https://www.halvorsen.blog

Create a minimal API with ASP.NET Core with CRUD functionality

Hans-Petter Halvorsen

Contents

- We will use ASP.NET Core to create a Web/REST/HTTP API.
  - ASP.NET is a server-side framework for creating web pages and web contents.
- We will create a so-called “Minimal API with ASP.NET Core” and we will use the “ASP.NET Core Empty” template in Visual Studio.
- The API will have CRUD functionality
  - We will implement a minimal CRUD API that Create, Read, Update and Delete data in the Database.
  - We will use SQL Server as the Database system.
- We will use Visual Studio as the code editor.

Introduction

Hans-Petter Halvorsen

Introduction

- We can create Web/REST/HTTP APIs in Visual Studio and C# using the ASP.NET Web framework.
- This can be done in many ways, and Microsoft also continuously updates Visual Studio with new approaches and new templates.
- A new approach from Microsoft is called “Minimal API with ASP.NET Core” and you use the “ASP.NET Core Empty” template in Visual Studio.

https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api

Minimal APIs

- ASP.NET Core supports two approaches to creating APIs: a controller-based approach and minimal APIs.
- Minimal APIs are architected to create HTTP APIs with minimal dependencies.
- They're ideal for microservices and apps that want to include only the minimum files, features, and dependencies in ASP.NET Core.
- Basically, Minimal APIs is a new simplified approach for creating APIs with ASP.NET Core.
- This tutorial teaches the basics of building a minimal API with ASP.NET Core.

https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api

ASP.NET

- ASP.NET is a framework for web development.
- You can use ASP.NET for creating Web Applications or Web/REST APIs.
- ASP.NET is based on .NET and C#.
- What is the difference between ASP.NET and .NET frameworks?
  - ASP.NET is specifically designed for web development, while the .NET framework covers a broader range of application types, including Windows desktop, mobile, and web applications.
- Homepage: https://dotnet.microsoft.com/en-us/apps/aspnet

API

- Application Programming Interface (API).
- An API is a way for two or more computer programs or components to communicate with each other.
- It is a type of software interface that offers a service to other software.
- APIs come in many shapes, some examples are SOAP API, REST API, GraphQL API, etc.
- Most programming languages today have components/libraries that can be used both to create APIs and to consume APIs (using existing APIs).

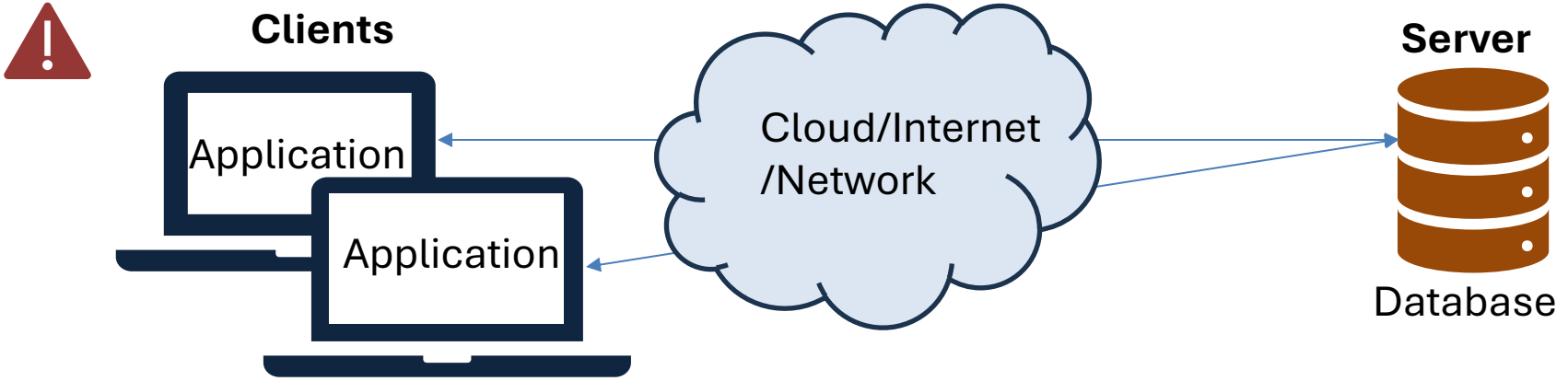
Web API

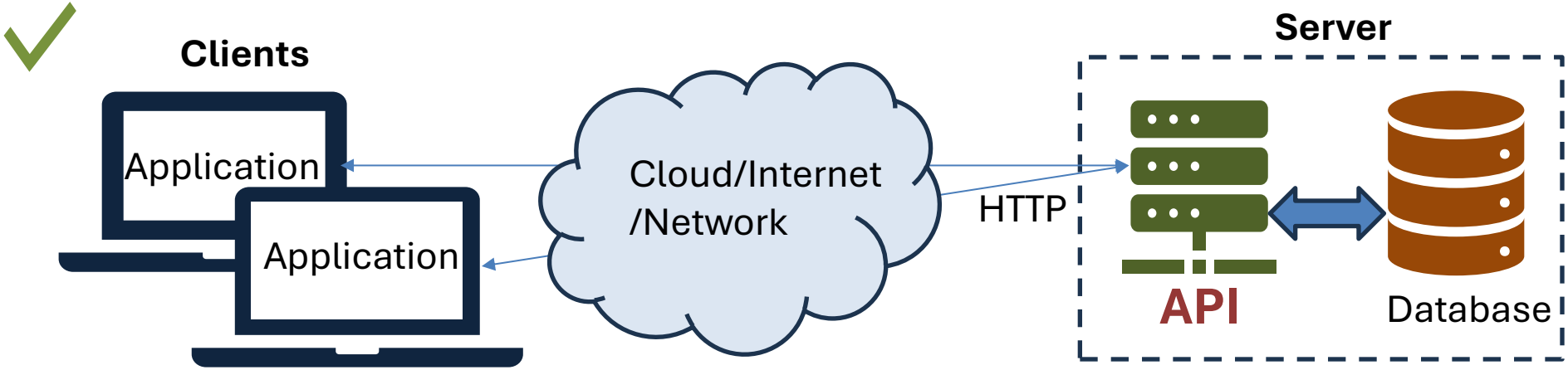
- We can create/use APIs for internal use inside an Application or between 2 or more Applications.
- Basically, an API can be just a Class with Methods that you use several places inside an Application or that you share between multiple Applications.
- A set of Stored Procedures in a Database can also be an API.
- When the Application that consume/use the API is on a local PC and the API itself is located on a Server, we can talk about so-called “Web APIs”.
- Such Web APIs also very often perform CRUD operations against a Database located on the Web.
- Normally it is not allowed to connect directly to a Database located in the Cloud from a local computer unless you configure and give access to the IP addresses for those clients.

