

Introducere in REST-ful API folosind ASP.NET Web API. Caracteristici. Avantajele utilizarii. Conventia REST-ful. Crearea unui Web API. Consumarea API-ului.

Introducere in REST-ful API folosind ASP.NET Web API

REST (**R**epresentational **S**tate **T**ransfer) este o noua abordare de dezvoltare a proiectelor si serviciilor web, care a schimbat complet dezvoltarea aplicatiilor software. In momentul de fata nu mai exista aplicatii care sa nu utilizeze **REST API (Twitter, YouTube, Facebook, etc)** fiind cel mai **logic** si **eficient** standard in crearea de API-uri.

Astfel, HTTP nu mai este doar un simplu protocol de transfer si de afisare a paginilor HTML, ci a devenit o platforma puternica in construirea API-urilor Web. Deoarece este construit peste framework-ul ASP.NET MVC, Web API se asigura in mod automat de toate necesitatile protocolului HTTP.

Caracteristici

- **Nu isi pastreaza starea** – toate requesturile se fac o singura data nepastrand o stare din trecut. De exemplu, un site sau o platforma web dezvoltata fara REST pastreaza la nivel de cookie-uri / sesiune istoricul de navigare, statusul daca un utilizator este sau nu logat, etc. In cazul platformelor dezvoltate utilizand arhitectura REST nu se stocheaza informatii, fiecare request fiind individual, comportandu-se ca si cum se apeleaza pentru prima oara de fiecare data.
- **Exista 4 tipuri de request** – **GET** (read), **POST** (create), **PUT** (update), **DELETE**.

- **Obiectele sunt manipulate prin intermediul URI (Uniform Resource Identifier)** – prin intermediul URI putem accesa, modifica, sterge informatia.

Avantajele utilizarii REST API in dezvoltarea aplicatiilor

- **Independenta fata de tipul platformei sau de limbaj** – REST API se adapteaza oricarui limbaj sau oricarei sintaxe, oferind astfel o libertate absoluta in utilizarea diverselor medii de dezvoltare (PHP, PYTHON, JAVA, etc). Singurul lucru pe care trebuie sa il avem in vedere este faptul ca raspunsul la request-uri trebuie sa fie intotdeauna in format XML sau **JSON (JavaScript Object Notation)**.
- **Separarea bine definita intre client si server** – protocolul REST separa interfata cu utilizatorul de partea de backend (partea de server si de stocare a datelor). Acest lucru aduce un avantaj deosebit **crescand scalabilitatea** - componentele pot fi dezvoltate diferit. Un API reprezinta un punct de comunicare intre una sau mai multe aplicatii si este folosit pentru a face posibila utilizarea datelor unui sistem in mai multe situatii (de exemplu: aceeasi logica se poate folosi atat intr-o aplicatie web, cat si intr-o aplicatie de mobil sau desktop)
- **Portabilitate** – interfata putand fi dezvoltata folosind tehnologii diferite. Faciliteaza dezvoltarea unei aplicatii pe mai multe platforme (iOS, android, desktop).
- **Flexibilitate** – componentele soft-ului dezvoltat (frontend si backend) pot sta pe servere diferite ceea ce duce la o securitate sporita .

