REST based Web Services

WHAT IS REST?

- REST stands for **RE**presentational **S**tate **T**ransfer.
- In traditional Web architecture, clients (our browsers) connect to Web servers for web pages.
- In REST architecture, clients (browsers, mobile apps) connect to Web servers for **Resources**.
- **Resource** — a resource can be any object the server can provide information about. For example, a resource can be a user, a photo, a comment. Each resource has a unique identifier.
- Example: http://www.curtin.edu.au/cse/students
- When a RESTful API is called, the server will transfer to the client the representation of the state of the requested resource.

REST REQUEST - GET

- Use HTTP GET requests **to retrieve resource representation/ information only.**
- For any given HTTP GET API, if the resource is found on the server, then it must return HTTP response code 200 (OK) – along with the response body (XML/JSON)
- In case the resource is NOT found on the server then it must return HTTP response code 404 (NOT FOUND).
- Example: GET http://www.curtin.edu.au/cse/students

REST REQUEST - POST

- POST methods are used to **create a new resource** into the collection of resources.
- It could be a new file created on the web server, or a database insert operation.

```
POST /student HTTP/1.1
Host: curtin.edu.au Accept: application/json
Content-Type: application/json
Content-Length: 81
{
        "Id": 287489,
        “Name": "Sajib Mistry",
        ”Address": “Perth”,
}
```

REST REQUEST - PUT

- update an existing resource (if the resource does not exist, then API may decide to create a new resource or not).

```
PUT /student/ 287489 HTTP/1.1
Host: curtin.edu.au Accept: application/json
Content-Type: application/json
Content-Length: 61
{
        ”Address": “Sydney”,
}
```

REST REQUEST - DELETE

- Use **to delete resources** (identified by the Request-URI).

DELETE /student/ 287489 HTTP/1.1

Host: curtin.edu.au Accept: application/json
Content-Type: application/json

REST REQUEST - PATCH

- PATCH lets you update a specific item in the resource
- Does not create it if it does not exist, that’s an error.
- If you’re using a Replace to Update sort of approach, may not be different to POST

DEMO – Creating REST Web service using NODE.JS

Lets assume you want to create your REST server in the folder “Rest-Node”. Open terminal

Navigate to the folder: cd <path to folder>/Rest-Node

npm init. This command will create the package.json. Give description as wish. By default the main file is index.js

npm install express –save (Our middleware to save a lot of time)

npm install --save body-parser (Will help to parse HTTP request)

npm install --save cors (Will help to handle cross origin requests)

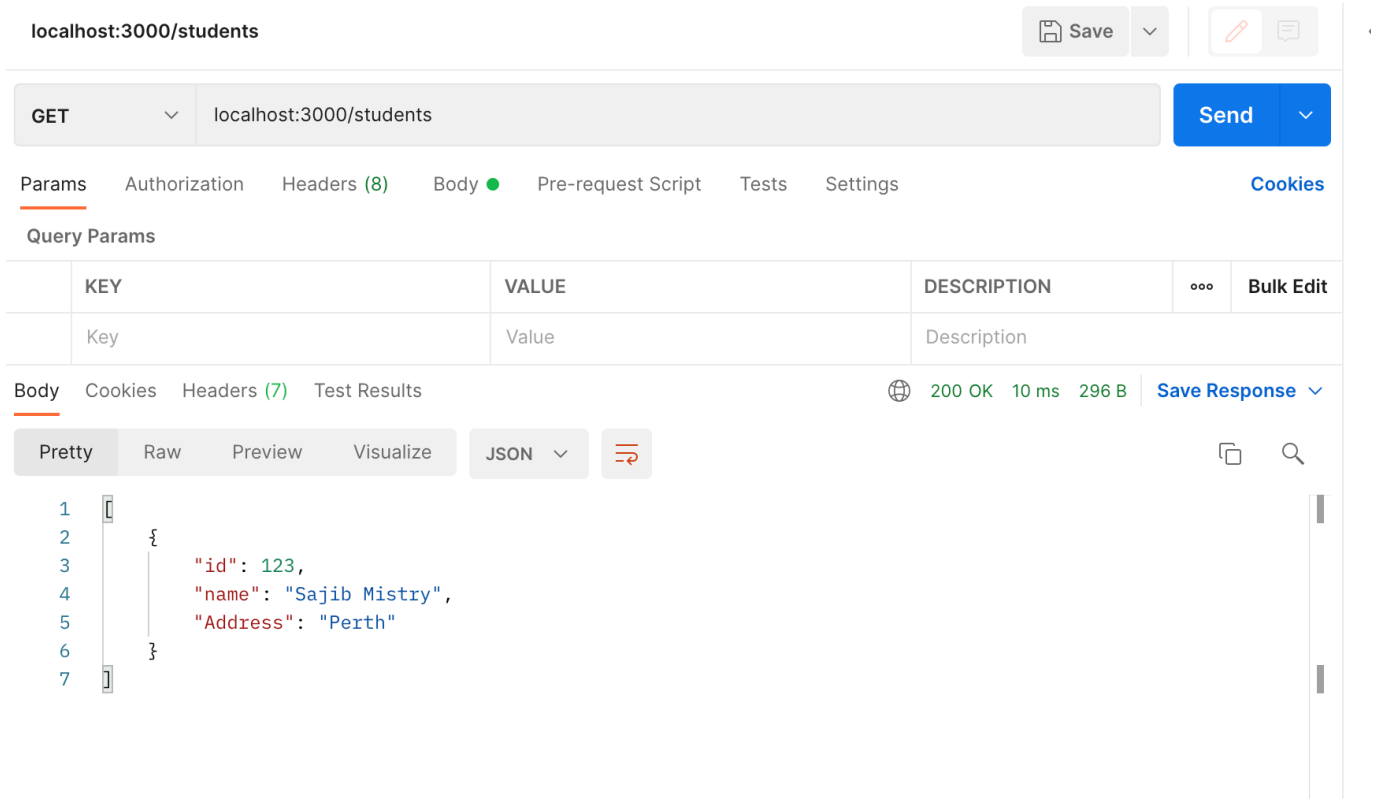
vi index.js (and copy the following code)

DEMO – Creating REST Web service using NODE.JS

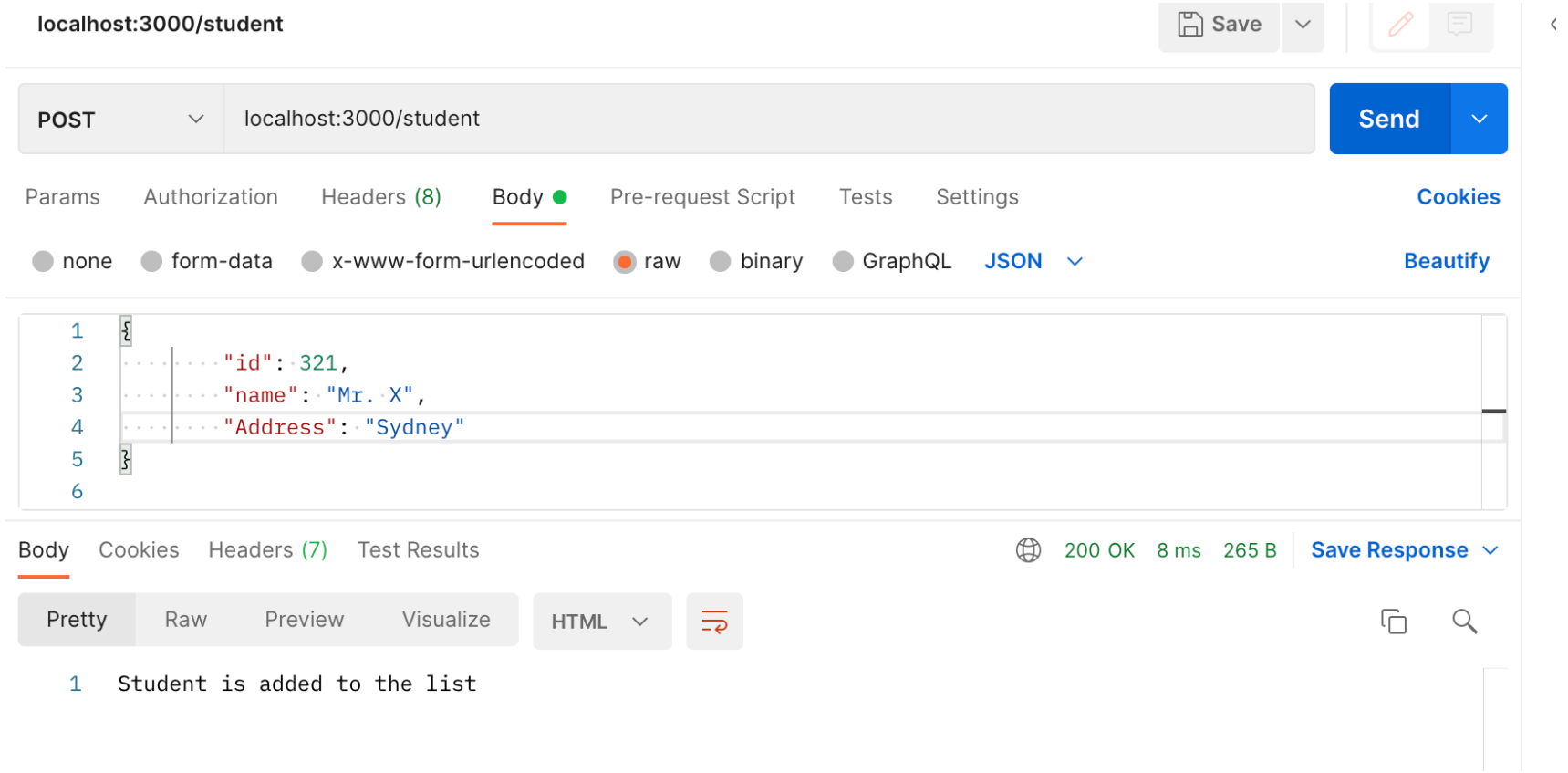
```
var express = require("express");
const bodyParser = require('body-parser');
const cors = require('cors');

var app = express();
app.use(cors());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

let students = [];
let student = {"id":123, "name": "Sajib Mistry", "Address": "Perth"};
students.push(student);

app.listen(3000, () => {
 console.log("Server running on port 3000");
});

app.get("/students", (req, res)=> {
    res.json(students);
});

app.post("/student", (req, res)=> {
    const newstudent = req.body;
    console.log(newstudent);
    students.push(newstudent);
    res.send('Student is added to the list');
});
```

DEMO – Creating REST Web service using NODE.JS

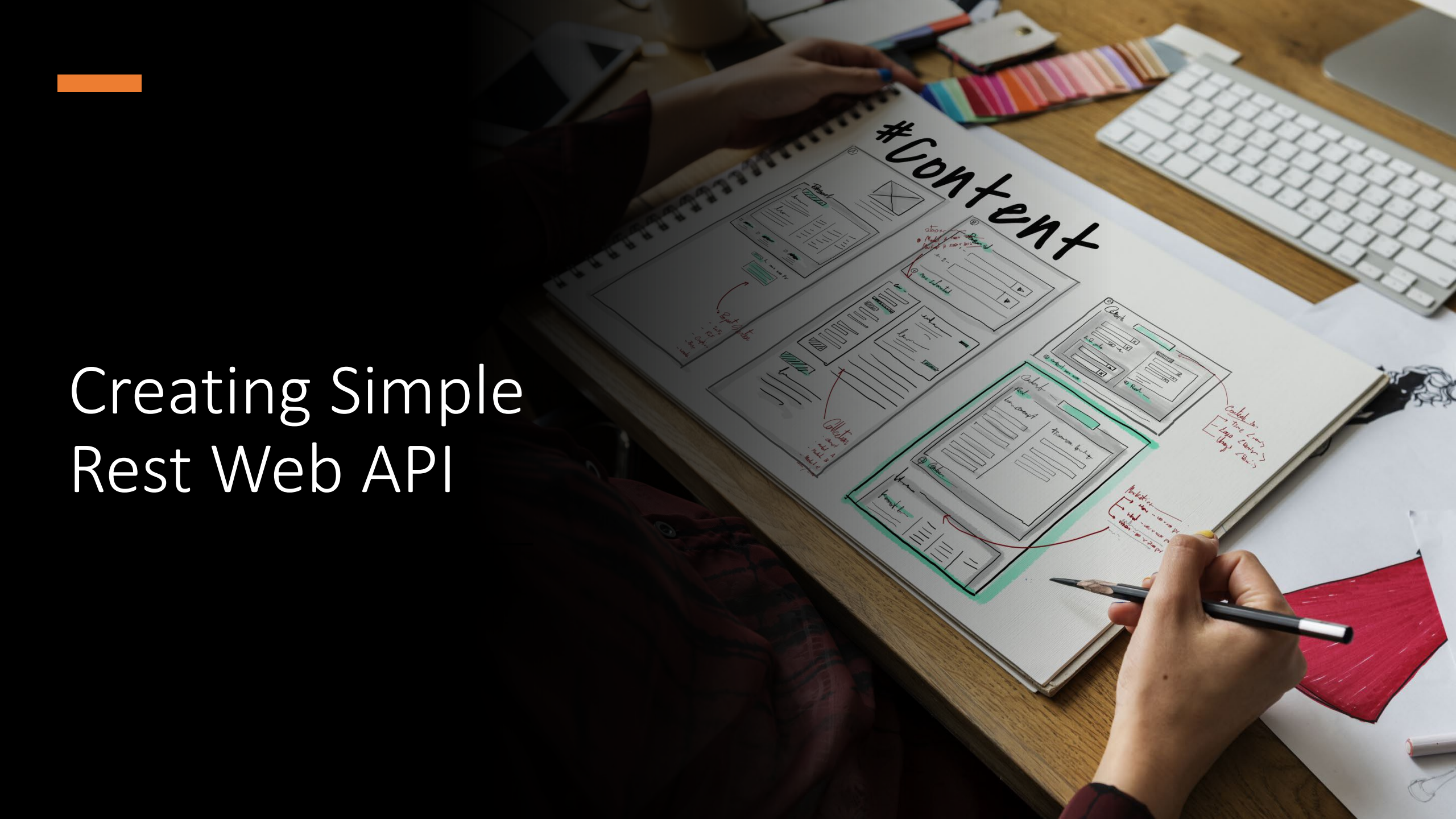
- Run server: node index.js
- Test GET in postman