CRUD: **C**reate, **R**ead, **U**pdate, **D**elete Data

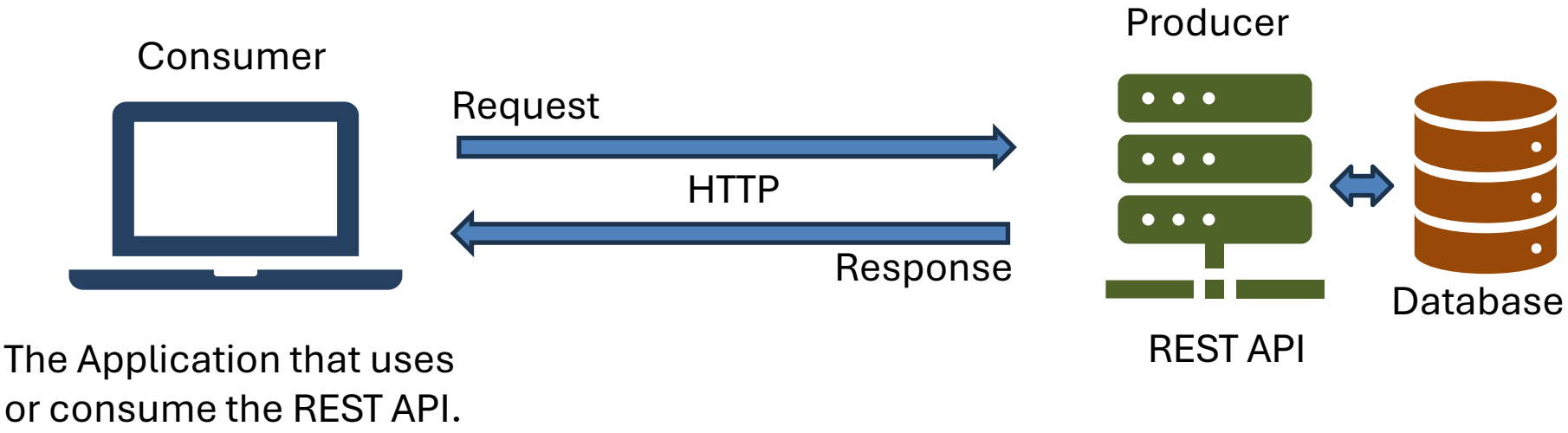
Web/REST API

Normally it is not allowed to connect directly to a Database located in the Cloud from a local computer unless you configure and give access to the IP addresses for those clients.





REST API

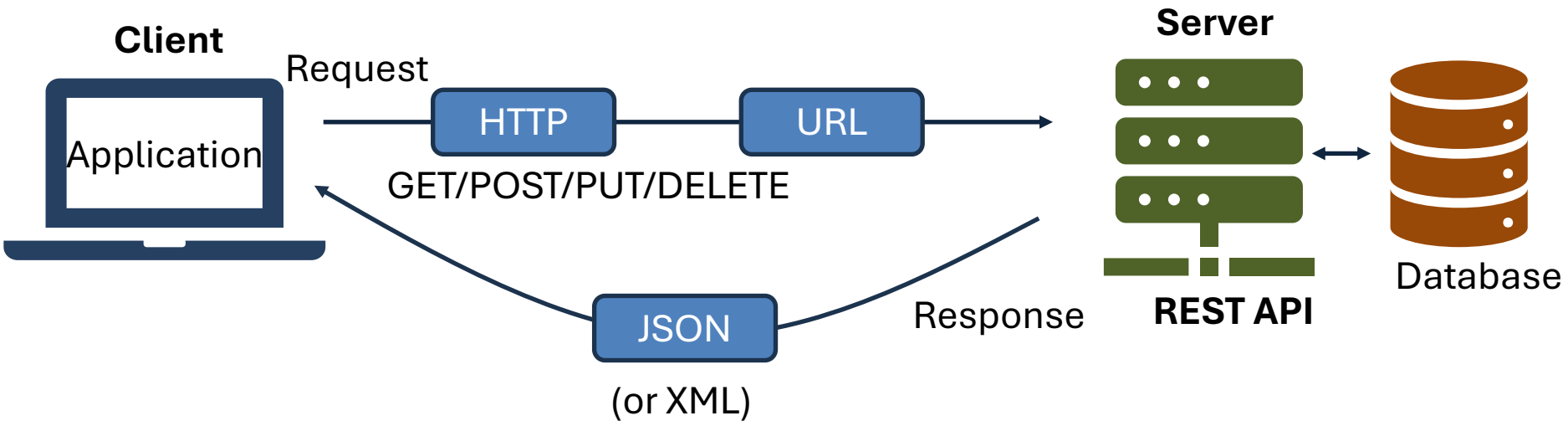

Consumer
Request
HTTP
Response
Producer
Database
REST API
The Application that uses
or consume the REST API.

HTTP/HTTPS

- HTTPS is not a separate protocol, but a combination of regular HTTP over an encrypted SSL (Secure Sockets Layer) or TLS (Transport Layer Security) connection.
- HTTP consists of different methods:
  - **GET** – This method is used to retrieve information from the server.
  - **POST** – This is used to send data to the server. Typically used to store data from a web page (an HTTML Form) to ,e.g., a database.
  - **PUT** – This is used to update information on the server.
  - **DELETE** – This is used to delete information on the server.
- You usually refer to these four methods as CRUD operations because they allow you to Create (POST), Read (GET), Update (PUT), and Delete (DELETE) resources, such as information in a database.

GET and **POST** are by far the most used of these HTTP methods

REST API


Client
Application
Request
HTTP
URL
GET/POST/PUT/DELETE
JSON
(or XML)
Response
Server
REST API
Database

JSON

- When it comes to Web APIs and REST APIs JSON is the standard for the data format.
- Example:

```
{
  "Name": "John Wayne",
  "Work": "Actor",
  "Age": 52
  "Children": [
    "Lisa",
    "Thomas",
    "Knut"
  ]
}
```

https://en.wikipedia.org/wiki/JSON

Why use an API?