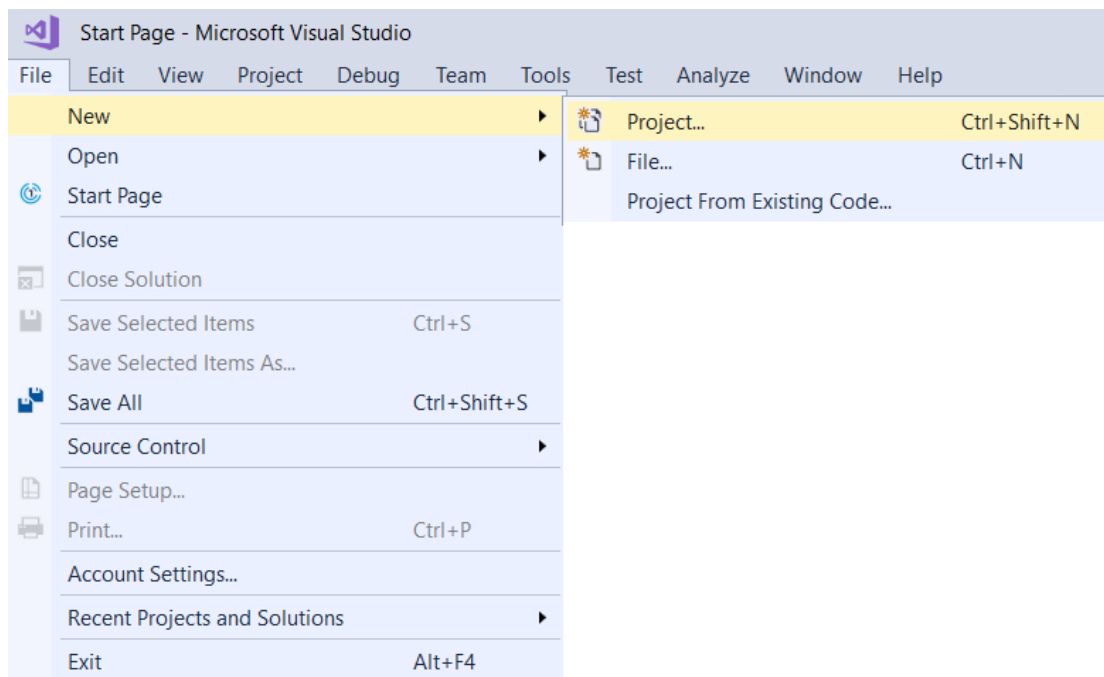
Conventia REST-ful

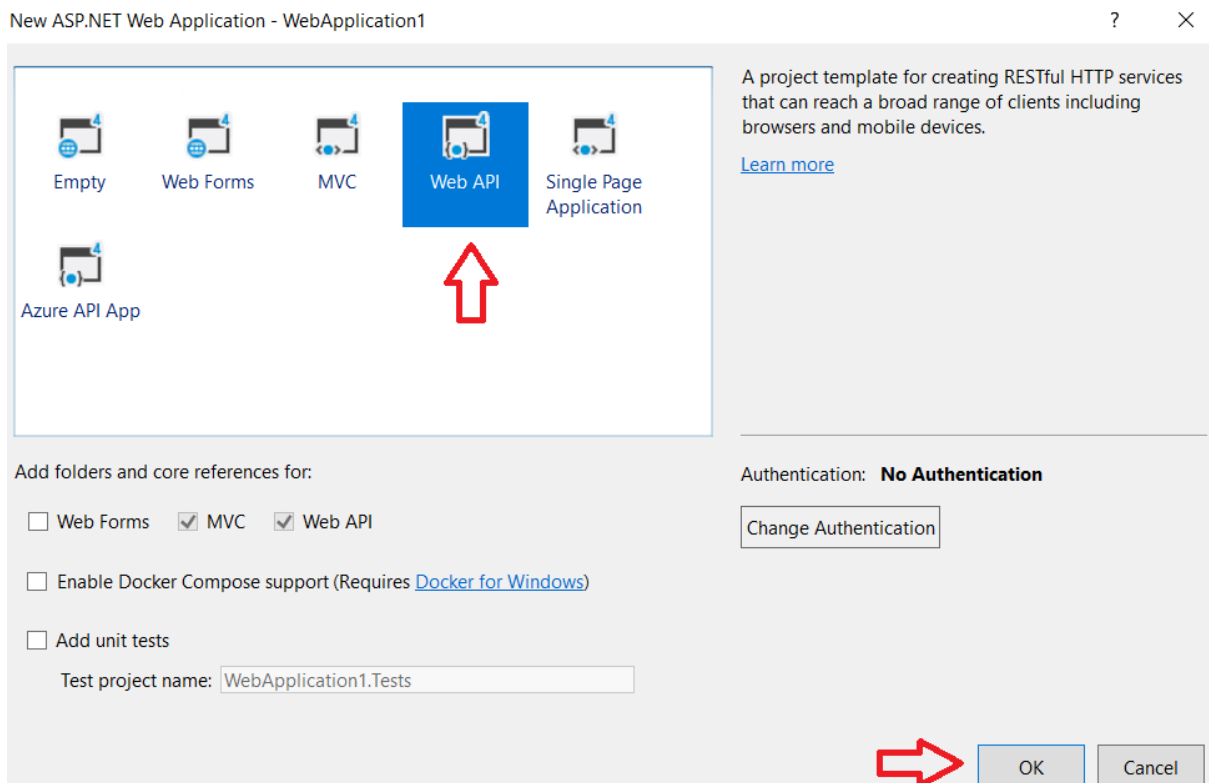
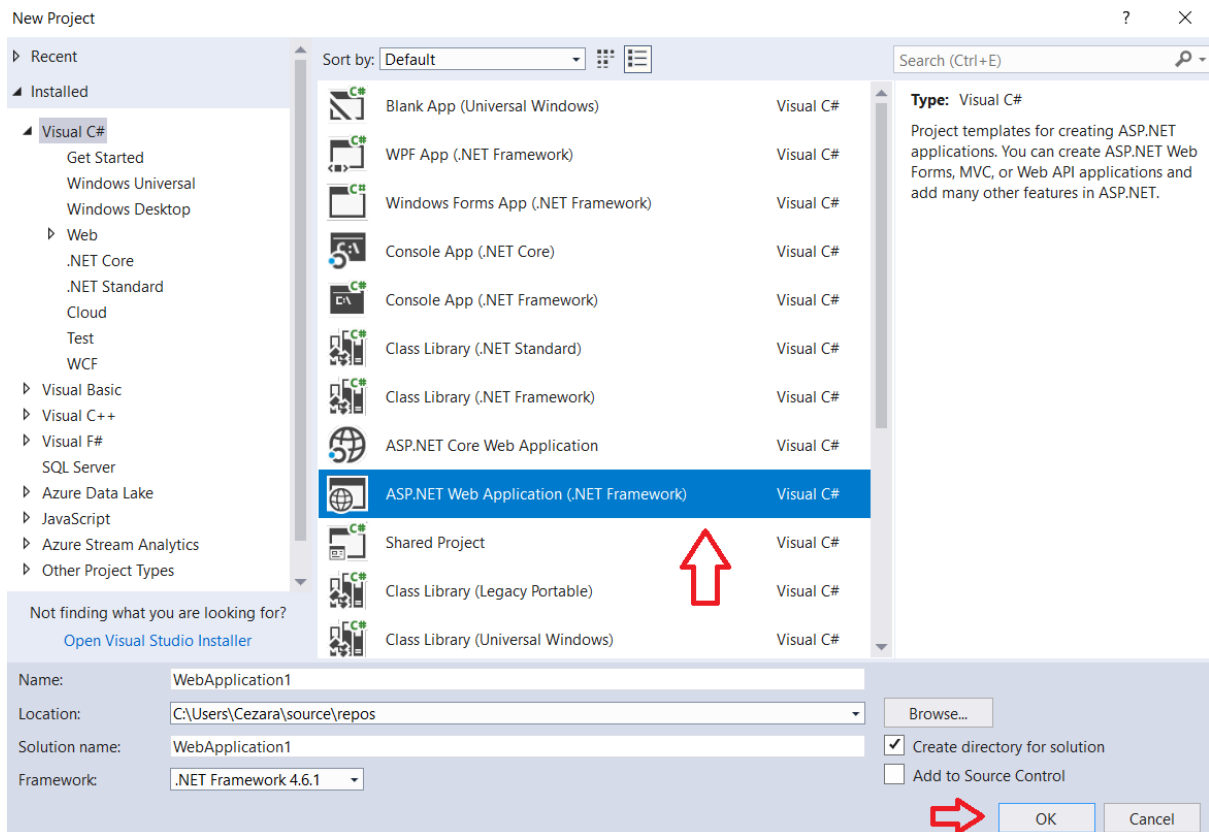
Exista cateva tipuri diferite de request. Ca exemplu folosim entitatea **Student** – presupunem ca avem o aplicatie care prelucreaza toti studentii prezenti in platforma: **afisarea** tuturor studentilor, **afisarea unui singur** student, **adaugarea** unui student nou, **modificarea** unui student existent si **stergerea** unui student. Rutele necesare vor fi:

