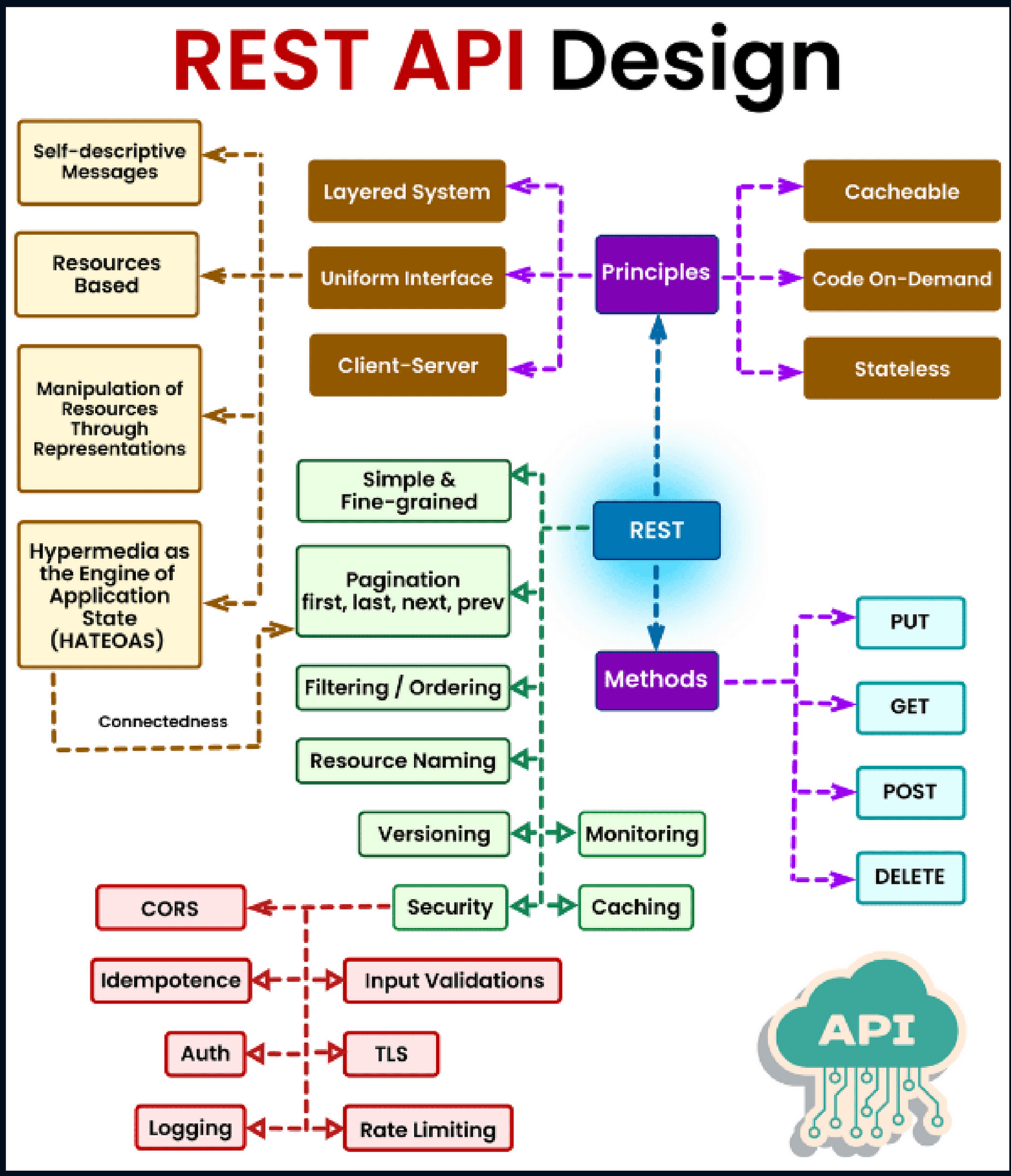
REST API Design



Design Rest APIs

Endpoints examples

In these examples, the attribute routing syntax is used to define the routes and HTTP methods for the respective endpoints. The [HttpPost], [HttpGet], and [HttpDelete] attributes are used to specify the HTTP method for each endpoint.

```
[HttpDelete("users/{id}")]
public IActionResult DeleteUser(int id)
{
    // Logic to find and delete the user from the database based on the
provided ID
    // ...

    if (user == null)
    {
        // Return a 404 Not Found if the user is not found
        return NotFound();
    }

    // Delete the user
    // ...

    // Return a 204 No Content status code to indicate successful deletion
    return NoContent();
}
```

