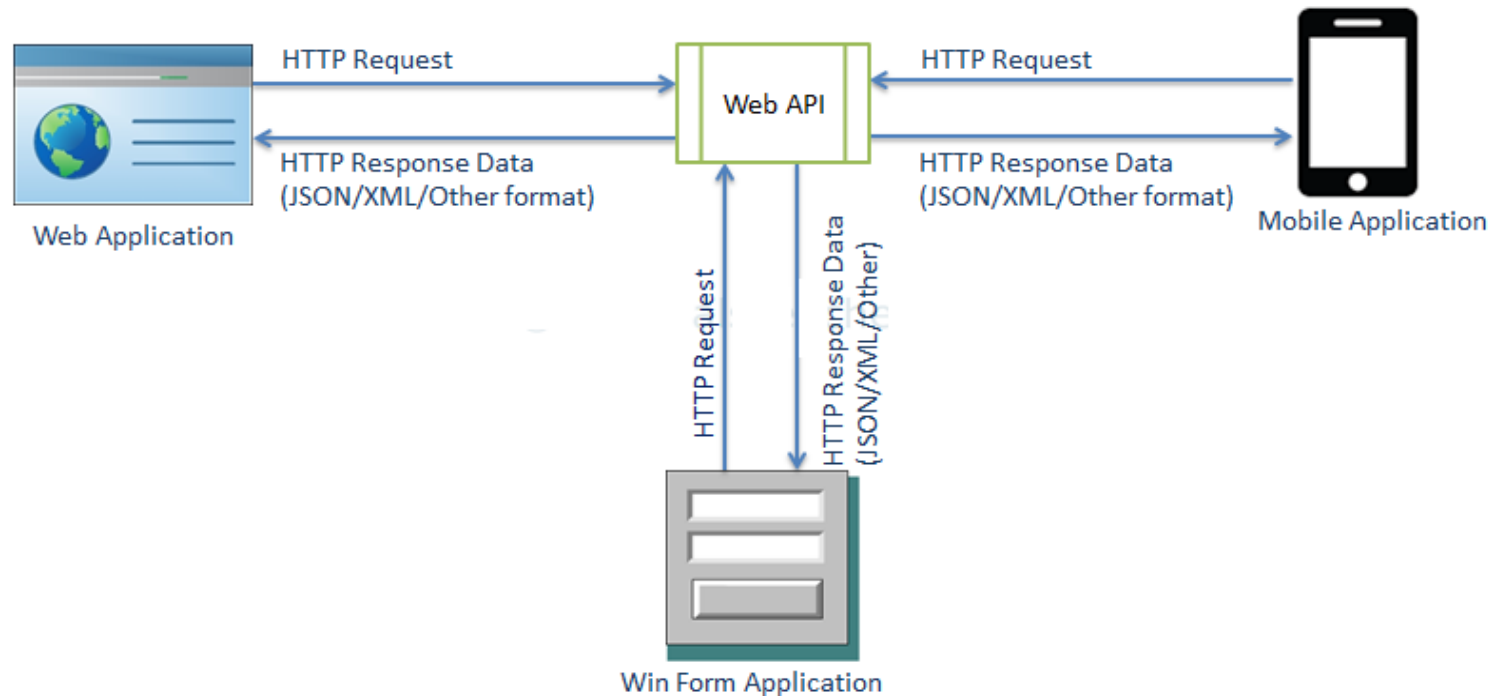


CHAPITRE 5 : ASP.NET Core Web API

Introduction

- Le Framework web ASP.NET Core MVC permet de créer des services web HTTP accessibles depuis n'importe quel client : navigateurs, appareils mobiles, applications IOT et autres.
- C'est une plate-forme idéale pour créer des services RESTful sous .NET Core.



RESTful API

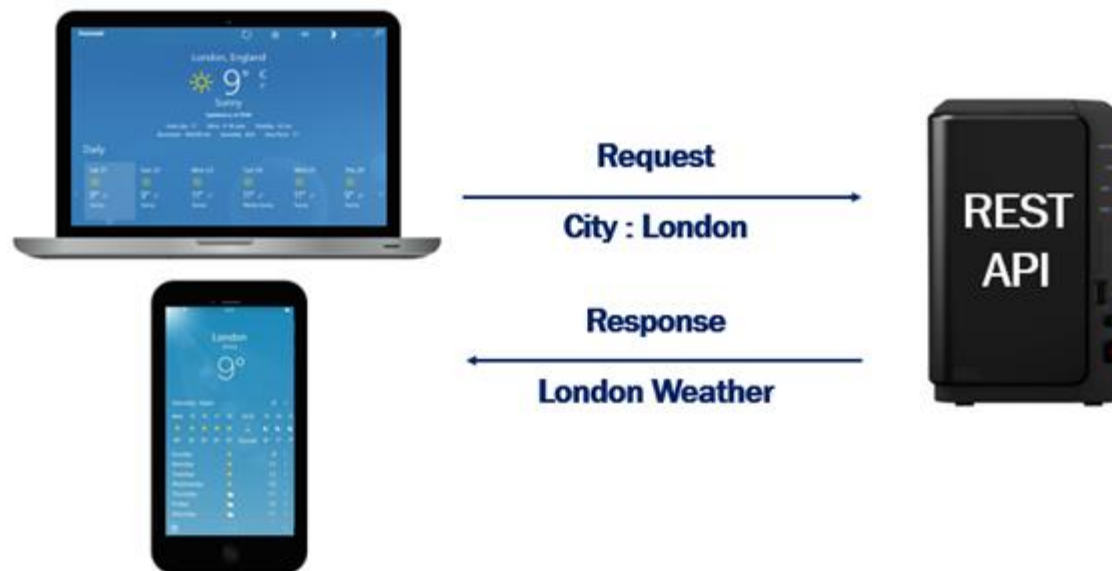
What are RESTful APIs

RE **RE**presentational
S **S**tate
T **T**ransfer

A **A**pplication
P **P**rogramming
I **I**nterface

Exemple Web API

- une API REST permet aux applications d'interagir les unes avec les autres et d'échanger des données.
- Par exemple, disons que vous créez une application mobile ou une application Web. Dans cette application, vous souhaitez afficher des données météorologiques telles que la température, l'humidité, la vitesse du vent, etc.