- **GET** – /api/students -> (afisarea listei cu toti studentii)
- **GET** – /api/students/1 -> (afisarea unui singur student)
- **POST** – /api/ students -> (adaugarea unui student nou)
- **PUT** – /api/students/1 -> (modificarea datelor unui student)
- **Delete** – /api/students/1 -> (stergerea unui singur student)

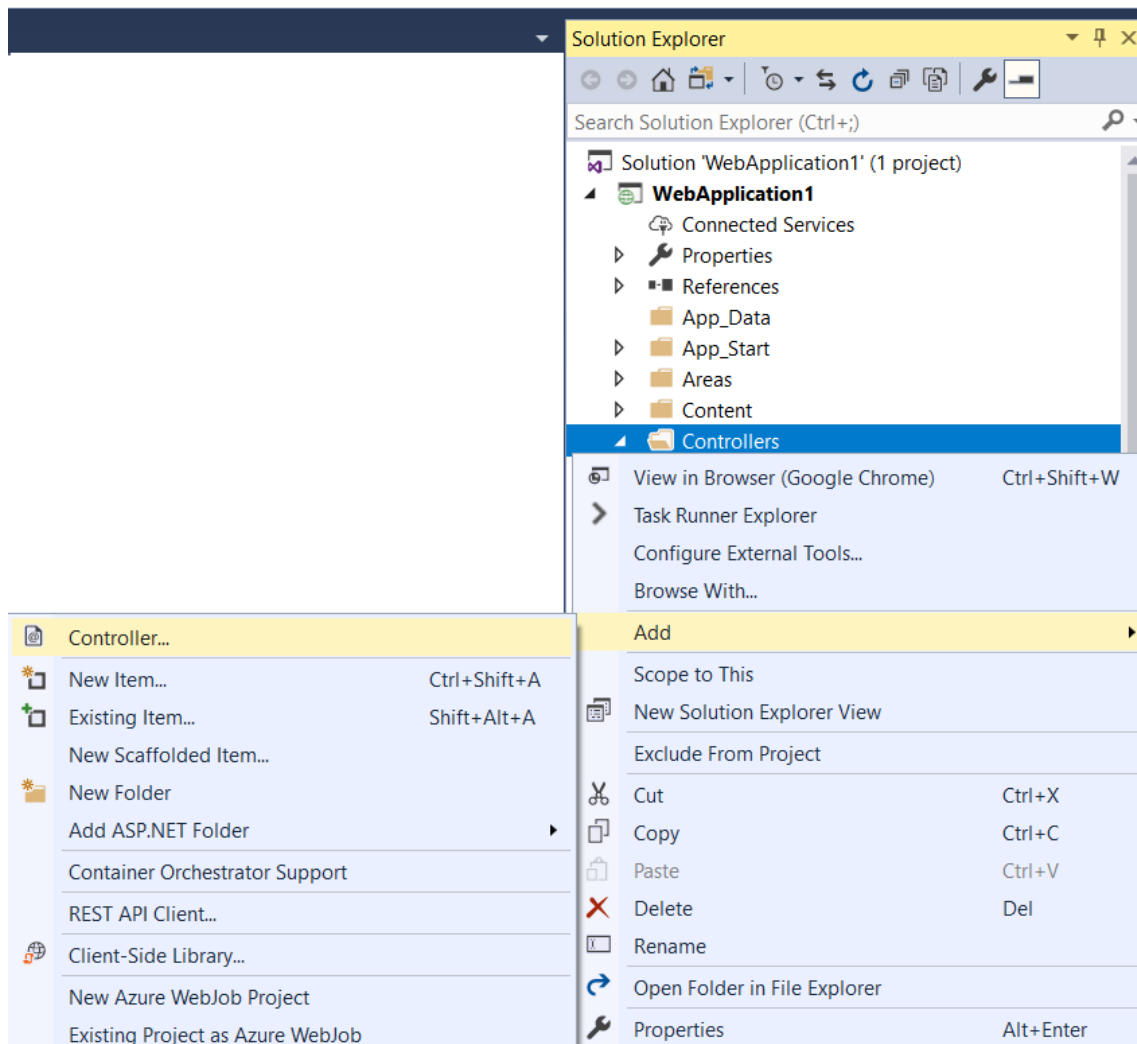
Crearea unui Web API

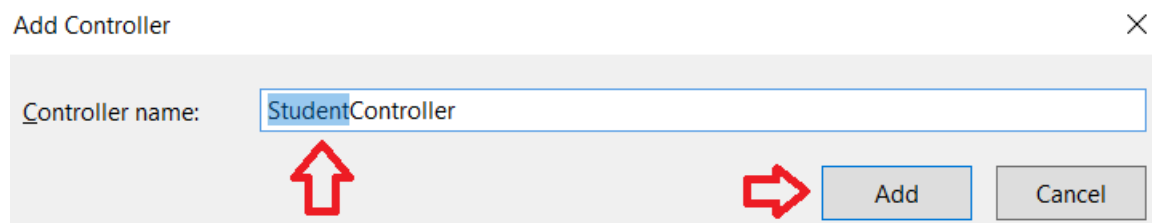
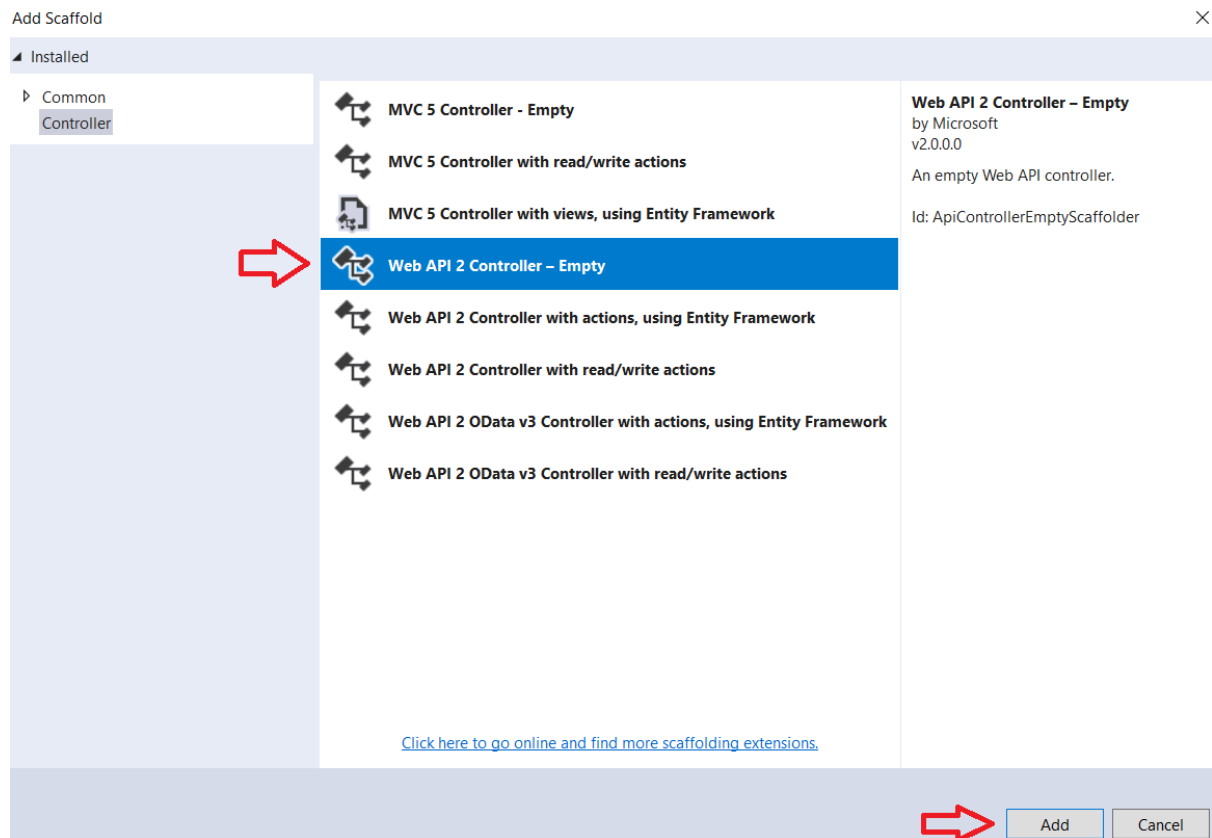
Pasul 1 – Crearea proiectului