DEMO – Creating REST Web service using NODE.JS



Creating Simple Rest Web API



Create a new project

asp.net

Clear all

C# | Windows | All project types

Recent project templates

- ASP.NET Core Web API — C#
- WPF App (.NET Framework) — C#
- Console App (.NET Framework) — C#
- Class Library (.NET Framework) — C#
- WPF Application — C#

An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

C# Linux macOS Windows Cloud Service Web

ASP.NET Core gRPC Service

A project template for creating a gRPC ASP.NET Core service.

C# Linux macOS Windows Cloud Service Web

ASP.NET Core with Angular

A project template for creating an ASP.NET Core application with Angular

C# Linux macOS Windows Cloud Service Web

ASP.NET Core with React.js

A project template for creating an ASP.NET Core application with React.js

C# Linux macOS Windows Cloud Service Web

Blazor Server App

Configure your new project

ASP.NET Core Web App (Model-View-Controller)

C# Linux macOS Windows Cloud Service Web

Project name

SimpleWebAPI

Location

C:\Users\283602K\Downloads\Core Web API\ ...

Solution

Create new solution

Solution name

SimpleWebAPI

Place solution and project in the same directory

Project will be created in "C:\Users\283602K\Downloads\Core Web API\SimpleWebAPI\SimpleWebAPI\"

Additional information

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Framework

.NET 6.0 (Long Term Support)

Authentication type

None

Configure for HTTPS

Enable Docker

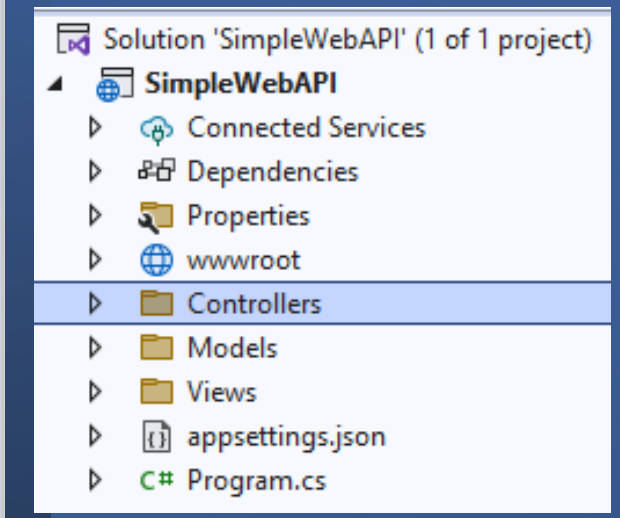
Docker OS

Linux

Do not use top-level statements

Uncheck. Since we are not managing HTTPS

Back Create



What is Model in MVC (Model-View-Controller)

The Model represents the application's data and business logic.

It is responsible for retrieving and storing data, as well as implementing the business rules and logic that manipulate the data.

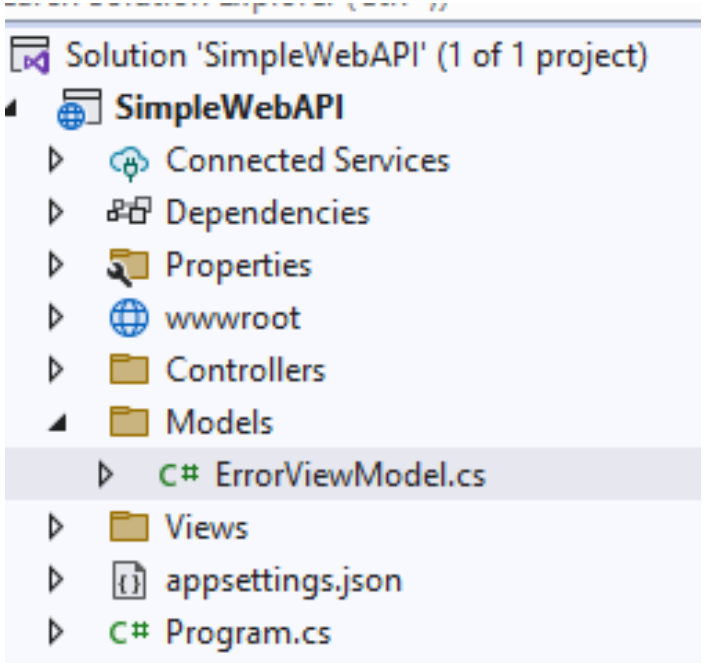
Models typically consist of classes that define the structure of your data entities, often interacting with databases or external data sources.

ASP.NET Core supports various data access technologies, such as Entity Framework Core, to work with databases seamlessly.

Getting used to the default files.. Default ErrorViewModel.cs

```
namespace SimpleWebAPI.Models
{
    4 references
    public class ErrorViewModel
    {
        3 references
        public string? RequestId { get; set; }

        1 reference
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

- RequestId { get; set; }: This line defines a public property named RequestId. The property has a getter and a setter. The type of the property is string?, where the ? indicates that the property can be nullable. This property is intended to hold a unique identifier for the user's request that resulted in an error. This identifier can help with debugging and tracking down specific issues.
- public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);: This is a computed property (read-only property with a calculated value) named ShowRequestId. It calculates its value based on whether the RequestId property is not null or empty. If RequestId has a value, ShowRequestId will be true, indicating that the identifier should be shown to the user. If RequestId is null or empty, ShowRequestId will be false, indicating that the identifier should not be shown.



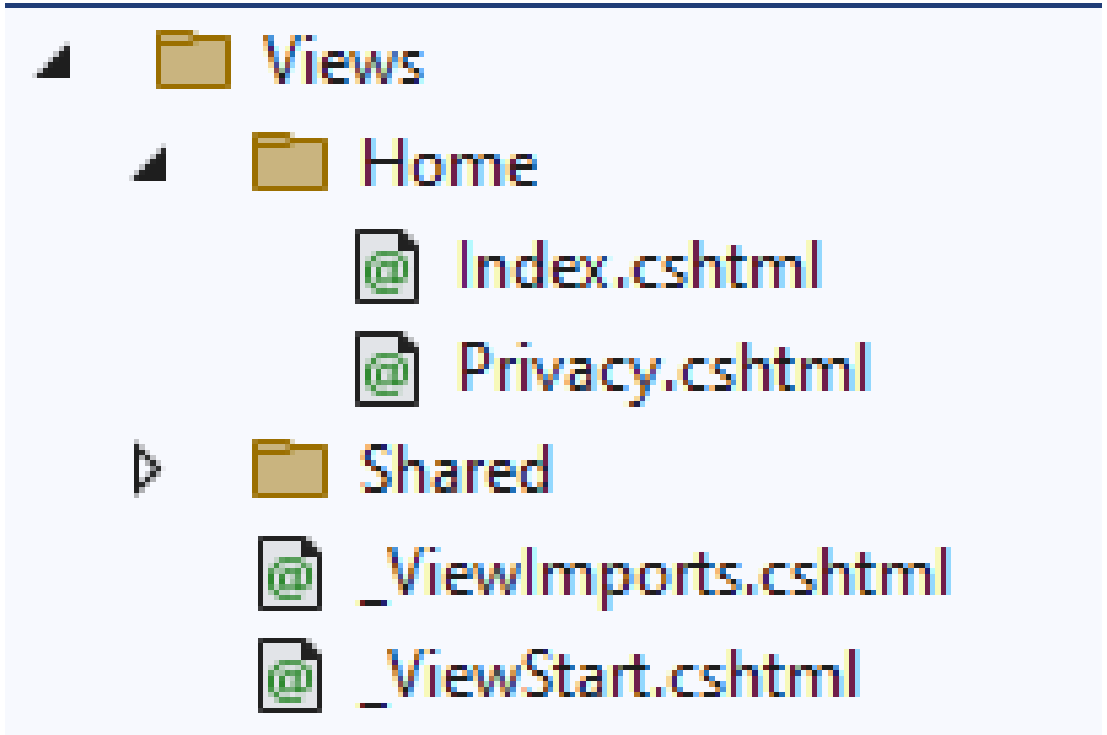
View in MVC

The View is responsible for presenting data to users.

It's the user interface that users interact with. Views are usually created using markup languages like HTML, along with embedded server-side code to render dynamic content.

In ASP.NET Core MVC, views are often associated with the Razor view engine, which combines HTML and C# code to create dynamic web pages.

Default View structure



- ‘Home’ folder means there is a controller named ‘HomeController’
- Index.cshtml means there is an action named Index in the HomeController
- Privacy.cshtml means that there is an action named Privacy in the HomeController
- The shared folder has views which are shared by the individual cshtml
- We are not working with view in this lecture since we are building a WebAPI.

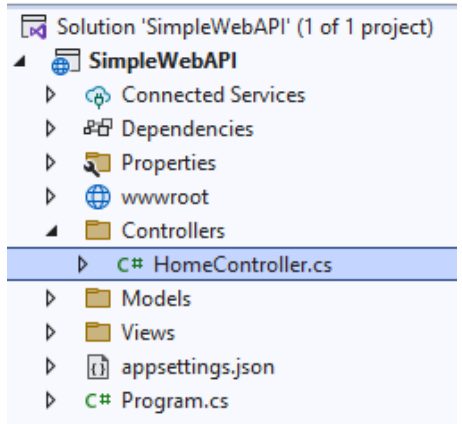
Controller in MVC

The Controller acts as an intermediary between the Model and the View.

It receives user requests, processes them, interacts with the Model to retrieve data, and then selects the appropriate View to render the response.

Controllers contain action methods that handle specific user interactions. These action methods are responsible for coordinating the data flow between the Model and the View

Default HomeController.cs



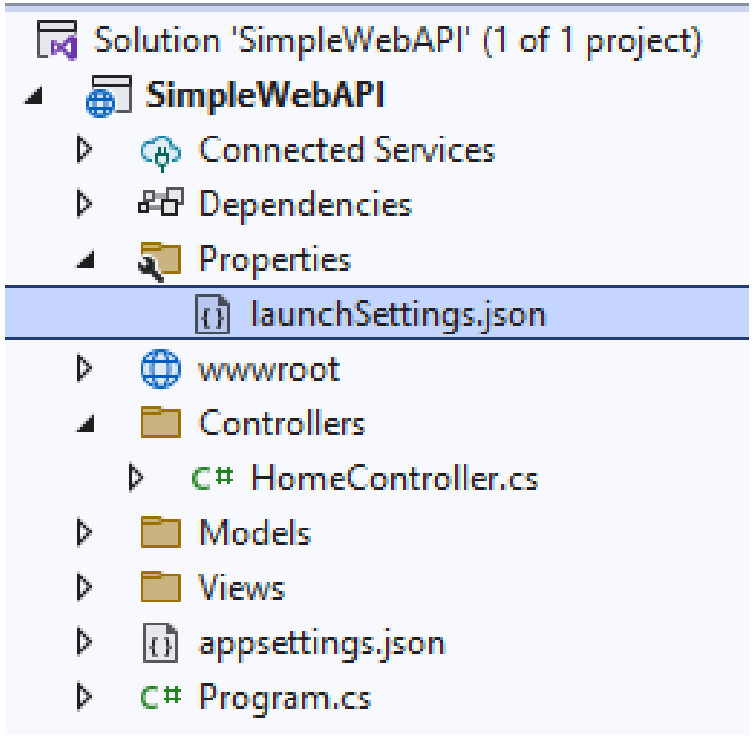
- Note, every controller has to end with “Controller” suffix. public IActionResult Index(): This method is an action method for the "Index" page. It returns an IActionResult, which is the base class for all possible action results. In this case, it returns the result of the View() method, indicating that the "Index" view should be rendered.
- public IActionResult Privacy(): Similar to the "Index" method, this action method returns the result of the View() method, indicating that the "Privacy" view should be rendered.

