Een CRUD-Controller maken in ASP.NET Core Web API

We maken in deze oefening een Controller class in een Asp.Net Core (3.1) Web API waarmee we een ToDo lijst kunnen beheren. De ToDo lijst kan worden opgevraagd, de nog openstaande ToDo items kunnen worden opgevraagd, of een ToDo met een bepaalde Id. Verder is het ook mogelijk om een ToDo Item te wijzigen en de Verwijderen.

Kloon het project **MyFirstWEBAPI** van github:

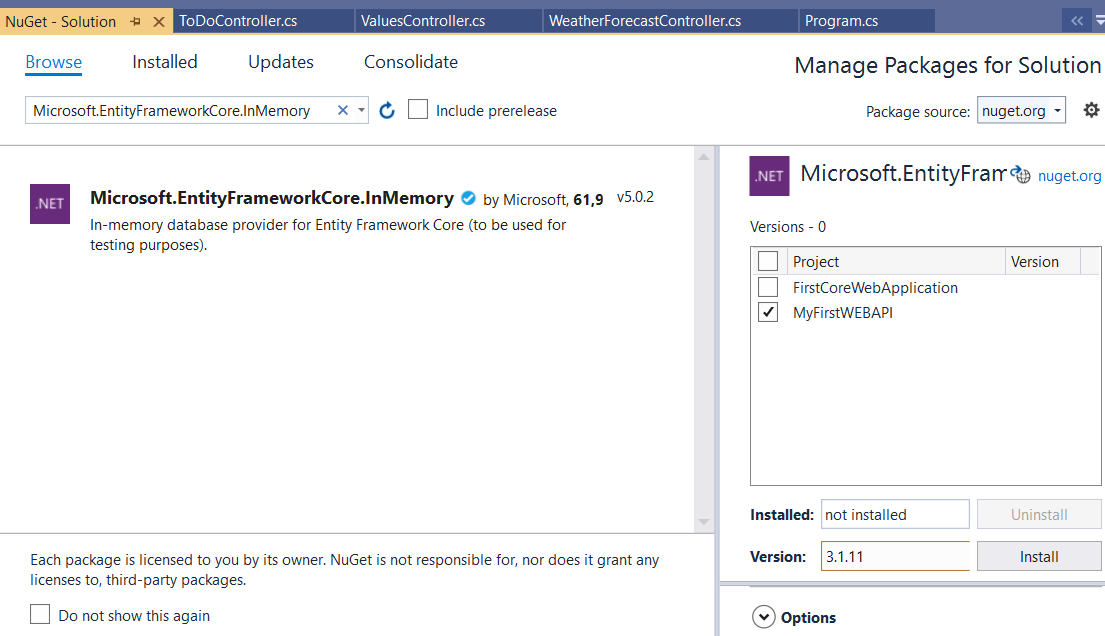
<https://github.com/CSharpSyntraWest/FirstCoreWebApplication/tree/master/MyFirstWEBAPI>

Om te beginnen zullen via een **InMemory** database gegevens bijhouden end us een database simuleren, later schakelen we over naar een echte SqlServer database.

* **Installeer de NuGet packages (versies 3.1.11)**

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.InMemory



* **Voeg een nieuwe folder met naam Models toe in het project en daarin een class ToDo:**

public enum PrioriteitsNiveau

{

Laag,

Middel,

Hoog

}

public class ToDo

{

public int Id { get; set; }

public string Titel { get; set; }

public string Omschrijving { get; set; }

public PrioriteitsNiveau Prioriteit { get; set; }

public bool IsGedaan { get; set; }

}

**Voeg onder de folder Models eveneens een class ToDoDbContext toe die afgeleid is van DbContext (voeg bovenaan een using directive toe naar de namespace Microsoft.EntityFrameworkCore** )

Zet hierin de volgende code :

using Microsoft.EntityFrameworkCore;

namespace MyFirstWEBAPI.Models

{

public class ToDoDbContext : DbContext

{

public ToDoDbContext(DbContextOptions<ToDoDbContext> options) : base(options) { }

public DbSet<ToDo> ToDos { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder

optionsBuilder)