- Normally it is not allowed to connect directly to a Database located in the Cloud from a local computer
  - unless you configure and give access to the IP addresses for those clients.
  - Typically, your IT Department don't allow that
- You can use the same API for multiple Applications, let say you have a Desktop App, an iPhone App and an Android App
  - All can use the same API
  - You save time and money by developing only once instead of specific code for each application.
- You want to expose data to externals services or persons, e.g., a Weather API that can be used by persons, external apps or services. Other examples: Hotel/plane reservations and ticket systems

API Summary

- “Web APIs”, “REST APIs” or “HTTP APIs” are basically the same.
- It is more or less just different names for the same.
- They use the request/response model.
- They all communicate via Internet and use HTTP as communication protocol.
- And they use JSON (or sometimes XML) as Data Format.
- Very often they implement CRUD functionality.

API Test Tools

- Postman. Homepage: https://www.postman.com
- Insomnia. Homepage: https://insomnia.rest

References

- **Tutorial: Create a minimal API with ASP.NET Core:** https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api
- Build a web API with minimal API, ASP.NET Core, and .NET: https://learn.microsoft.com/en-gb/training/modules/build-web-api-minimal-api/
- Back-end Web Development with .NET for Beginners: https://www.youtube.com/playlist?list=PLdo4fOcmZ0oWunQnm3WnZxJrseIw2zSAk
- Use .http files in Visual Studio 2022: https://learn.microsoft.com/en-us/aspnet/core/test/http-files

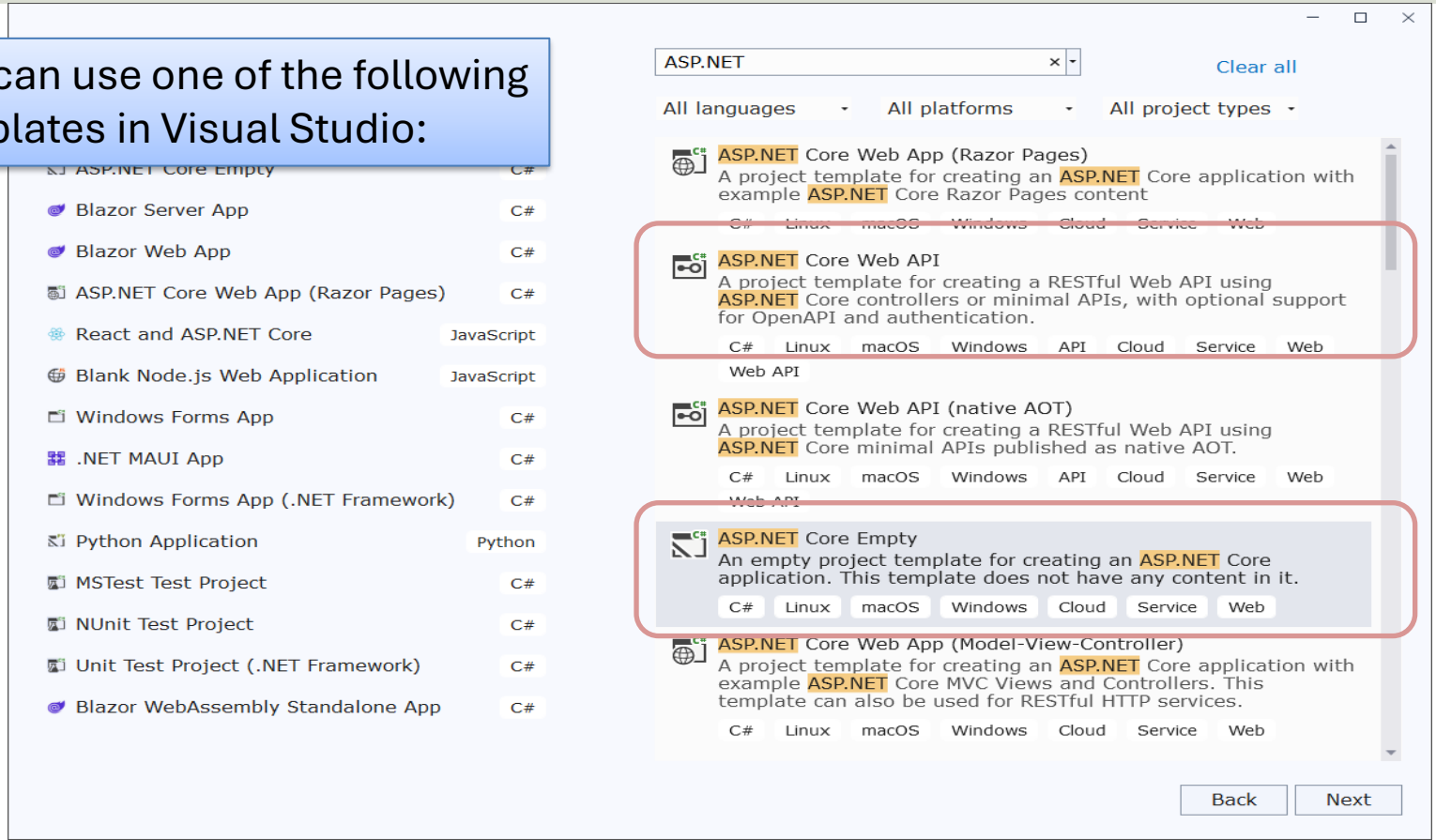
Getting Started

Hans-Petter Halvorsen

Table of Contents

Visual Studio Template

You can use one of the following templates in Visual Studio:



Visual Studio Project

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search WebAPI

Debug Any CPU https GitHub Copilot

Endpoints Explorer

- WebAPI (1)
  - GET /

Program.cs

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

100 % No issues found Ln: 4 Ch: 39 SPC CRLF

Error List

Entire Solution | 0 Errors | 0 Warnings | 0 Messages | Build + IntelliSe | Search Error List

Code	Description	Project	File	L...