```
namespace SimpleWebAPI.Controllers
{
    3 references
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        0 references
        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        0 references
        public IActionResult Index()
        {
            return View();
        }

        0 references
        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        0 references
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

Properties – launchSettings.json

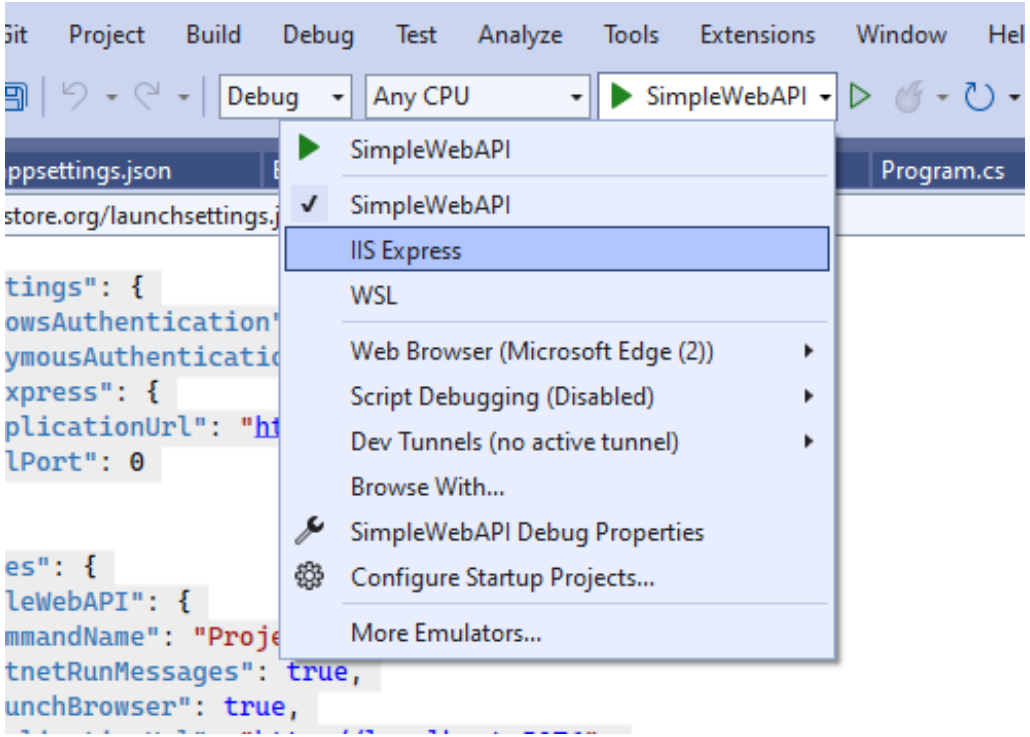
- This file is commonly found in ASP.NET Core projects and is used to configure how the application is launched during development, including various settings for debugging, environment variables, and more.

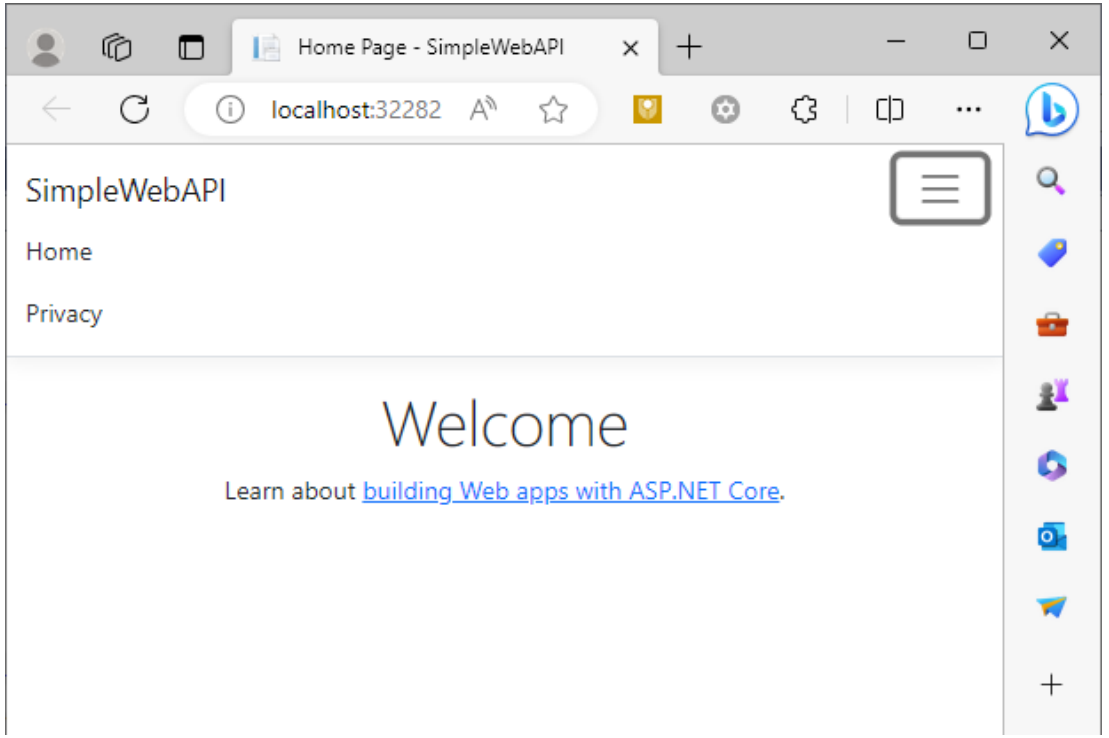
```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:32282",
      "sslPort": 0
    }
  },
  "profiles": {
    "SimpleWebAPI": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5076",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```



Running the App in IIS Express

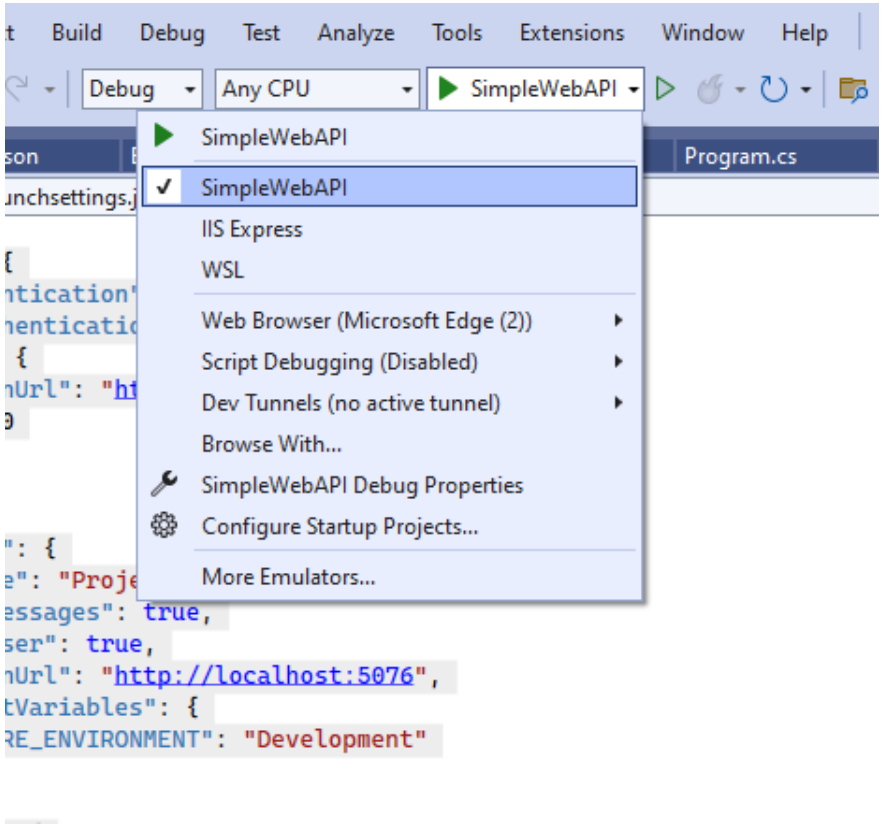
- IIS Express (Internet Information Services Express) is a lightweight, self-contained version of Microsoft's Internet Information Services (IIS) web server.

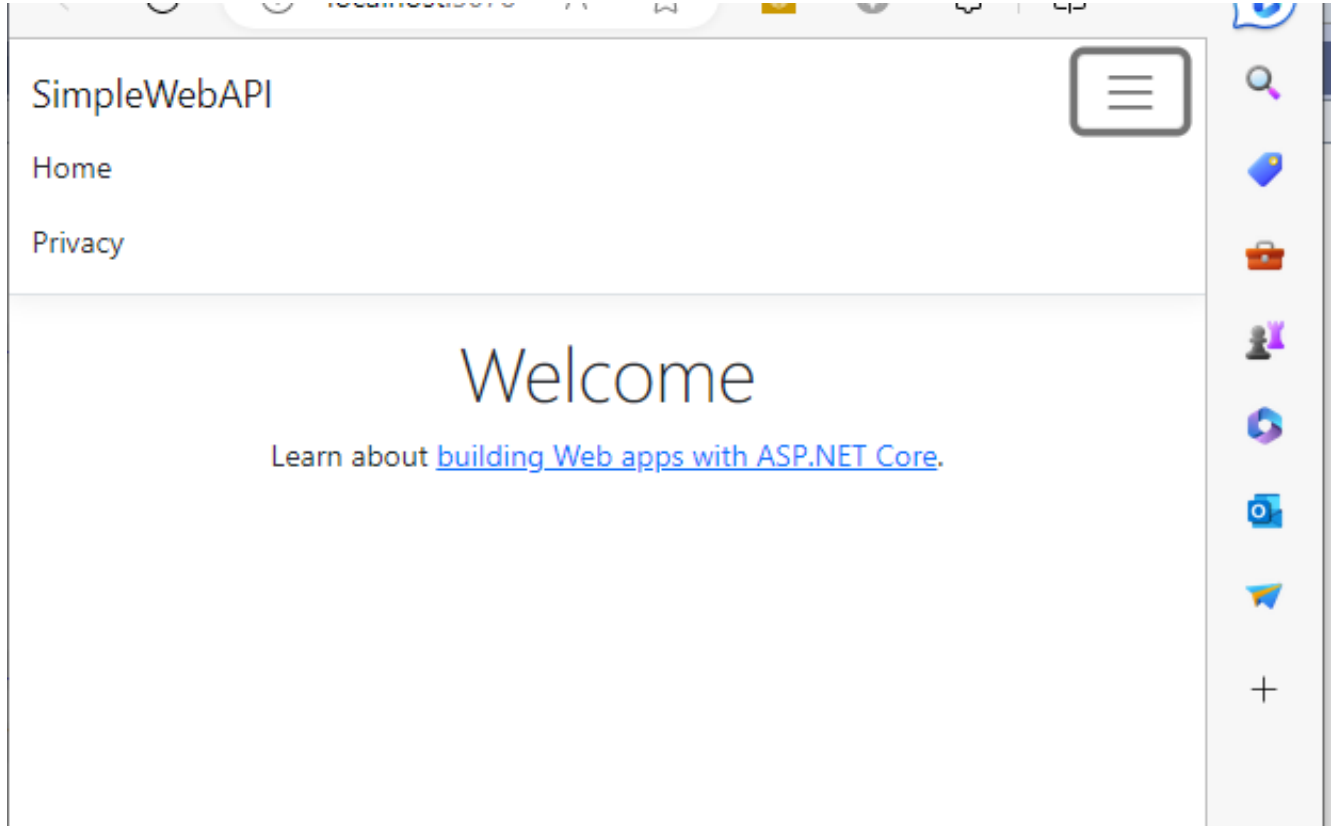




Running App in the Project profile

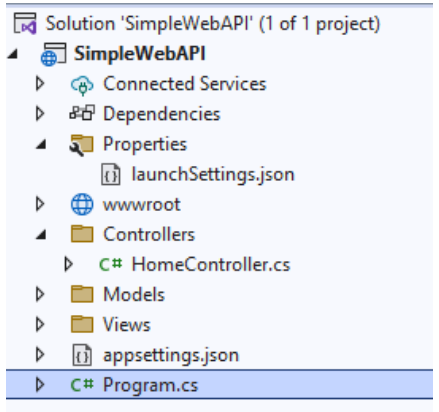
- Check the different in port number





Understanding Convention-based Routing

- **Convention-based routing is a routing mechanism in ASP.NET Core MVC that allows you to define routes based on conventions and patterns.**
- **Default Route: Convention-based routing often starts with a default route. The default route template specifies placeholders for the controller and action names, along with optional route parameters. This route acts as a catch-all for incoming requests that don't match any specific routes.**
- **Controller and Action Naming: The framework assumes certain naming conventions for controllers and actions. For example, a URL segment like /Home/Index is automatically mapped to the Index action within the HomeController.**
- **Route Parameters: Convention-based routing also handles route parameters, allowing you to capture values from the URL and pass them to action methods. For example, a URL like /Products/Details/5 maps to the Details action within the ProductsController and passes the value 5 as a parameter.**



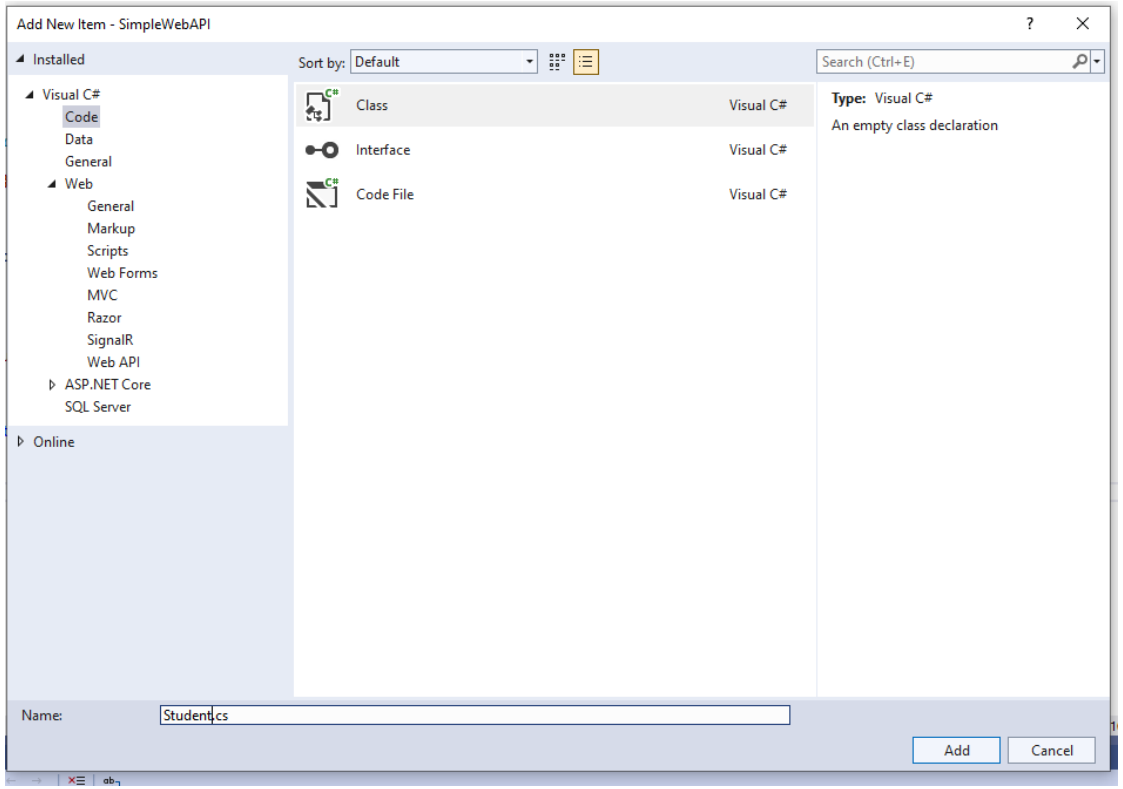

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
}
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