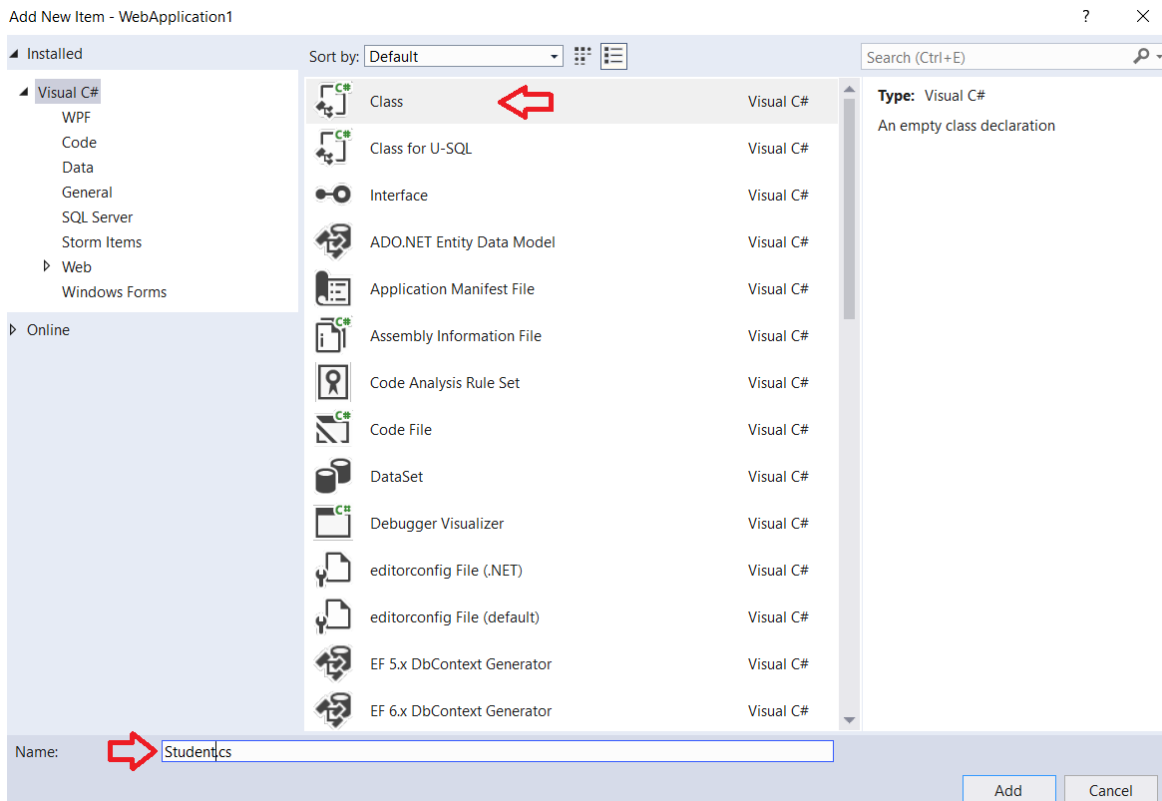
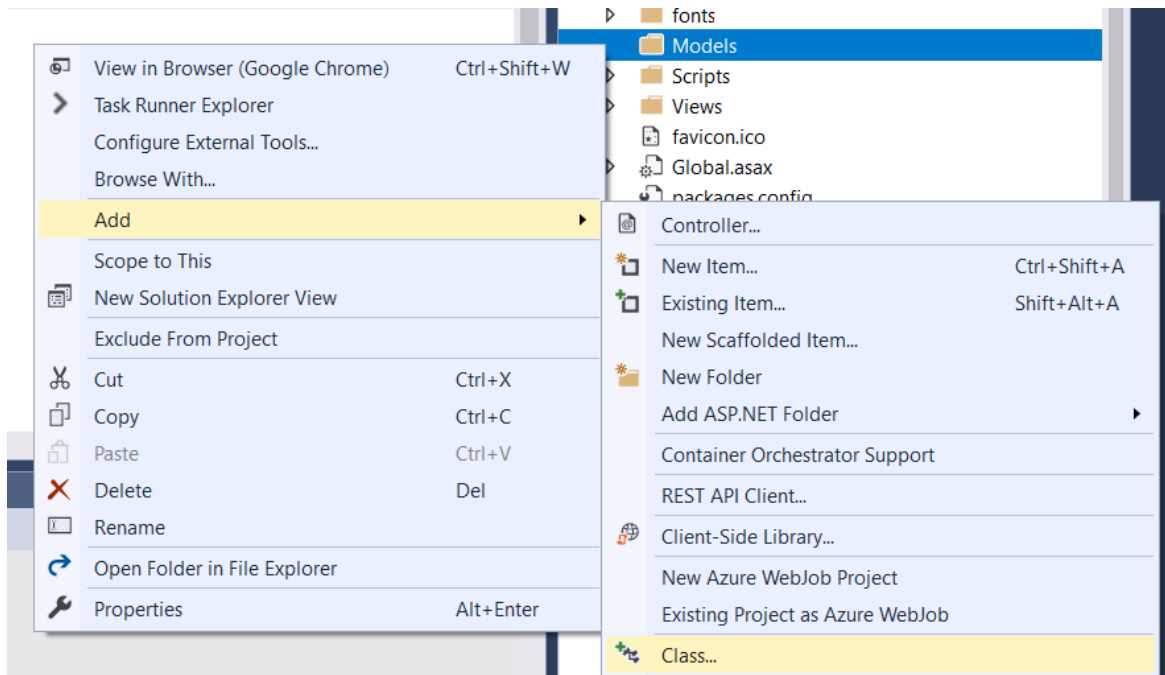