Error List | Output

Solution Explorer

Search Solution Explorer (Ctrl+¨)

- Solution 'WebAPI' (1 of 1 project)
  - WebAPI
    - Connected Services
    - Dependencies
    - Properties
      - launchSettings.json
    - appsettings.json
      - appsettings.Development.json
    - Program.cs

Properties

Program.cs File Properties

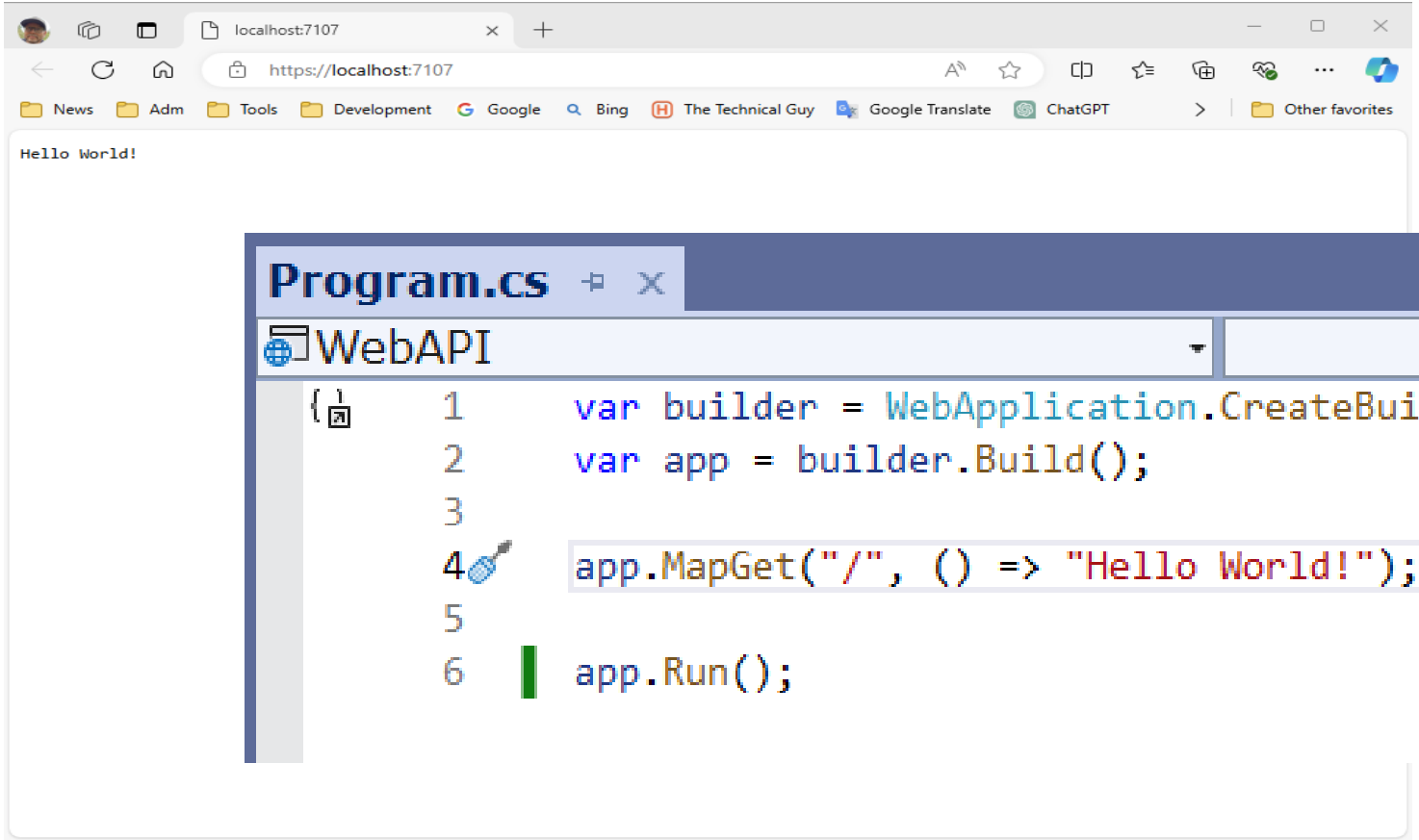
Build Action	C# compiler
Copy to Output Dire	Do not copy
Custom Tool	
Custom Tool Namesp	
File Name	Program.cs
Full Path	C:\Temp\WebAPI\WebA

Build Action
How the file relates to the build and depl...