```
app.MapControllerRoute(
    name:"api",
    pattern:"api/{controller}/{action}/{id?}");
```

Optional: Add a new MapController Route for API

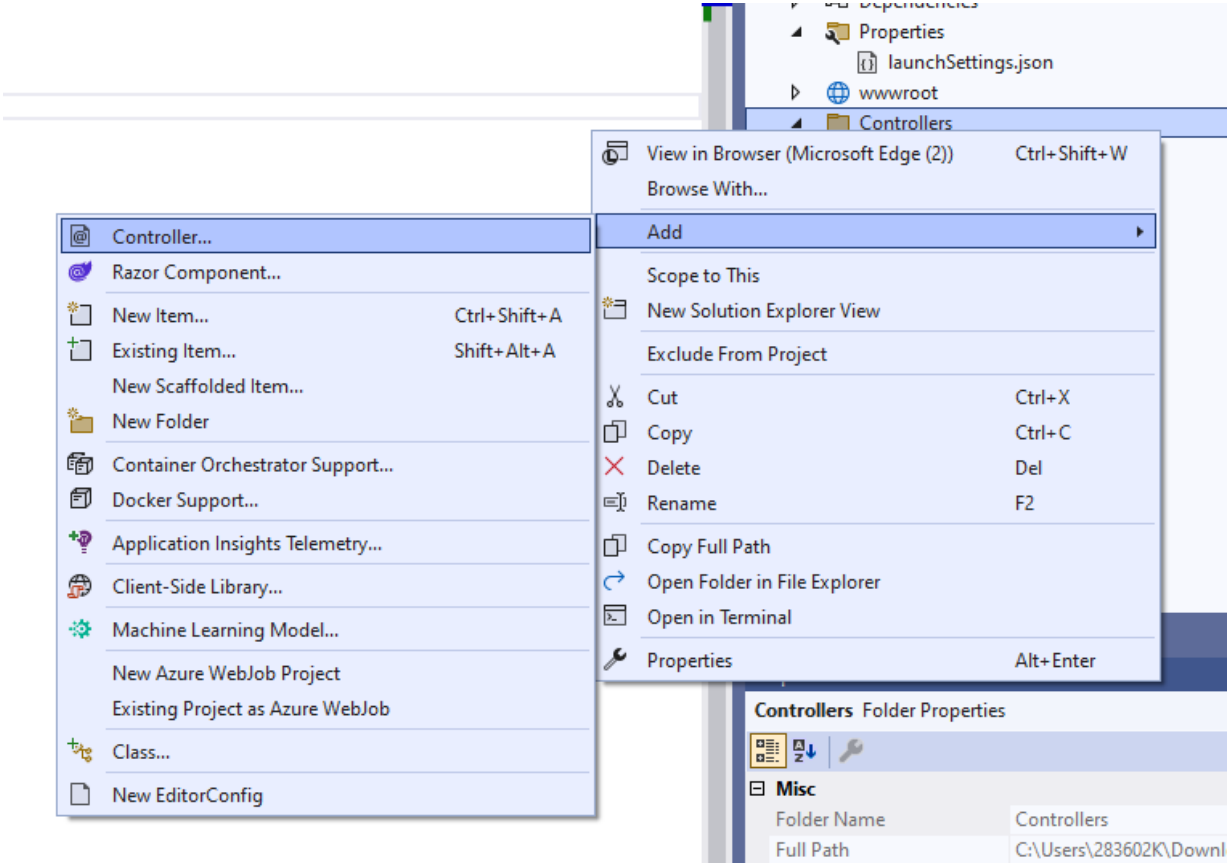
- In this example, a new custom route named “api" is added. This route template defines a pattern that starts with “api/" followed by the controller, action, and an optional id parameter. So, URLs like /api/students/details/5 would match this route and be mapped to the appropriate controller action.

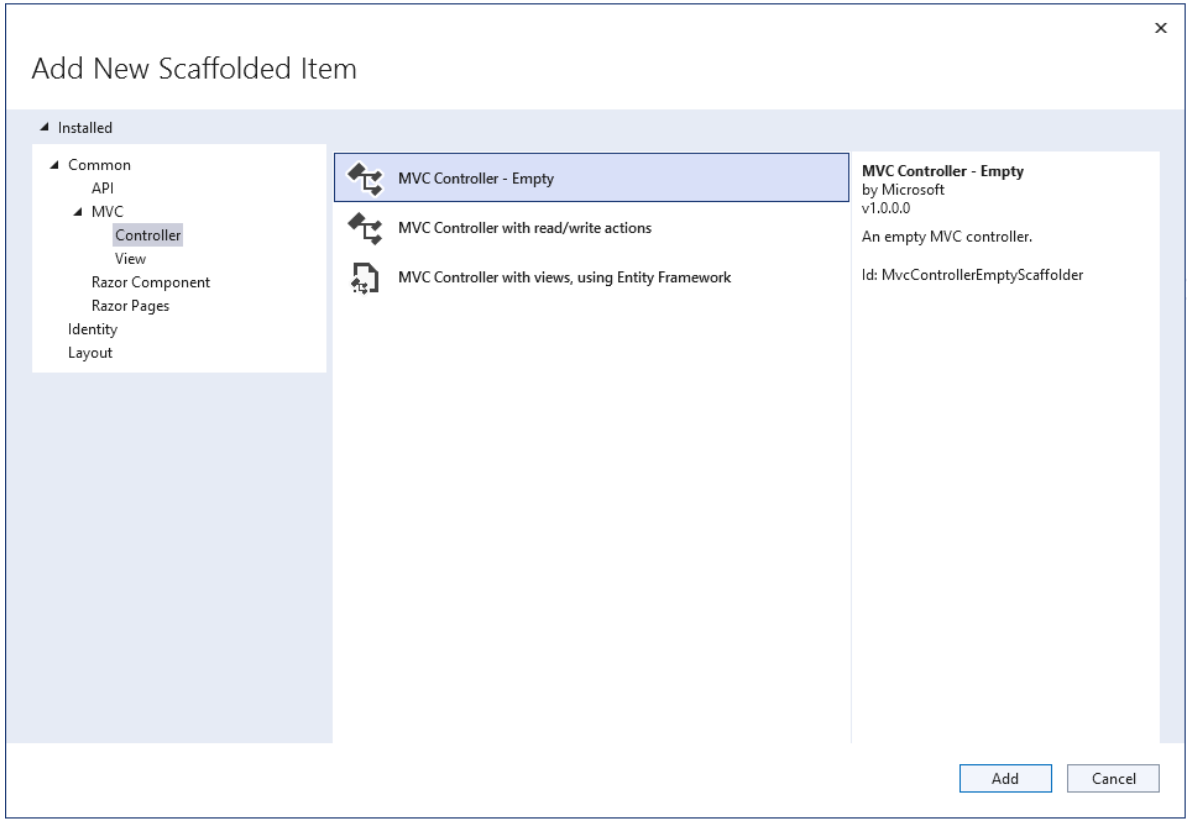
Add a student Model and Stu
Model/Student folder – Stud
in the model folder




```
0 references
public class StudentList
{
    static List<Student> students = new List<Student>();
    0 references
    public static List<Student> AllStudents()
    {
        return students;
    }
    0 references
    public static void AddStudent(Student student)
    {
        students.Add (student);
    }
    0 references
    public static Student GetStudent(int id)
    {
        foreach (Student student in students)
        {
            if (student.Id == id)
            {
                return student;
            }
        }
        return null;
    }
    0 references
    public static void Generate()
    {
        Student s1 = new Student();
        s1.Id = 1;
        s1.Name = "Sajib";
        s1.Age = 30;
        students.Add(s1);

        Student s2 = new Student();
        s2.Id = 2;
        s2.Name = "Mistry";
        s2.Age = 30;

        students.Add(s2);
    }
}
```

Create a new controller
i.e., Student Controller





StudentController.cs