Delete Request

```
[HttpPut("users/{id}")]
public IActionResult UpdateUser(int id, [FromBody] User updatedUser)
{
    // Logic to fetch the user from the database based on the provided ID
    // ...

    if (user == null)
    {
        // Return a 404 Not Found if the user is not found
        return NotFound();
    }

    // Update the properties of the existing user with the provided data
    user.Name = updatedUser.Name;
    user.Email = updatedUser.Email;
    // ...

    // Save the changes to the database
    // ...

    // Return the updated user with a 200 OK status code
    return Ok(user);
}
```

Put Request

Resource Naming guidelines

Nouns: things, not actions

- ~~api/getauthors~~
- GET api/authors
- GET api/authors/{authorId}

Convey meaning when choosing nouns

Follow through on this principle for predictability

- ~~api/something/somethingelse/employees~~
- api/employees
- ~~api/id/employees~~
- api/employees/{employeeId}

Represent hierarchy when naming resources

- api/authors/{authorId}/courses
- api/authors/{authorId}/courses/{courseId}

Resource identifiers

- Use pluralized nouns that convey meaning
- Represent model hierarchy
- Be consistent

Filters, sorting orders, ... aren't resources

- ~~api/authors/orderby/name~~
- api/authors?orderby=name

Sometimes, RPC-style calls don't easily map to pluralized resource names

- ~~api/authors/{authorId}/pagetotals~~
- ~~api/authorpagetotals/{id}~~
- api/authors/{authorId}/totalamountofpages

Content negotiation

Formatters and content negotiation - the media type is passed through via the Accept header of the request. application/json, application/xml, are some of the example.

Output formatter: deals with the output, the header that it uses is accept header
Input formatter: deals with the input, the header that it usesis Content-type

```
[HttpGet("{fileId}")]
0 references
public ActionResult GetFile(string fileId)
{
    // look up the actual file, depending on the fileId...
    // demo code
    var pathToFile = "getting-started-with-rest-slides.pdf";

    // check whether the file exists
    if (!System.IO.File.Exists(pathToFile))
    {
        return NotFound();
    }

    if (!_fileExtensionContentTypeProvider.TryGetContentType(
        pathToFile, out var contentType))
    {
        contentType = "application/octet-stream";
    }

    var bytes = System.IO.File.ReadAllBytes(pathToFile);
    return File(bytes, contentType, Path.GetFileName(pathToFile));
}
```

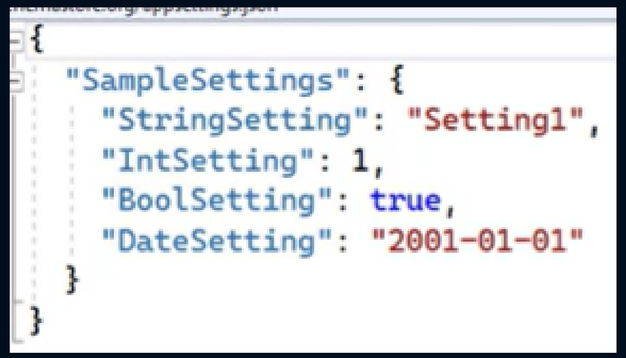
Settings to log later of debugging exp

```
builder.Services.AddSingleton<FileExtensionContentTypeProvider>();

private readonly FileExtensionContentTypeProvider _fileExtensionContentTypeProvider;

0 references
public FilesController(
    FileExtensionContentTypeProvider fileExtensionContentTypeProvider)
{
    _fileExtensionContentTypeProvider = fileExtensionContentTypeProvider
        ?? throw new System.ArgumentNullException(
            nameof(fileExtensionContentTypeProvider));
}
```

