

PeopleApp

Setup

Project

- Voeg een nieuwe ASP.NET Core Web API applicatie toe aan de bestaande solution met de naam PeopleApp.Api.

Template	Configure	Additional info
<div><h3>Create a new project</h3><p>Recent project templates</p><ul style="list-style-type: none">ASP.NET Core Web APIClass LibraryASP.NET Core Web API (Blazor)ASP.NET Core Web API (Blazor) (New features)Blank Solution</div>	<div><h3>Configure your new project</h3><p>ASP.NET Core Web API C#</p><p>Project name PeopleApp.Api</p><p>Location C:\Users\20004343\source\repos\PeopleApp\PeopleApp.Api</p><p>Solution name ⓘ PeopleApp</p><p><input type="checkbox"/> Place solution and project in the same directory</p><p>Project will be created in "C:\Users\20004343\source\repos\PeopleApp\PeopleApp.Api"</p></div>	<div><h3>Additional information</h3><p>ASP.NET Core Web API C# Linux macOS</p><p>Framework ⓘ .NET 8.0 (Long Term Support)</p><p>Authentication type ⓘ None</p><p><input checked="" type="checkbox"/> Configure for HTTPS ⓘ</p><p><input type="checkbox"/> Enable container support ⓘ</p><p>Container OS ⓘ Linux</p><p>Container build type ⓘ Dockerfile</p><p><input checked="" type="checkbox"/> Enable OpenAPI support ⓘ</p><p><input type="checkbox"/> Do not use top-level statements ⓘ</p><p><input checked="" type="checkbox"/> Use controllers ⓘ</p><p><input type="checkbox"/> Enlist in .NET Aspire orchestration ⓘ</p><p>Aspire version ⓘ</p></div>

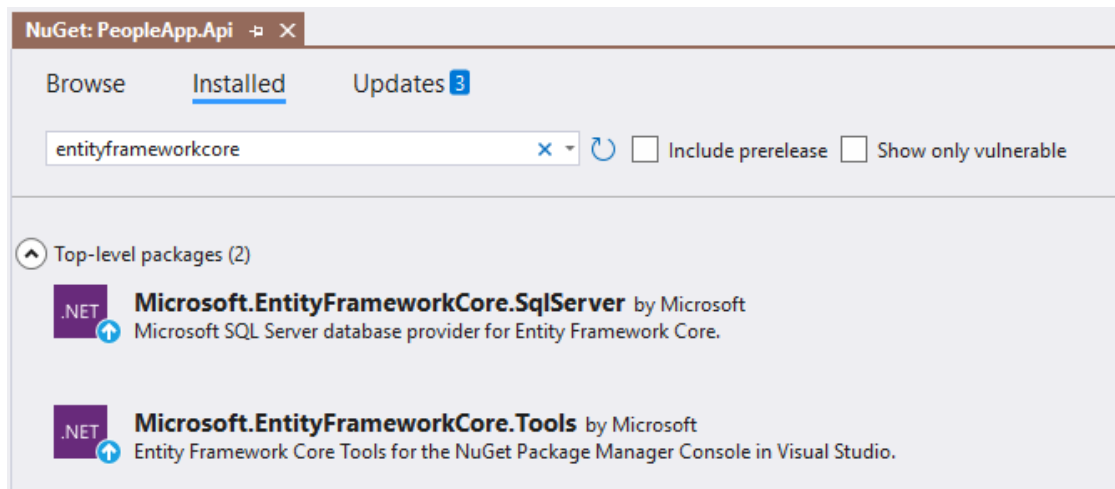
Folders & bestanden

- Verwijder de bestaande WeatherForecastController.cs en WeatherForecast.cs bestanden
- Zorg voor de volgende folder-structuur en bestanden:
PeopleApp.Api
 - | - Controllers (empty)
 - | - Data
 - | - ApplicationDbContext.cs
 - | - Entities
 - | - Department.cs
 - | - Location.cs
 - | - Person.cs

Packages

- Installeer onderstaande NuGet packages:
 - Microsoft.EntityFrameworkCore.SqlServer
 - Microsoft.EntityFrameworkCore.Tools

[!CAUTION] Let op dat je de juiste versie selecteert!



nuget packages to install

DbContext

- Vervolledig de AppDbContext klasse in de Data folder en registreer deze in de DI container van de applicatie. Gebruik hiervoor de PeopleConnection uit het appsettings.json bestand.

```
public class AppDbContext : DbContext { public  
AppDbContext(DbContextOptions options) : base(options) { }  
  
    public DbSet<Department> Departments { get; set; }  
    public DbSet<Location> Locations { get; set; }  
    public DbSet<Person> People { get; set; }  
  
}  
  
var connectionString =  
builder.Configuration.GetConnectionString("PeopleConnection");  
builder.Services.AddDbContext(options => {  
    options.UseSqlServer(connectionString);  
  
});
```

Entities

- Vervolledig de entity classes
- "Department.cs"

```
public class Department
{
    public long Id { get; set; }
    public string Name { get; set; }
    public IEnumerable<Person> People { get; set; }
}
```
- "Location.cs"

```
public class Location
{
    public long Id { get; set; }
    public string City { get; set; }
    public string State { get; set; }
    public IEnumerable<Person>? People { get; set; }
}
```
- "Person.cs"

```
public class Person
{
    public long Id { get; set; }
    public string Firstname { get; set; }
    public string Surname { get; set; }
    public Department Department { get; set; }
    public long DepartmentId { get; set; }
    public Location Location { get; set; }
    public long LocationId { get; set; }
}
```

Migrations

- Voer de volgende commando's uit in de Package Manager Console:
 - Add-Migration Initial
 - Update-Database

Seed Data

- Maak een nieuw class aan in de Data folder met de naam

DbInitializer.cs

```
public static class DbInitializer
{
    public static void SeedData(this WebApplication app)
    {
        using (var scope = app.Services.CreateScope())
        {
            AppDbContext context = scope.ServiceProvider.GetRequiredService();
            context.SeedPeopleData();
        }
    }

    private static void SeedPeopleData(this AppDbContext context)
    {
        context.Database.Migrate();
        if (!context.People.Any() && !context.Departments.Any() &&
            !context.Locations.Any())
        {
            Department d1 = new Department { Name = "Sales" };
            Department d2 = new Department { Name = "Development" };
            Department d3 = new Department { Name = "Support" };
            Department d4 = new Department { Name = "Facilities" };

            context.Departments.AddRange(d1, d2, d3, d4);
            context.SaveChanges();

            Location l1 = new Location { City = "Oakland", State = "CA" };
            Location l2 = new Location { City = "San Jose", State = "CA" };
            Location l3 = new Location { City = "New York", State = "NY" };
            context.Locations.AddRange(l1, l2, l3);

            context.People.AddRange(
                new Person
                {
                    Firstname = "Francesca", Surname = "Jacobs",
                    Department = d2, Location = l1
                },
```

```

        new Person
        {
            Firstname = "Charles", Surname = "Fuentes",
            Department = d2, Location = l3
        },
        new Person
        {
            Firstname = "Bright", Surname = "Becker",
            Department = d4, Location = l1
        },
        new Person
        {
            Firstname = "Murphy", Surname = "Lara",
            Department = d1, Location = l3
        },
        new Person
        {
            Firstname = "Beasley", Surname = "Hoffman",
            Department = d4, Location = l3
        },
        new Person
        {
            Firstname = "Marks", Surname = "Hays",
            Department = d4, Location = l1
        },
        new Person
        {
            Firstname = "Underwood", Surname = "Trujillo",
            Department = d2, Location = l1
        },
        new Person
        {
            Firstname = "Randall", Surname = "Lloyd",
            Department = d3, Location = l2
        },
        new Person
        {
            Firstname = "Guzman", Surname = "Case",
            Department = d2, Location = l2
        }
    });
    context.SaveChanges();
}
}

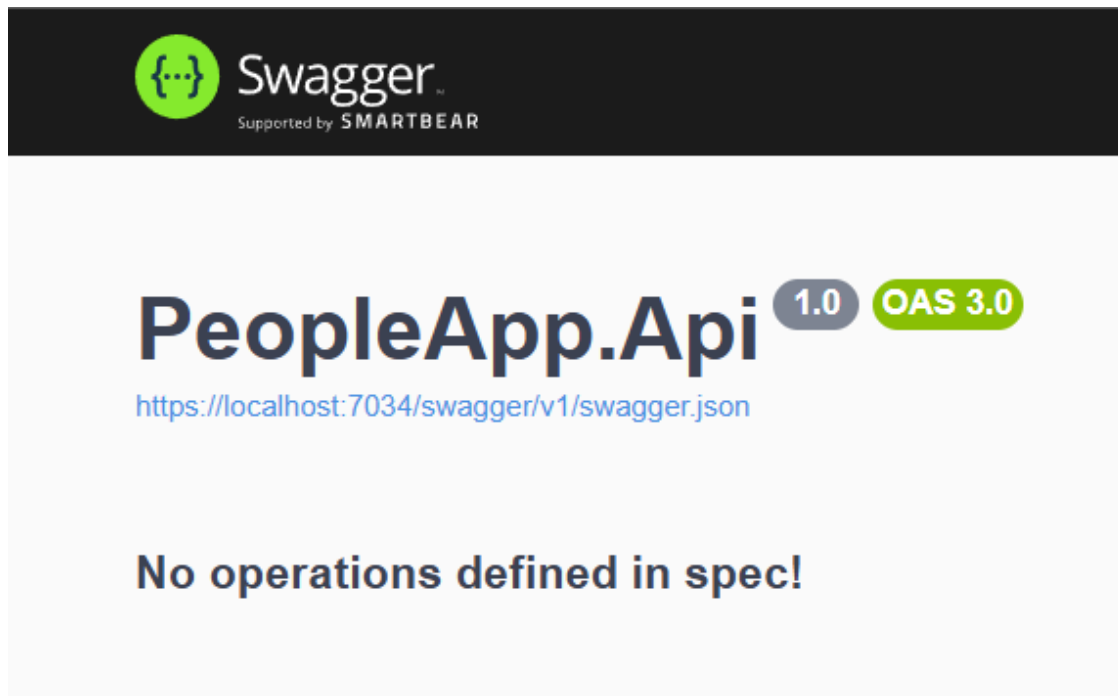
```

- Zorg nu dat deze methode wordt aangeroepen in de Program class net voor de applicatie wordt gestart:

```
app.SeedData();
```

```
app.Run();
```

De applicatie kan nu gestart worden zonder fouten **MAAR** de API heeft nog geen functionaliteit.

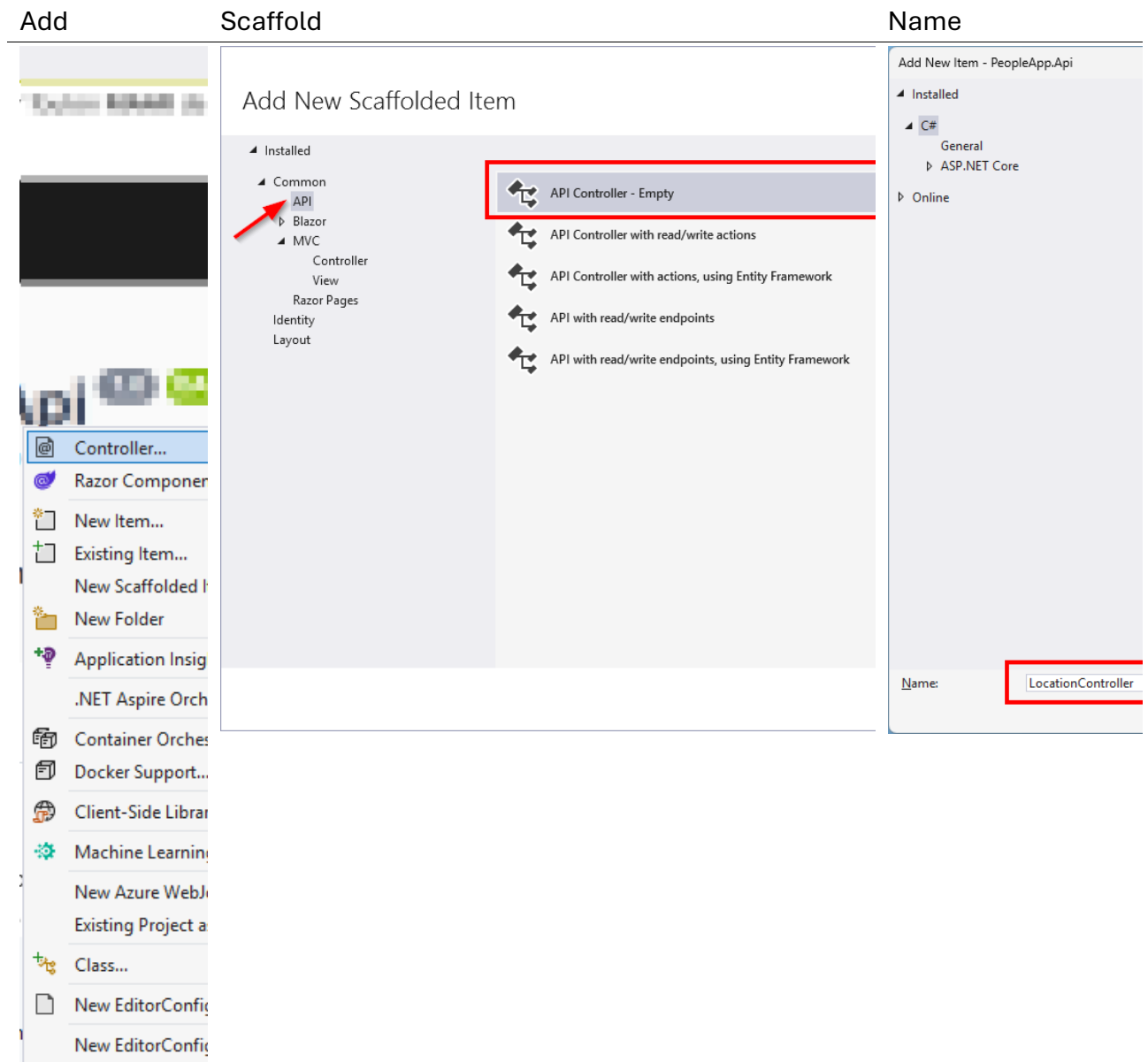


empty swagger page

LocationController

- Voeg een folder Apitoe aan de Controllers folder
- Voeg aan deze folder een nieuwe controller toe met de naam LocationController

Add	Scaffold	Name
-----	----------	------



[!TIP] Dankzij het `[Route("api/[controller]")]` attribuut en de extra endpoint configuratie in Program.cs zullen alle HTTP requests van `/api/location` naar deze controller worden geleid.

Dependency Injection

- Voeg de ApplicationDbContext toe aan de constructor van de LocationController en wijs deze toe aan een readonly instance variabele. ````cs private readonly ApplicationDbContext _context;

public LocationController(ApplicationDbContext context) { _context = context; } ```` ### Request Elke HTTP request heeft een URL en een *verb* (= methode).
- De URL bepaalt over welke data het gaat (bv. /api/location)
- De verb bepaalt welke operatie er moet worden uitgevoerd:
 - GET: haal 1 of meer data object op
 - POST: voeg een data object toe
 - PUT: pas een bestaand data object aan
 - DELETE: verwijder een bestaand data object

GetLocations

- Maak een GET methode aan in de LocationController die alle locaties ophaalt uit de database en teruggeeft als JSON object:

```
[HttpGet]  
public async Task<ActionResult<IEnumerable<Location>>> GetLocations()  
{  
    try  
    {  
        var locations = await _context.Locations.ToListAsync();  
        return Ok(locations);  
    }  
    catch (Exception)  
    {  
        return BadRequest();  
    }  
}
```

[!NOTE]

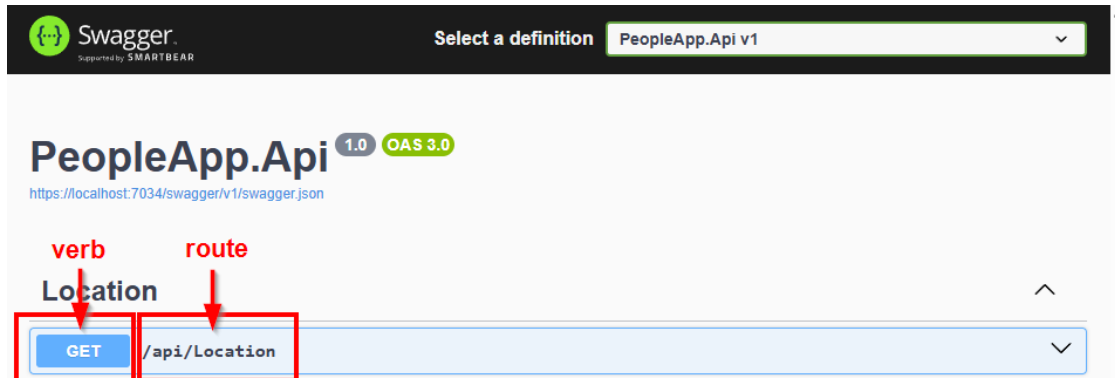
Deze functie geeft een ActionResult terug. Dit [geniet de voorkeur](#) ten opzicht van een IActionResult wanneer het type van het resultaat vast staat. In dit voorbeeld is het type altijd een IEnumerable<Location> en is een ActionResult dus de beste keuze.

[!NOTE]

Omdat een API steeds beschikbaar moet zijn voor inkomende request te behandelen zorgen we er voor dat de functies steeds asynchroon worden uitgevoerd!

Run!

Start de applicatie nogmaals en bekijk het resultaat!



swagger endpoint

JSON

De data die wordt geretourneerd in een RESTful web service heeft het JSON-formaat (JavaScript Object Notation): - Object aangeduid door {} - Lijst (reeks) aangeduid door [] - Eigenschappen aangeduid door "key": "value"

```
[
  {
    "id": 1,
    "city": "Oakland",
    "state": "CA",
    "people": null
  },
  {
    "id": 2,
    "city": "San Jose",
    "state": "CA",
    "people": null
  },
  {
    "id": 3,
    "city": "New York",
    "state": "NY",
    "people": null
  }
]
```

Postman

Een API kan (beperkt) getest worden met Swagger maar om uitgebreide testen te kunnen doen maken we beter gebruik van een andere tool om HTTP requests op te stellen.

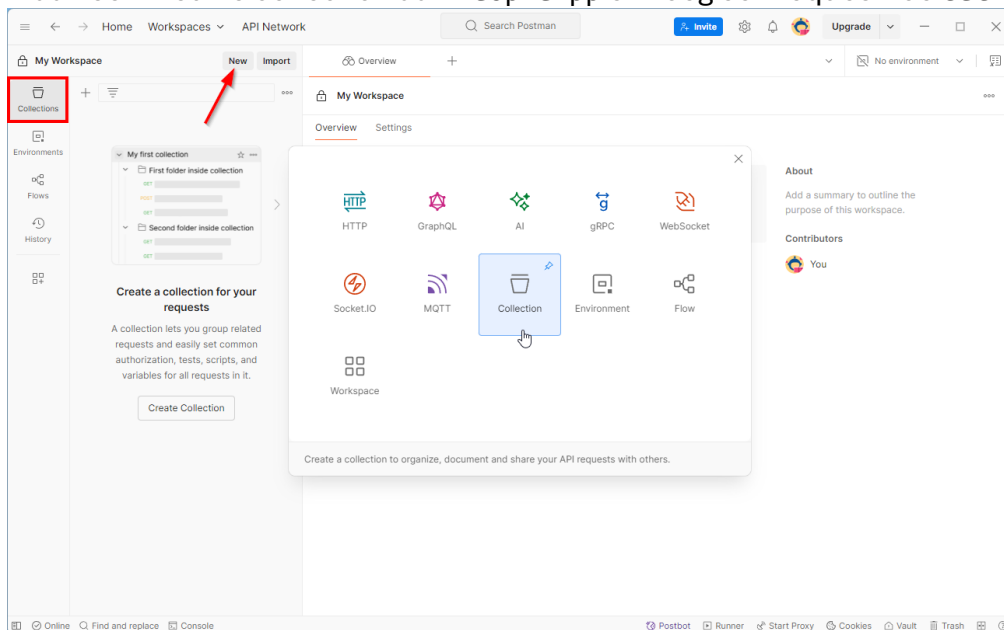
[!NOTE] Een GET request kan zelfs eenvoudig getest worden met de browser. Start je applicatie opnieuw en surf naar <https://localhost:7034/api/Location> om het resultaat te bekijken.

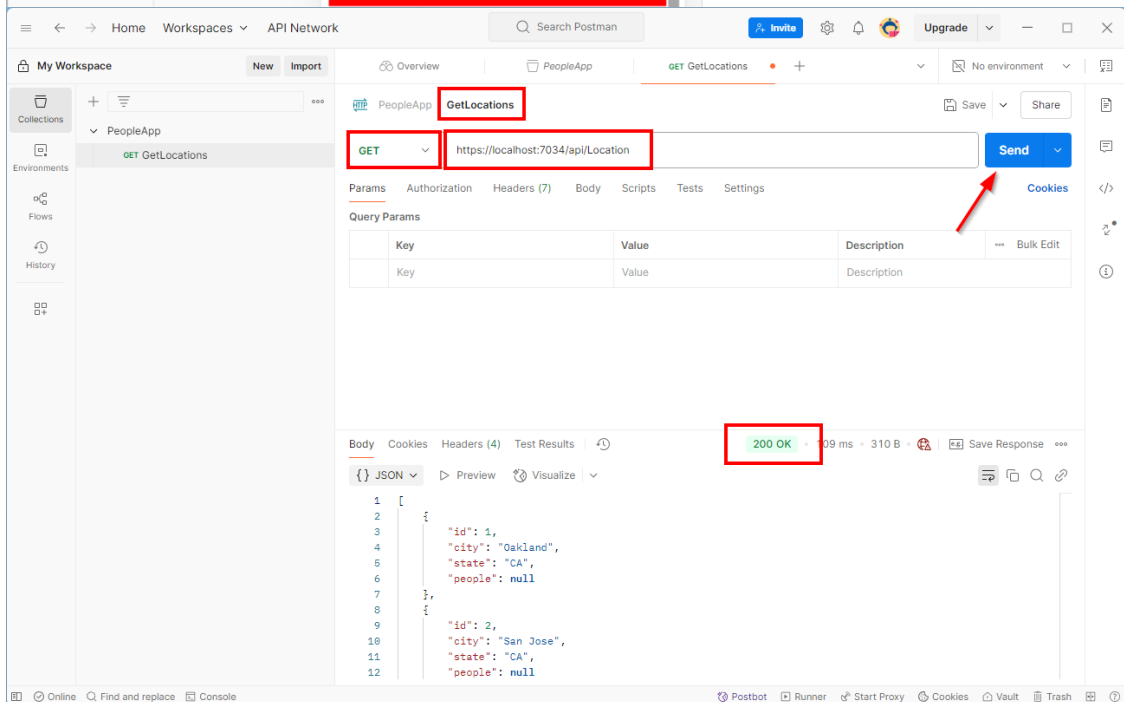
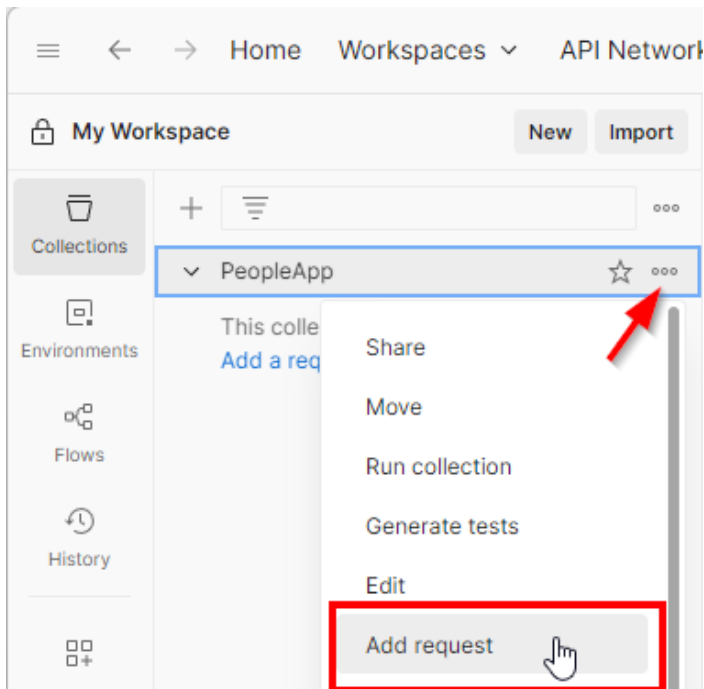
Download

Maak een account op <https://www.postman.com> en installeer de desktop agent via <https://www.postman.com/downloads>

Collections & request

- Maak een nieuwe collection aan PeopleApp en voeg een request toe GetLocations





GetDetails

- Voeg een GetDetails action toe aan de LocationController

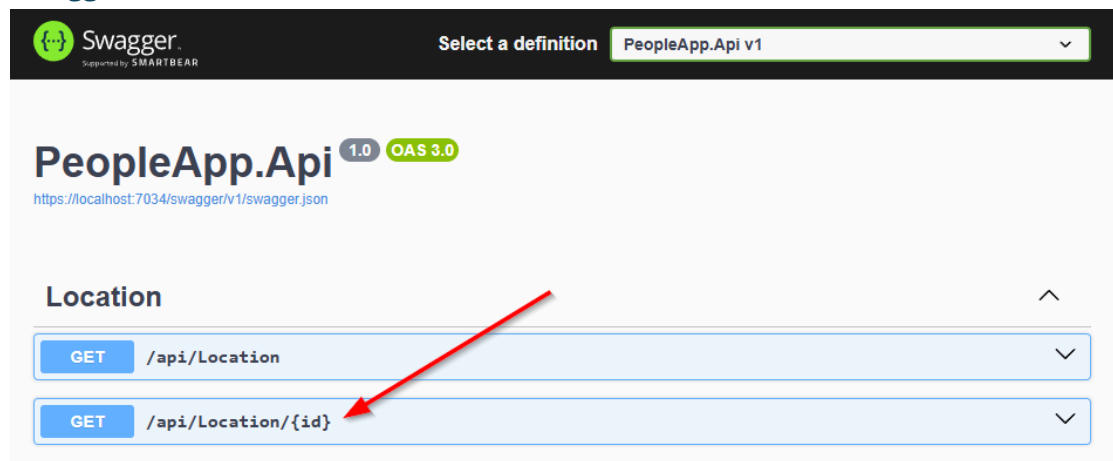
```
[HttpGet("{id}")]
public async Task<ActionResult<Location>> GetDetails(long id)
{
    Location? location = await _context.Locations.FindAsync(id);
    if (location == null)
    {
        return NotFound();
    }
    return Ok(location);
}
```

[!TIP] Het [HttpGet] attribuut (in combinatie met het [Route] attribuut van de controller) zorgt ervoor dat een HTTP GET request naar bv. /api/location/1 naar deze action worden geleid. - Als de locatie niet gevonden wordt geven we het resultaat van de NotFound methode terug. Dit gaat zorgen voor een 404 code in de response. - De Ok methode zorgt ervoor dat de HTTP response een 200 code gaat hebben en in de body een JSON-representatie van het model zal bevatten.

Run!

- Test de nieuwe action in je browser en met Swagger
- Maak ten slotte een nieuw request aan in de PeopleApp-collection van Postman

Swagger



swagger getdetails

Postman

The image displays two screenshots of the Postman API client interface, illustrating a successful GET request and a 404 Not Found error.

Top Screenshot: Successful GET Request

- Collection:** PeopleApp
- Environment:** No environment
- Request:** GET `https://localhost:7034/api/Location/2` (The ID '2' is highlighted with a red box).
- Response:** 200 OK (35 ms, 201 B). The response body is JSON:

```
1 {
2   "id": 2,
3   "city": "San Jose",
4   "state": "CA",
5   "people": null
6 }
```

Bottom Screenshot: 404 Not Found Error

- Collection:** PeopleApp
- Environment:** No environment
- Request:** GET `https://localhost:7034/api/Location/27` (The ID '27' is highlighted with a red box). A red arrow points to the URL with the text "Location met Id = 27 bestaat niet".
- Response:** 404 Not Found (19 ms, 325 B). The response body is JSON:

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-16.6.6",
3   "title": "Not Found",
4   "status": 404,
5   "traceId": "00-afda14c59f70d2368ed56d06aed425df-aedbca4ff135bb09-00"
6 }
```

AddLocation

Model

[!IMPORTANT]

Omdat we onze entities niet altijd willen blootstellen aan de buitenwereld kunnen we ook gebruik maken van specifieke request en response modellen.

- Maak een nieuwe folder Models aan in de root van de applicatie
- Voeg een nieuwe class CreateLocationRequest toe aan deze folder csharp

```
public class CreateLocationRequest {      public string City { get; set; }      public string State { get; set; } } ### Action
```
- Maak een nieuwe action aan in de LocationController die een nieuwe locatie toevoegt aan de database `` `cs

```
[HttpPost] public async Task<ActionResult> AddLocation(CreateLocationRequest request) { try { Location location = new Location { City = request.City, State = request.State }; await _context.Locations.AddAsync(location); await _context.SaveChangesAsync();
```

```
        return Ok(location);
    }
    catch (Exception)
    {
        return BadRequest();
    }
}
```

} `` ` > [!TIP] > - Het [HttpPost] attribuut (in combinatie met het [Route] attribuut van de controller) zorgt ervoor een HTTP POST request naar “/api/location” naar deze action worden geleid. > - Het parameter model wordt opgevuld met gegevens die MVC in de request body (json) vindt. > - De Ok methode zorgt ervoor dat de HTTP response een 200 code gaat terug geven en in de body een JSON-representatie van het output model zal bevatten.

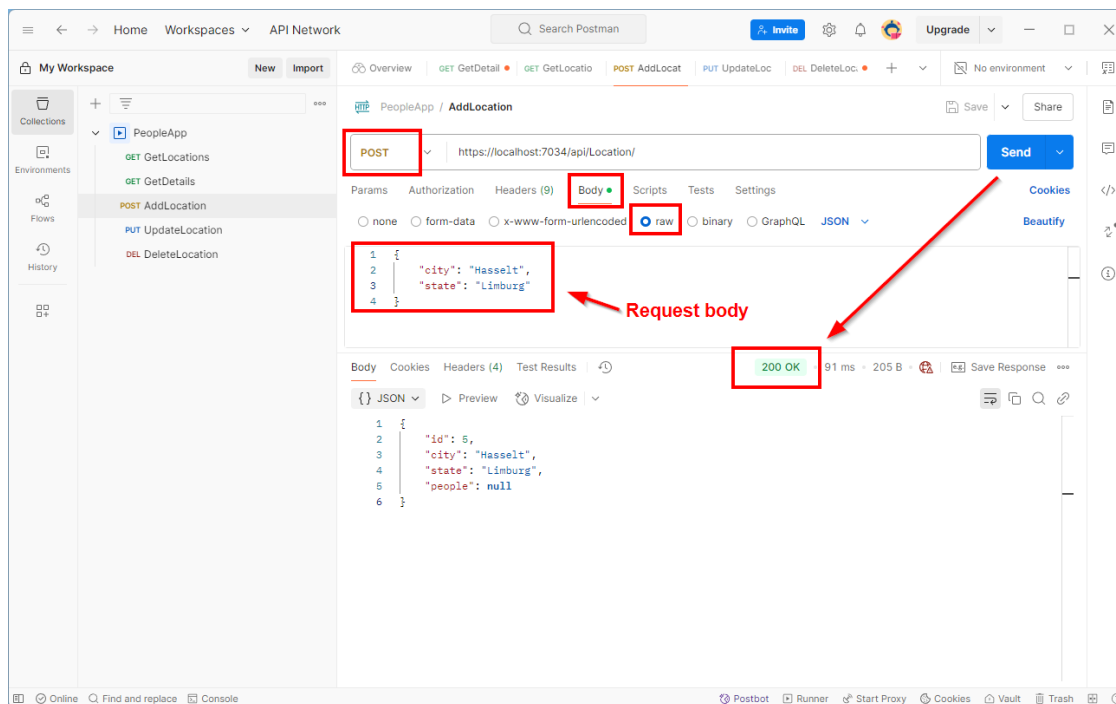
Run!

Swagger



add location

Postman



postman addlocation

UpdateLocation

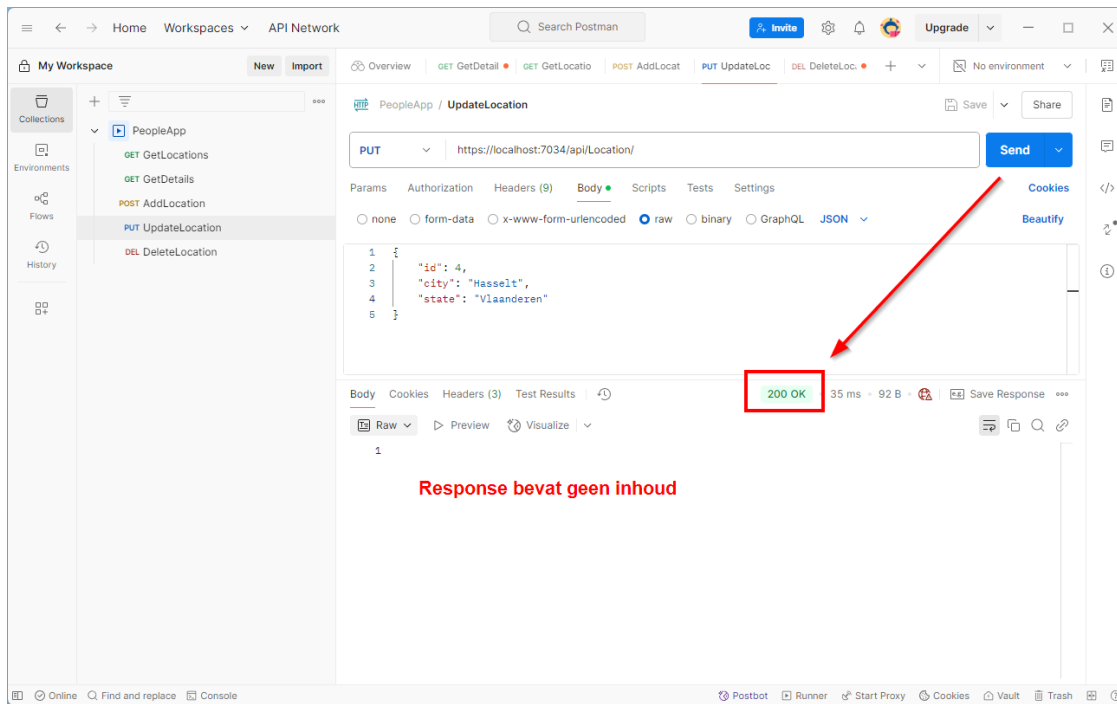
```
[HttpPut]
public async Task<ActionResult> UpdateLocation(Location model)
{
    try
    {
        _context.Locations.Update(model);
        int rows = await _context.SaveChangesAsync();
        if (rows == 0)
        {
            return NotFound();
        }
        return Ok();
    }
    catch (Exception)
    {
        return BadRequest();
    }
}
```

[!CAUTION] Zorg ervoor dat de People-property van de Location entity **null** is, indien dit een lege lijst is (new List<Person>()) zal EF Core namelijk de bestaande records in de database ‘updaten’ (en dus schrappen aangezien de lijst leeg is).

[!TIP] Indien er geen content in de response wordt verwacht kan je ook de NoContent methode gebruiken in plaats van de Ok methode. Dit zullen we toepassen in het volgende voorbeeld.

Run!

Postman



update request


DeleteLocation


```
[HttpDelete("{id}")]
public async Task<ActionResult> DeleteLocation(long id)
{
    try
    {
        Location? location = await _context.Locations.FindAsync(id);
        if (location == null)
        {
            return NotFound();
        }
        _context.Locations.Remove(location);
        await _context.SaveChangesAsync();
        return NoContent();
    }
    catch (Exception)
    {
        return BadRequest();
    }
}
```

[!TIP] - Het [HttpDelete] attribuut (in combinatie met het [Route] attribuut van de controller) zorgt ervoor een HTTP DELETE request naar bv. “/api/location/1” wordt geleid. - Als de locatie niet gevonden wordt geven we het resultaat van de NotFound methode terug. Dit gaat zorgen voor een 404 code in de response. - De NoContent methode zorgt ervoor dat de HTTP response een 204 code gaat hebben (met een lege body).

Run!

- Maak ook voor de PUT en de DELETE request een postman-request aan

 **Swagger**
Supported by SMARTBEAR


Select a definition **PeopleApp.Api v1** 


PeopleApp.Api


1.0 OAS 3.0


<https://localhost:7034/swagger/v1/swagger.json>


Location




GET /api/Location 

POST /api/Location 

PUT /api/Location 

GET /api/Location/{id} 

DELETE /api/Location/{id} 

update & delete request

Postman

The image displays two screenshots of the Postman API client interface, illustrating a DELETE request to a location endpoint.

Top Screenshot: The interface shows a workspace named "PeopleApp" with a collection of API endpoints. The selected endpoint is "DeleteLocation" (DELETE) with the URL `https://localhost:7034/api/Location/5`. The response is "204 No Content" (117 ms, 81 B). A red arrow points to the response status.

Bottom Screenshot: The interface shows the same workspace, but the selected endpoint is "DeleteLocation" (DELETE) with the URL `https://localhost:7034/api/Location/6`. The response is "404 Not Found" (69 ms, 325 B). A red arrow points to the URL, and a red text label "Location met Id = 6 bestaat niet" is displayed next to it. The response body is shown in JSON format:

```
1 {
2   "type": "https://tools.ietf.org/html/rfc9110#section-16.6.6",
3   "title": "Not Found",
4   "status": 404,
5   "traceId": "00-e9a9adc129477260db6cf24ae545ae0e-b27e412a690346a-00"
6 }
```

Best practices

[!IMPORTANT]

- **Gebruik bij voorkeur een repository- of een service-class om de database logica (dbContext) te encapsuleren.** Enkel voor demo doeleinden zoals dit labo wordt een dbContext rechtstreeks in een controller geïnjecteerd. - Gebruik altijd asynchrone methodes/functies in een API, zeker bij het werken met databases en/of bestanden. - Gebruik altijd de juiste HTTP status codes in je responses. - Gebruik altijd de juiste HTTP verbs voor de juiste acties.

Oefening

- Maak 2 nieuwe controllers aan voor de entities Department en Person
- Voeg de nodige actions toe om deze entities te beheren