```
public class StudentController : Controller
{
    [HttpGet]
    0 references
    public IEnumerable<Student> Details()
    {
        List<Student> students = StudentList.AllStudents();
        return students;
    }
```

this method **Details()** retrieves a list of **Student** objects, likely from some data source, and returns them as an **IEnumerable<Student>**.

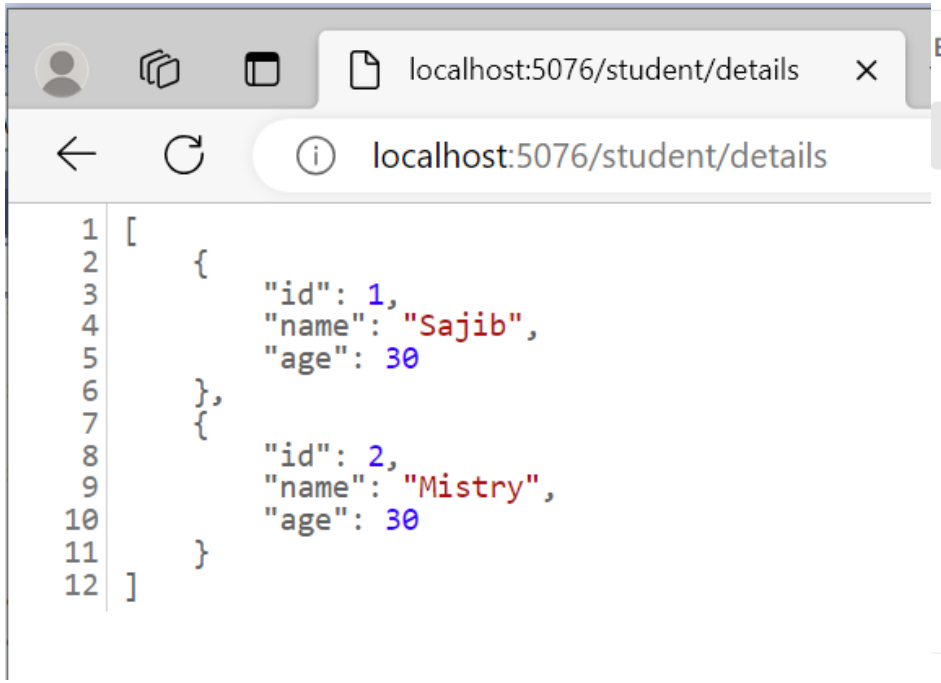
IEnumerable<T> in an ASP.NET Web API context is used to represent collections of data that are returned as responses to HTTP requests.

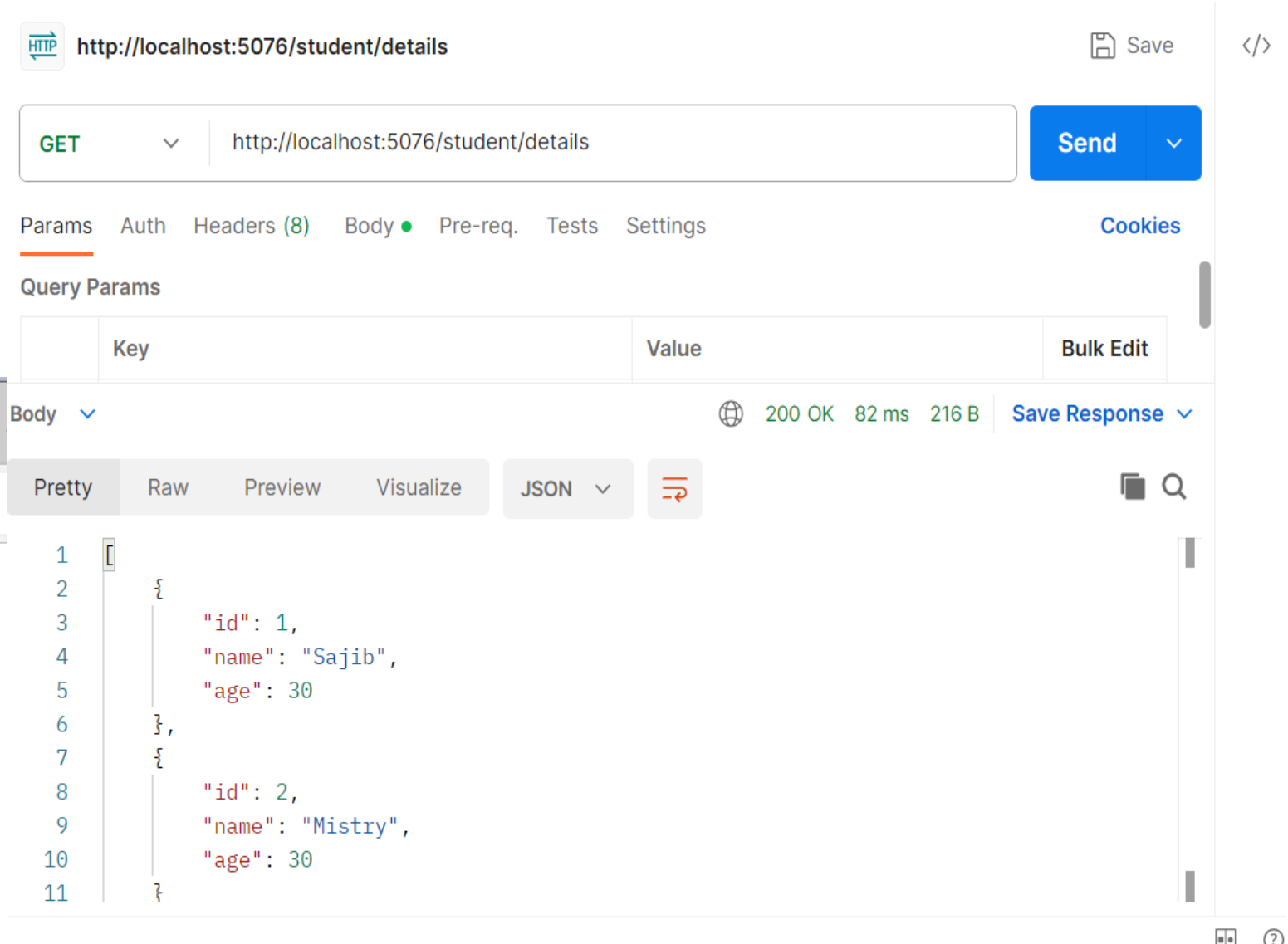
Program.cs

- We generate the data when the program runs!! (optional)

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");


StudentList.Generate();
```

Run the app:





StudentController.cs

```
[HttpGet]
0 references
public IActionResult Detail(int id)
{
    Student student = StudentList.GetStudent(id);
    if(student == null)
    {
        return NotFound();
    }
    else
    {
        return new ObjectResult(student) { StatusCode = 200 };
    }

}
```

IActionResult allows you to encapsulate various response types and status codes in a clean and consistent manner.

NotFound method returns an **NotFoundResult** with a 404 status code.

Using **ObjectResult** allows you to return structured data to clients in a standardized format (JSON), making it suitable for APIs. It also gives you control over the HTTP status code and additional response properties.

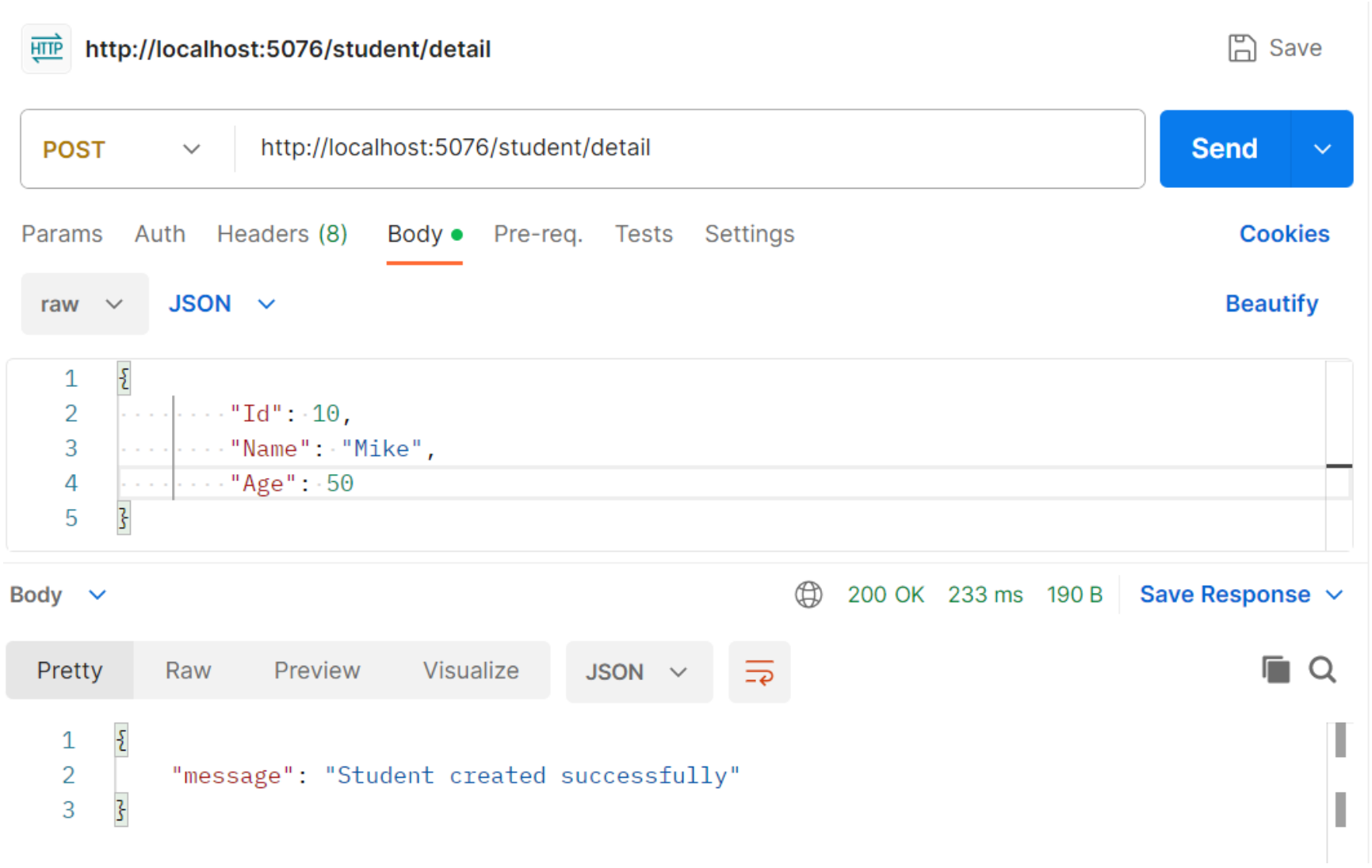
StudentController.cs

```
[HttpPost]
0 references
public IActionResult Detail([FromBody] Student student)
{
    StudentList.AddStudent(student);
    var response = new { Message = "Student created successfully" };
    return new ObjectResult(response)
    {
        StatusCode = 200,
        ContentTypes = { "application/json" }
    };
}
```

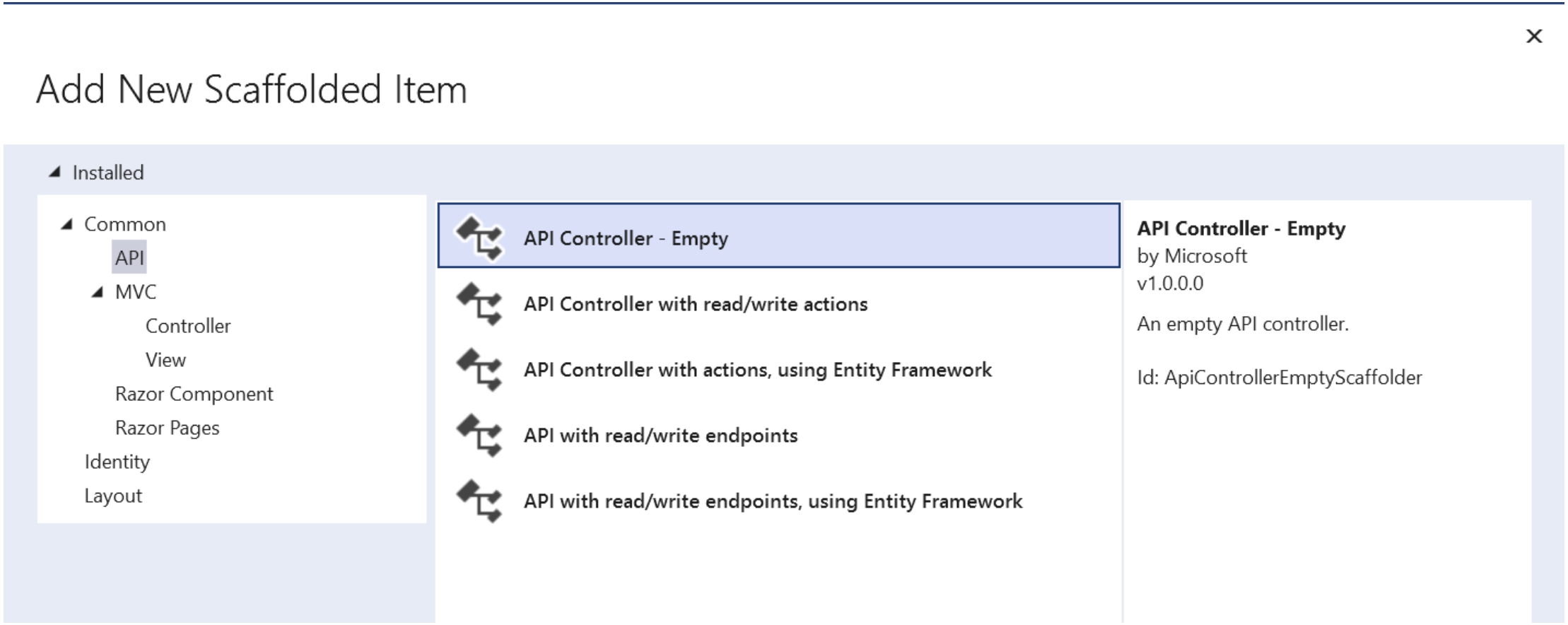
[HttpPost] attribute is an attribute in ASP.NET MVC and ASP.NET Core MVC that is used to indicate that a controller action method should only respond to HTTP POST requests.