Example code

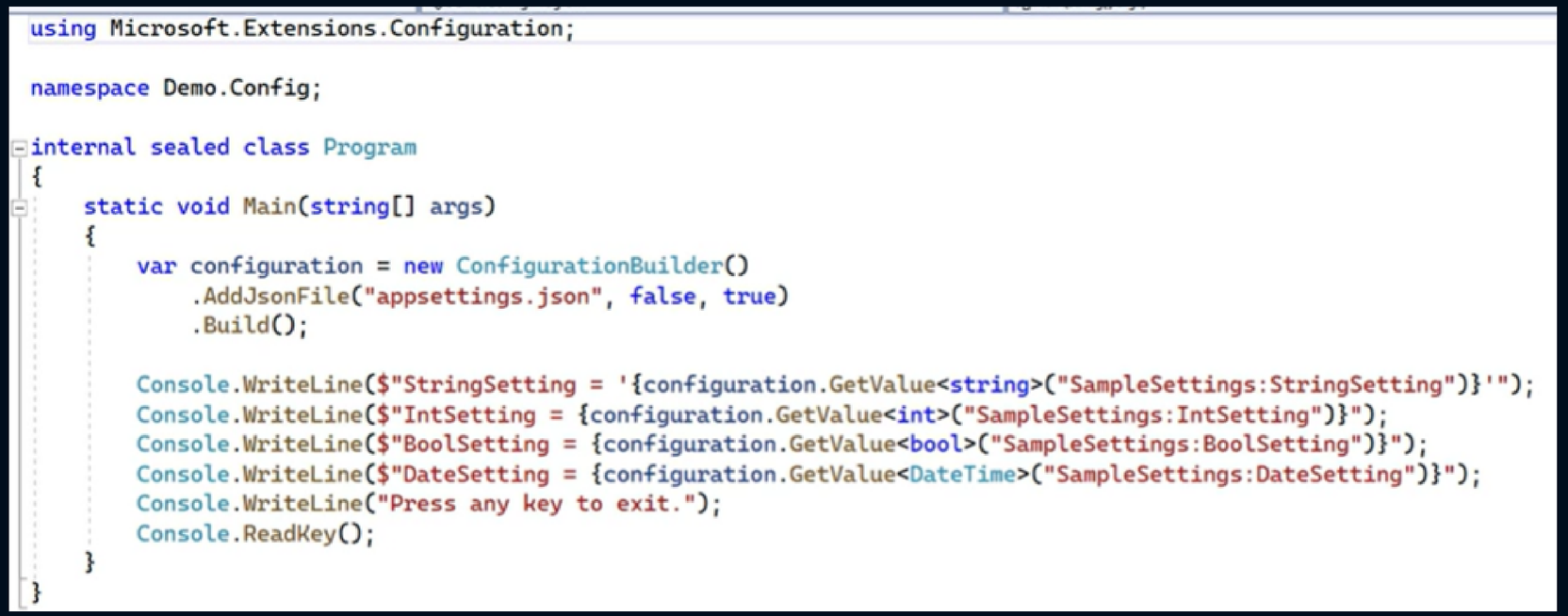
App settings

By using the appsettings.json file, you can centralize and manage your application's configuration settings in a structured and flexible way. App settings typically store information such as connection strings, API keys, feature flags, and other configuration values that your application needs to function correctly. By storing these values in app settings, you can easily manage and update them without modifying your application's code.



```
{
  "SampleSettings": {
    "StringSetting": "Setting1",
    "IntSetting": 1,
    "BoolSetting": true,
    "DateSetting": "2001-01-01"
  }
}
```

App settings example



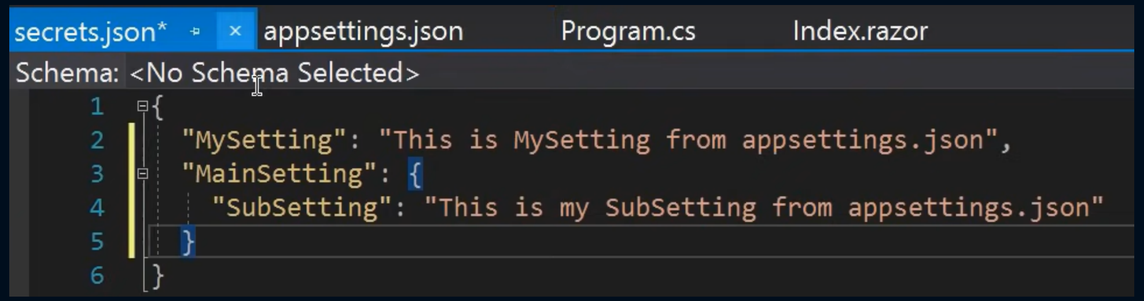
```
using Microsoft.Extensions.Configuration;

namespace Demo.Config;

internal sealed class Program
{
    static void Main(string[] args)
    {
        var configuration = new ConfigurationBuilder()
            .AddJsonFile("appsettings.json", false, true)
            .Build();

        Console.WriteLine($"StringSetting = '{configuration.GetValue<string>("SampleSettings:StringSetting")}'");
        Console.WriteLine($"IntSetting = {configuration.GetValue<int>("SampleSettings:IntSetting")}");
        Console.WriteLine($"BoolSetting = {configuration.GetValue<bool>("SampleSettings:BoolSetting")}");
        Console.WriteLine($"DateSetting = {configuration.GetValue<DateTime>("SampleSettings:DateSetting")}");
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
```