Conventions pour les Web API

- Dans le contexte d'une API REST, une ressource est une entité de données telle que Produit, Employé, Client, Commande, etc.
- Par exemple, une API REST qui fournit des données sur les employés rend la liste des employés disponible à l'URI (Uniform Resource Identifier) suivant:

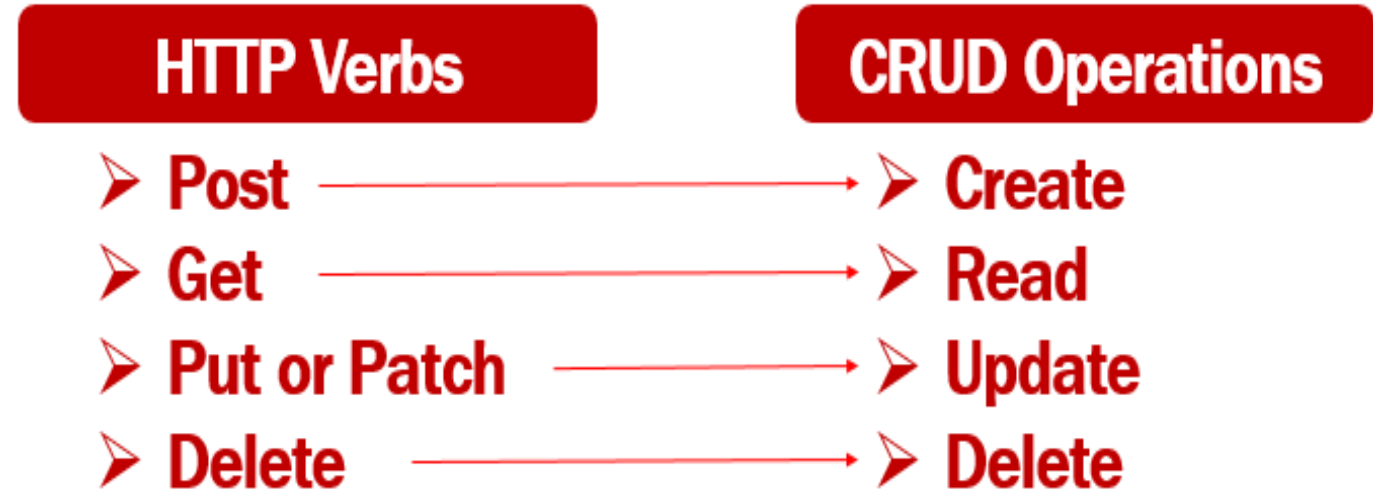
Protocol ↑
http:// **localhost:65180** **/api/employees**
↓
Domain Name

Optional ↑
/api/employees
↓
Endpoint

Conventions pour les Web API

- En plus de l'URI, nous devons également envoyer un HTTP Verb au serveur. Voici les HTTP Verb courants:

- **GET**
- **POST**
- **PUT**
- **PATCH**
- **DELETE**



- C'est ce verbe HTTP qui indique à l'API quoi faire avec la ressource, par exemple, créer un nouvel employé, consulter, supprimer ou modifier un employé existant.

Conventions pour les Web API









- la combinaison de l'URI et du verbe HTTP, qui est envoyé avec chaque requête, indique au serveur (c'est-à-dire à l'API REST) ce qu'il doit faire avec la ressource. Par exemple,


URI	HTTP Verb	Outcome
.../api/employees	GET	Gets list of employees
.../api/employees/1	GET	Gets employee with Id = 1
.../api/employees	POST	Creates a new employee
.../api/employees/1	PUT	Updates employee with Id = 1
.../api/employees/1	DELETE	Deletes employee with Id = 1

Créer un Projet WEBAPI

Créer un projet


Modèles de projet récents

-  API web ASP.NET Core C#
-  Application WebAssembly Blazor C#
-  Bibliothèque de classe C#
-  Nouvelle solution
-  Application web ASP.NET Core (modèle-vue-contrôleur) C#
-  ASP.NET Core vide C#
-  Application Windows Forms (.NET Framework) C#
-  Application Windows Forms C#

Rechercher des modèles (Alt+S) 

Tout effacer


C# Toutes les plateformes Web



Application web ASP.NET Core (modèle-vue-contrôleur)

Modèle de projet permettant de créer une application ASP.NET Core avec des exemples de vues et de contrôleurs ASP.NET Core MVC. Vous pouvez également utiliser ce modèle pour les services HTTP RESTful.


C# Linux macOS Windows Cloud Service Web



Application Blazor Server

Modèle de projet permettant de créer une application Blazor Server qui s'exécute côté serveur dans une application ASP.NET Core, et qui gère les interactions utilisateur via une connexion SignalR. Vous pouvez utiliser ce modèle pour les applications web ayant des IU (interfaces utilisateur) dynamiques riches.


C# Linux macOS Windows Cloud Web



API web ASP.NET Core

Modèle de projet permettant de créer une application ASP.NET Core avec un exemple de contrôleur pour un service HTTP RESTful. Vous pouvez également utiliser ce modèle pour les vues et contrôleurs ASP.NET Core MVC.

C# Linux macOS Windows Cloud Service Web



Service gRPC ASP.NET Core

Modèle de projet pour la création d'un service gRPC ASP.NET Core.

