

# **Developing a Web App using ASP.NET Core Web API**

## **Summer Training Report**

**CMSE 400**

**Fares Arnous 20801168**

**Number of working days:** 40

**Period of Training:** 10/07/2023 – 25/08/2023

**Company Name:** Cloud Soft

**Company Address:** Ismet Inonu Bulvarı, Gritili Apt. 1st  
Floor, Famagusta, TRNC, 99450

**Computer Engineering Department**

**Eastern Mediterranean University**

## **ACKNOWLEDGEMENT**

I would like to thank **Mr. Kheder Kasem** who is the manager of Cloud soft company and my supervisor through my internship. He used to give me tutorials every week to help me improve my coding skills and allowed me to ask many questions. And I would like to thank my friends who were working in the company, they helped me a lot by answering all my questions and guiding me if I have any difficulties. I really had a good time at Cloud soft company with great and helpful people.

## ABSTRACT

Firstly, I spent the first week of my internship studying the fundamentals of C# language through a YouTube course by building console application, the duration of the course was 9 hours. First thing I had to do is download some useful tools like postman and Visual Studio 2022. I learned so many things about C# during this course like, class basics, Variables, Casting, Data type and many other things. After that I continued learning more about C# by going over “**Ultimate ASP.NET Core Web API**” book, I started to learn what is API, then I learned what is project configuration and how to configure the logging services of the project, after that I learned about the onion architecture, its layers, and the advantages of using it, then I applied the database models, the repository patterns, and global handling errors. Then I worked with Get, Post, Update, Patch, and Delete requests. Then I learned about what validation, asynchronous code does and applied it on my code, and I applied action filters, paging, filtering, searching, and sorting on my project. Then the last topic I learned was data shaping for the listed methods applied JWT (JSON Web Tokens) and Identity on the project. And of course, I had to test each step of building my project and the HTTP methods so, I used postman tool for the testing part.

**Keywords:** C Sharp, API, Onion architecture, Repository, Services, Postman, Visual Studio, Ultimate ASP.NET Core Web API, HTTP methods, JWT, JSON.

<b>TABLE OF CONTENTS</b>	<b>Page</b>
ACKNOWLEDGEMENT .....	II
ABSTRACT .....	III
TABLE OF CONTENTS .....	IV
LIST OF FIGURES .....	V
1. INTRODUCTION .....	1
2. COMPANY OVERVIEW .....	2
3. OBJECTIVES / PROBLEM DEFINITION.....	4
4. WORK DONE AND METHODOLOGY APPLIED .....	5
5. RESULTS AND DISCUSSION.....	16
6. NEW KNOWLEDGE GAINED THROUGH THE EXPERIENCE .....	23
7. CONCLUSIONS .....	24
8. REFERENCES .....	25
9. APPENDIX .....	26

## LIST OF FIGURES

## Page

<b>Figure 1:</b> Real Estate System-----	2
<b>Figure 2:</b> Exchange system-----	2
<b>Figure 3:</b> XML code -----	4
<b>Figure 4:</b> Paging -----	5
<b>Figure 5:</b> Metadata class -----	6
<b>Figure 6:</b> Request Parameters class-----	6
<b>Figure 7:</b> flowchart of the paging algorithm -----	7
<b>Figure 8:</b> filtering method in Employee Parameter Class. -----	8
<b>Figure 9:</b> filtering method in Employee repository class. -----	8
<b>Figure 10:</b> filtering method in Repository Employee Extensions class. -----	9
<b>Figure 11:</b> searching method in Employee Parameters class. -----	9
<b>Figure 12:</b> searching method in Employee repository class-----	9.
<b>Figure 13:</b> searching method in Repository Employee Extensions class -----	10.
<b>Figure 14:</b> sorting method in Request parameters class-----	10
<b>Figure 15:</b> sorting method in Employee parameters class-----	11
<b>Figure 16:</b> sort method in Repository Employee Extensions class-----	11
<b>Figure 17:</b> safety and Idempotency -----	12
<b>Figure 18:</b> The implementation of GET method-----	13
<b>Figure 19:</b> The implementation of delete request -----	14
<b>Figure 20:</b> The implementation of put request-----	14
<b>Figure 21:</b> The implementation of patch request-----	14

<b>Figure 22:</b> The operations of PATCH request-----	15
<b>Figure 23:</b> The output of the GET request-----	16
<b>Figure 24:</b> The output of getting any company by his Id, -----	17
<b>Figure 25:</b> The query that I wrote to add an employee-----	17
<b>Figure 26:</b> The output of the PUT method -----	18
<b>Figure 27:</b> The replace query of PATCH method -----	18
<b>Figure 28:</b> getting the same employee-----	19
<b>Figure 29:</b> The removing query-----	20
<b>Figure 30:</b> The result after removing the age of the employee -----	20
<b>Figure 31:</b> The adding query -----	20
<b>Figure 32:</b> The result after Adding the age of the employee -----	20
<b>Figure 33:</b> The result of getting all employees before deleting any of them -----	21
<b>Figure 34:</b> Sending deleting request to delete the employee -----	21
<b>Figure 35:</b> Getting all employees after deleting the employee -----	22

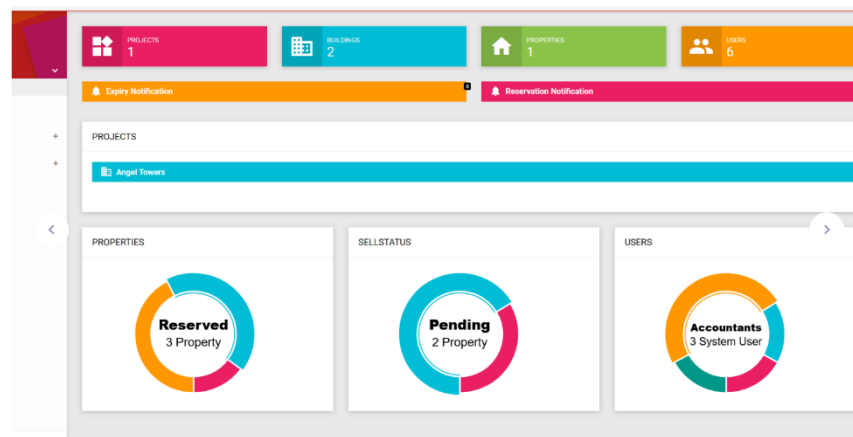
## 1. INTRODUCTION

During my internship in Cloud soft company, I learned a lot of things and I got a small knowledge of how the real life of software engineer is going to be, how to start my practical life, and what kind of work I am going to do and too many things about work in the real life. I figured out that it is better to focus on mastering only one programming language rather than trying to learn many at once. That allows me to go through the chosen language deeply and understanding it very well, that does not mean I should study only one programming language. Once I feel confident and good enough in the chosen language, I will start learning another one, in this way I will improve my skills without feeling confused. Learning one programming language completely makes it easier to learn new languages in the future. It's a step-by-step approach to become a proficient programmer, and it helps prevent confusion. Through my internship I was asked to learn C# language so, I started to take online courses and **Mr. Kheder Kasem** helped me to choose appropriate sources to start study from. It took me almost one week to understand the basics of C#, Then I read book for Code Maze organization which was “**Ultimate ASP.NET Core Web API**”, I went over the book and start to apply the project that the book was explaining which was about “Company and Employee” system, the main idea of the project is to store the Companies’ information and the data of each employees that work for each company.

In addition to that, I learnt a very important thing, which is “TIME”. During my internship I became aware of how much the time is important, and thanks for Mr. Kheder who was very strict with the time, and it is the most important thing for him. Since we had to be in the office at 8:30 am and leave at 5:00 pm, that’s helped me a lot with knowing how time is valuable.