{

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<ToDo>(toDo =>

{

toDo.Property(c => c.Id)

.ValueGeneratedOnAdd(); // autonummering

toDo.HasKey(s => s.Id);

toDo.Property(s => s.Titel).IsRequired().HasMaxLength(50);

toDo.Property(s => s.Omschrijving).IsRequired().HasMaxLength(255);

});

base.OnModelCreating(modelBuilder);

}

}

}

* **Zet in Startup.cs in de ConfigureServices te volgende code :**

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ToDoDbContext>(builder =>

builder.UseInMemoryDatabase("ToDosDatabase"));

services.AddControllers();

}

* **Voeg een nieuwe Lege API Controller met naam ToDosController toe onder de folder Controllers en zet hierin de volgende code:**

using System.Collections.Generic;

using System.Linq;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyFirstWEBAPI.Models;

namespace MyFirstWEBAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ToDosController : ControllerBase

{

private readonly ToDoDbContext \_context;

public ToDosController(ToDoDbContext context)

{

\_context = context;

}

// GET: api/ToDos

[HttpGet]

public ActionResult<IEnumerable<ToDo>> GetToDos()

{

return \_context.ToDos.ToList();

}

// GET: api/ToDos/5

[HttpGet("{id}")]

public ActionResult<ToDo> GetToDo(int id)

{

var toDo = \_context.ToDos.Find(id);

if (toDo == null)

{

return NotFound();

}

return toDo;

}

// PUT: api/ToDos/1

[HttpPut("{id}")]

public IActionResult PutToDo(int id, ToDo toDo)

{

if (id != toDo.Id)

{

return BadRequest();

}

\_context.Entry(toDo).State = EntityState.Modified;

try

{

\_context.SaveChanges();

}

catch (DbUpdateConcurrencyException)

{

if (!ToDoExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/ToDos

[HttpPost]

public ActionResult<ToDo> PostToDo(ToDo toDo)

{

\_context.ToDos.Add(toDo);

\_context.SaveChanges();

return CreatedAtAction("GetToDo", new { id = toDo.Id }, toDo);

}

// DELETE: api/ToDos/1

[HttpDelete("{id}")]

public ActionResult<ToDo> DeleteToDo(int id)

{

var toDo = \_context.ToDos.Find(id);

if (toDo == null)

{

return NotFound();

}

\_context.ToDos.Remove(toDo);

\_context.SaveChanges();

return toDo;

}

private bool ToDoExists(int id)

{

return \_context.ToDos.Any(e => e.Id == id);

}

}

}

* **Start de MyfirstWebAPI op met IIS Express en navigeer naar url**

<http://localhost:55342/api/todos>

je krijgt een lege todo lijst te zijn (json) :

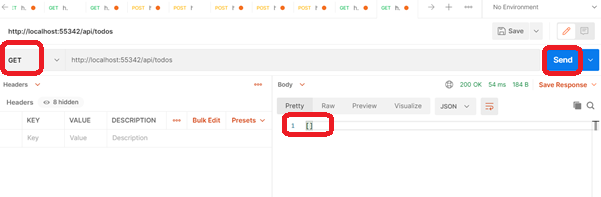
**[]**

Laat de browser open staan en open Postman

* **Voeg een tabblad toe om een HTTP GET Request te sturen naar dezelfde url:**

Je krijgt als HTTP Response hetzelfde als in de Browser:

[]



* Om een nieuwe ToDo item toe te voegen:
* Stuur nu via Postman een HTTP **Post** Request naar dezelfde URL, maar

Zet eerst de volgende json code in de **Body** van de http Request (zet de body van de request op Raw en daarnaast op JSON):

{

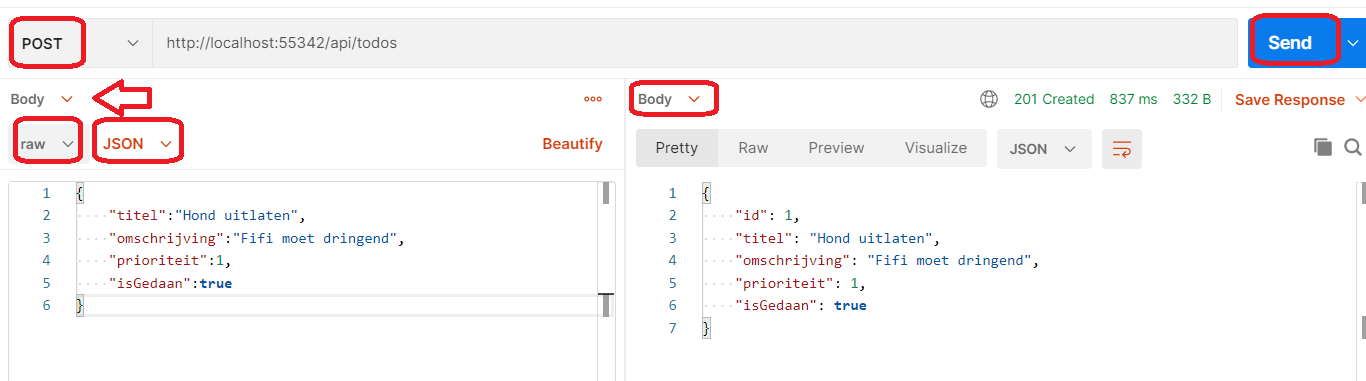
    "titel":"Hond uitlaten",

    "omschrijving":"Fifi moet dringend",

    "prioriteit":1,

    "isGedaan":**true**

}



Aan de rechterkant zie de body van de http Response. Er is automatisch een id gegenereerd (autonummering) en deze wordt eveneens teruggegeven

* **Voeg nog een 2de ToDo** item toe met een http **POST** request naar hetzelfde url:

{

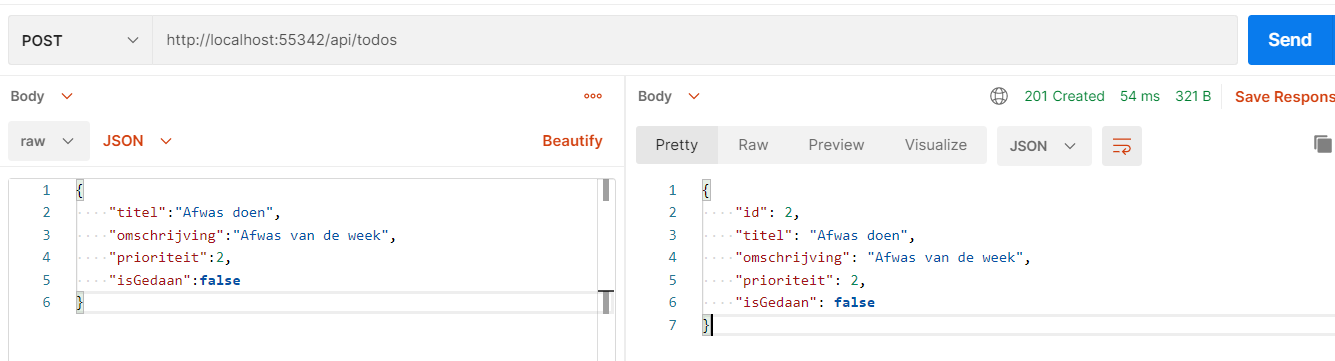
    "titel":"Afwas doen",

    "omschrijving":"Afwas van de week",

    "prioriteit":2,

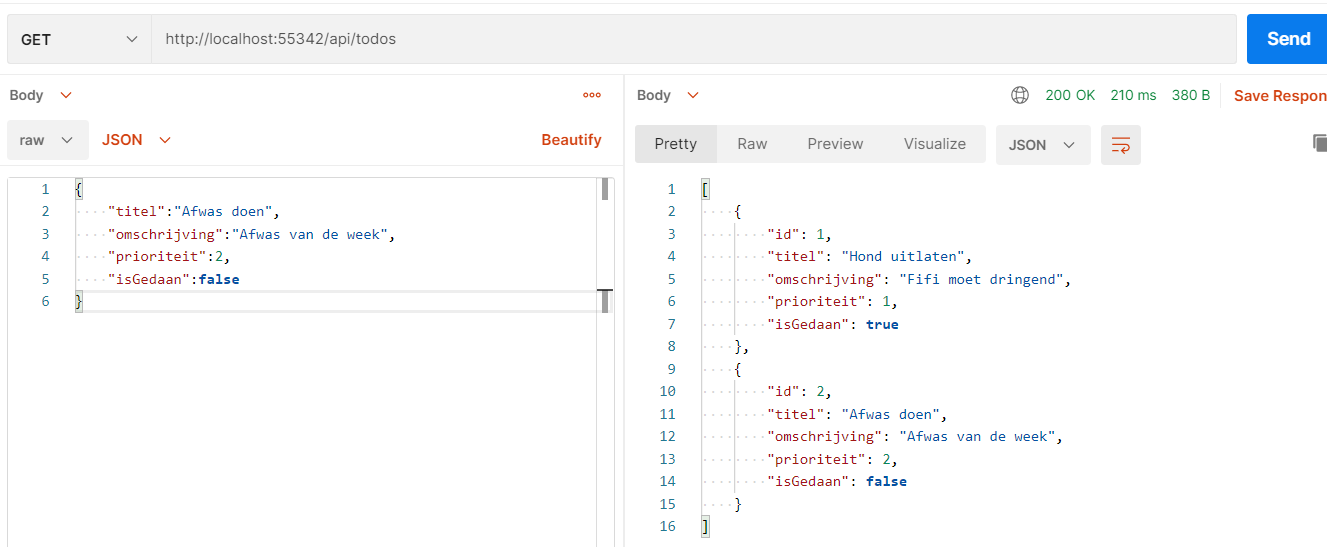
    "isGedaan":**false**

}



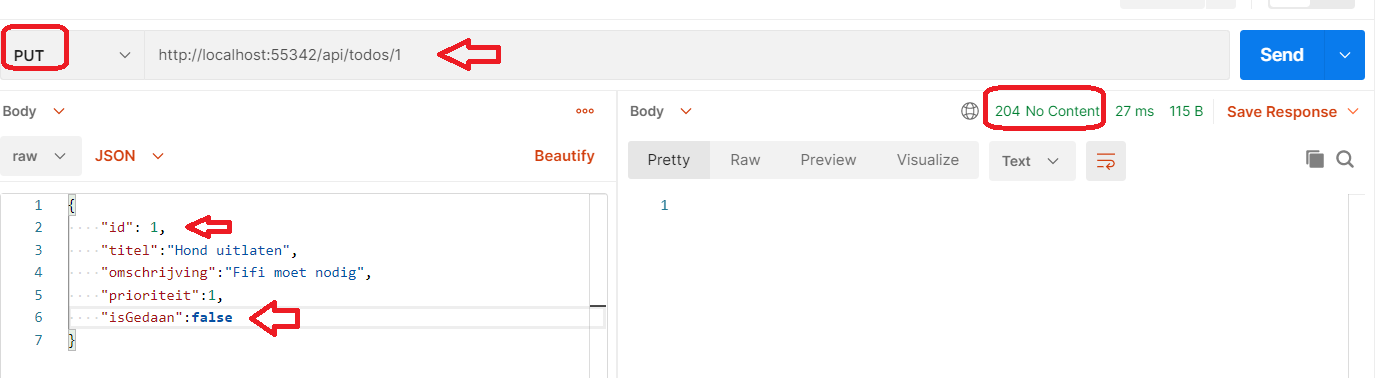
* **Stuur opnieuw een HTTP GET Request met Postman naar dezelfde url:**

Je krijgt als HTTP Response body nu de 2 ToDo items in Jason code:



* Om de **ToDo** met Id=1 te **wijzigen**, stuur met postman de volgende json body met een **http PUT request naar de volgende Url :**

<http://localhost:55342/api/todos>**/1**

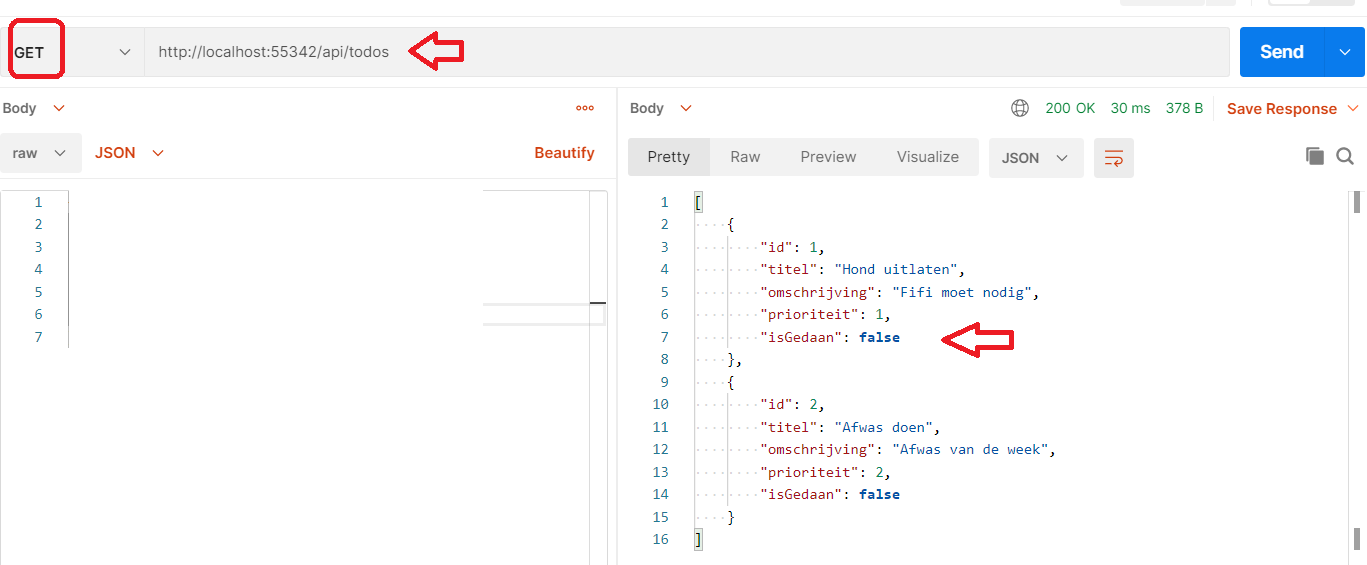


Deze http Response body is leeg, maar we krijgen wel een Success status (204 No content)

* **Stuur opnieuw een HTTP GET Request met Postman naar de url:** <http://localhost:55342/api/todos>

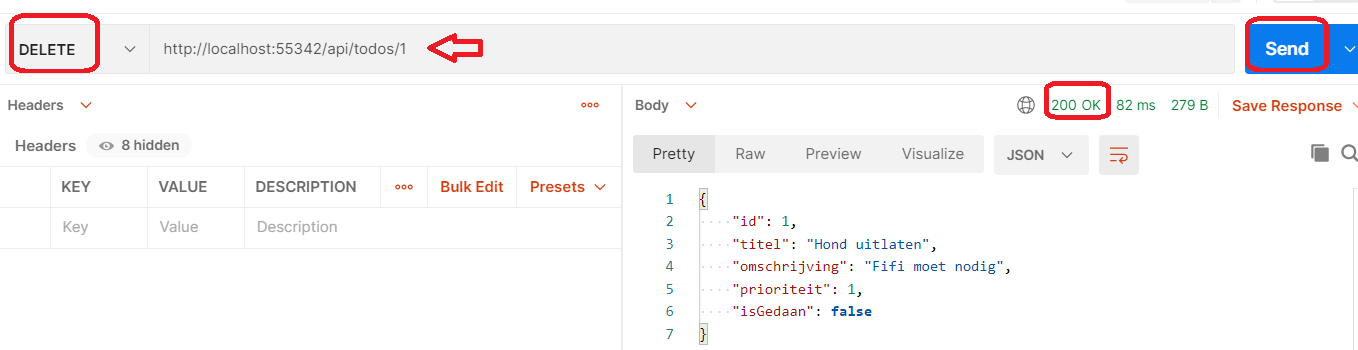
Je krijgt als HTTP Response body nu de 2 ToDo items in Json

Je zie dat bij ToDo met id 1 isGedaan nu op **false** is gezet:



* Om de **ToDo** met Id=1 te **verwijderen**, stuur met postman de volgende json body met een **http DELETE request naar de volgende Url :**

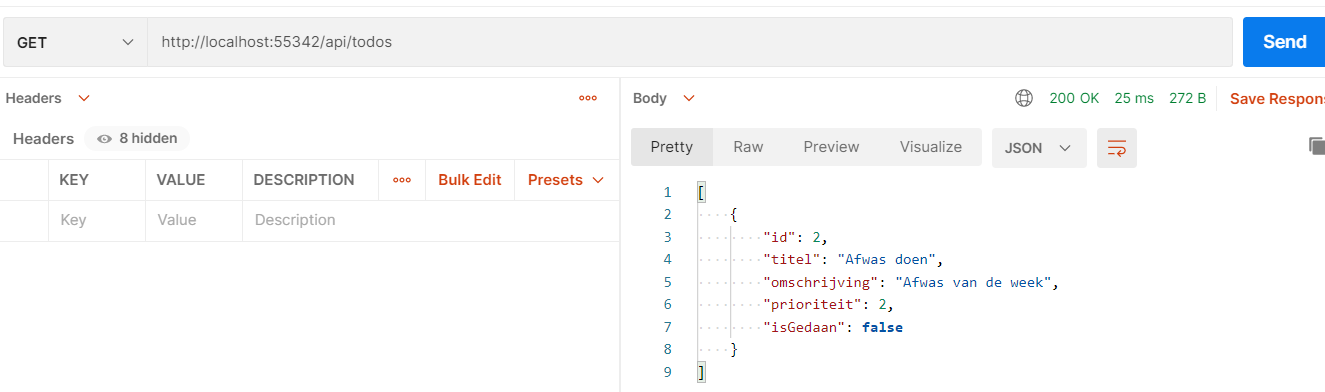
<http://localhost:55342/api/todos>**/1**



Als Http Response krijg je een 200 OK statuscode en eveneens het verwijderde item in de body

* **Stuur opnieuw een HTTP GET Request met Postman naar de url:** <http://localhost:55342/api/todos>

Je krijgt als HTTP Response body maar 1 ToDo item meer terug



**Om de database asynchroon aan te spreken en eveneens de action methoden in de ToDosController asynchroon te maken, voeg de volgende code in TodosController:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using MyFirstWEBAPI.Models;

namespace MyFirstWEBAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ToDosController : ControllerBase

{

private readonly ToDoDbContext \_context;

public ToDosController(ToDoDbContext context)

{

\_context = context;

}

// GET: api/ToDos

[HttpGet]

public async Task<ActionResult<IEnumerable<ToDo>>> GetToDos()

{

return await \_context.ToDos.ToListAsync();

}

// GET: api/ToDos/1

[HttpGet("{id}")]

public async Task<ActionResult<ToDo>> GetToDo(int id)

{

var toDo = await \_context.ToDos.FindAsync(id);

if (toDo == null)

{

return NotFound();

}

return toDo;

}

// PUT: api/ToDos/1

[HttpPut("{id}")]

public async Task<IActionResult> PutToDo(int id, ToDo toDo)

{

if (id != toDo.Id)

{

return BadRequest();

}

\_context.Entry(toDo).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ToDoExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/ToDos

[HttpPost]

public async Task<ActionResult<ToDo>> PostToDo(ToDo toDo)

{

\_context.ToDos.Add(toDo);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetToDo", new { id = toDo.Id }, toDo);

}

// DELETE: api/ToDos/1

[HttpDelete("{id}")]

public async Task<ActionResult<ToDo>> DeleteToDo(int id)

{

var toDo = await \_context.ToDos.FindAsync(id);

if (toDo == null)

{

return NotFound();

}

\_context.ToDos.Remove(toDo);

await \_context.SaveChangesAsync();

return toDo;

}

private bool ToDoExists(int id)

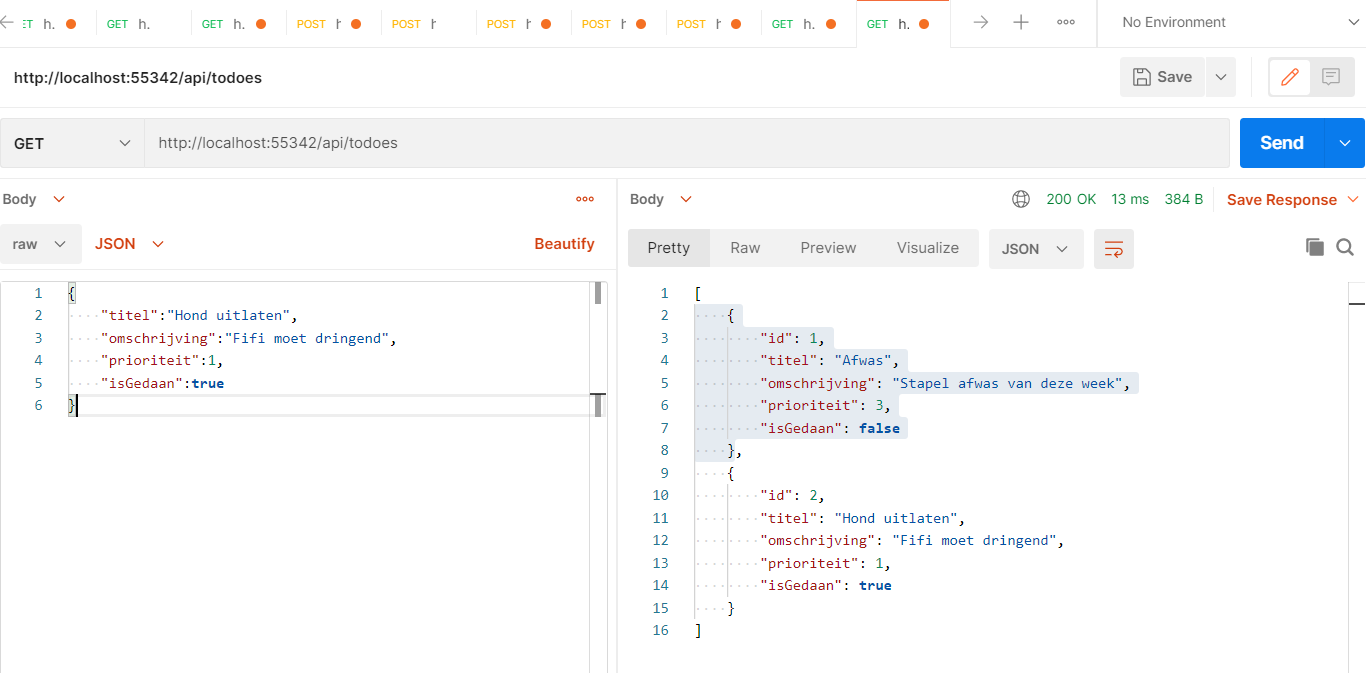
{

return \_context.ToDos.Any(e => e.Id == id);

}

}

}



[

    {

        "id": 1,

        "titel": "Afwas",

        "omschrijving": "Stapel afwas van deze week",

        "prioriteit": 3,

        "isGedaan": **false**

    },

    {

        "id": 2,

        "titel": "Hond uitlaten",

        "omschrijving": "Fifi moet dringend",

        "prioriteit": 1,

        "isGedaan": **true**

    }

]

* **Voeg onder de folder Controllers een nieuwe Empty API Controller toe met naam ToDoController:**

namespace MyFirstWEBAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ToDoController : ControllerBase

{

}

}

**Met deze controller gaan we CRUD operaties toelaten voor ToDo items**

We starten met het toevoegen van een nieuwe Action methode om een HttpGet request af te handelen:

# Referenties

<https://medium.com/imaginelearning/asp-net-core-3-1-microservice-quick-start-c0c2f4d6c7fa>