Pasul 2 – Crearea controller-ilor, unde vom scrie metodele necesare API-ului



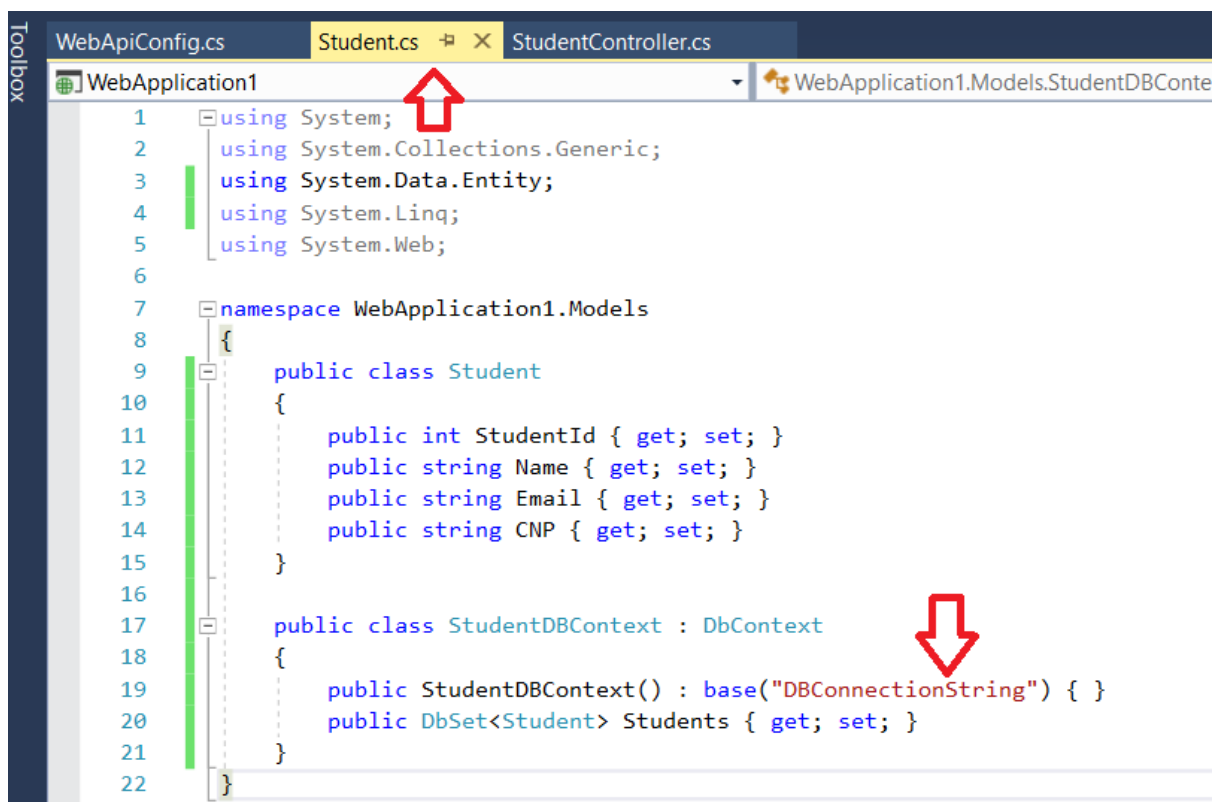


Pasul 3 – Crearea modelelor (a bazei de date)



In continuare o sa folosim entitatea **Student** avand urmatoarele attribute in baza de date: StudentId, Name, Email, CNP

```
public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string CNP { get; set; }
}
```



Pentru crearea si modelarea bazei de date Vezi Curs 4 – Model, sectiunea Entity Framework.

Un Controller Web API este similar cu unul ASP.NET MVC. Acesta este o clasa creata in folderul Controller, al carei nume trebuie sa se termine in cuvantul Controller si care deriva din clasa **System.Web.Http.ApiController**.

In momentul in care cream un nou proiect Web API, Visual Studio adauga by default un Controller numit **ValuesController.cs** in care se regaseste urmatoarea structura de baza:

```
7
8 namespace WebApplication1.Controllers
9 {
10     public class ValuesController : ApiController
11     {
12         // GET api/values
13         public IEnumerable<string> Get()
14         {
15             return new string[] { "value1", "value2" };
16         }
17
18         // GET api/values/5
19         public string Get(int id)
20         {
21             return "value";
22         }
23
24         // POST api/values
25         public void Post([FromBody]string value)
26         {
27         }
28
29         // PUT api/values/5
30         public void Put(int id, [FromBody]string value)
31         {
32         }
33
34         // DELETE api/values/5
35         public void Delete(int id)
36         {
37         }
38     }
39 }
```

Legarea parametrilor (parameter binding) in ASP.NET Web API atunci cand API-ul apeleaza metodele din Controller, utilizeaza urmatoarele doua reguli de baza:

- Daca tipul de date al parametrului este unul **simplu** (int, bool, double, decimal, GUID, DateTime, string) atunci API-ul Web incearca sa ia valoarea acestui parametru din **URI**
- Pentru tipuri complexe de date (ex: obiecte – Student in exemplul nostru) API-ul Web incearca sa citeasca valoarea parametrilor din **body**

Astfel, daca avem un parametru care are un tip primitiv de date si nu dorim ca acesta sa fie preluat din URI, conform primei reguli de sus, atunci trebuie sa adaugam **[FromBody]** in fata tipurilor primitive de date in metodele din controller-ul API-ului Web.

Daca avem un parametru care are un tip complex de date si nu dorim ca acesta sa fie preluat din body, ci din URI, atunci trebuie sa adaugam **[FromUri]** in fata acestuia.

OBSERVATIE:

/! De asemenea, se poate observa, pentru fiecare metoda in parte, denumirea acesteia (Get, Post, Put, Delete). **Modul in care este denumita o metoda joaca un rol foarte important.** Numele metodelor trebuie sa corespunda verbelor HTTP sau sa inceapa cu verbul HTTP corespunzator. De exemplu, **metoda Get** se poate numi: **Get()**, **get()**, **GET()**, **GetStudents()**, **GetAllArticles()**, etc. La fel se procedeaza pentru toate metodele existente.

/! A doua varianta pe care o putem utiliza atunci cand denumim metode este de a atribui acestora orice nume, dar trebuie **obligatoriu** sa utilizam inaintea metodei, la fel ca in cazul unui Controller MVC, verbul HTTP corespunzator. De exemplu, pentru metoda de creare a unui student, utilizam verbul **[HttpPost]** inaintea metodei, iar in acest moment metoda poate avea orice denumire (ex: **CreateNewStudent**).

