```csharp
/*
In this project we will implement a RESTFul Web API, i.e. a program that:
1) exposes an API in form of a set of HTTP endpoints
2) the different HTTP methods are used coherently for different kind of operations, mapping the CRUD (for example, POST for the addition of resources)

This app follows the MVC pattern:
Controller: classes to handle the routing and the application logic
Model: classes to handle the domain data, the domain logic (validations on the data) and the storing (database?)
View: classes to handle the UI (the representation/formatting of the model and the interaction with the user)
Every type of the application should have a single role.
If a class acts as a controller (example: a PersonController to handle a list of people), it shouldn't contain data of the domain (for example properties Name, Surname, etc.)

*/

var builder = WebApplication.CreateBuilder(args);

// In order to enable the controller system, we must add the controllers as services:
builder.Services.AddControllers();

var app = builder.Build();

// and we must add the controllers middleware:
app.MapControllers();

app.Run();

```

```csharp
using Microsoft.AspNetCore.Mvc;
using S05_P03_MVC1.Models;

namespace S05_P03_MVC1.Controllers;

// The attribute [ApiController] makes automatically the filling of the input parameters from the body (see the Edit() method),
// and the automatic generation of a 400 error if the model is not valid.
[ApiController]
public class SuperheroesController : ControllerBase
{
    /*
    This is the lifecycle of a controller:
    1) Given the request's path, the system finds the controller and the method mapped to that path
    2) The controller is instantiated
    3) If it derives from ControllerBase, its web properties are filled (Connection, HttpContext, Request, Response, ...)
    4) The input is instantiated and filled with the values from the request (could be the body, the path, ...)
    5) If it derives from ControllerBase, the input is validated and the ModelState property is filled
    6) The method is invoked
    */

    // We want to implement a CRUD in memory.
    // This does not work, because a new controller instance is created for EVERY request:
    // private readonly List<Superhero> _superHeroes;
    // public SuperheroesController() {
    //     _superHeroes = new();
    // }

    // For this simple exercise, we could use static lists:
    private static readonly List<Superhero> _superHeroes;
    static SuperheroesController() {
        _superHeroes = new();
    }

    // Through HttpGet, HttpPost ecc. attributes I can set the allowed HTTP verbs
    // (if the client uses another one, a 405 error is returned).
    // In the attribute I can specify the path.
    [HttpPost("/superheroes/add")]
    public void Add(Superhero model) {
        _superHeroes.Add(model);
    }

    [HttpGet("/superheroes/all")]
    public List<Superhero> GetAll() {
        return _superHeroes;
    }

    // Sometimes a path segment must be parametrized.
    // For example, we want all the call to /superheroes/edit/<the-id-of-the-superhero> to be processed by the below method.
    // We can set a parametrized segment in braces: {}
    [HttpPut("/superheroes/edit/{id}")]
    // The method must have a parameter with the same name:
    public void Edit(int id, Superhero model) {
        // Simplest idea: I find the corresponding model and I replace it with the input:
        var index = _superHeroes.FindIndex(s => s.Id == id);
        if (index == -1) {
            Response.StatusCode = 404;
            return;
        }
        _superHeroes[index] = model;

        // Nonetheless, it's not a good idea.
        // Usually an API exposes multiple PUT operations, to edit different parts of a model.
        // Therefore it's better to map the exact properties on the saved model:
        var saved = _superHeroes.FirstOrDefault(s => s.Id == id);
```

```
            if (saved == null) {
                Response.StatusCode = 404;
                return;
            }
            saved.Nickname = model.Nickname;
            saved.SecretName = model.SecretName;
            saved.Assets = model.Assets;
            saved.CanFly = model.CanFly;
            saved.Birth = model.Birth;
            saved.City = model.City;
            saved.Gender = model.Gender;
        }

        [HttpDelete("/superheroes/delete/{id}")]
        public void Delete(int id) {
            _superHeroes.RemoveAll(s => s.Id == id);
            // Here a 404 is not required:
            // If a deletion is requested for a model that does not exist on the system,
            // the goal of deleting it is already satisfied.
            // No error required.
        }
    }
```