[FromBody] attribute in ASP.NET Core MVC is used to bind the content of an HTTP request's body to a parameter in a controller action method. It's commonly used when you want to receive data from the request body, such as JSON or XML data, and deserialize it into a parameter of a method. This is especially useful when handling HTTP POST or PUT requests where the data is sent in the request body.

Checking [Post] with PostMan



Creating API using API controller



API controller needs Attribute Routing

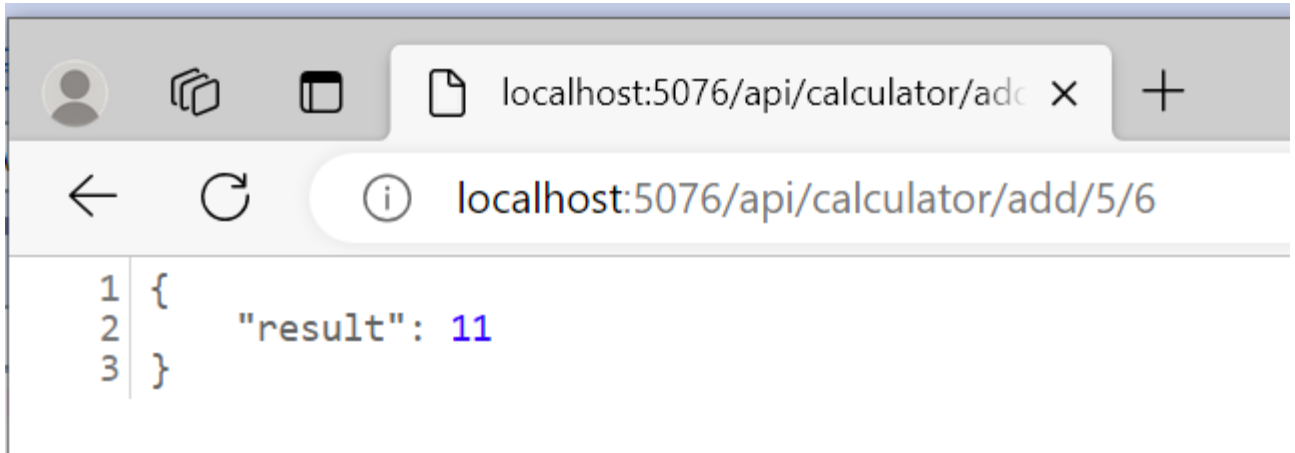
- Enable Attribute Routing: In ASP.NET Core, Attribute Routing is enabled by default. In earlier versions of ASP.NET MVC, you need to enable it by calling the MapMvcAttributeRoutes method in the RouteConfig or Startup class.

- Controller-Level Routing: You can apply a route template at the controller level using the [Route] attribute on the controller class. This template will be the prefix for all action methods within the controller.

- Action-Level Routing: You can further customize the route for individual action methods using the [HttpGet], [HttpPost], or other HTTP method attributes combined with the [Route] attribute.

CalculatorController.cs

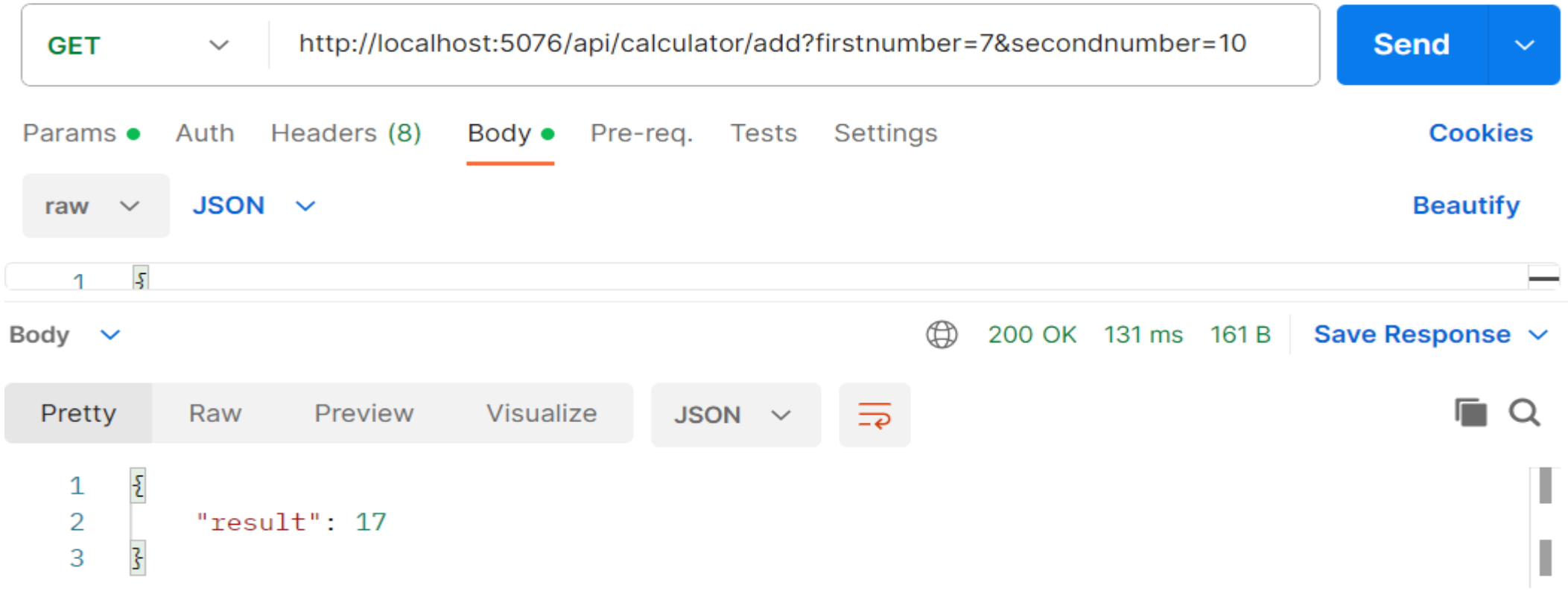
```
[Route("api/[controller]")]
[ApiController]
0 references
public class CalculatorController : ControllerBase
{
    [HttpGet]
    [Route("add/{firstNumber}/{secondNumber}")]
    0 references
    public IActionResult Add(int firstNumber, int secondNumber)
    {
        int result = firstNumber + secondNumber;
        var response = new { Result = result};
        return Ok(response);
    }
```



CalculatorController.cs

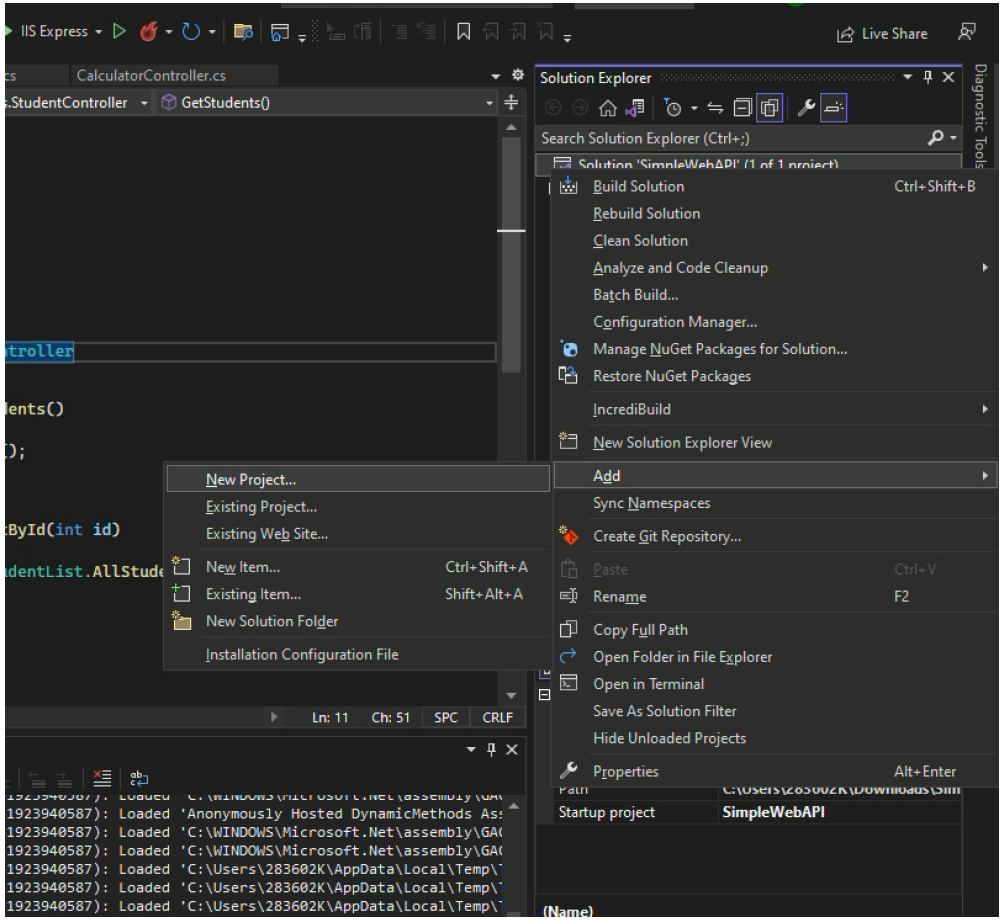
- Using query parameter

```
[Route("add")]
[HttpGet]
0 references
public IActionResult AddFromQuery([FromQuery]int firstNumber,
    [FromQuery] int secondNumber)
{
    int result = firstNumber + secondNumber;
    var response = new { Result = result };
    return Ok(response);
}
```