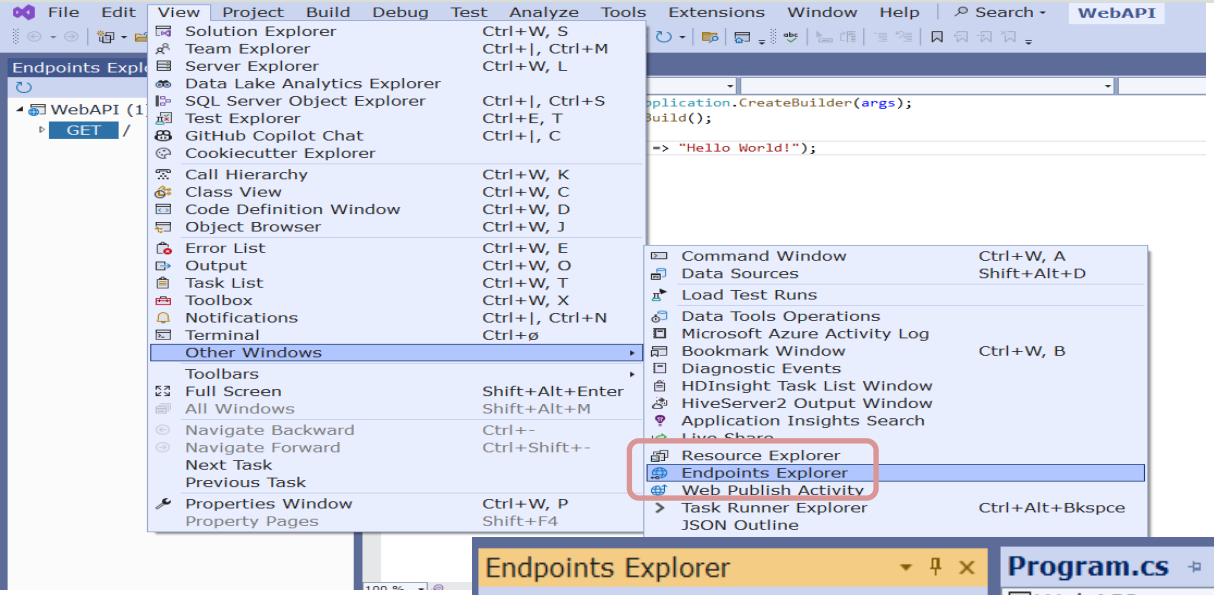
Notifications

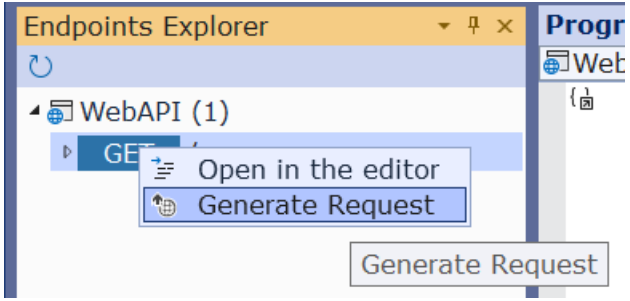
Ready Add to Source Control

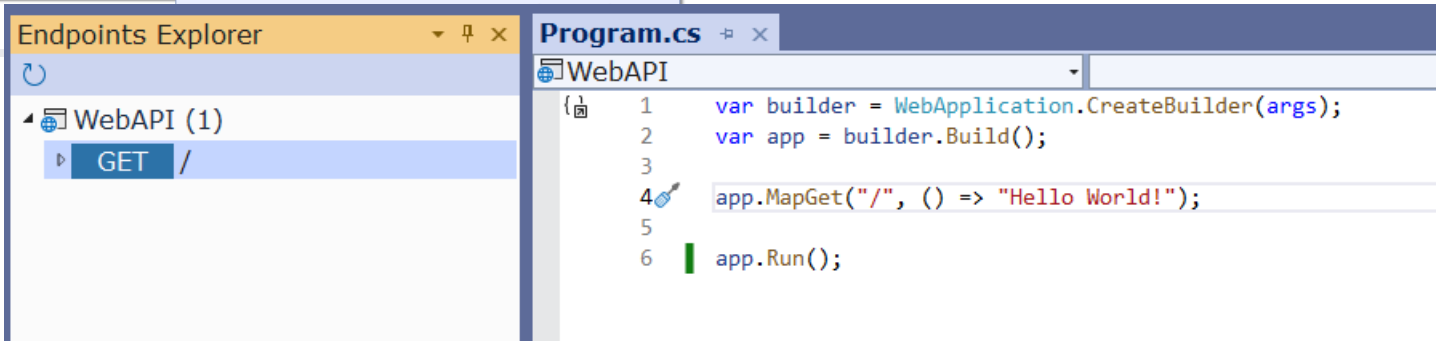
Hello World


localhost:7107
https://localhost:7107
News
Adm
Tools
Development
Google
Bing
The Technical Guy
Google Translate
ChatGPT
Other favorites
Hello World!
Program.cs
WebAPI
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.MapGet("/", () => "Hello World!");
app.Run();

Endpoints Explorer







Test the API Endpoints

Endpoints Explorer

WebAPI (1)

GET /

Open in the editor

Generate Request

Generate Request

WebAPI.http

GET {{WebAPI_HostAddress}}/

```
@WebAPI_HostAddress = https://localhost:7107

Send request | Debug
GET {{WebAPI_HostAddress}}/

###
```

WebAPI.http

GET {{WebAPI_HostAddress}}/

```
@WebAPI_HostAddress = https://localhost:7107

Send request | Debug
GET {{WebAPI_HostAddress}}/

###
```

Status: 200 OK Time: 104,45 ms Size: 12 bytes

Formatted Raw Headers Request

Body

text/plain; charset=utf-8, 12 bytes

```
Hello World!
```

CRUD Example

Hans-Petter Halvorsen

Table of Contents

CRUD API Example

- We will create a Web/REST/HTTP API with CRUD functionality.
  - Meaning we will Insert (**C**reate), **R**ead, **U**pdate and **D**elete data in a Database.
- We will start by creating a Database and Table using SQL Server and SQL Server Management Studio.
- Then we will create the ASP.NET/C# code for the Web API.
- We will test the API using the Endpoints Explorer in Visual Studio.

Tools

The following tools will be used in this example:

- SQL Server
  - SQL Server Management Studio
- Visual Studio
- ASP.NET
- C#

Database

Hans-Petter Halvorsen

Table of Contents

CRUD and Database

- We will create an API with CRUD functionality
- We will implement a minimal CRUD API that Create, Read, Update and Delete data in the Database.
- We will use SQL Server as the Database system.

Database

```
CREATE TABLE [AUTHOR]
(
    [AuthorId] [int] IDENTITY(1, 1) NOT
    [AuthorName] [varchar](50) NOT NULL
    [Address] [varchar](50) NULL,
    [Phone] [varchar](50) NULL,
    [PostCode] [varchar](50) NULL,
    [PostAddress] [varchar](50) NULL,
)
```

```
CREATE TABLE [PUBLISHER]
(
    [PublisherId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [PublisherName] [varchar](50) NOT NULL UNIQUE,
    [Description] [varchar](1000) NULL,
    [Address] [varchar](50) NULL,
    [Phone] [varchar](50) NULL,
    [PostCode] [varchar](50) NULL,
    [PostAddress] [varchar](50) NULL,
    [EMail] [varchar](50) NULL,
)
```

```
CREATE TABLE [BOOK]
(
    [BookId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [Title] [varchar](50) NOT NULL UNIQUE,
    [ISBN] [varchar](20) NOT NULL,
    [PublisherId] [int] NOT NULL F
    [AuthorId] [int] NOT NULL FORE
    [CategoryId] [int] NOT NULL FO
    [Description] [varchar](1000)
    [Year] [date] NULL,
    [Edition] [int] NULL,
    [AverageRating] [float] NULL,
)
```

```
CREATE TABLE [CATEGORY]
(
    [CategoryId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [CategoryName] [varchar](50) NOT NULL UNIQUE,
    [Description] [varchar](1000) NULL,
)
```

Visual Studio

Hans-Petter Halvorsen

Table of Contents

Visual Studio

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search MinimalBookWebAPI

Debug Any CPU https GitHub Copilot

Endpoints Explorer

- MinimalBookWebAPI (6)
  - GET /
  - GET /book/{id}
  - DELETE /book/{id}
  - PUT /book{id}
  - GET /books
  - POST /newbook

Book.cs | **Program.cs**

MinimalBookWebAPI

```
16
17  app.MapGet("/book/{id}", (int id) =>
18  {
19      Book book = new Book();
20
21      book = book.GetBookData(id);
22      return book;
23  });
24
25
26  app.MapPost("/newbook", (Book book) =>
27  {
28      book.CreateBook(book);
29  });
30
31
32  app.MapPut("/book{id}", (int id, Book book) =>
33  {
34      book.EditBook(id, book);
35  });
36
37
38  app.MapDelete("/book/{id}", (int id) =>
39  {
40      Book book = new Book();
41
42      book.DeleteBook(id);
43  });
44
45
46
47  app.MapGet("/", () => "Welcome to Minimal Book Web API");
48
49  app.Run();
```

100 % No issues found Ln: 34 Ch: 29 SPC CRLF

Error List

Entire Solution | 0 Errors | 0 of 3 Warnings | 0 of 27 Messages | Build + IntelliSe | Search Error List

Code	Description	Project	File	L...	Suppression...