Pasul 4 – Realizarea operatiilor CRUD (Create, Read, Update, Delete) asupra entitatii Student

```
public class StudentController : ApiController
{
    private StudentDbContext db = new StudentDbContext();

    // get all students
    public Student[] Get()
    {
        var students = from student in db.Students
                        orderby student.Name
                        select student;

        return students.ToArray();
    }

    // get one student
    public Student Get(int id)
    {
        Student student = db.Students.Find(id);
        return student;
    }

    // create student
    public Student Post([FromBody]Student s)
    {
        try
        {
            db.Students.Add(s);
            db.SaveChanges();
            return s;
        }
        catch (Exception e)
        {
            return new Student();
        }
    }

    // update student
    public Student Put(int id, [FromBody] Student requestStudent)
    {
        try
        {
            Student student = db.Students.Find(id);
            student.Name = requestStudent.Name;
            student.Email = requestStudent.Email;
            student.CNP = requestStudent.CNP;
            db.SaveChanges();
            return student;
        }
    }
}
```

```

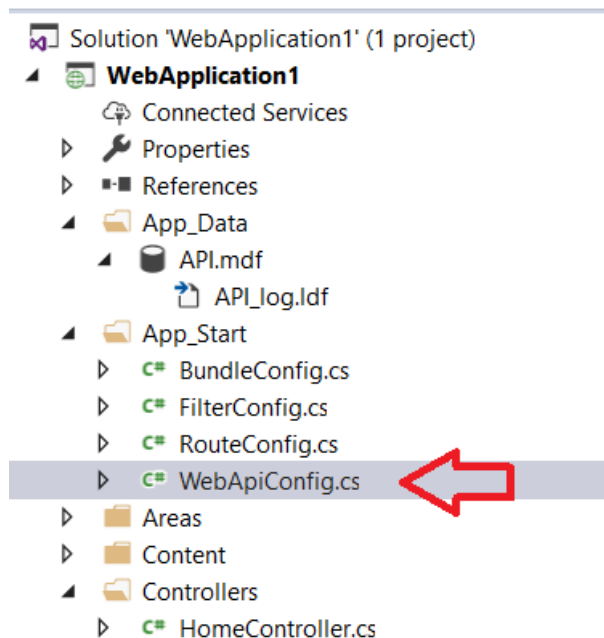
        catch (Exception e)
        {
            return new Student();
        }
    }

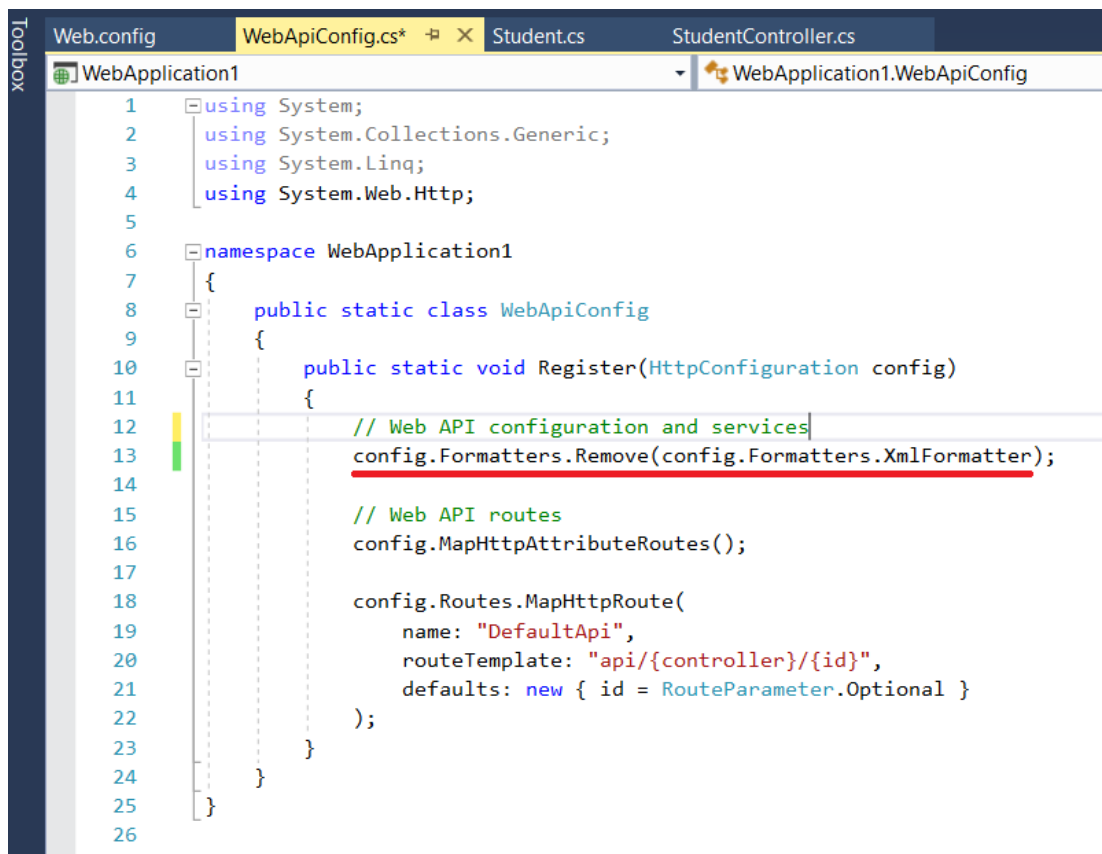
    // delete student
    public Student Delete(int id)
    {
        Student student = db.Students.Find(id);
        db.Students.Remove(student);
        db.SaveChanges();
        return new Student();
    }
}

```

Pasul 5 – modificarea configuratiei pentru a ne asigura ca prelucram JSON si nu XML

De cele mai multe ori, by default, datele sunt in format XML. Pentru a schimba aceasta configuratie astfel incat sa returneze JSON, trebuie sa atasam urmatoarea secventa de cod in fisierul **WebApiConfig.cs**, dupa cum urmeaza:



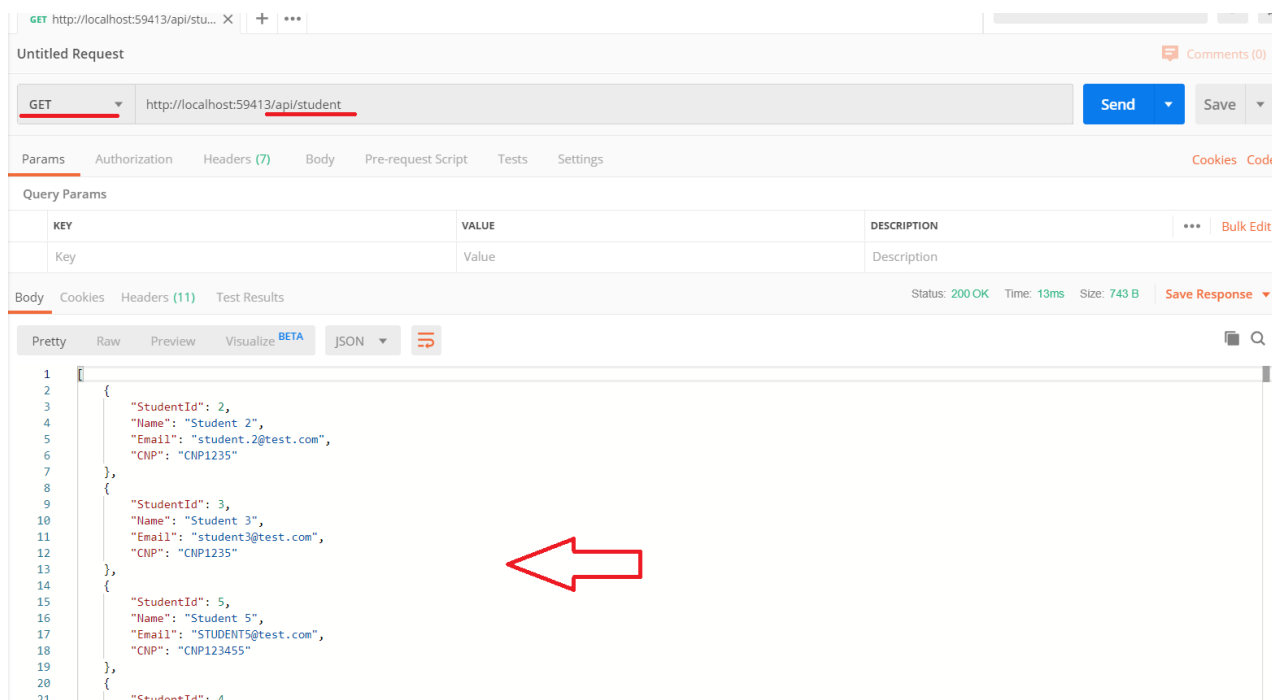


```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web.Http;
5
6 namespace WebApplication1
7 {
8     public static class WebApiConfig
9     {
10         public static void Register(HttpConfiguration config)
11         {
12             // Web API configuration and services
13             config.Formatters.Remove(config.Formatters.XmlFormatter);
14
15             // Web API routes
16             config.MapHttpAttributeRoutes();
17
18             config.Routes.MapHttpRoute(
19                 name: "DefaultApi",
20                 routeTemplate: "api/{controller}/{id}",
21                 defaults: new { id = RouteParameter.Optional }
22             );
23         }
24     }
25 }
26
```

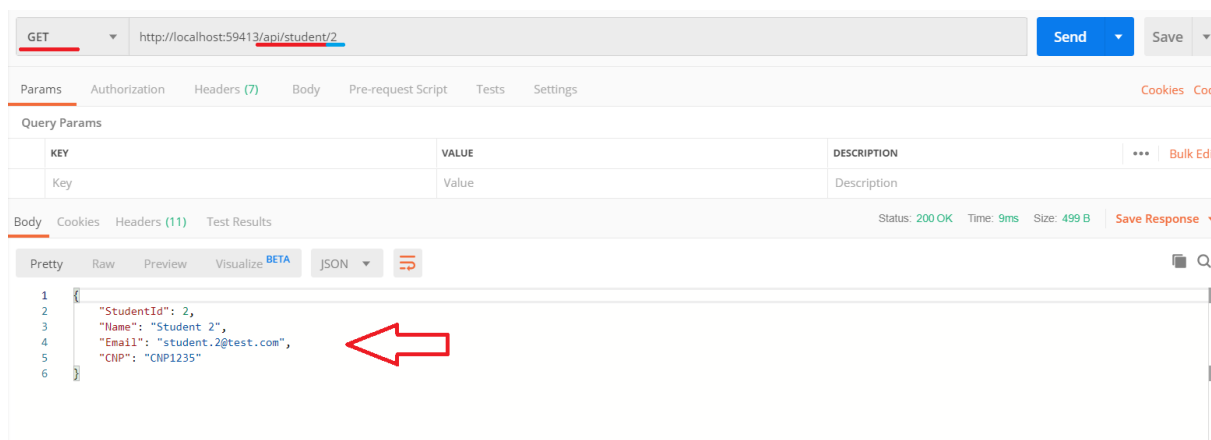
Consumarea API-ului

Pentru consumarea REST API-ului o sa folosim pe post de client Postman (<https://www.getpostman.com/>) – aplicatie cu ajutorul careia se pot face request-uri HTTP (implicit catre un API).

Request de tip **GET** pentru preluarea tuturor studentilor:



Request de tip **GET** pentru preluarea unui singur student:



Request de tip **POST** pentru crearea unui student:

POST <http://localhost:59413/api/student> Send Save

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies Code

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

```
1 {
2   "Name": "Student 1",
3   "Email": "student1@test.com",
4   "CNP": "CNP1234"
5 }
```

Body Cookies Headers (11) Test Results Status: 200 OK Time: 51ms Size: 494 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "StudentId": 6,
3   "Name": "Student 1",
4   "Email": "student1@test.com",
5   "CNP": "CNP1234"
6 }
```

Request de tip **PUT** pentru modificarea datelor unui student:

PUT <http://localhost:59413/api/student/2> Send Save

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies Code

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

```
1 {
2   "CNP": "CNP_MODIFICAT",
3   "Email": "student_2@test.com",
4   "Name": "Student 2 - Update"
5 }
```

Body Cookies Headers (11) Test Results Status: 200 OK Time: 1829ms Size: 514 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "StudentId": 2,
3   "Name": "Student 2 - Update",
4   "Email": "student_2@test.com",
5   "CNP": "CNP_MODIFICAT"
6 }
```

Request de tip **DELETE** pentru stergerea unui student:

DELETE <http://localhost:59413/api/student/7> Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (11) Test Results Status: 200 OK Time: 121ms Size: 471 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "StudentId": 0,
3   "Name": null,
4   "Email": null,
5   "CNP": null
6 }
```