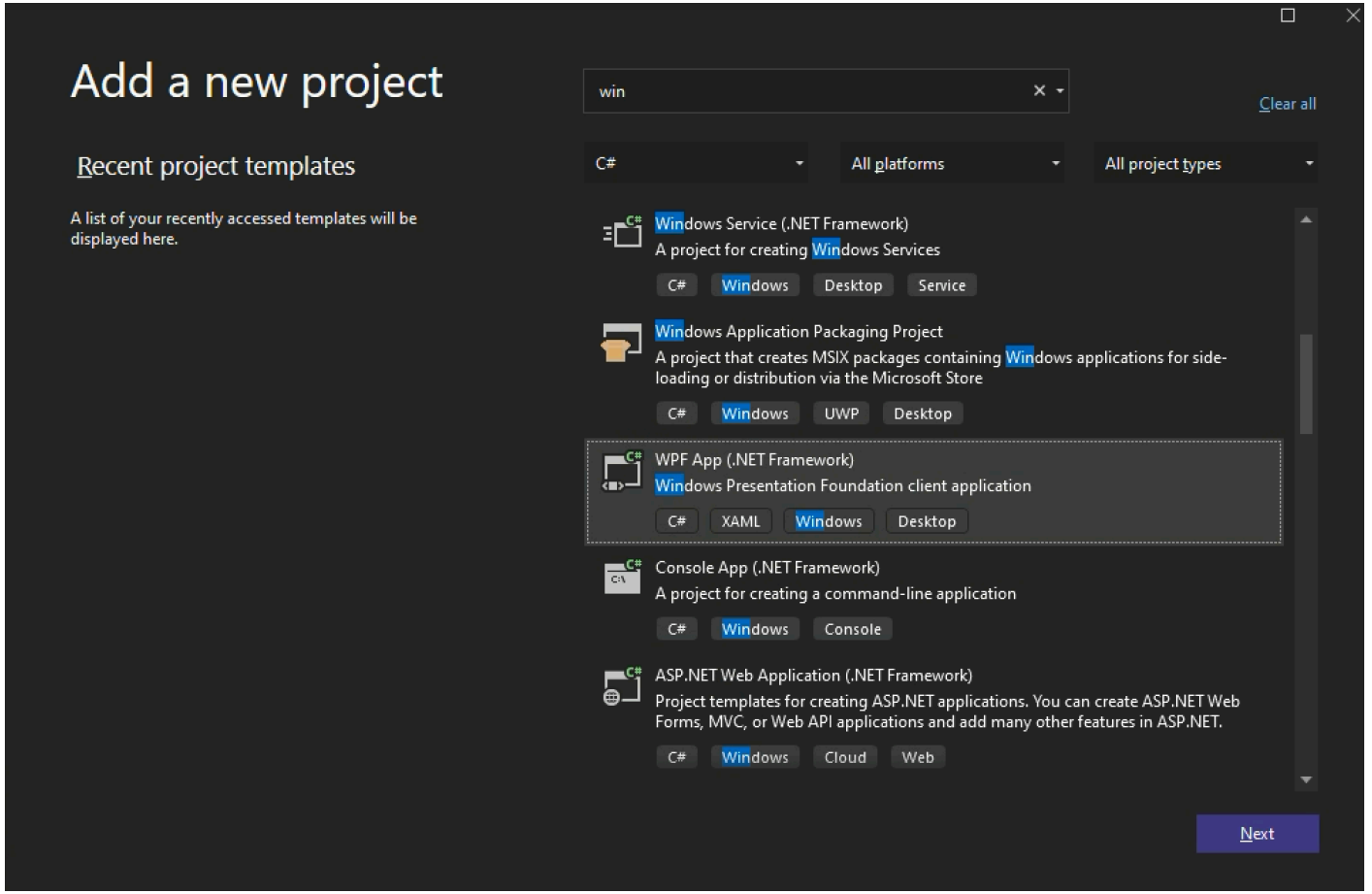


The WEB API is ready. Now we want to call the services from a simple WPF client

Add a new project in the solution



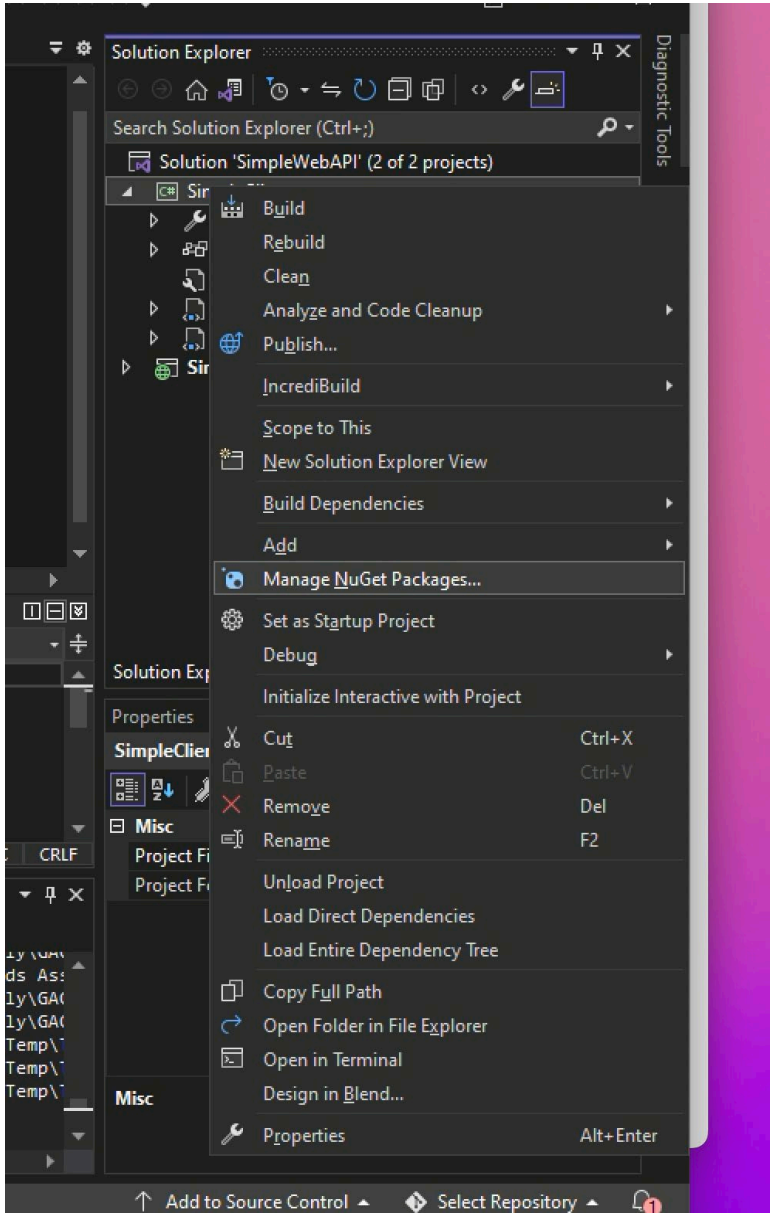
Make sure it is WPF APP .NET Framework

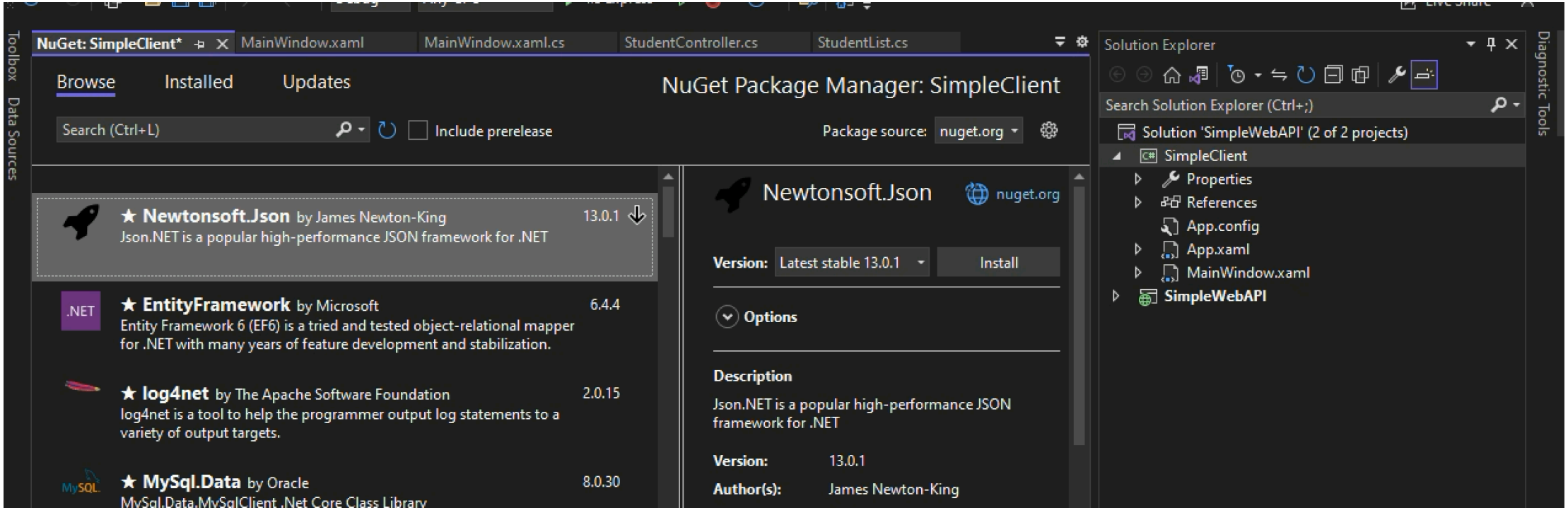


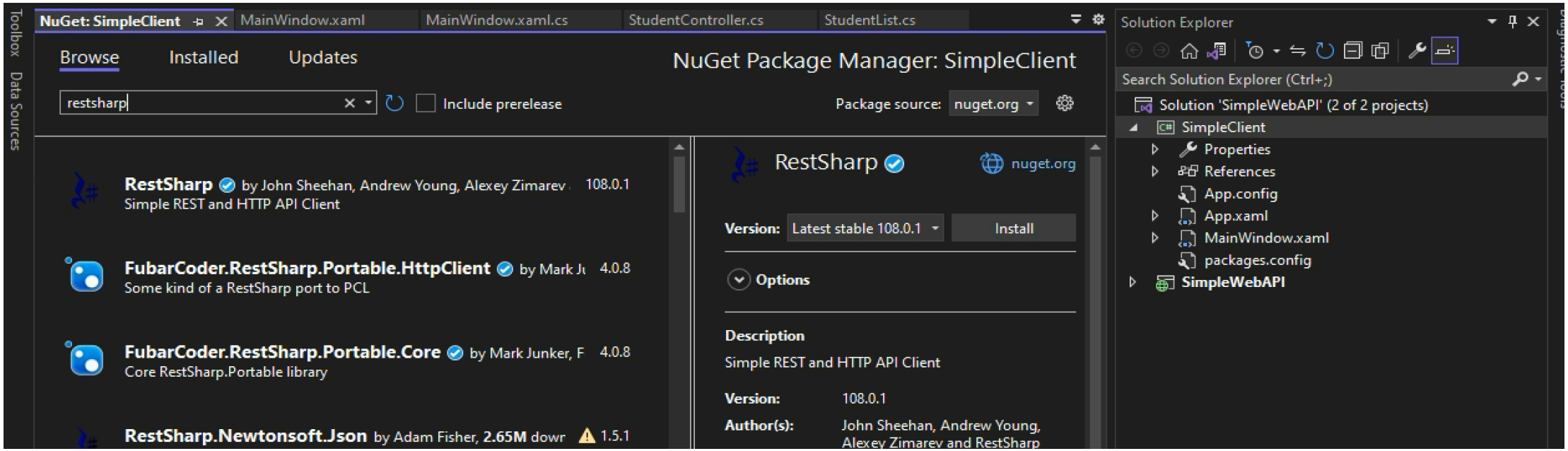
You can add external libraries to your project using NuGet

We will add RestSharp (for dealing with REST API)

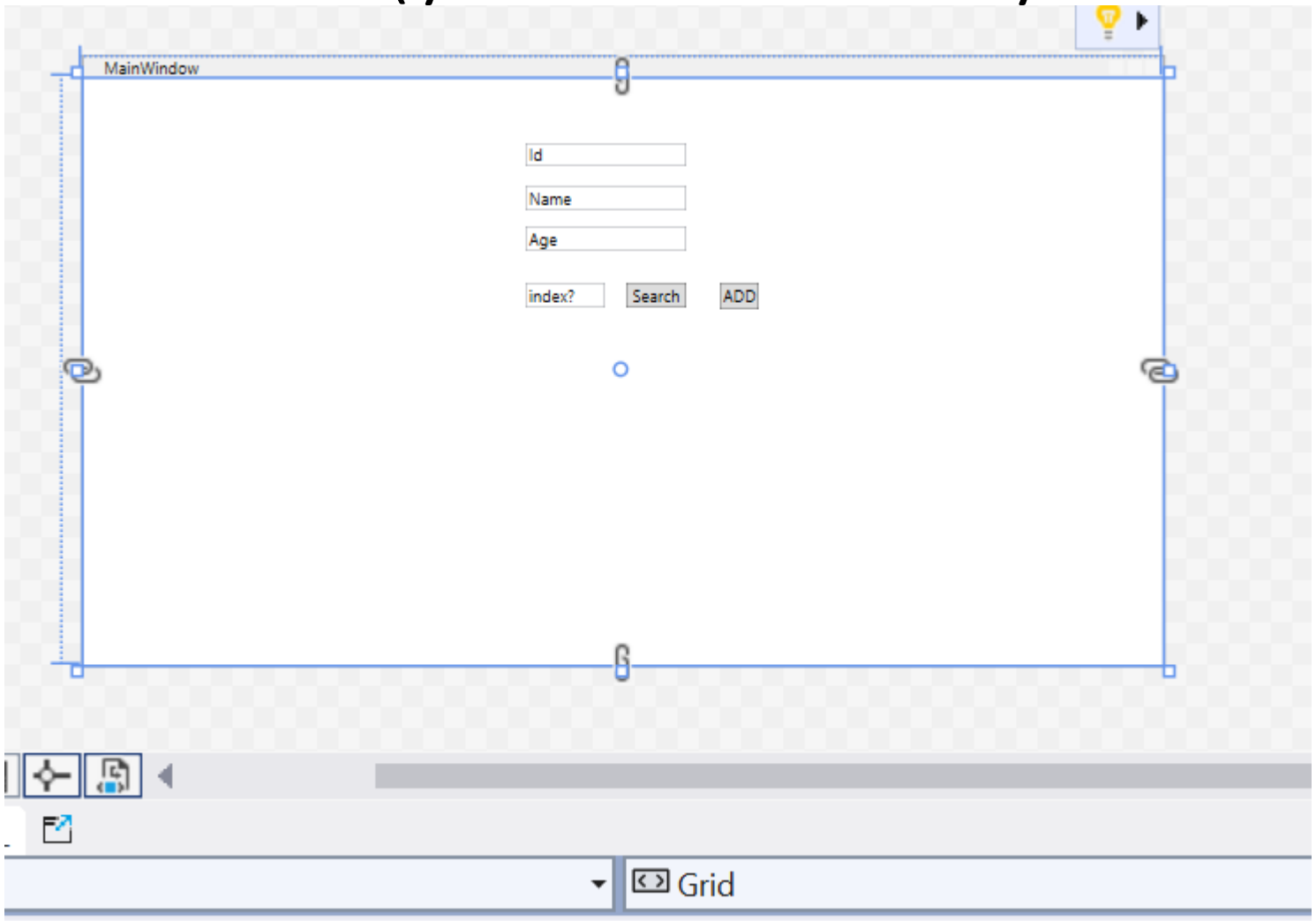
and NewtonSoft.JSON (for dealing with json messages from the APIs)