Error List | Output

Solution Explorer

Search Solution Explorer (Ctrl+¨)

- Solution 'MinimalBookWebAPI' (1 of 1 proj
  - MinimalBookWebAPI
    - Connected Services
    - Dependencies
    - Properties
    - Models
      - Book.cs
    - appsettings.json
    - Program.cs

Properties

Notifications | Diagnostic Tools

Ready Add to Source Control

CRUD Database Class

Book.cs

MinimalBookWebAPI | MinimalBookWebAPI.Models.Book | BookId

Books.cs

```
using System;
using System.Data;
using System.Collections.Generic;
using Microsoft.Data.SqlClient;

namespace MinimalBookWebAPI.Models
{
    20 references
    public class Book
    {
        2 references
        public int BookId { get; set; }
        4 references
        public string? Title { get; set; }
        4 references
        public string? Isbn { get; set; }
        4 references
        public string? PublisherName { get; set; }
        4 references
        public string? AuthorName { get; set; }
        4 references
        public string? CategoryName { get; set; }

        private string connectionString = "Data Source=XPS15HPH\\SQLEXPRESS;Initial Catalog=BOOKS;Integrated Security=True; TrustServerCertificate=True";


        1 reference
        public List<Book> GetBooks()...

        1 reference
        public Book GetBookData(int bookId)...

        1 reference
        public void CreateBook(Book book)...

        1 reference
        public void EditBook(int bookId, Book book)...

        1 reference
        public void DeleteBook(int bookId)...

    }
}
```

GetBooks

Books.cs

```
public List<Book> GetBooks()
{
    List<Book> bookList = new List<Book>();

    SqlConnection con = new SqlConnection(connectionString);
    string selectSQL = "select BookId, Title, Isbn, PublisherName, AuthorName, CategoryName  from GetBookData";
    con.Open();
    SqlCommand cmd = new SqlCommand(selectSQL, con);
    SqlDataReader dr = cmd.ExecuteReader();

    if (dr != null)
    {
        while (dr.Read())
        {
            Book book = new Book();
            book.BookId = Convert.ToInt32(dr["BookId"]);
            book.Title = dr["Title"].ToString();
            book.Isbn = dr["ISBN"].ToString();
            book.PublisherName = dr["PublisherName"].ToString();
            book.AuthorName = dr["AuthorName"].ToString();
            book.CategoryName = dr["CategoryName"].ToString();
            bookList.Add(book);
        }
    }
    return bookList;
}
```

GetBookData

Books.cs

```
public Book GetBookData(int bookId)
{

    SqlConnection con = new SqlConnection(connectionString);
    string selectSQL = "select BookId, Title, Isbn, PublisherName, AuthorName, CategoryName  from GetBookData where BookId = " +
                           bookId;
    con.Open();
    SqlCommand cmd = new SqlCommand(selectSQL, con);
    SqlDataReader dr = cmd.ExecuteReader();

    Book book = new Book();

    if (dr != null)
    {
        while (dr.Read())
        {
            book.BookId = Convert.ToInt32(dr["BookId"]);
            book.Title = dr["Title"].ToString();
            book.Isbn = dr["ISBN"].ToString();
            book.PublisherName = dr["PublisherName"].ToString();
            book.AuthorName = dr["AuthorName"].ToString();
            book.CategoryName = dr["CategoryName"].ToString();
        }
    }
    return book;
}
```

CreateBook

Books.cs

```
public void CreateBook(Book book)
{
    try
    {
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            SqlCommand cmd = new SqlCommand("CreateBook", con);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.Add(new SqlParameter("@Title", book.Title));
            cmd.Parameters.Add(new SqlParameter("@Isbn", book.Isbn));
            cmd.Parameters.Add(new SqlParameter("@PublisherName", book.PublisherName));
            cmd.Parameters.Add(new SqlParameter("@AuthorName", book.AuthorName));
            cmd.Parameters.Add(new SqlParameter("@CategoryName", book.CategoryName));

            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

EditBook

Books.cs

```
public void EditBook(int bookId, Book book)
{
    try
    {
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            SqlCommand cmd = new SqlCommand("UpdateBook", con);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.Add(new SqlParameter("@BookId", bookId));
            cmd.Parameters.Add(new SqlParameter("@Title", book.Title));
            cmd.Parameters.Add(new SqlParameter("@Isbn", book.Isbn));
            cmd.Parameters.Add(new SqlParameter("@PublisherName", book.PublisherName));
            cmd.Parameters.Add(new SqlParameter("@AuthorName", book.AuthorName));
            cmd.Parameters.Add(new SqlParameter("@CategoryName", book.CategoryName));

            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

DeleteBook

Books.cs

```
public void DeleteBook(int bookId)
{
    try
    {
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            SqlCommand cmd = new SqlCommand("DeleteBook", con);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.Add(new SqlParameter("@BookId", bookId));


            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Program.cs

Endpoints Explorer

- MinimalBookWebAPI (6)
  - GET /
  - PUT /book/{id}
  - DELETE /book/{id}
  - GET /books
  - GET /books/{id}
  - POST /newbook

Program.cs | MinimalB...API.http

MinimalBookWebAPI