App settings in console app

Other configurations sources

DOTNET variables are environment variables specific to .NET applications. They can be used to configure various aspects of the .NET runtime and development environment.
User secrets are a mechanism provided by .NET Core for storing sensitive configuration data during development. They are meant to be used in local development environments and are not deployed with the application

DOTNET Environment variables

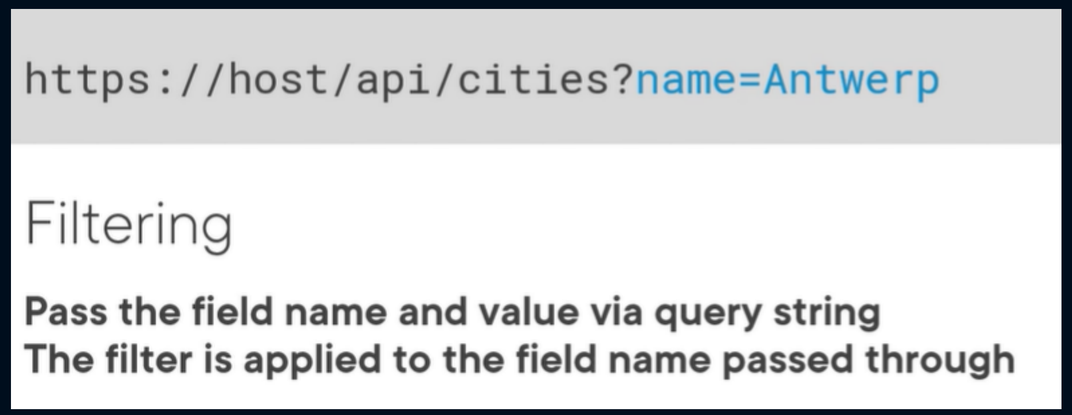


User Secrets

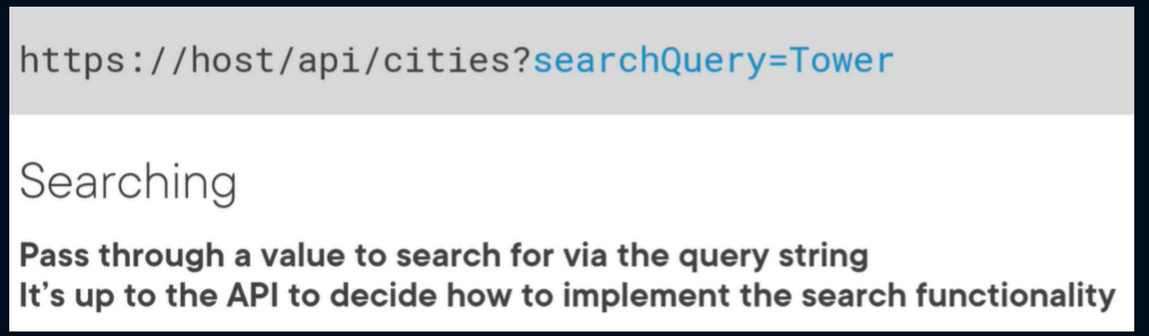
Filtering and searching

Filtering allows you to be precise by adding filters until you get exactly the result you want.

Searching allows you to go wider, it is used when you don't exactly know which items will be in the collection.