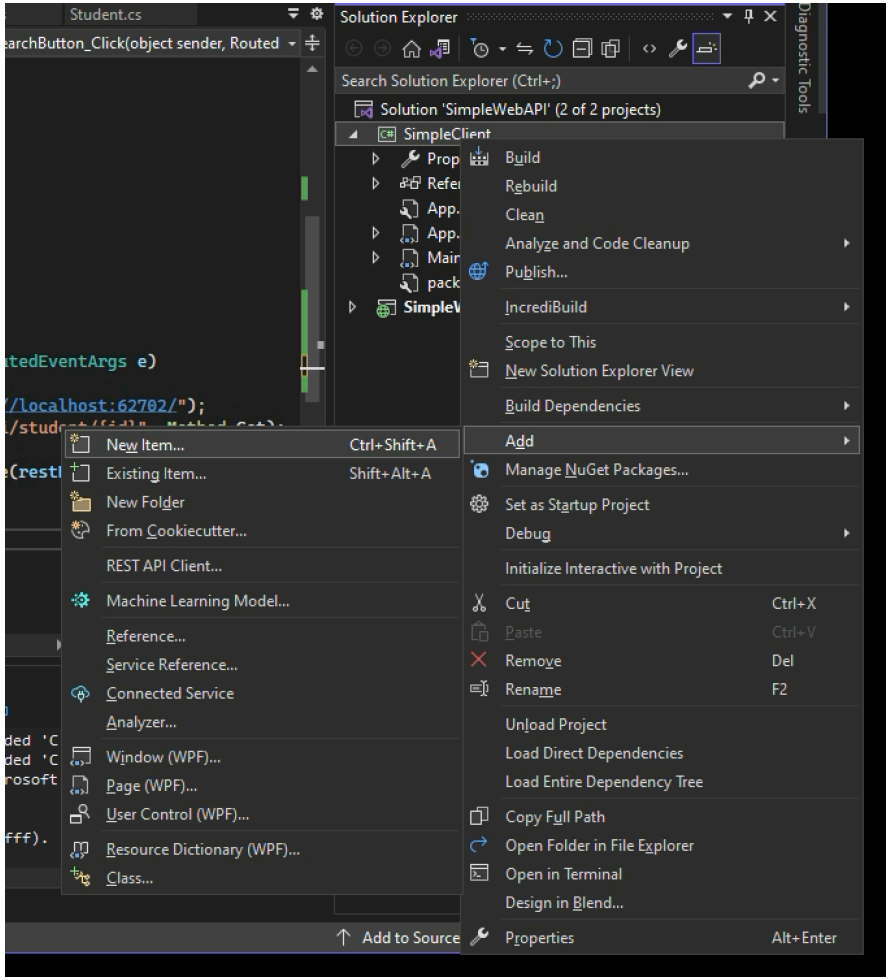

NuGet: SimpleClient*
MainWindow.xaml
MainWindow.xaml.cs
StudentController.cs
StudentList.cs
Browse
Installed
Updates
NuGet Package Manager: SimpleClient
Search (Ctrl+L)
Include prerelease
Package source: nuget.org
★ Newtonsoft.Json by James Newton-King 13.0.1
Json.NET is a popular high-performance JSON framework for .NET
★ EntityFramework by Microsoft 6.4.4
Entity Framework 6 (EF6) is a tried and tested object-relational mapper for .NET with many years of feature development and stabilization.
★ log4net by The Apache Software Foundation 2.0.15
log4net is a tool to help the programmer output log statements to a variety of output targets.
★ MySql.Data by Oracle 8.0.30
MySql.Data.MySqlClient .Net Core Class Library
Newtonsoft.Json
nuget.org
Version: Latest stable 13.0.1
Install
Options
Description
Json.NET is a popular high-performance JSON framework for .NET
Version: 13.0.1
Author(s): James Newton-King
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SimpleWebAPI' (2 of 2 projects)
SimpleClient
Properties
References
App.config
App.xaml
MainWindow.xaml
SimpleWebAPI
Toolbox
Data Sources
Diagnostic Tools


NuGet: SimpleClient
MainWindow.xaml
MainWindow.xaml.cs
StudentController.cs
StudentList.cs
Browse
Installed
Updates
NuGet Package Manager: SimpleClient
restsharp
Include prerelease
Package source: nuget.org
RestSharp by John Sheehan, Andrew Young, Alexey Zimarev 108.0.1
Simple REST and HTTP API Client
FubarCoder.RestSharp.Portable.HttpClient by Mark Ju 4.0.8
Some kind of a RestSharp port to PCL
FubarCoder.RestSharp.Portable.Core by Mark Junker, F 4.0.8
Core RestSharp.Portable library
RestSharp.Newtonsoft.Json by Adam Fisher, 2.65M dowr 1.5.1
RestSharp
nuget.org
Version: Latest stable 108.0.1
Install
Options
Description
Simple REST and HTTP API Client
Version: 108.0.1
Author(s): John Sheehan, Andrew Young, Alexey Zimarev and RestSharp
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SimpleWebAPI' (2 of 2 projects)
SimpleClient
Properties
References
App.config
App.xaml
MainWindow.xaml
packages.config
SimpleWebAPI
Toolbox
Data Sources

Build the XAML (you are already familiar)



Add the Student Class again. Or you may refer to the WEB API project models. However, I am not a big fan of reference when doing WEB API




```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Call the WEB API when the search button is pressed and deserialise the return json into objects and update the GUI

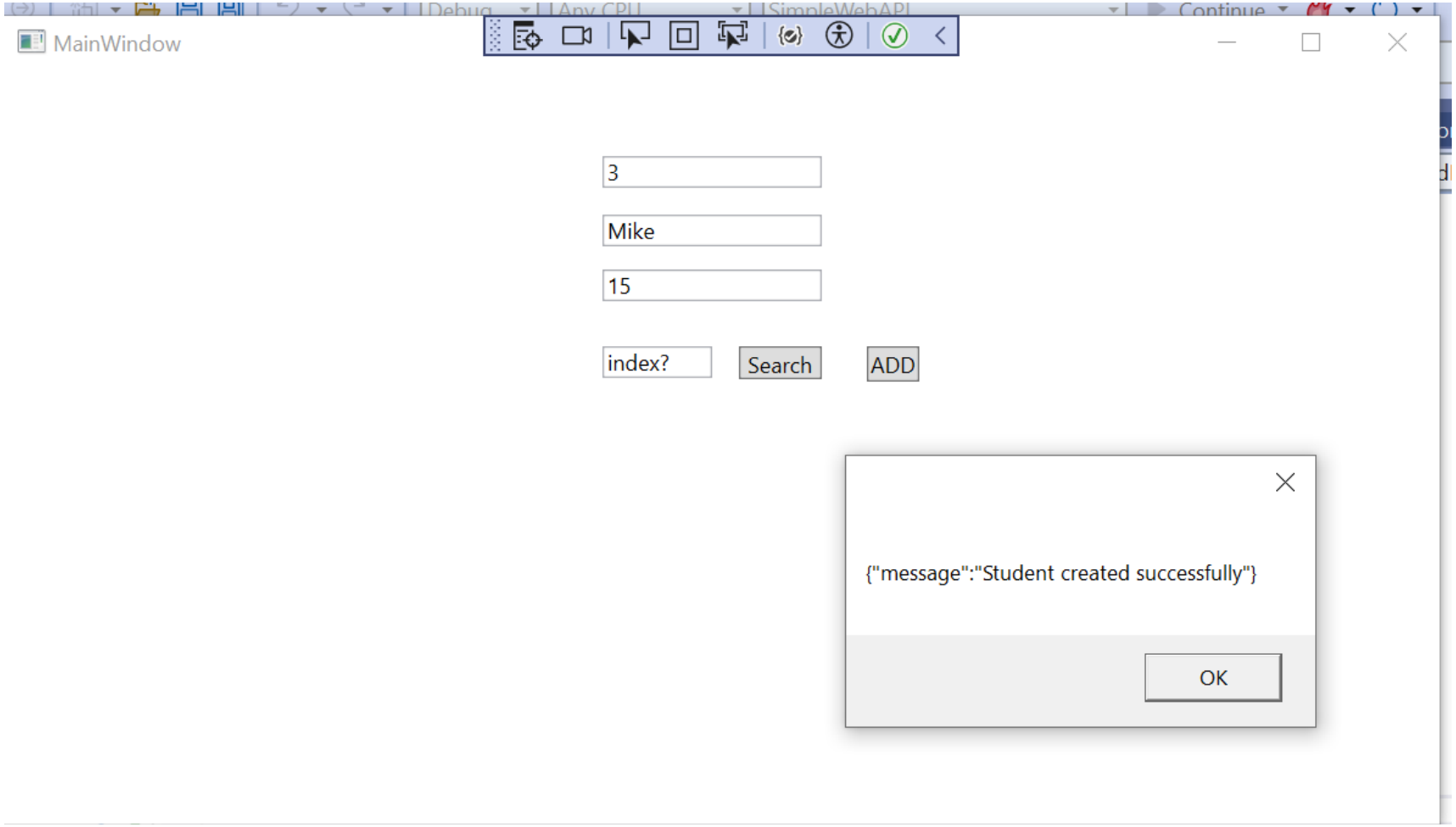
```
using RestSharp;
using Newtonsoft.Json;
```

```
1 reference
private void SearchButton_Click(object sender, RoutedEventArgs e)
{
    RestClient restClient = new RestClient("http://localhost:5076");
    RestRequest restRequest = new RestRequest("/student/detail/{id}", Method.Get);
    restRequest.AddUrlSegment("id", SIndex.Text);
    RestResponse restResponse = restClient.Execute(restRequest);
    // Console.WriteLine(restResponse.Content);
    Student student = JsonConvert.DeserializeObject<Student>(restResponse.Content);
    SID.Text = student.Id.ToString();
    SName.Text = student.Name;
    SAge.Text = student.Age.ToString();
}
```

When post, use the request body

```
private void AddButton_Click(object sender, RoutedEventArgs e)
{
    RestClient restClient = new RestClient("http://localhost:5076");
    RestRequest restRequest = new RestRequest("/student/detail", Method.Post);
    Student student = new Student();
    student.Age = Int32.Parse(SAge.Text);
    student.Name = SName.Text;
    student.Id = Int32.Parse(SID.Text);

    restRequest.RequestFormat = RestSharp.DataFormat.Json;
    restRequest.AddBody(student);

    RestResponse restResponse = restClient.Execute(restRequest);
    // Console.WriteLine(restResponse.Content);
    MessageBox.Show(restResponse.Content);
}
```

Make sure the Web API is running before you run the client



How to use RestSharp and NewtonSoft.json

- https://restsharp.dev/usage.html#recommended-usage
- https://www.newtonsoft.com/json/help/html/Samples.htm