```
using MinimalBookWebAPI.Models;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/books", () =>
{
    List<Book> bookList = new List<Book>();
    Book book = new Book();

    bookList = book.GetBooks();
    return bookList;
});

app.MapGet("/books/{id}", (int id) =>
{
    Book book = new Book();

    book = book.GetBookData(id);
    return book;
});

app.MapPost("/newbook", (Book book) =>
{
    book.CreateBook(book);

    return "Book has been created";
});

app.MapPut("/book/{id}", (int id, Book book) =>
{
    book.EditBook(id, book);
    return "Book has been updated";
});

app.MapDelete("/book/{id}", (int id) =>
{
    Book book = new Book();

    book.DeleteBook(id);

    return "Book has been deleted";
});

app.MapGet("/", () => "Welcome to Minimal Book Web API");

app.Run();
```

Program.cs

```
using MinimalBookWebAPI.Models;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/books", () =>
{
    List<Book> bookList = new List<Book>();
    Book book = new Book();
    bookList = book.GetBooks();
    return bookList;
});

app.MapGet("/books/{id}", (int id) =>
{
    Book book = new Book();
    book = book.GetBookData(id);
    return book;
});

app.MapPost("/newbook", (Book book) =>
{
    book.CreateBook(book);

    return "Book has been created";
});

app.MapPut("/book/{id}", (int id, Book book) =>
{
    book.EditBook(id, book);
    return "Book has been updated";
});

app.MapDelete("/book/{id}", (int id) =>
{
    Book book = new Book();
    book.DeleteBook(id);
    return "Book has been deleted";
});

app.MapGet("/", () => "Welcome to Minimal Book Web API");

app.Run();
```

Testing the API functions

- We will test the different API functions
- We can test the functions using the Endpoints Explorer in Visual Studio
- We can test it using the Web Browser
- We can also test it using an AI tool like Postman, etc.
- We can test it using different programming languages, like Python, etc.

GET /books

localhost:7277/books

https://localhost:7277/books

News | Adm | Tools | Development | Google | Bing | The Technical Guy | Other favorites


```
[
    {
        "bookId": 3,
        "title": "The Lord of the Rings",
        "isbn": "2-09-066556-2",
        "publisherName": "McGraw-Hill",
        "authorName": "J.R.R Tolkien",
        "categoryName": "Novel"
    },
    {
        "bookId": 4,
        "title": "Arduino",
        "isbn": "12345678",
        "publisherName": "Wiley",
        "authorName": "Hans-Petter",
        "categoryName": "IoT"
    },
    {
        "bookId": 6,
        "title": "Python Programming",
        "isbn": "12345678",
        "publisherName": "Wiley",
        "authorName": "Hans-Petter",
        "categoryName": "Programming"
    }
]
```


We can test the GET method in the Web Browser

GET /books

Minimal...API.http | Program.cs

GET {{MinimalBookWebAPI_HostAddress}}/books | env: <none>

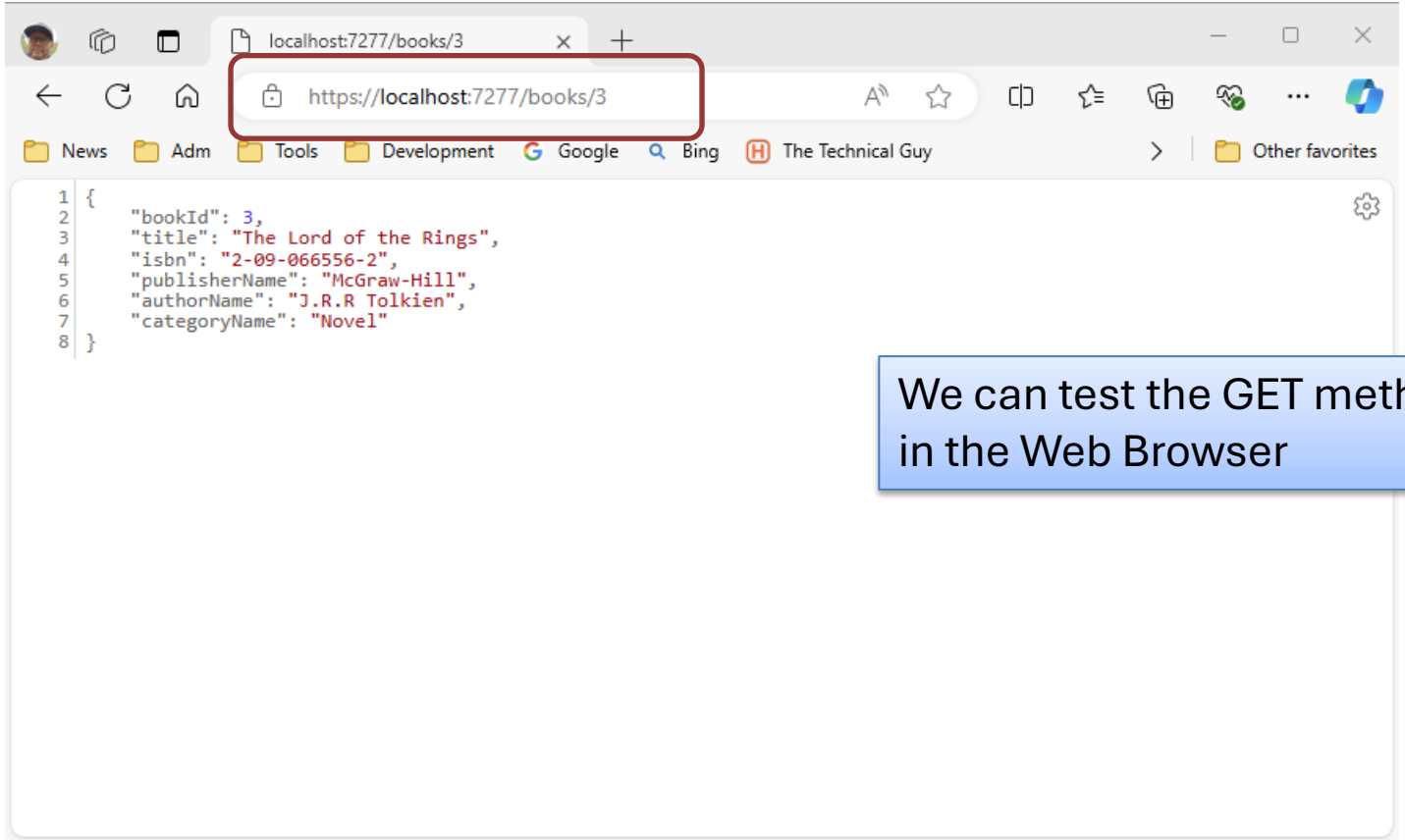
```
@MinimalBookWebAPI_HostAddress = https://localhost:7277

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books

###

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books/3

###

Send request | Debug
POST {{MinimalBookWebAPI_HostAddress}}/newbook

###

Send request | Debug
PUT {{MinimalBookWebAPI_HostAddress}}/book{id}

###

Send request | Debug
DELETE {{MinimalBookWebAPI_HostAddress}}/book/{id}

###
```

We can test the method using the **Endpoints Explorer** in Visual Studio

application/json; charset=utf-8, 412 bytes