C# Linux macOS Windows Cloud Service Web

Retour Suivant

Créer un Projet WEBAPI

Configurer votre nouveau projet

Application web ASP.NET Core

C#

Linux

macOS

Windows

Cloud

Service


Web

Nom du projet

EmployeeWebAPI

Emplacement

C:\Users\USER\source\repos

Nom de la solution 

EmployeeWebAPI

☐ Placer la solution et le projet dans le même répertoire

Créer un Projet WEBAPI

Informations supplémentaires

API web ASP.NET Core

C#

Linux

macOS

Windows

Cloud

Service


Web

Framework 

.NET 6.0 (Prise en charge à long terme)

Type d'authentification 


Aucun


☒ Configurer pour HTTPS 

☐ Activer Docker 

OS Docker 

Linux

☒ Utiliser des contrôleurs (décocher pour utiliser un minimum d'API) 

☒ Activer la prise en charge d'OpenAPI 

Créer un Projet WEBAPI

The screenshot shows the Visual Studio IDE interface. The top menu bar includes options like Fichier, Edition, Affichage, Git, Projet, Générer, Débuguer, Test, Analyser, Outils, Extensions, Fenêtre, and Aide. The search bar contains 'Rechercher (Ctrl+Q)'. The title bar shows 'WebApplication1'. The left sidebar has a 'Vue d'ensemble' tab. The main area is titled 'ASP.NET Core' and contains the text 'Découvrez la plateforme .NET, créez votre première application et étendez-la au cloud.' Below this are three large icons representing different actions: 'Générer votre application', 'Se connecter à Azure', and 'Découvrir votre IDE'. Each icon has a corresponding link below it. The right sidebar is the 'Explorateur de solutions' (Solution Explorer), showing the project structure for 'WebApplication1' (1 sur 1 projet). The structure includes 'Connected Services', 'Dépendances', 'Properties', 'Controllers', and a sub-folder 'C# WeatherForecastController.c' containing 'appsettings.json', 'Program.cs', and 'WeatherForecast.cs'.

WeatherForecastController.cs | WeatherForecast.cs | **WebApplicati...e d'ensemble** X


Vue d'ensemble

Services connectés

Publier

ASP.NET Core


Découvrez la plateforme .NET, créez votre première application et étendez-la au cloud.



Générer votre application

[Parcourir les documentations, les exemples et les didacticiels](#)


[Architecture d'application .NET](#)



Se connecter à Azure

[Publier votre site web sur Azure](#)

[Bien démarrer avec ASP.NET sur Azure](#)



Découvrir votre IDE

[Consulter notre guide de productivité](#)

[Écrire du code plus rapidement](#)

Explorateur de solutions

Rechercher dans Explorateur de solution

- Solution 'WebApplication1' (1 sur 1 pr)
- WebApplication1
 - Connected Services
 - Dépendances
 - Properties
 - Controllers
 - C# WeatherForecastController.c
 - appsettings.json
 - Program.cs
 - WeatherForecast.cs

Créer les classes de Modèle

- Dans le dossier Models, Ajouter les classes suivantes:

```
public class Employee
{
    public int EmployeeId { get; set; }
    [Required]
    [StringLength(100, MinimumLength = 2)]
    public string FirstName { get; set; }
    [Required]
    public string LastName { get; set; }
    [Required][EmailAddress]
    public string Email { get; set; }
    public DateTime DateOfBrith { get; set; }
    public Gender Gender { get; set; }
    public int DepartmentId { get; set; }
    public string PhotoPath { get; set; }
    public Department Department { get; set; }
}

public class Department
{
    public int DepartmentId { get; set; }
    public string DepartmentName { get; set; }
}

public enum Gender
{
    Male,
    Female
}
```

Installer le package Entity Framework Core

The screenshot shows the Visual Studio IDE with the NuGet Package Manager window open. The left pane displays a list of packages, and the right pane shows the details for the selected package, **Microsoft.EntityFrameworkCore**.

Left Pane (Package List):

Package Name	Author	Downloads	Version
Microsoft.EntityFrameworkCore	Microsoft	490M	6.0.9
Microsoft.EntityFrameworkCore.Relational	Microsoft	483M	6.0.9
Microsoft.EntityFrameworkCore.Abstractions	Microsoft	450M	6.0.9
Microsoft.EntityFrameworkCore.Analyzers	Microsoft	436M	6.0.9
Microsoft.EntityFrameworkCore.SqlServer	Microsoft	243M	6.0.9
Microsoft.EntityFrameworkCore.Design	Microsoft	236M	6.0.9

Right Pane (Package Details for Microsoft.EntityFrameworkCore):

Source de package : **nuget.org**

Versions - 0

	Projet	Version	Installée
<input checked="" type="checkbox"/>	Projet		
<input checked="" type="checkbox"/>	WebApplication1		

Installé : non installé **Désinstaller**

Version : Dernière version stable 6.0.9 **Installer**

Options

Description

Installer SQL SERVER Provider pour EF Core

The screenshot shows the Visual Studio IDE with the NuGet Package Manager window open. The 'Solution' tab is selected, showing a list of packages. The package 'Microsoft.EntityFrameworkCore.SqlServer' is highlighted. A tooltip for 'Microsoft.EntityFrameworkCore.Relational' is visible. The right sidebar shows the package details for 'Microsoft.EntityFrameworkCore.SqlServer', including version 6.0.9 and an 'Installer' button.

Package Name	Version	Downloads
Microsoft.EntityFrameworkCore	6.0.9	490M
Microsoft.EntityFrameworkCore.Relational	6.0.9	483M
Microsoft.EntityFrameworkCore.Abstractions	6.0.9	450M
Microsoft.EntityFrameworkCore.Analyzers	6.0.9	436M
Microsoft.EntityFrameworkCore.SqlServer	6.0.9	243M
Microsoft.EntityFrameworkCore.Design	6.0.9	236M

Microsoft.EntityFrameworkCore.SqlServer
Version: 6.0.9
Status: non installé
Action: Installer

```
PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Installer EF Core Tools

The screenshot shows the Visual Studio interface with the NuGet Package Manager console open. The console displays a list of packages for the selected solution. The first package, **Microsoft.EntityFrameworkCore.Tools**, is highlighted. The Package Manager UI on the right shows the details for this package, including the version 6.0.9 and the 'Install' button.

NuGet - Solution WeatherForecastController.cs WeatherForecast.cs WebApplicati...ue d'ensemble

Parcourir Installé Mises à jour **1** Consolider

MICROSOFT.ENTITYFRAMEWORK x ↻ ☐ Inclure la version préliminaire

Source de package : nuget.org

.NET	Package Name	Author	Downloads	Version
.NET	Microsoft.EntityFrameworkCore.Tools ✓	par Microsoft	168M téléchargements	6.0.9
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.				
.NET	Microsoft.EntityFrameworkCore.InMemory ✓	par Microsoft	124M téléchargements	6.0.9
In-memory database provider for Entity Framework Core (to be used for testing purposes).				
.NET	Microsoft.EntityFrameworkCore.Sqlite.Core ✓	par Microsoft	88.4M téléchargements	6.0.9
SQLite database provider for Entity Framework Core. This package does not include a copy of the native SQLite library.				
.NET	Microsoft.EntityFrameworkCore.Sqlite ✓	par Microsoft	82M téléchargements	6.0.9
SQLite database provider for Entity Framework Core.				
.NET	EntityFramework ✓	par Microsoft	178M téléchargements	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.				

Microsoft.EntityFrameworkCore.Tools nuget.org

Versions - 0

<input checked="" type="checkbox"/>	Projet	Version	Installée
<input checked="" type="checkbox"/>	WebApplication1		

Installé : non installé **Désinstaller**

Version : Dernière version stable 6.0.9 **Installer**

Créer la classe de contexte

```
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options) :base(options)
    {
    }

    public DbSet<Employee> Employees { get; set; }
    public DbSet<Department> Departments { get; set; }
}
```

Dans le fichier de configuration appsettings.json, ajouter la chaîne de connexion de la base SQL SERVER

```
"ConnectionStrings": {
    "DBConnection":
    "server=(localdb)\\MSSQLLocalDB;database=APIEmployeeDB;Trusted_Connection=true"
}
```


Inscription du service de contexte

- Inscrire le service de contexte dans le fichier Program.cs, :

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

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DBConnection")));
```

EF Migration

- Dans Package Manager Console, Exécuter les commandes de Migration pour créer la base de données :

Add-Migration InitialCreate

Update-Database

Repository Pattern Interface

- Dans le dossier Models, Ajouter un dossier Repositories.
- Créer l'interface IEmployeeRepository qui définit les opérations de base sur les employés :

```
public interface IEmployeeRepository
{
    2 références
    Task<IEnumerable<Employee>> GetEmployees();
    4 références
    Task<Employee> GetEmployee(int employeeId);
    2 références
    Task<Employee> AddEmployee(Employee employee);
    2 références
    Task<Employee> UpdateEmployee(Employee employee);
    2 références
    Task<Employee> DeleteEmployee(int employeeId);
    2 références
    Task<Employee> GetEmployeeByEmail(string email);
}
```

Repository Pattern – Implémentation SQL Server

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace EmployeeWebAPI.Models.Repositories
```

```
{
    1 référence
    public class EmployeeRepository : IEmployeeRepository
    {
        private readonly AppDbContext appDbContext;
```

```
        0 références
        public EmployeeRepository(AppDbContext appDbContext)
        {
            this.appDbContext = appDbContext;
        }
```

```
        1 référence
        public async Task<IEnumerable<Employee>> GetEmployees()
        {
            return await appDbContext.Employees.ToListAsync();
        }
    }
}
```

1 référence

```
public async Task<IEnumerable<Employee>> GetEmployees()
{
    return await appDbContext.Employees.ToListAsync();
}
```

1 référence

```
public async Task<Employee> GetEmployee(int employeeId)
{
    return await appDbContext.Employees
        .FirstOrDefaultAsync(e => e.EmployeeId == employeeId);
}
```

1 référence

```
public async Task<Employee> AddEmployee(Employee employee)
{
    var result = await appDbContext.Employees.AddAsync(employee);
    await appDbContext.SaveChangesAsync();
    return result.Entity;
}
```

Repository Pattern – Implémentation SQL Server

```
public async Task<Employee> UpdateEmployee(Employee employee)
{
    var result = await appDbContext.Employees
        .FirstOrDefaultAsync(e => e.EmployeeId == employee.EmployeeId);

    if (result != null)
    {
        result.FirstName = employee.FirstName;
        result.LastName = employee.LastName;
        result.Email = employee.Email;
        result.DateOfBirth = employee.DateOfBirth;
        result.Gender = employee.Gender;
        result.DepartmentId = employee.DepartmentId;
        result.PhotoPath = employee.PhotoPath;

        await appDbContext.SaveChangesAsync();

        return result;
    }

    return null;
}
```

```
public async Task<Employee> DeleteEmployee(int employeeId)
{
    var result = await appDbContext.Employees
        .FirstOrDefaultAsync(e => e.EmployeeId == employeeId);
    if (result != null)
    {
        appDbContext.Employees.Remove(result);
        await appDbContext.SaveChangesAsync();
        return result;
    }

    return null;
}
```

Repository Pattern Interface

- Créer l'interface `IDepartmentRepository` qui définit les opérations sur les départements :

```
public interface IDepartmentRepository
{
    IEnumerable<Department> GetDepartments();
    Department GetDepartment(int departmentId);
}
```

Repository Pattern – Implémentation SQL Server

```
public class DepartmentRepository : IDepartmentRepository
{
    private readonly AppDbContext appDbContext;

    0 références
    public DepartmentRepository(AppDbContext appDbContext)
    {
        this.appDbContext = appDbContext;
    }

    1 référence
    public Department GetDepartment(int departmentId)
    {
        return appDbContext.Departments
            .FirstOrDefault(d => d.DepartmentId == departmentId);
    }

    1 référence
    public IEnumerable<Department> GetDepartments()
    {
        return appDbContext.Departments;
    }
}
```

Injection des services – Classes Repository

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

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DBConnection")));

builder.Services.AddScoped<IDepartmentRepository, DepartmentRepository>();
builder.Services.AddScoped<IEmployeeRepository, EmployeeRepository>();

var app = builder.Build();
```


Avantages du Repository Pattern

- Le code est plus propre et plus facile à réutiliser et à maintenir.
- Nous permet de créer des systèmes faiblement couplés.
- Par exemple, si nous voulons que notre application fonctionne avec oracle au lieu de SQL SERVER, implémentez un `OracleRepository` qui sait lire et écrire dans la base de données Oracle et enregistrez `OracleRepository` dans le système d'injection de dépendances.
- Dans un projet de test unitaire, il est facile de remplacer un vrai repository par une fausse implémentation pour les tests.

Les contrôleurs WEBAPI

- Un contrôleur WebAPI est une classe qui hérite de ControllerBase,
- ControllerBase se trouve dans l' espace de noms Microsoft.AspNetCore.Mvc .
- Créez donc un contrôleur qui dérive de la classe Controller si vous créez une application Web MVC.



Ajouter un contrôleur API

```
[Route("api/[controller]")]
[ApiController]
public class EmployeesController : ControllerBase
{
    //Injection de dépendance du service IEmployeeRepository
    private readonly IEmployeeRepository employeeRepository;

    public EmployeesController(IEmployeeRepository employeeRepository)
    {
        this.employeeRepository = employeeRepository;
    }
}
```

La méthode d'action- GetEmployees

```
[HttpGet]
public async Task<ActionResult> GetEmployees()
{
    try
    {
        return Ok(await employeeRepository.GetEmployees());
    }
    catch (Exception)
    {
        return StatusCode(StatusCode.Status500InternalServerError,
            "Error retrieving data from the database");
    }
}
```

Http Status codes

- Fournit au client de l'API, le statut de la requête.

Level 200

200 OK

201 Created

204 No Content

Level 400

400 Bad Request

401 Unauthorized

404 Not Found

Level 500

500 Internal Server Error

501 Not Implemented


503 Service Unavailable

Http Status codes

- ASP.NET Core fournit les méthodes d'assistance suivantes pour renvoyer les codes d'état HTTP:

HTTP Status Code	Helper Method
200	Ok
201	Created
204	NoContent
400	BadRequest
401	Unauthorized
404	NotFound
500	StatusCode
501	StatusCode
503	StatusCode

Any request you send in this workspace will appear here.

 Show me how 

Comments

Save

[Cookies](#) [Code](#)

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Save Response ▼


```

1  {
2      {
3          "employeeId": 1,
4          "firstName": "ZIRIBI",
5          "lastName": "MALEK",
6          "email": "malek_zribi@yahoo.fr",
7          "dateOfBrith": "1976-05-10T00:00:00",
8          "gender": 1,
9          "departmentId": 1,
10         "photoPath": null
11     },
12     {
13         "employeeId": 2,
14         "firstName": "TRIKI",
15         "lastName": "MOHAMED ALI",
16         "email": "triki_ali@gmail.com",
17         "dateOfBrith": "1988-05-22T00:00:00",
18         "gender": 0,
19         "departmentId": 2,
20         "photoPath": null

```

Les contrôleurs WEBAPI

GetEmployee(id)

URI	HTTP Verb	Outcome
.../api/employees	GET	Gets list of employees
.../api/employees/1	GET	Gets employee with Id = 1

```
[HttpGet("{id:int}")]
public async Task<ActionResult<Employee>> GetEmployee(int id)
{
    try
    {
        var result = await employeeRepository.GetEmployee(id);
        if (result == null) return NotFound();
        return result;
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError,
            "Error retrieving data from the database");
    }
}
```


Filter

History Collections APIs BETA

Save Responses Clear all

Today

GET https://localhost:44369/api/Employees/1

DEL http://localhost:65180/api/Empl... [CONFLICT] GET http://localhost:651... GET http://localhost:65180/api/Empl... + ...

No Environment

Comments

GET https://localhost:44369/api/Employees/1

Send Save

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

Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 1309ms Size: 349 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "employeeId": 1,
3   "firstName": "ZRIBI",
4   "lastName": "MALEK",
5   "email": "malek_zribi@yahoo.fr",
6   "dateOfBrith": "1976-05-10T00:00:00",
7   "gender": 1,
8   "departmentId": 1,
9   "photoPath": null
10 }
```

Méthode Post – Créer un employé

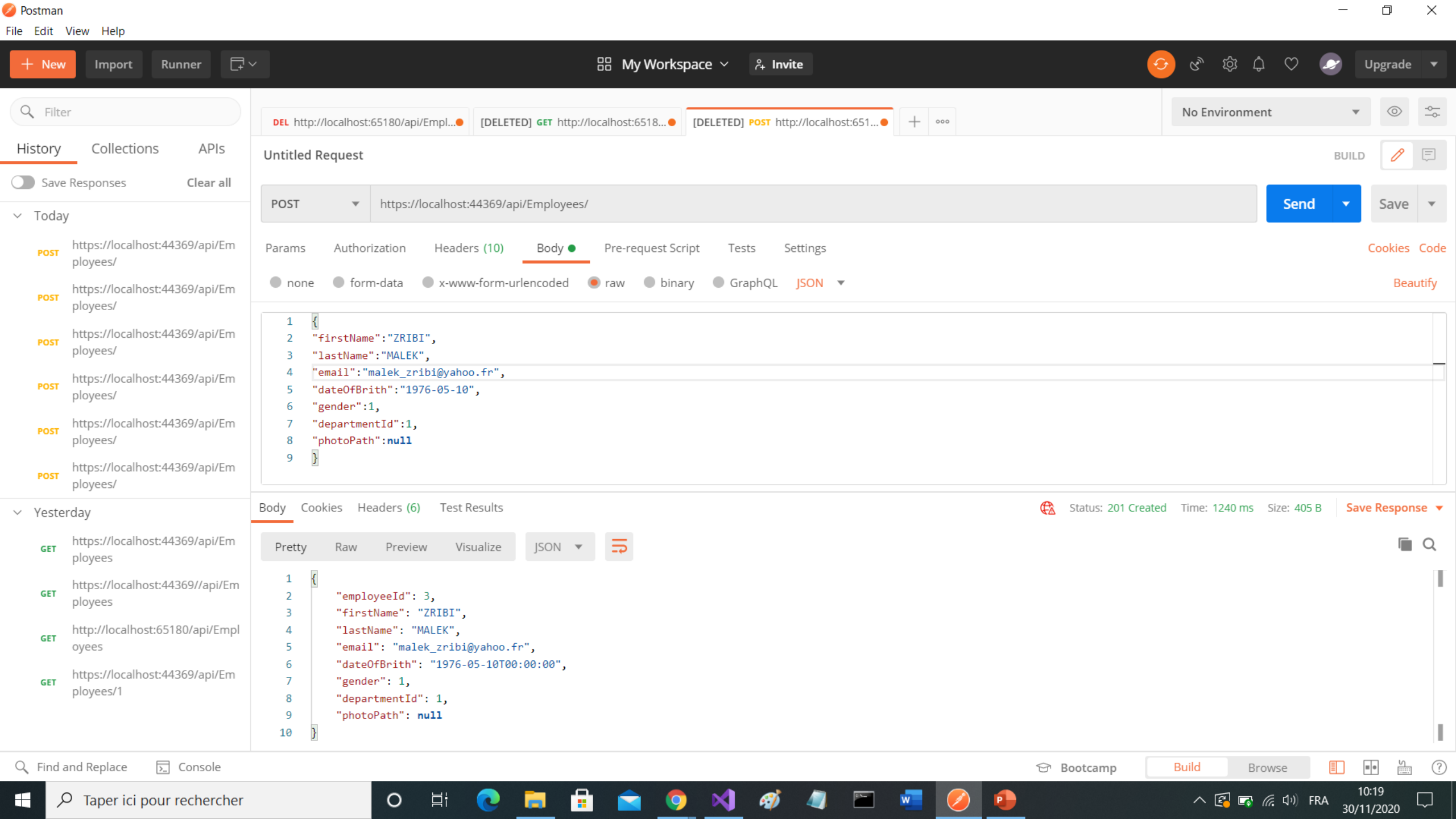
```
[HttpPost]
public async Task<ActionResult<Employee>> CreateEmployee([FromBody] Employee employee)
{
    try
    {
        if (employee == null)
            return BadRequest();
        var createdEmployee = await employeeRepository.AddEmployee(employee);

        return CreatedAtAction(nameof(GetEmployee),
            new { id = createdEmployee.EmployeeId }, createdEmployee);
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError,
            "Error creating new employee record");
    }
}
```

Méthode Post – Créer un employé

Lorsqu'une nouvelle ressource est créée, les 3 choses suivantes se produisent généralement :

- Renvoyer le code d'état http 201 pour indiquer que la ressource a été créée avec succès.
- Renvoyer la ressource nouvellement créée. Dans notre cas, le nouvel employé.
- Ajoutez un en-tête Location à la réponse. L'en-tête Location spécifie l'URI de l'objet employé nouvellement créé.
- La méthode CreatedAtAction nous aide à réaliser les trois choses ci-dessus. Nous utilisons l'opérateur nameof au lieu d'inclure le nom de la méthode (GetEmployee) dans une chaîne.



Validation de Modèle dans ASP.NET Core Rest API

- ASP.NET Core fournit plusieurs attributs intégrés pour la validation de modèle
- Required : Spécifie que le champ est obligatoire
- Range : Spécifie la valeur minimale et maximale autorisée
- MinLength : Spécifie la longueur minimale d'une chaîne
- MaxLength : Spécifie la longueur maximale d'une chaîne
- Compare : Compare 2 propriétés d'un modèle. Par exemple, comparer les propriétés Email et ConfirmEmail.
- RegularExpression : Valide si la valeur fournie correspond au modèle spécifié par l'expression régulière
- Ces attributs de validation se trouvent dans l'espace de noms `System.ComponentModel.DataAnnotations`

Validation de Modèle dans ASP.NET Core Rest API

- Pour implémenter la validation de modèle dans une API REST ASP.NET Core, décorez les propriétés respectives avec les attributs de validation.
- Dans l'exemple suivant, FirstName est une propriété obligatoire. Doit contenir un minimum de 2 caractères et ne doit pas dépasser 100 caractères.

```
public class Employee
{
    5 références
    public int EmployeeId { get; set; }
    [Required]
    [StringLength(100, MinimumLength = 2)]
    2 références
    public string FirstName { get; set; }
    [Required]
    2 références
    public string LastName { get; set; }
    [Required]
    2 références
    public string Email { get; set; }
    2 références
    public DateTime DateOfBrith { get; set; }
    2 références
    public Gender Gender { get; set; }
    2 références
    public int DepartmentId { get; set; }
    2 références
    public string PhotoPath { get; set; }
}
```

- Si on saisit des données non conformes pour les colonnes firstname et lastname on aura les messages d'erreurs suivants :

The screenshot shows the Postman application interface. On the left, there is a sidebar with 'History', 'Collections', and 'APIs' tabs. The 'History' tab is active, showing a list of requests. The main area displays an 'Untitled Request' for a POST method to the endpoint `https://localhost:44369/api/Employees/`. The request body is in JSON format, containing the following data:

```
{
  "firstName": "B",
  "email": "malek_zribi@yahoo.fr",
  "dateOfBrith": "1976-05-10",
  "gender": 1,
  "departmentId": 1,
  "photoPath": null
}
```

The response is displayed below the request, showing a status of 400 Bad Request. The response body is in JSON format, containing the following data:

```
{
  "status": 400,
  "traceId": "|871a158d-454303cd000a058f.",
  "errors": {
    "LastName": [
      "The LastName field is required."
    ],
    "FirstName": [
      "The field FirstName must be a string with a minimum length of 2 and a maximum length of 100."
    ]
  }
}
```

The bottom of the screen shows the Windows taskbar with various application icons and the system clock indicating 10:49 on 01/12/2020.

Erreurs de validation de modèle

- Pour ajouter une erreur de validation de modèle personnalisé, utilisez la méthode **AddModelError ()** de l'objet **ModelState**.
- Dans une API REST ASP.NET Core, il n'est pas nécessaire de vérifier explicitement si l'état du modèle est valide.
- La classe de contrôleur est décorée avec l'attribut [ApiController], elle prend soin de vérifier si l'état du modèle est valide et renvoie automatiquement l'état 400 (Bad Request) avec les erreurs de validation.
- Apportons à la méthode CreateEmployee les modifications nécessaires pour valider les données envoyés et vérifier si l'email de l'employé existe déjà.

2 références

```
public async Task<Employee> GetEmployeeByEmail(string email)
{
    return await appDbContext.Employees
        .FirstOrDefaultAsync(e => e.Email == email);
}
```



```

public async Task<ActionResult<Employee>> CreateEmployee(Employee employee)
{
    try
    {
        if (employee == null)
        {
            return BadRequest();
        }
        else
        // Add custom model validation error
        {
            var emp = await employeeRepository.GetEmployeeByEmail(employee.Email);
            if (emp != null)
            {
                ModelState.AddModelError("email", "Employee email already in use");
                return BadRequest(ModelState);
            }
            else
            {
                var createdEmployee = await employeeRepository.AddEmployee(employee);

                return CreatedAtAction(nameof(GetEmployee), new { id = createdEmployee.EmployeeId },
                    createdEmployee);
            }
        }
    }
    catch (Exception)
    {
        return StatusCode(StatusCode.Status500InternalServerError, "Error retrieving data from the database");
    }
}

```

- Si on envoie à l'API un nouvel employé (POST) avec un email existant, on aura comme réponse un Bad Request (400) avec le message d'erreur de validation.

The screenshot shows a REST client interface with a top bar containing several request tabs. The active tab is a POST request to `https://localhost:44369/api/employees/`. The interface includes tabs for Params, Authorization, Headers (10), Body, Pre-request Script, Tests, and Settings. The Body tab is selected, showing a JSON payload. Below the request editor, there are tabs for Body, Cookies, Headers (5), and Test Results. The Test Results tab is active, displaying the response status and body. The response status is `400 Bad Request` with a time of `235 ms` and a size of `234 B`. The response body is a JSON object indicating that the email is already in use.

DEL http://localhost:65180/api/Emp... [DELETED] GET http://localhost:6518... [DELETED] POST http://localhost:651...

Untitled Request BUILD

POST https://localhost:44369/api/employees/ Send Save

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

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

```
1 {
2   "firstName": "BEN ALI",
3   "lastName": "SOFIEN",
4   "email": "malek_zribi@yahoo.fr",
5   "dateOfBrith": "1976-05-10",
6   "gender": 1,
7   "departmentId": 1,
8   "photoPath": null
9 }
```

Body Cookies Headers (5) Test Results Status: 400 Bad Request Time: 235 ms Size: 234 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "email": [
3     "Employee email already in use"
4   ]
5 }
```

Mettre à jour les données (la requête HTTP PUT)

- Pour mettre à jour les données d'un employé, ajouter un http put request, et donner dans l'url l'ID de l'employé à modifier.

URI	HTTP Verb	Outcome
.../api/employees	GET	Gets list of employees
.../api/employees/1	GET	Gets employee with Id = 1
.../api/employees	POST	Creates a new employee
.../api/employees/1	PUT	Updates employee with Id = 1

- Le paramètre de la méthode UpdateEmployee doit être décoré avec l'attribut [FromBody] mais ceci n'est pas obligatoire si vous avez ajouté l'attribut [ApiController] au niveau du contrôleur.

Mettre à jour les données (la requête HTTP PUT)

```
[HttpPut("{id:int}")]
```

0 références

```
public async Task<ActionResult<Employee>> UpdateEmployee(int id, Employee employee)
{
    try
    {
        if (id != employee.EmployeeId)
            return BadRequest("Employee ID mismatch");

        var employeeToUpdate = await employeeRepository.GetEmployee(id);

        if (employeeToUpdate == null)
            return NotFound($"Employee with Id = {id} not found");

        return await employeeRepository.UpdateEmployee(employee);
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError,
            "Error updating data");
    }
}
```

Mettre à jour les données (la requête HTTP PUT)

Untitled Request BUILD

PUT ▼ `https://localhost:44369/api/Employees/3` Send Save ▼

Params Authorization Headers (10) **Body** ● Pre-request Script Tests Settings Cookies Co

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ▼ Beautify

```
1 {
2   "employeeId": 3,
3   "firstName": "Akroute",
4   "lastName": "Salah",
5   "email": "ak_sal@yahoo.fr",
6   "dateOfBrith": "1976-05-10T00:00:00",
7   "gender": 1,
8   "departmentId": 1,
9   "photoPath": null
10 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 1449 ms Size: 346 B Save Response

Pretty Raw Preview Visualize **JSON** ▼ ≡

```
1 {
2   "employeeId": 3,
3   "firstName": "Akroute",
4   "lastName": "Salah",
5   "email": "ak_sal@yahoo.fr",
6   "dateOfBrith": "1976-05-10T00:00:00",
7   "gender": 1,
8   "departmentId": 1,
9   "photoPath": null
10 }
```

Requête Delete dans ASP.NET Core Rest API

- Pour supprimer une ressource, envoyez une requête HTTP DELETE à l'URI /api/Employees/ID . L' ID de l'employé à supprimer doit être passé dans l'URI.

URI	HTTP Verb	Outcome
.../api/employees	GET	Gets list of employees
.../api/employees	POST	Creates a new employee
.../api/employees/1	GET	Gets employee with Id = 1
.../api/employees/1	PUT	Updates employee with Id = 1
.../api/employees/1	DELETE	Deletes employee with Id = 1



Requête HTTP Delete ASP.NET Core Rest API

```
[HttpDelete("{id:int}")]
```

0 références

```
public async Task<ActionResult<Employee>> DeleteEmployee(int id)
{
    try
    {
        var employeeToDelete = await employeeRepository.GetEmployee(id);

        if (employeeToDelete == null)
        {
            return NotFound($"Employee with Id = {id} not found");
        }

        return await employeeRepository.DeleteEmployee(id);
    }
    catch (Exception)
    {
        return StatusCode(StatusCodes.Status500InternalServerError,
            "Error deleting data");
    }
}
```

[DELETED] GET http://localhost:651... [DELETED] POST http://localhost:65... POST https://localhost:44369/api/E... DEL https://localhost:44369/api/Em...

Untitled Request

DELETE

https://localhost:44369/api/Employees/3

Send

Save

- Params
- Authorization
- Headers (8)
- Body
- Pre-request Script
- Tests
- Settings
- none
- form-data
- x-www-form-urlencoded
- raw
- binary
- GraphQL
- Cookies
- Cc

This request does not have a body

Body Cookies Headers (5) Test Results Status: 200 OK Time: 2.30 s Size: 346 B Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "employeeId": 3,
3   "firstName": "Akroute",
4   "lastName": "Salah",
5   "email": "ak_sal@yahoo.fr",
6   "dateOfBrith": "1976-05-10T00:00:00",
7   "gender": 1,
8   "departmentId": 1,
9   "photoPath": null
10 }
```


Requête HTTP Search

- Pour implémenter le service de recherche, on commence par ajouter la méthode Search dans EmployeeRepository.

```
public interface IEmployeeRepository
{
    2 références
    Task<IEnumerable<Employee>> GetEmployees();
    4 références
    Task<Employee> GetEmployee(int employeeId);
    2 références
    Task<Employee> AddEmployee(Employee employee);
    2 références
    Task<Employee> UpdateEmployee(Employee employee);
    2 références
    Task<Employee> DeleteEmployee(int employeeId);
    2 références
    Task<Employee> GetEmployeeByEmail(string email);
    1 référence
    Task<IEnumerable<Employee>> Search(string name, Gender? gender);
}
```

Requête HTTP Search

- Code de la méthode Search dans EmployeeRepository

```
public async Task<IEnumerable<Employee>> Search(string name, Gender? gender)
{
    IQueryable<Employee> query = appDbContext.Employees;

    if (!string.IsNullOrEmpty(name))
    {
        query = query.Where(e => e.FirstName.Contains(name)
                               || e.LastName.Contains(name));
    }

    if (gender != null)
    {
        query = query.Where(e => e.Gender == gender);
    }

    return await query.ToListAsync();
}
```

Requête HTTP Search

- Code de la méthode d'action Search dans EmployeesController

```
[HttpGet("{search}")]
```

0 références

```
public async Task<ActionResult<IEnumerable<Employee>>> Search(string name, Gender? gender)
{
    try
    {
        var result = await employeeRepository.Search(name, gender);

        if (result.Any())
        {
            return Ok(result);
        }

        return NotFound();
    }
    catch (Exception)
    {
        return StatusCode(StatusCode.Status500InternalServerError,
            "Error retrieving data from the database");
    }
}
```

Requête HTTP Search

[DELETED] PUT http://... [DELETED] POST http:... POST https://localho... DEL https://localhost... POST https://localho... GET https://localhost... + ... No Environment

Untitled Request BUILD

GET https://localhost:44369/api/Employees/Search?name=zribi&gender=0 Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	name	zribi			
<input checked="" type="checkbox"/>	gender	0			
	Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 29 ms Size: 351 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "employeeId": 1,
4     "firstName": "ZRIBI",
5     "lastName": "MALEK",
6     "email": "malek_zribi@yahoo.fr",
7     "dateOfBrith": "1976-05-10T00:00:00",
8     "gender": 0,
9     "departmentId": 1,
10    "photoPath": null
11  }
12 ]
```

Requête HTTP Search

[DELETED] PUT http://... [DELETED] POST http://... POST https://localho... DEL https://localhost... POST https://localho... GET https://localhost... + ...

No Environment

Build

Untitled Request

GET https://localhost:44369/api/Employees/Search?name=turki

Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Ed
<input checked="" type="checkbox"/>	name	turki			
	Key	Value	Description		

Body Cookies Headers (5) Test Results

Status: 404 Not Found Time: 72 ms Size: 330 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "type": "https://tools.ietf.org/html/rfc7231#section-6.5.4",
3   "title": "Not Found",
4   "status": 404,
5   "traceId": "|ba9c9e70-44871144cbef37ad."
```