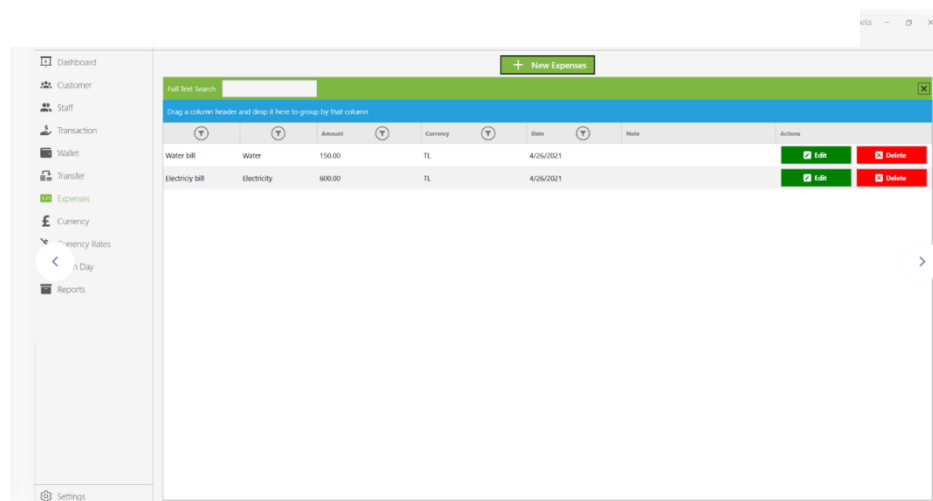
## 2. COMPANY OVERVIEW

As I mentioned earlier, I completed my summer internship at Cloud Soft Company, which offers a wide range of services including Web Applications, Hosting solutions, System Integration, Desktop applications, Mobile Applications, UI/UX Design, and Digital Marketing. The company has served over 100 clients, and successfully completed more than 200 projects, thanks to a team of experts proficient in over 30 different skills. **Figures 1, and 2** below shows some projects done by the company:



**Figure 1:** Real Estate System

shows a system designed to help real estate offices to manage and operate their business operation, manage their users, bonuses, reports, and more.



**Figure 2:** Exchange system

Figure2 shows a desktop application that is designed to help users manage and monitor all aspects of their business.



The company has 4 employees, and they are experts with ASP.Net Core, C#, UI/UX design, Python, and many other programming languages. And there were 2 trainers that were training in the company.

Company address: **“İsmet İnönü Bulvarı, Giritli Apt. 1st Floor, Famagusta, TRNC, 99450”**.

Company phone number: (+90) 392 3653003.

Company E-mail: [admin@cloudsoft.com](mailto:admin@cloudsoft.com).

Company website: [Cloud soft](#)

### 3. OBJECTIVES / PROBLEM DEFINITION

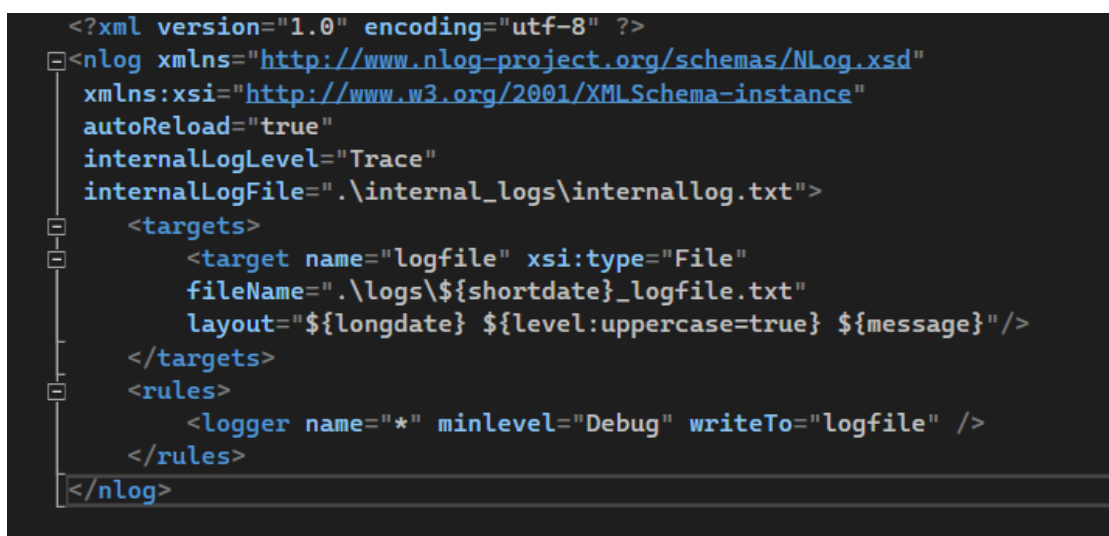
I was assigned during my summer practicing to create a small web API which is “Company Employee system”. But before that they asked me to go over an online course to get an Idea about C# language, The course duration was 9 hours so, I took almost one week to finish the course.

After I finished the online course, I start working on the project by writing the backend code which has been done on .NET platform by using C# language, I wrote the models that I will use in the system, the Database has been created automatically from the .NET platform by MSSQL database language,

I was asked to create HTTP methods (Get, Post, Put, Patch, and Delete) and test them using postman tool. I used Asynchronous code, Validation, action filters, Paging, Filtering, Searching, storing, and Data shaping to enhance the HTTP methods.

Furthermore, I used global error handling to deal with errors and implemented logger services to save information, errors, debugging data, and warning messages to an external file. To set up these logger services

I wrote C# and XML code in the backend. Additionally, I implemented JWT (JSON Web Token) and Identity to enhance the security of the API. **Figure 3** below is the XML code I created for logger services:

The image shows a code editor with XML code for configuring a logger. The code is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
autoReload="true"
internalLogLevel="Trace"
internalLogFile=".\internal_logs\internallog.txt">
  <targets>
    <target name="logfile" xsi:type="File"
fileName=".\logs\${shortdate}_logfile.txt"
layout="${longdate} ${level:uppercase=true} ${message}" />
  </targets>
  <rules>
    <logger name="*" minlevel="Debug" writeTo="logfile" />
  </rules>
</nlog>
```

**Figure 3:** XML code

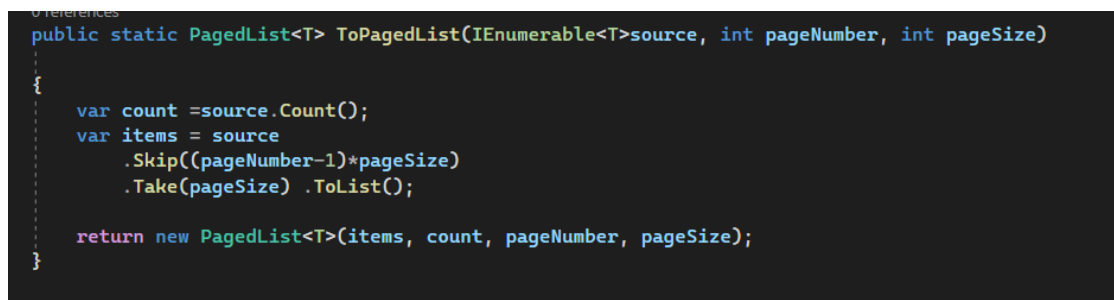
shows the XML code that I created for logger services.

## 4. WORK DONE AND METHODOLOGY APPLIED

As I mentioned earlier, I was working on web API by using “**Ultimate ASP.NET core Web API**” book as the main reference and I used Microsoft documents that are related to the book that’s I could refer to them in case I faced any problem. The book had 32 chapters, but I could not finish them because of the time, I could complete only 20 chapters in 40 days and the last topic I reached was Data shaping.

The Company was Implementing Agile as their chosen methodology, enabling them to adapt quickly to changing customer requirements. In my situation, I was required to test all additions to my system and any modifications I made to it.

During my coding phase I did not use calculation a lot, because in my project there was no need to use calculation except for applying the paging method. **Figure 4** below shows how did I use calculation and how I implemented the paging method:

A screenshot of a code editor showing a C# method named ToPagedList. The code is as follows:

```
public static PagedList<T> ToPagedList(IEnumerable<T>source, int pageNumber, int pageSize)
{
    var count = source.Count();
    var items = source
        .Skip((pageNumber-1)*pageSize)
        .Take(pageSize) .ToList();

    return new PagedList<T>(items, count, pageNumber, pageSize);
}
```

**Figure 4:** Paging

This figure shows the code I wrote for the paging, let’s say we want to retrieve the results for the third page with 20 results per page, so we would skip the first 40 results. It is 40 because of the “Skip” equation  $((3-1) * 20) = 40$  and then we take the next 20 results.

The paging algorithm is designed with a maximum of 50 rows per page. If the caller doesn't specify values for the page number and page size, the system will set the page size to 10 and a page number to 1 by default. If the caller assigns a page number

greater than the maximum (50), the system will automatically set the page number to the maximum value (50).

**Figures 5, 6 and 7** below show the implementation of the paging algorithm and the flowchart of it:

```
public class MetaData
{
    3 references
    public int CurrentPage { get; set; }
    2 references
    public int TotalPages { get; set; }
    1 reference
    public int PageSize { get; set; }

    1 reference
    public int TotalCount { get; set; }

    0 references
    public bool HasPrevious => CurrentPage > 1;

    0 references
    public bool HasNext => CurrentPage < TotalPages;
}
```

**Figure 5:** Metadata class

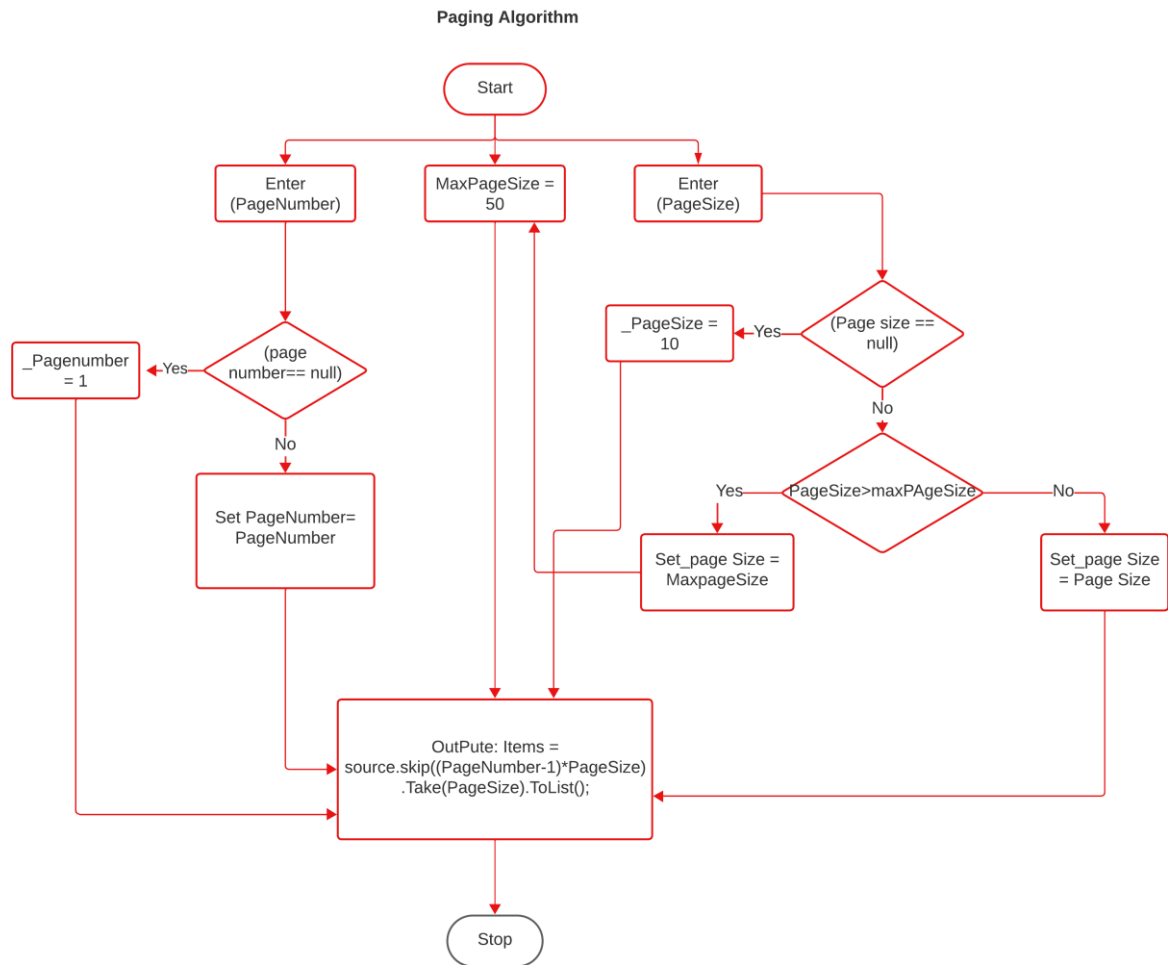
This figure shows the Metadata class that was created to manage and to provide metadata related to pagination in my system.

```
1 reference
public abstract class RequestParameters
{
    const int maxPageSize = 50;
    1 reference
    public int PageNumber { get; set; } = 1;

    private int _pageSize = 10;
    1 reference
    public int PageSize
    {
        get
        {
            return _pageSize;
        }
    }
}
```

**Figure 6:** Request Parameters class

This figure shows the request parameters class that is used to allow the caller to specify things like the page number, page size, which fields to include in the response when interacting with the system, and sorting criteria.



**Figure 7:** flowchart of the paging algorithm.

This diagram explains how dose the paging algorithm works. If the caller did not enter any values for the page number and paging size, the system will directly set the paging number to **1** and the page size to **10**.

In addition to that, I applied the searching method to my system, which is a mechanism to retrieve specific information from the database. And I applied the filtering method, which is a mechanism to retrieve specific information from the database depending on some kind of criterion. I have added the sort method as well to

order the results in a preferred way using query string parameters, because It makes finding information way easier.

Sometimes people cannot distinguish between filtering and searching. Basically, in the searching method we usually have only one input to search for anything on the website. In other words, you send a string to the API and the API is responsible for using that string to find any results that match it. On the contrary, for the filtering method we have more than one input to retrieve specific data from the database.

**Figures** below shows the implementation of the searching, filtering, and sorting methods:

- **Filtering method:**

```
5 references
public class EmployeeParameters : RequestParameters
{
    2 references
    public uint MinAge { get; set; }
    2 references
    public uint MaxAge { get; set; } = int.MaxValue;

    1 reference
    public bool ValidAgeRange => MaxAge > MinAge;
}
```

**Figure 8:** filtering method in Employee Parameter Class.

This figure shows how did I defined the minimum age and the maximum age. I used “uint” (unsigned integer) properties to avoid negative year value.

```
public async Task<PagedList<Employee>> GetEmployeesAsync(Guid companyId,
    EmployeeParameters employeeParameters, bool trackChanges)
{
    var employees = await FindByCondition(e => e.CompanyId.Equals(companyId) && (e.Age
    >= employeeParameters.MinAge && e.Age <= employeeParameters.MaxAge), trackChanges)
    .OrderBy(e => e.Name)
    .ToListAsync();

    return PagedList<Employee>
    .ToPagedList(employees, employeeParameters.PageNumber,
    employeeParameters.PageSize);
}
```

**Figure 9:** filtering method in Employee repository class.

This figure shows the implementation of our filtering method in the employee repository class. It is very simple because I am using the “FindByCondition” method to find all the employees with an Age between the Max Age and the Min Age.

```
0 references
public static class RepositoryEmployeeExtensions
{
    1 reference
    public static IQueryable<Employee> FilterEmployees(this IQueryable<Employee>
employees, uint minAge, uint maxAge) =>
    employees.Where(e => (e.Age >= minAge && e.Age <= maxAge));
    1 reference
}
```

**Figure10:** filtering method in Repository Employee Extensions class.

This figure explains what will happen if the caller calls the filtering method, it will return a filtered “**IQueryable<Employee>**” containing only the employees whose ages are within the specified range.

- **Searching method:**

```
2 references
public uint MinAge { get; set; }
2 references
public uint MaxAge { get; set; } = int.MaxValue;

1 reference
public bool ValidAgeRange => MaxAge > MinAge;

1 reference
public string? SearchTerm { get; set; }
}
```

**Figure11:** searching method in Employee Parameters class.

This figure shows that I have added the “**search term**” property, which allows us to write queries, and retrieve data from the database.

```
public async Task<PagedList<Employee>> GetEmployeesAsync(Guid companyId,
EmployeeParameters employeeParameters, bool trackChanges)
{
    var employees = await FindByCondition(e => e.CompanyId.Equals(companyId),
trackChanges)
    .FilterEmployees(employeeParameters.MinAge, employeeParameters.MaxAge)
    .Search(employeeParameters.SearchTerm)
    .OrderBy(e => e.Name)
    .ToListAsync();

    var count = await FindByCondition(e => e.CompanyId.Equals(companyId), trackChanges)
    .CountAsync();

    return new PagedList<Employee>(employees, count,
employeeParameters.PageNumber, employeeParameters.PageSize);
}
```

**Figure12:** searching method in Employee repository class.

I have made two changes here. The first is modifying the filter logic and the second is adding the Search method for the searching functionality.

```
public static IQueryable<Employee> Search(this IQueryable<Employee> employees,
string searchTerm)
{
    if (string.IsNullOrEmpty(searchTerm))
        return employees;
    var lowerCaseTerm = searchTerm.Trim().ToLower();
    return employees.Where(e => e.Name.ToLower().Contains(lowerCaseTerm));
}
```

**Figure13:** searching method in Repository Employee Extensions class.

The code in this figure is about returning only the employees whose names contain the provided search term and if the search term is empty, it will return all employees.

- **Sorting method:**

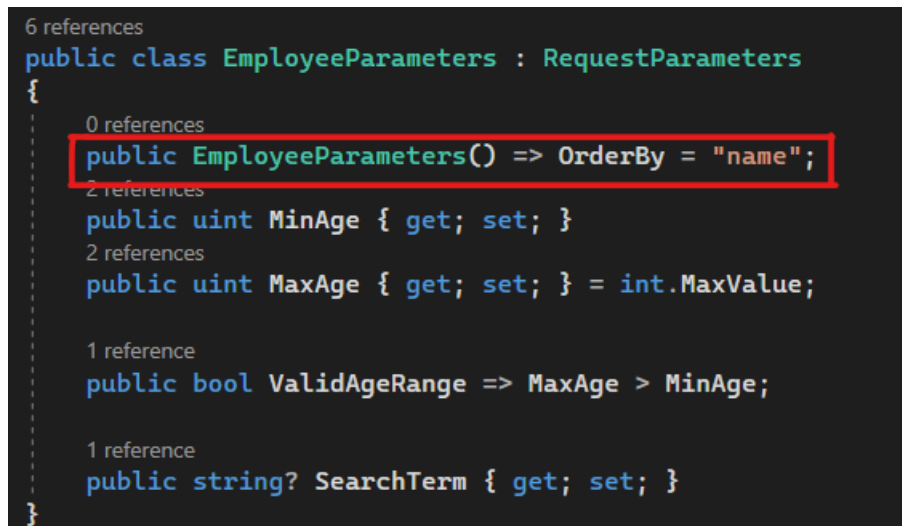
```
public abstract class RequestParameters
{
    const int maxPageSize = 50;
    1 reference
    public int PageNumber { get; set; } = 1;

    private int _pageSize = 10;
    1 reference
    public int PageSize
    {
        get
        {
            return _pageSize;
        }
        set
        {
            _pageSize = (value > maxPageSize) ? maxPageSize : value;
        }
    }
    2 references
    public string? OrderBy { get; set; }
```

**Figure14:** sorting method in Request parameters class



As you can see in the figure above, I added to the Request parameters class “**orderby**” property, it is used to specify a sorting criterion when we call the sorting method.

A screenshot of a code editor showing the C# code for the EmployeeParameters class. The class inherits from RequestParameters. The constructor is highlighted with a red rectangle and contains the lambda expression 'OrderBy = "name";'. Other properties include MinAge, MaxAge, ValidAgeRange, and SearchTerm.

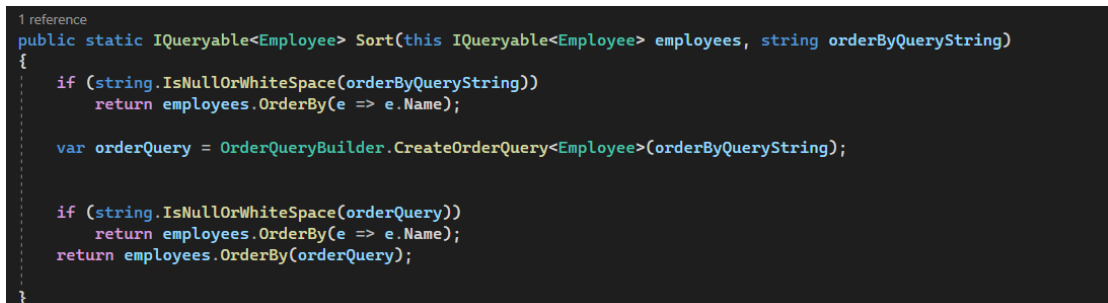
```
6 references
public class EmployeeParameters : RequestParameters
{
    0 references
    public EmployeeParameters() => OrderBy = "name";
    2 references
    public uint MinAge { get; set; }
    2 references
    public uint MaxAge { get; set; } = int.MaxValue;

    1 reference
    public bool ValidAgeRange => MaxAge > MinAge;

    1 reference
    public string? SearchTerm { get; set; }
}
```

**Figure15:** sorting method in Employee parameters class

I have enabled the default sorting condition for Employee if none was set by the caller, which is the employee’s name.

A screenshot of a code editor showing the C# code for the Sort method in the Repository Employee Extensions class. The method takes an IQueryable of Employees and a string orderByQueryString. It checks if the string is null or whitespace, and if so, returns employees ordered by name. Otherwise, it creates an OrderQuery and returns employees ordered by that query.

```
1 reference
public static IQueryable<Employee> Sort(this IQueryable<Employee> employees, string orderByQueryString)
{
    if (string.IsNullOrEmpty(orderByQueryString))
        return employees.OrderBy(e => e.Name);

    var orderQuery = OrderQueryBuilder.CreateOrderQuery<Employee>(orderByQueryString);

    if (string.IsNullOrEmpty(orderQuery))
        return employees.OrderBy(e => e.Name);
    return employees.OrderBy(orderQuery);
}
```

**Figure16:** sort method in Repository Employee Extensions class.

This figure shows how does the sorting method work, when we call the sorting method it will sort the employees based on the specified sorting criteria, and if there are no sorting criteria, it will sort the employee based on their names in ascending order by default.

Before I start showing how did I created Get, Update, Delete, Put, and patch actions, there are two important principles in HTTP standard I should explain. Those two standards are **Method Safety** and **method Idempotency**. Method safety refers to the HTTP method that does not change the resource, in other words, the resources should not be changed after the method is executed. In this case we can consider the method as a safety method.

On the other hand, we have the Idempotency method, we can consider the method Idempotency, if we call the method many times and we still get the same result. That means, wither you call the method once or multiple times, the result would be the same. **Figure 17 below** shows the safety and Idempotency HTTP methods:

HTTP Method	Is Safe?	Is Idempotent?
GET	YES	YES
OPTIONS	YES	YES
HEAD	NO	YES
POST	NO	NO
DELETE	NO	YES
PUT	NO	YES
PATCH	NO	YES

**Figure 17:** safety and Idempotency

It is possible for the HTTP method to be both safety and Idempotency, consider **Get** method as an example, if we execute the Get method one time the resource would not change, and even if we call it many times, we will get the same result. Because we are not modifying the resources. This also applies for the **OPTIONS** and **HEAD** method.

We cannot consider the **POST** method neither safe nor idempotent, because when we call it one time it will change the resource, because it creates them. And every time we call it, it will create a new resource.

The **DELETE** is not safe, because it removes the resource, but it is Idempotent, because if we remove the same resource multiple times, we will get the same result as removing it one time. The **PUT** method is not safe as well, because when we update our resource it will make changes, but it is idempotent, because no matter how many times we update our resource with the same request we will get the same result.

Finally, the **PATCH** method is similar to the **PUT** method, the only difference between them is that the **PATCH** method is used to apply partial modifications to a resource. On the contrary, the **PUT** method is used to update all entities of the resources. **Figures** below shows the implementation of each request:

- GET request:

```
[Route("api/companies")]
[ApiController]
1 reference
public class CompaniesController : ControllerBase
{
    private readonly IServiceManager _service;

    0 references
    public CompaniesController(IServiceManager service) => _service = service;

    [HttpGet()]
    0 references
    public async Task<IActionResult> GetCompanies()
    {
        var companies = await _service.CompanyService.GetAllCompaniesAsync(trackChanges: false);
        return Ok(companies);
    }

    [HttpGet("{id:guid}", Name = "CompanyById")]
    0 references
    public async Task<IActionResult> GetCompany(Guid id)
    {
        var company = await _service.CompanyService.GetCompanyAsync(id, trackChanges: false);
        return Ok(company);
    }
}
```

**Figure 18:** shows the implementation of GET method, we can get all companies, or we can get a specific company by ID.

- Delete request:

```
[HttpDelete("{id:guid}")]
0 references
public async Task<IActionResult> DeleteCompany(Guid id)
{
    await _service.CompanyService.DeleteCompanyAsync(id, trackChanges: false);
    return NoContent();
}
```

**Figure 19:** The implementation of delete request.

- PUT request:

```
[HttpPut("{id:guid}")]
[ServiceFilter(typeof(ValidationFilterAttribute))]
0 references
public async Task<IActionResult> UpdateCompany(Guid id, [FromBody] CompanyForUpdateDto company)
{
    if (company is null)
        return BadRequest("CompanyForUpdate object is null. ");
    await _service.CompanyService.UpdateCompanyAsync(id, company, trackChanges: true);
    return NoContent();
}
```

**Figure 20:** The implementation of put request.

- PATCH request:

```
[HttpPatch("{id:guid}")]
0 references
public async Task<IActionResult> PartiallyUpdateEmployeeForCompany(Guid companyId, Guid id,
    [FromBody] JsonPatchDocument<EmployeeForUpdateDto> patchDoc)
{
    if (patchDoc is null)
        return BadRequest("patchDoc object sent from client is null.");
    var result = await _service.EmployeeService.GetEmployeeForPatchAsync(companyId, id,
        compTrackChanges: false,
        empTrackChanges: true);
    patchDoc.ApplyTo(result.employeeToPatch, ModelState);
    TryValidateModel(result.employeeToPatch);
    if (!ModelState.IsValid)
        return UnprocessableEntity(ModelState);
    await _service.EmployeeService.SaveChangesForPatchAsync(result.employeeToPatch, result.employeeEntity);
    return NoContent();
}
```

**Figure 21:** The implementation of patch request.

There are Additional Information about PATCH request, I mentioned before the difference between PUT and PATCH requests, but now I am going to explain the operations of the PATCH, there are **6** different operations for the Patch requests as shown in **Figure 22**:

OPERATION	REQUEST BODY	EXPLANATION
<b>Add</b>	<pre>{   "op": "add",   "path": "/name",   "value": "new value" }</pre>	Assigns a new value to a required property.
<b>Remove</b>	<pre>{   "op": "remove",   "path": "/name" }</pre>	Sets a default value to a required property.
<b>Replace</b>	<pre>{   "op": "replace",   "path": "/name",   "value": "new value" }</pre>	Replaces a value of a required property to a new value.
<b>Copy</b>	<pre>{   "op": "copy",   "from": "/name",   "path": "/title" }</pre>	Copies the value from a property in the "from" part to the property in the "path" part.
<b>Move</b>	<pre>{   "op": "move",   "from": "/name",   "path": "/title" }</pre>	Moves the value from a property in the "from" part to a property in the "path" part.
<b>Test</b>	<pre>{   "op": "test",   "path": "/name",   "value": "new value" }</pre>	Tests if a property has a specified value.

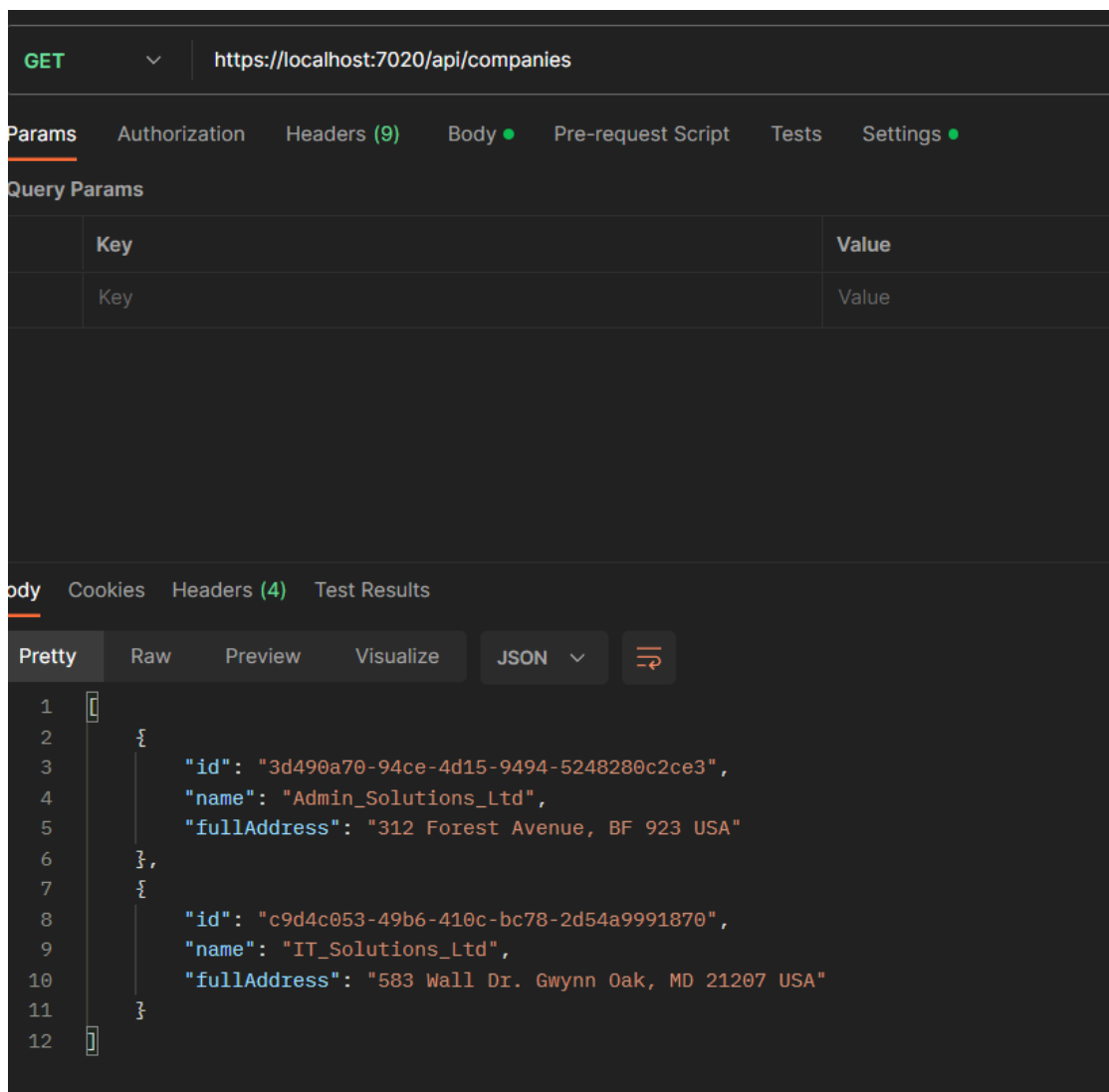
**Figure 22:** shows the operations, the request body, and the explanation of each one.

The only requests I was required to implement are GET, PUT, PATCH, and DELETE. I was not required to implement the HEAD, POST, and OPTION requests.

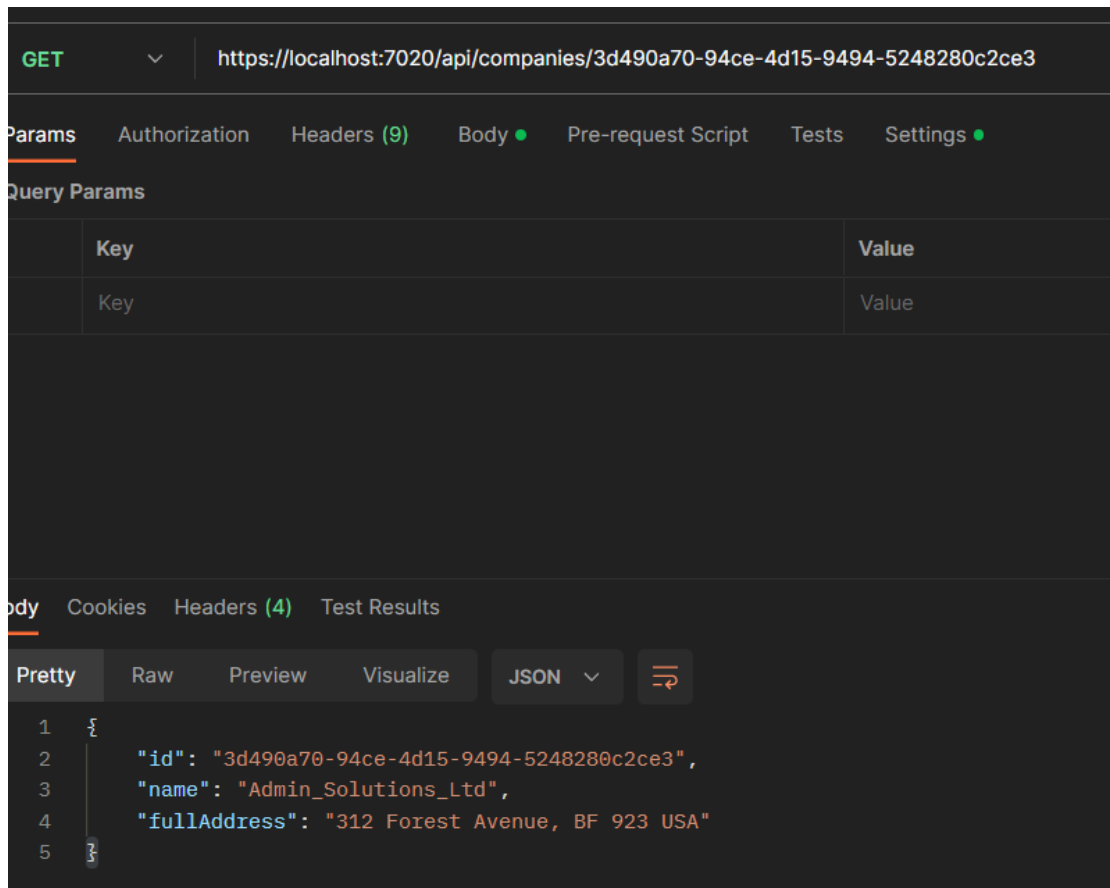
## 5. RESULTS AND DISCUSSION

In this section, I am going to show the results of the HTTP method that I implemented, which are GET, PUT, PATCH, and DELETE requests. In order to get the results, I was asked to use postman tool. And I am going to provide the URL request that I used to request specific data from the database, **Figures** below shows the result of calling each method:

I am going to start with the GET method:

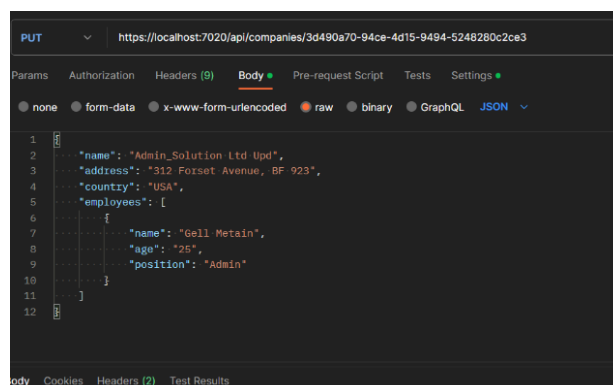


**Figure 23:** shows the output of the GET request, which get all companies in the Database, I used “https://localhost:7020/api/companies” URL.



**Figure 24:** shows the output of getting any company by its Id, I used “https://localhost:7020/api/companies/3d490a70-94ce-4d15-9494-5248280c2ce3” URL.

PUT request:



**Figure 25:** shows the query that I wrote to add an employee to a chosen company. I used “https://localhost:7020/api/companies/3d490a70-94ce-4d15-9494-5248280c2ce3” URL request.

```
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (14ms) [Parameters=[@p2='?' (DbType = Guid), @p0='?' (Size = 60), @p1='?' (Size = 60), @p3='?' (DbType = Guid), @p4='?' (DbType = Int32), @p5='?' (DbType = Guid), @p6='?' (Size = 30), @p7='?' (Size = 20)], CommandType='Text', CommandTimeout='30']
      SET NOCOUNT ON;
      UPDATE [Companies] SET [Address] = @p0, [Name] = @p1
      OUTPUT 1
      WHERE [CompanyId] = @p2;
      INSERT INTO [Employees] ([EmployeeId], [Age], [CompanyId], [Name], [Position])
      VALUES (@p3, @p4, @p5, @p6, @p7);
```

**Figure 26:** shows the output of the PUT method, it shows how it adds the new employee to the chosen company.

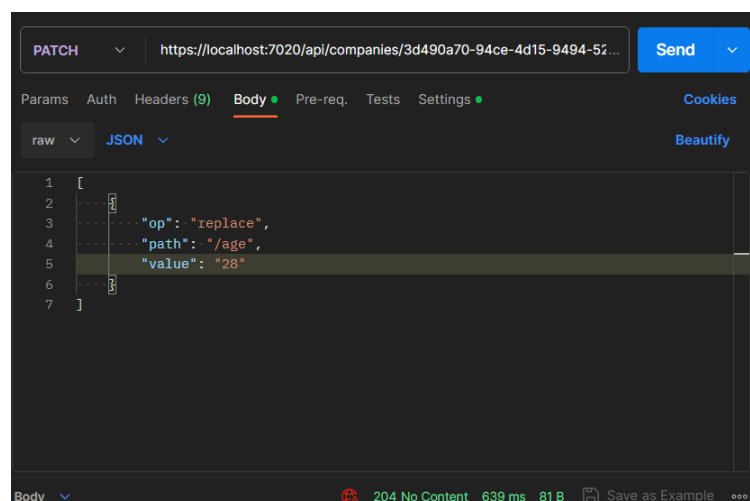
PATCH request:

As I said before there are 6 different operations In PATCH request, Add, Remove, Replace, Copy, Move, and Test. I am going to show some implementation of them, starting with the Replace operation, as you can see in the figure below the employee's age is 25 and I want to change it to 28 for example.

```
{
  "Id": "9503b52c-30b1-44e6-073e-08dbc4311866",
  "Name": "Gell Metain",
  "Age": 25,
  "Position": "Admin"
},
```

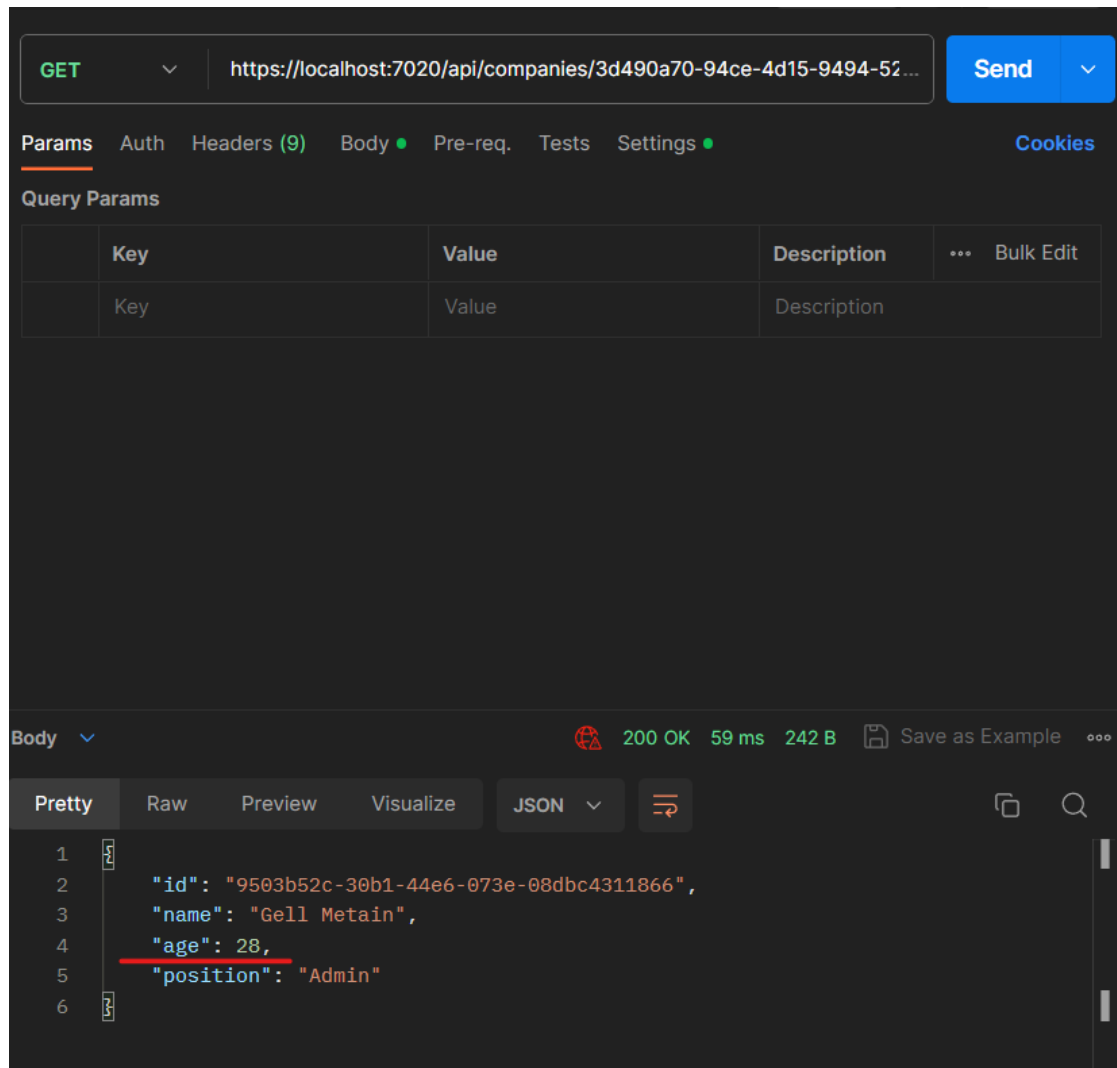
**Figure:** The Employee's age that we want to change.

Basically, I am going to write the query of replacement in PATCH request which shown in the figure below:





**Figure 27:** shows the query, “op” stands for operation, and I declared the path which is age then I assigned the new value.



**Figure 28:** shows that when we get the same employee, his age will be assigned to the new value which is 28.

And if we want to remove his age, we can change the operation to “remove” and send the request again, it will remove the employee’s age and assign it to zero. And if we want to reassign him age again, we can replace the operation “remove” to “add” and it will assign a new value for the age. **Figure 29, and 30** below show the result of “remove” and “add” operation.



**Figure 29:** shows the removing query.

```

1 {
2   "id": "80abbca8-664d-4b20-b5de-024705497d4a",
3   "name": "Sam Raiden",
4   "age": 0,
5   "position": "Software developer"
6 }

```

**Figure 30:** shows the result after removing the age of the employee.



**Figure 31:** shows the adding query.

```

1 {
2   "id": "80abbca8-664d-4b20-b5de-024705497d4a",
3   "name": "Sam Raiden",
4   "age": 28,
5   "position": "Software developer"
6 }

```

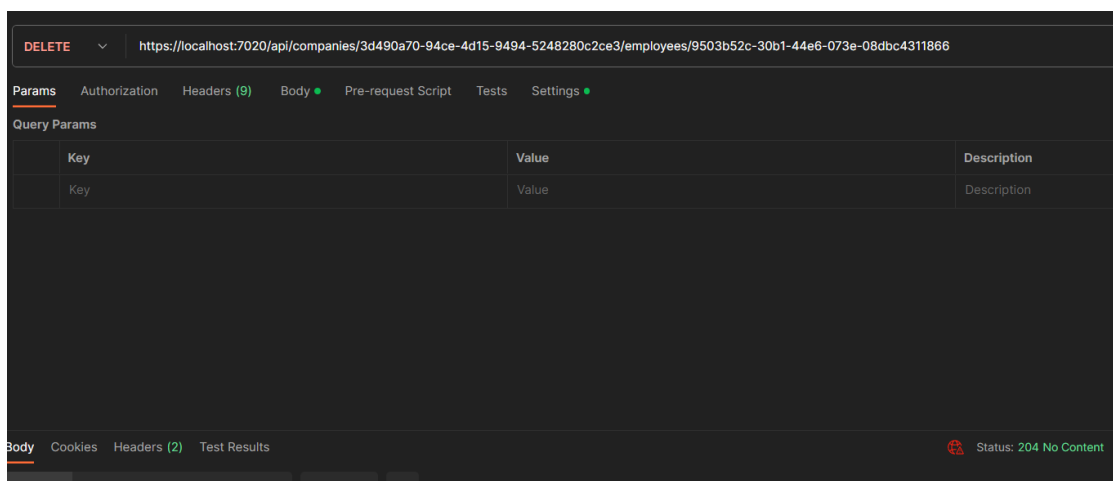
**Figure 32:** shows the result after Adding the age of the employee.

DELETE request:

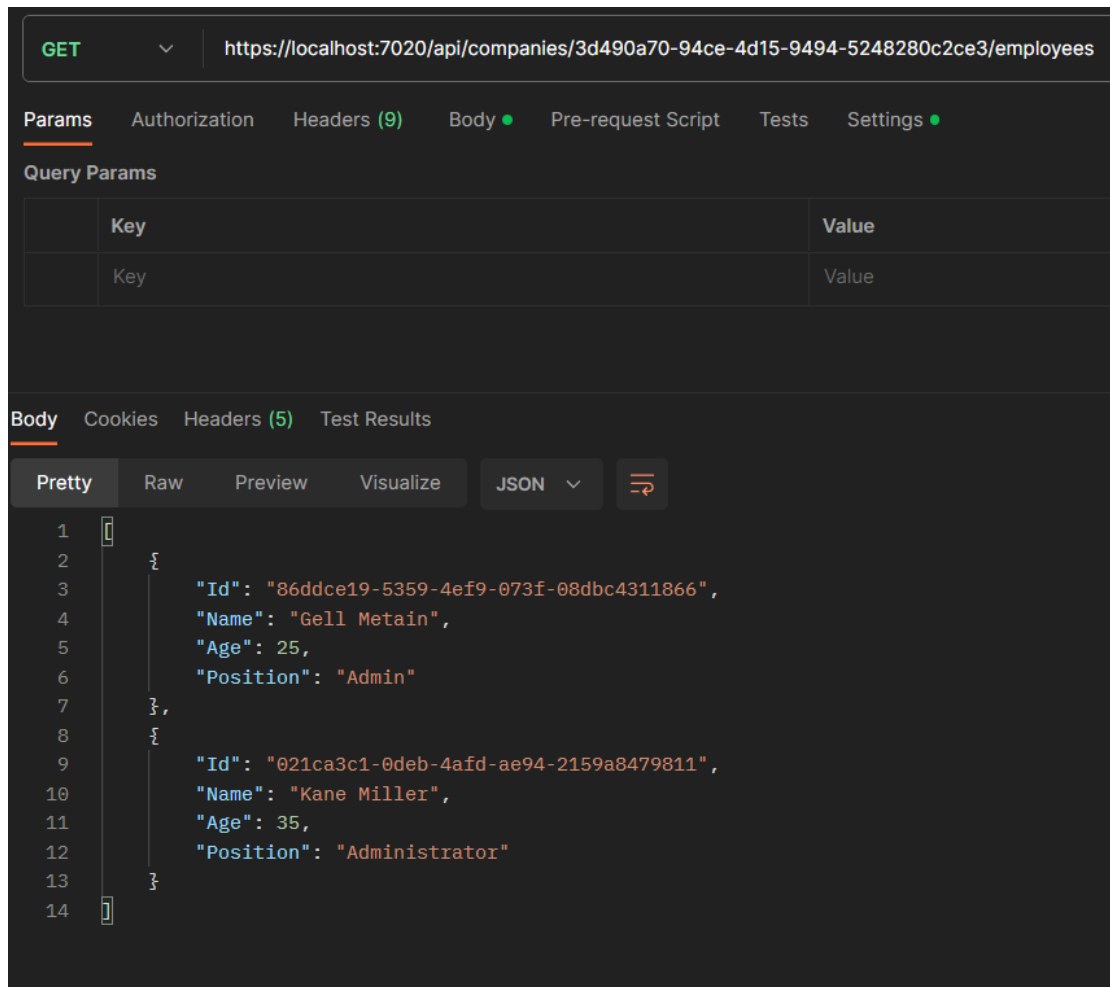
If we want to delete an employee from the company, we can use DELETE request. Now I am going to delete an employee from a company and show you the difference between getting all employee after and before deleting as shown in the **Figures 33, 34, and 35**.