```
[
  {
    "bookId": 3,
    "title": "The Lord of the Rings",
    "isbn": "2-09-066556-2",
    "publisherName": "McGraw-Hill",
    "authorName": "J.R.R Tolkien",
    "categoryName": "Novel"
  },
  {
    "bookId": 4,
    "title": "Arduino",
    "isbn": "12345678",
    "publisherName": "Wiley",
    "authorName": "Hans-Petter",
    "categoryName": "IoT"
  },
  {
    "bookId": 6,
    "title": "Python Programming",
    "isbn": "12345678",
    "publisherName": "Wiley",
    "authorName": "Hans-Petter",
    "categoryName": "Programming"
  }
]
```

GET /books/{id}



We can test the GET method in the Web Browser

GET /books/{id}

Minimal...API.http | Program.cs

GET {{MinimalBookWebAPI_HostAddress}}/books | env: <none>

```
@MinimalBookWebAPI_HostAddress = https://localhost:7277

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books

###

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books/3

###

Send request | Debug
POST {{MinimalBookWebAPI_HostAddress}}/newbook

###

Send request | Debug
PUT {{MinimalBookWebAPI_HostAddress}}/book{id}

###

Send request | Debug
DELETE {{MinimalBookWebAPI_HostAddress}}/book/{id}

###
```

Status: 200 OK Time: 12,96 ms Size: 149 bytes

Formatted Raw Headers Request

Body

application/json; charset=utf-8, 149 bytes


```
{
  "bookId": 3,
  "title": "The Lord of the Rings",
  "isbn": "2-09-066556-2",
  "publisherName": "McGraw-Hill",
  "authorName": "J.R.R Tolkien",
  "categoryName": "Novel"
}
```


We can test the method using the **Endpoints Explorer** in Visual Studio

POST /newbook

Program.cs | Minimal...API.http

DELETE {{MinimalBookWebAPI_HostAddress}}/book/6 | env: <none>

```
Send request | Debug
POST {{MinimalBookWebAPI_HostAddress}}/newbook
Content-Type: application/json
{
    "Title" : "API Book",
    "Isbn" : "12345678",
    "PublisherName" : "Wiley",
    "AuthorName" : "Knut Hamsun",
    "CategoryName" : "Science"
}

###

Send request | Debug
PUT {{MinimalBookWebAPI_HostAddress}}/book/8
Content-Type: application/json
{
    "Title" : "API Book2",
    "Isbn" : "5555555",
    "PublisherName" : "Wiley",
    "AuthorName" : "Knut Hamsun",
    "CategoryName" : "Science"
}

###

Send request | Debug
DELETE {{MinimalBookWebAPI_HostAddress}}/book/6

###
```

Status: 200 OK Time: 82,67 ms Size: 21 bytes

Formatted Raw Headers Request

Body

text/plain; charset=utf-8, 21 bytes

```
Book has been created
```

We can test the method using the **Endpoints Explorer** in Visual Studio

PUT /book/{id}

Program.cs | Minimal...API.http

DELETE {{MinimalBookWebAPI_HostAddress}}/book/6 | env: <none>

```
Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books/3

###

Send request | Debug
POST {{MinimalBookWebAPI_HostAddress}}/newbook
Content-Type: application/json
{
    "Title" : "API Book",
    "Isbn" : "12345678",
    "PublisherName" : "Wiley",
    "AuthorName" : "Knut Hamsun",
    "CategoryName" : "Science"
}

###

Send request | Debug
PUT {{MinimalBookWebAPI_HostAddress}}/book/8
Content-Type: application/json
{
    "Title" : "API Book2",
    "Isbn" : "5555555",
    "PublisherName" : "Wiley",
    "AuthorName" : "Knut Hamsun",
    "CategoryName" : "Science"
}

###
```

Status: 200 OK Time: 7,47 ms Size: 21 bytes

Formatted Raw Headers Request

Body

text/plain; charset=utf-8, 21 bytes

```
Book has been updated
```

We can test the method using the **Endpoints Explorer** in Visual Studio

DELETE /book/{id}

Minimal...PI.http* | Program.cs

DELETE {{MinimalBookWebAPI_HostAddress}}/book/6 | env: <none>

```
@MinimalBookWebAPI_HostAddress = https://localhost:7277

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books

###

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books/3

###

Send request | Debug
POST {{MinimalBookWebAPI_HostAddress}}/newbook

###

Send request | Debug
PUT {{MinimalBookWebAPI_HostAddress}}/book{id}

###

Send request | Debug
DELETE {{MinimalBookWebAPI_HostAddress}}/book/6

###
```

Status: 200 OK Time: 111,51 ms Size: 0 bytes

Formatted Raw Headers Request

Body

0 bytes

We see Book with BookId=6 has been deleted

Minimal...PI.http* | Program.cs

DELETE {{MinimalBookWebAPI_HostAddress}}/book/6 | env: <none>

```
@MinimalBookWebAPI_HostAddress = https://localhost:7277

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books

###

Send request | Debug
GET {{MinimalBookWebAPI_HostAddress}}/books/3

###

Send request | Debug
POST {{MinimalBookWebAPI_HostAddress}}/newbook

###

Send request | Debug
PUT {{MinimalBookWebAPI_HostAddress}}/book{id}

###

Send request | Debug
DELETE {{MinimalBookWebAPI_HostAddress}}/book/6

###
```

Status: 200 OK Time: 2,78 ms Size: 272 bytes

Formatted Raw Headers Request

Body

application/json; charset=utf-8, 272 bytes


```
[
  {
    "bookId": 3,
    "title": "The Lord of the Rings",
    "isbn": "2-09-066556-2",
    "publisherName": "McGraw-Hill",
    "authorName": "J.R.R Tolkien",
    "categoryName": "Novel"
  },
  {
    "bookId": 4,
    "title": "Arduino",
    "isbn": "12345678",
    "publisherName": "Wiley",
    "authorName": "Hans-Petter",
    "categoryName": "IoT"
  }
]
```


We can test the method using the **Endpoints Explorer** in Visual Studio

Summary

- We have created a so-called Minimal API in Visual Studio using ASP.NET Core and C#.
- The API has basic CRUD functionality that Create, Read, Update and Delete data in the SQL Server database.
- We tested the API methods using the Endpoint Explorer in Visual Studio.
- The code is very simplified for showing the basic principles of creating such a CRUD API.

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog