = false},
21             new Book {Id = 6, Title = "The Martian", Author = "Weir, Andy", PublicationYear = 2011, CallNumber = "SF WEIR", IsAvailable = false},
22             new Book {Id = 7, Title = "Me Before You", Author = "Moyes, Jojo", PublicationYear = 2012, CallNumber = "F MOYES", IsAvailable = false},
23             new Book {Id = 8, Title = "Alexander Hamilton", Author = "Chernow, Ron", PublicationYear = 2004, CallNumber = "B HAMILTO A", IsAvailable = false},
24             new Book {Id = 9, Title = "Before the Fall", Author = "Hawley, Noah", PublicationYear = 2016, CallNumber = "F HAWLE", IsAvailable = false}
25         };
26
27         2 Verweise
28         public List<Book> GetAllBooks()
29         {
30             return books;
31         }
32
33         2 Verweise
34         public Book GetBook(int id)
35         {
36             return books.FirstOrDefault(x => x.Id == id);
37         }
38     }
39 }

```

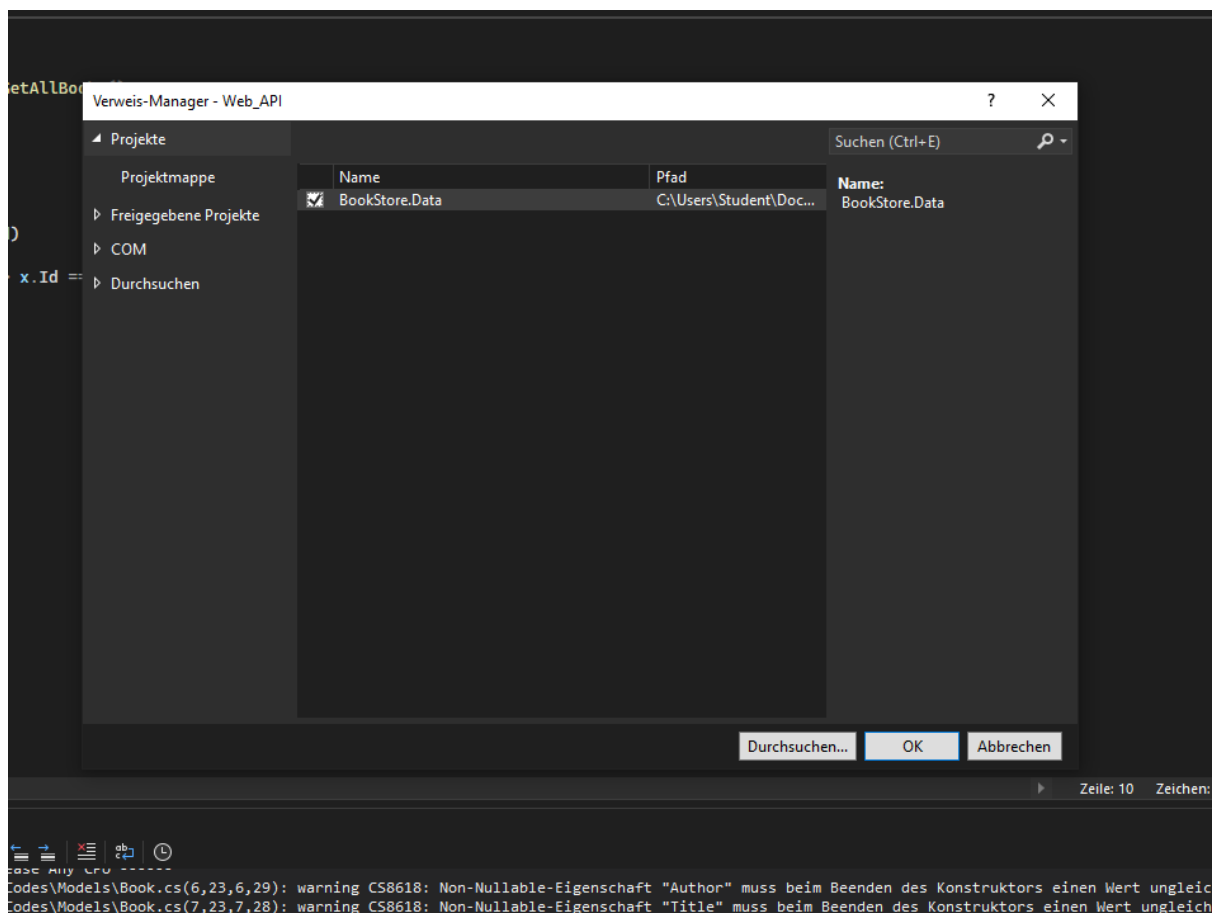
Img 21

Solltest du bei FirstOrDefault einen fehler bekommen dann musst du using System.Linq; ganz oben in der Datei hinzufügen.