```
1  [
2    {
3      "Id": "9503b52c-30b1-44e6-073e-08dbc4311866",
4      "Name": "Gell Metain",
5      "Age": 28,
6      "Position": "Admin"
7    },
8    {
9      "Id": "86ddce19-5359-4ef9-073f-08dbc4311866",
10     "Name": "Gell Metain",
11     "Age": 25,
12     "Position": "Admin"
13   },
14 ]
```

**Figure 33:** shows the result of getting all employees before deleting any of them.



**Figure 34:** sending deleting request to delete the employee with Id “9503b52c-30b1-44e6-073e-08dbc4311866”.



**Figure 35:** getting all employees after deleting the employee with the Id “9503b52c-30b1-44e6-073e-08dbc4311866”.

## **6. NEW KNOWLEDGE GAINED THROUGH THE EXPERIENCE**

- What new knowledge did you learn in your summer training?

During my training I learned how to manage my time better and became more aware of time management. Furthermore, I gained more experience when it comes to working in a team and dealing with people from different backgrounds. In addition, I learned more about the basics of C# programming language and when to use it, and I discovered what is API and how to create an API, I understood how HTTP methods work and how to use postman tool for testing. These learning experiences have helped me grow and become more skilled in these areas.

- How did you use the new knowledge?

I used this knowledge very well, where I firstly learnt the C# Programming language and build many consoles application to enhance myself more and more, then I went to REST Full APIs with ASP.NET Core, here I had to learn about ASP.NET Core, and .NET Family, I searched about it more in order to build the Company Employee System that I mentioned in the report previously. Finally, I learnt how to use Postman in order to test my results (sending request and receiving responses) thorough them.

- What strategies did you use in learning the new knowledge?

The strategies that I used while learning these new knowledges are YouTube videos, reading books and articles, reading Microsoft documents about C#, .NET, ASP.NET Core, etc. and asking people who are already experienced in this field.

## **7. CONCLUSIONS**

In conclusion, my internship at Cloud Soft Company was a transformative experience that equipped me not only with technical skills but also with valuable life lessons. I learned the significance of mastering one programming language at a time to ensure a strong foundation and prevent confusion. My journey into C# and web development, guided by dedicated mentors, allowed me to apply my knowledge in practical projects. Furthermore, I gained a deep appreciation for the importance of time management, a skill that I now consider essential for success in any professional endeavor. As I move forward in my career, I carry these lessons with me, confident in my ability to navigate the dynamic world of software engineering. These 40 days, have changed me a lot and I'm sure it will have a good contribution to my future career, and even future studies.

## **8. REFERENCES**

[.NET documentation | Microsoft Learn](#)

[ASP.NET documentation | Microsoft Learn](#)

[C# docs - get started, tutorials, reference. | Microsoft Learn](#)

[.NET Core Tutorial - Creating a Restful Web API - Code Maze \(code-maze.com\)](#)

[Download Postman | Get Started for Free](#)

[Cloudsoft \(cloud-soft.tech\)](#)

## 9. APPENDIX

### APPENDIX A: some other codes

- **Program.cs file:**

```
using CompanyEmployees.Extension;
using Contracts;
using Microsoft.AspNetCore.HttpOverrides;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;
using NLog;
using Microsoft.AspNetCore.Mvc.Formatters;
using CompanyEmployees.Presentation.ActionFilters;
using Service.DataShaping;
using Shared.DataTransferObjects;

var builder = WebApplication.CreateBuilder(args);

LogManager.LoadConfiguration(string.Concat(Directory.GetCurrentDirectory(),
"/nlog.config"));

NewtonsoftJsonPatchInputFormatter GetJsonPatchInputFormatter() =>
new ServiceCollection().AddLogging().AddMvc().AddNewtonsoftJson()
.Services.BuildServiceProvider()
.GetRequiredService<IOptions<MvcOptions>>().Value.InputFormatters
.Of<NewtonsoftJsonPatchInputFormatter>().First();

// Add services to the container.
builder.Services.ConfigureCors();
builder.Services.ConfigureIISIntegration();
builder.Services.ConfigureLoggerService();
builder.Services.ConfigureRepositoryManager();
builder.Services.ConfigureServiceManager();
builder.Services.AddAutoMapper(typeof(Program));

builder.Services.Configure<ApiBehaviorOptions>(options =>
{
    options.SuppressModelStateInvalidFilter = true;
});
builder.Services.AddScoped<ValidationFilterAttribute>();

builder.Services.AddControllers(config =>
{
    config.RespectBrowserAcceptHeader = true;
    config.ReturnHttpNotAcceptable = true;
    config.InputFormatters.Insert(0, GetJsonPatchInputFormatter());
}).AddXmlDataContractSerializerFormatters()
.AddCustomCSVFormatter()
.AddApplicationPart(typeof(CompanyEmployees.Presentation.AssemblyReference).Assembly);

builder.Services.AddScoped<IDataShaper<EmployeeDto>, DataShaper<EmployeeDto>>();

builder.Services.ConfigureSqlContext(builder.Configuration);

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



```

var logger = app.Services.GetRequiredService<ILoggerManager>();
app.ConfigureExceptionHandler(logger);

if (app.Environment.IsProduction())
    app.UseHsts();

// Configure the HTTP request pipeline.

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.All
});
app.UseCors("CorsPolicy");

app.UseAuthorization();

app.MapControllers();

app.Run();

```

- **Configuration file:**

- a) **For the company:**

```

using Entites.Models;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Microsoft.EntityFrameworkCore;

namespace Repository_.Configuration
{
    public class CompanyConfiguration : IEntityTypeConfiguration<Company>
    {
        public void Configure(EntityTypeBuilder<Company> builder)
        {
            builder.HasData
            (
                new Company
                {
                    Id = new Guid("c9d4c053-49b6-410c-bc78-2d54a9991870"),
                    Name = "IT_Solutions_Ltd",
                    Address = "583 Wall Dr. Gwynn Oak, MD 21207",
                    Country = "USA"
                },
                new Company
                {
                    Id = new Guid("3d490a70-94ce-4d15-9494-5248280c2ce3"),
                    Name = "Admin_Solutions_Ltd",
                    Address = "312 Forest Avenue, BF 923",
                    Country = "USA"
                }
            );
        }
    }
}

```

- b) **for the Employee**

```

using Entites.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Repository_.Configuration
{
    public class EmployeeConfiguration : IEntityTypeConfiguration<Employee>
    {
        public void Configure(EntityTypeBuilder<Employee> builder)
        {
            builder.HasData
            (
                new Employee
                {
                    Id = new Guid("80abbca8-664d-4b20-b5de-024705497d4a"),
                    Name = "Jana McLeaf",
                    Age = 26,
                    Position = "Software developer",
                    CompanyId = new Guid("c9d4c053-49b6-410c-bc78-2d54a9991870")
                },
                new Employee
                {
                    Id = new Guid("86dba8c0-d178-41e7-938c-ed49778fb52a"),
                    Name = "Sam Raiden",
                    Age = 30,
                    Position = "Software developer",
                    CompanyId = new Guid("c9d4c053-49b6-410c-bc78-2d54a9991870")
                },
                new Employee
                {
                    Id = new Guid("021ca3c1-0deb-4afd-ae94-2159a8479811"),
                    Name = "Kane Miller",
                    Age = 35,
                    Position = "Administrator",
                    CompanyId = new Guid("3d490a70-94ce-4d15-9494-5248280c2ce3")
                }
            );
        }
    }
}

```

## APPENDIX B: Entities and models

### a) for Company

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

```

```

namespace Entites.Models
{
    public class Company
    {
        [Column("CompanyId")]
        public Guid Id { get; set; }
        [Required(ErrorMessage = "Company name is a required field.")]
        [MaxLength(60, ErrorMessage = "Maximum length for the Name is 60 characters.")]
        public string? Name { get; set; }
        [Required(ErrorMessage = "Company address is a required field.")]
        [MaxLength(60, ErrorMessage = "Maximum length for the Address is 60 characters")]
        public string? Address { get; set; }
        public string? Country { get; set; }
        public ICollection<Employee>? Employees { get; set; }
    }
}

```

## b) for Employee

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Entites.Models
{
    public class Employee
    {
        [Column("EmployeeId")]
        public Guid Id { get; set; }
        [Required(ErrorMessage = "Employee name is a required field.")]
        [MaxLength(30, ErrorMessage = "Maximum length for the Name is 30 characters.")]
        public string? Name { get; set; }
        [Required(ErrorMessage = "Age is a required field.")]
        public int Age { get; set; }
        [Required(ErrorMessage = "Position is a required field.")]
        [MaxLength(20, ErrorMessage = "Maximum length for the Position is 20 characters.")]
        public string? Position { get; set; }
        [ForeignKey(nameof(Company))]
        public Guid CompanyId { get; set; }
        public Company? Company { get; set; }
    }
}

```