Filtering



Searching

Paging

Collection resources often grow quite large, that is why it is recommended to implement paging. Paging helps avoid performance issues.
Page size should be limited
You should select a page by default
Page all the way through the underlying data store

```
https://host/api/cities?pageNumber=1&pageSize=5
```

Paging through Resources

Pass parameters via the query string

Using paging via query params

```
[HttpGet]
0 references
public async Task<ActionResult<IEnumerable<CityWithoutPointsOfInterestDto>>> GetCities(
    string? name, string? searchQuery, int pageNumber = 1, int pageSize = 10)
{
    if (pageSize > maxCitiesPageSize)
    {
        pageSize = maxCitiesPageSize;
    }

    var cityEntities = await _cityInfoRepository
        .GetCitiesAsync(name, searchQuery, pageNumber, pageSize);
    return Ok(_mapper.Map<IEnumerable<CityWithoutPointsOfInterestDto>>(cityEntities));
}
```

Get with filter, search and paging

```
if (!string.IsNullOrWhiteSpace(searchQuery))
{
    searchQuery = searchQuery.Trim();
    collection = collection.Where(a => a.Name.Contains(searchQuery)
        || (a.Description != null && a.Description.Contains(searchQuery)));
}

return await collection.OrderBy(c => c.Name)
    .Skip(pageSize * (pageNumber - 1))
    .Take(pageSize)
    .ToListAsync();
```

Implement searching and paging

Paging metadata

When requesting application/json, paging metadata isn't part of the resource representation. Use a custom header, like X-Pagination

```
1 reference
public class PaginationMetadata
{
    1 reference
    public int TotalItemCount { get; set; }
    1 reference
    public int TotalPageCount { get; set; }
    1 reference
    public int PageSize { get; set; }
    1 reference
    public int CurrentPage { get; set; }

    0 references
    public PaginationMetadata(int totalItemCount, int pageSize, int currentPage)
    {
        TotalItemCount = totalItemCount;
        PageSize = pageSize;
        CurrentPage = currentPage;
        TotalPageCount = (int)Math.Ceiling(totalItemCount / (double)pageSize);
    }
}
```

Pagination metadata definition

```
[HttpGet]
0 references
public async Task<ActionResult<IEnumerable<CityWithoutPointsOfInterestDto>>> GetCities(
    string? name, string? searchQuery, int pageNumber = 1, int pageSize = 10)
{
    if (pageSize > maxCitiesPageSize)
    {
        pageNumber = maxCitiesPageSize;
    }

    var (cityEntities, paginationMetadata) = await _cityInfoRepository
        .GetCitiesAsync(name, searchQuery, pageNumber, pageSize);

    Response.Headers.Add("X-Pagination",
        JsonSerializer.Serialize(paginationMetadata));

    return Ok(_mapper.Map<IEnumerable<CityWithoutPointsOfInterestDto>>(cityEntities));
}
```

Controller method configuration

```
var totalItemCount = await collection.CountAsync();

var paginationMetadata = new PaginationMetadata(
    totalItemCount, pageSize, pageNumber);

var collectionToReturn = await collection.OrderBy(c => c.Name)
    .Skip(pageSize * (pageNumber - 1))
    .Take(pageSize)
    .ToListAsync();

return (collectionToReturn, paginationMetadata);
```

Returning pagination metadata

Demo

We will check the following:

Showing App settings and secrets

Showing a controller implementation

Controller has GET, POST, PUT

You can check the following Repository for some examples:

C# fundamentals

You can check the following Repository(ies) for some examples:

ASP NET Core 6 Web API

Homework

Exercise/ Homework

This is a continuation of the previous exercise.

Implement the API for a specific Domain model. You can implement for more than one but **at least the implementation for one is required**

- Implement DELETE and PUT endpoints
- Implement/update a GET request endpoint that supports paging. If it also supports filtering then is a plus.
- In case you are using a database ensure that ConnectionString is an appsetting, and the value is not in your remote repository but under a secret. In case you are using a in memory collection as database, add to the constructor a required parameter that will be called ConnectionString, and ensure that if the ConnectionString value is not valid then and exception is thrown; handle this Connection string as defined before for the database case.

Note: You can use other ideas generated from the design process from the previous workflow if it helps

The focus of this exercise is on the implementation of the API, so it is not necessary to use a database, since we will be covering that topic later. But as mentioned it is allowed