Und jetzt haben wir auch das Logic Model erreicht.



Jetzt Navigieren wir in Projekt 1 zu der BooksController.cs Datei und Löschen erstmal den Teil raus wo die ganzen Bücher drinnen stehen. Dann müssen wir Projekt 2 mit Projekt 1 Verknüpfen damit die miteinander kommunizieren können. Das kann man machen, indem man bei Projekt 1 bei **Abhängigkeiten** Rechtsklick macht und dann auf **Projektverweis hinzufügen**. Dann sollte sich noch ein Fenster öffnen. In diesem Fenster klickt man dann in den Weißen Viereck rein und klickt dann auf Fertig.



Visual Studio - Verweis Manager

Sollte das nicht geklappt haben und die Auswahl im Weißen Viereck ist nicht mehr drinnen dann musst du bei Projekt 2 bei Dependencies rechtsklick machen und dort dann auf **Projektverweis hinzufügen** und dann dort den Haken wegmachen und dann sollte das bei Projekt 1 dann auch gehen.

Nun können wir den **books** und **Book** Fehlern nach gehen. Um die Fehler zu fixen, müssen wir dann einfach nur ganz oben bei den jeweiligen Dateien **using BookStore.Data.Models;** Hinzufügen. Mit diesem fix ist dann der Fehler mit **Book** gefixt.

Um den Fehler von den **books** zu fixen, müssen wir folgende Zeile über dem **[HttpGet]** hinzufügen:

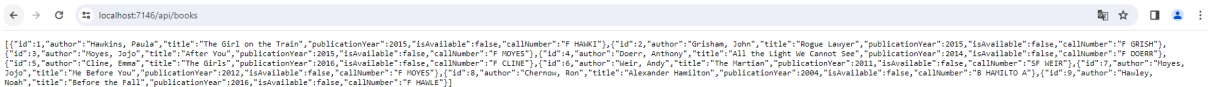
```
private BookRepository books = new BookRepository();
```

als Nächstes müssen wir bei **books**, **getAllBooks()** Hinzufügen. Und anstatt **var book = books.FirstOrDefault(x => x.Id == id);** machen wir dann **var book = books.GetBook(id);**

Nun können wir das Programm wieder starten, indem wir auf den Grünen Pfeil klicken.

**⚠** Solltest du Fehler bekommen, überprüfe, ob du bei der **BooksController.cs** Datei **using BookStoreAPI.Models;** entfernt hast.

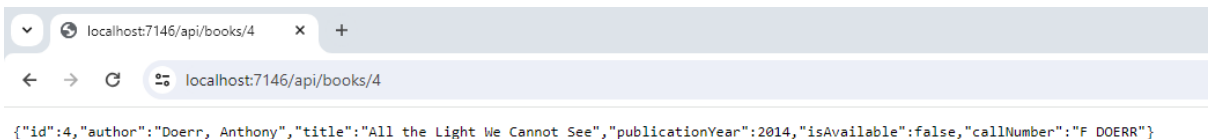
Nun kann man das Projekt Starten und dann sollte folgende Antwort kommen im Browser, sobald man **/api/books** eingibt.



```
[{"id":1,"author":"Hawkins, Paula","title":"The Girl on the Train","publicationYear":2015,"isAvailable":false,"callNumber":"F HAWKI"}, {"id":2,"author":"Grisham, John","title":"Rogue Lawyer","publicationYear":2015,"isAvailable":false,"callNumber":"F GRISH"}, {"id":3,"author":"Hoyes, Jojo","title":"After You","publicationYear":2015,"isAvailable":false,"callNumber":"F HOYES"}, {"id":4,"author":"Doerr, Anthony","title":"All the Light We Cannot See","publicationYear":2014,"isAvailable":false,"callNumber":"F DOERR"}, {"id":5,"author":"Cline, Emma","title":"The Girls","publicationYear":2016,"isAvailable":false,"callNumber":"F CLINE"}, {"id":6,"author":"Heir, Andy","title":"The Mortal","publicationYear":2011,"isAvailable":false,"callNumber":"F HEIR"}, {"id":7,"author":"Hoyes, Jojo","title":"We Before You","publicationYear":2012,"isAvailable":false,"callNumber":"F HOYES"}, {"id":8,"author":"Chernow, Ron","title":"Alexander Hamilton","publicationYear":2004,"isAvailable":false,"callNumber":"B HAMILTO A"}, {"id":9,"author":"Mauley, Noah","title":"Before the Fall","publicationYear":2016,"isAvailable":false,"callNumber":"F MAULEY"}]
```

Img 22

und folgendes bei **/api/books/4**:



```
{"id":4,"author":"Doerr, Anthony","title":"All the Light We Cannot See","publicationYear":2014,"isAvailable":false,"callNumber":"F DOERR"}
```

img\_23.png