# ASP.NET WEBAPI INTRO

AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER
ENGINEERING

ST3ITS3
26 OCTOBER 2023

HENRIK BITSCH KIRK
ASSOCIATE PROFESSOR

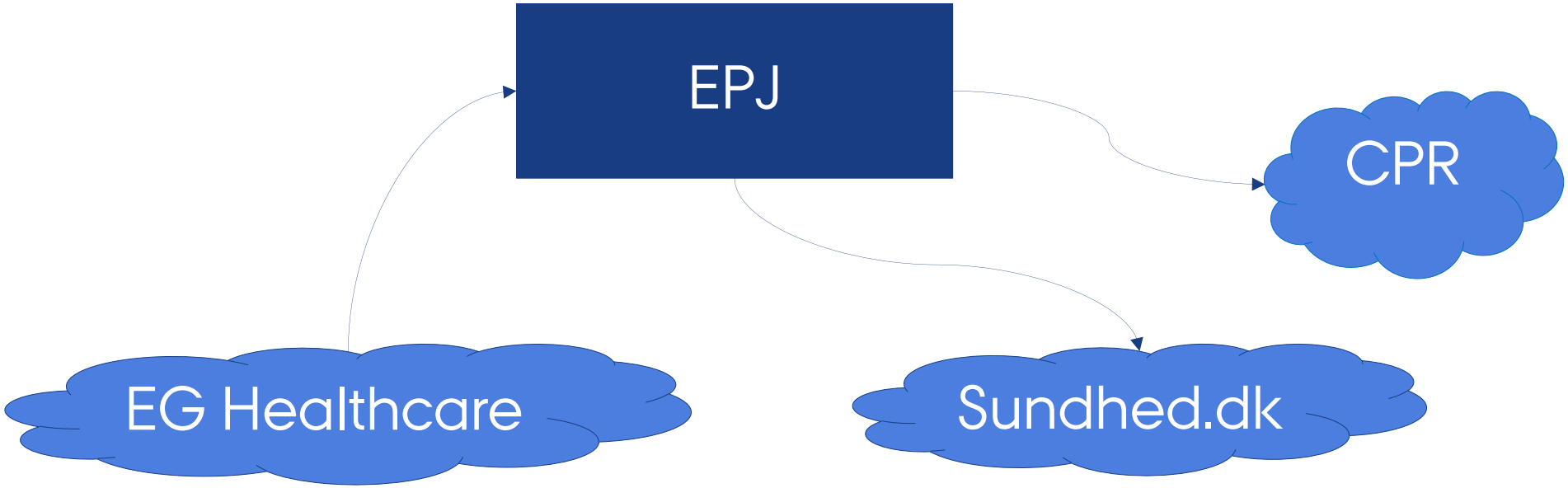
# AGENDA

- Problem
- XML / JSON
- API
- ASP.NET

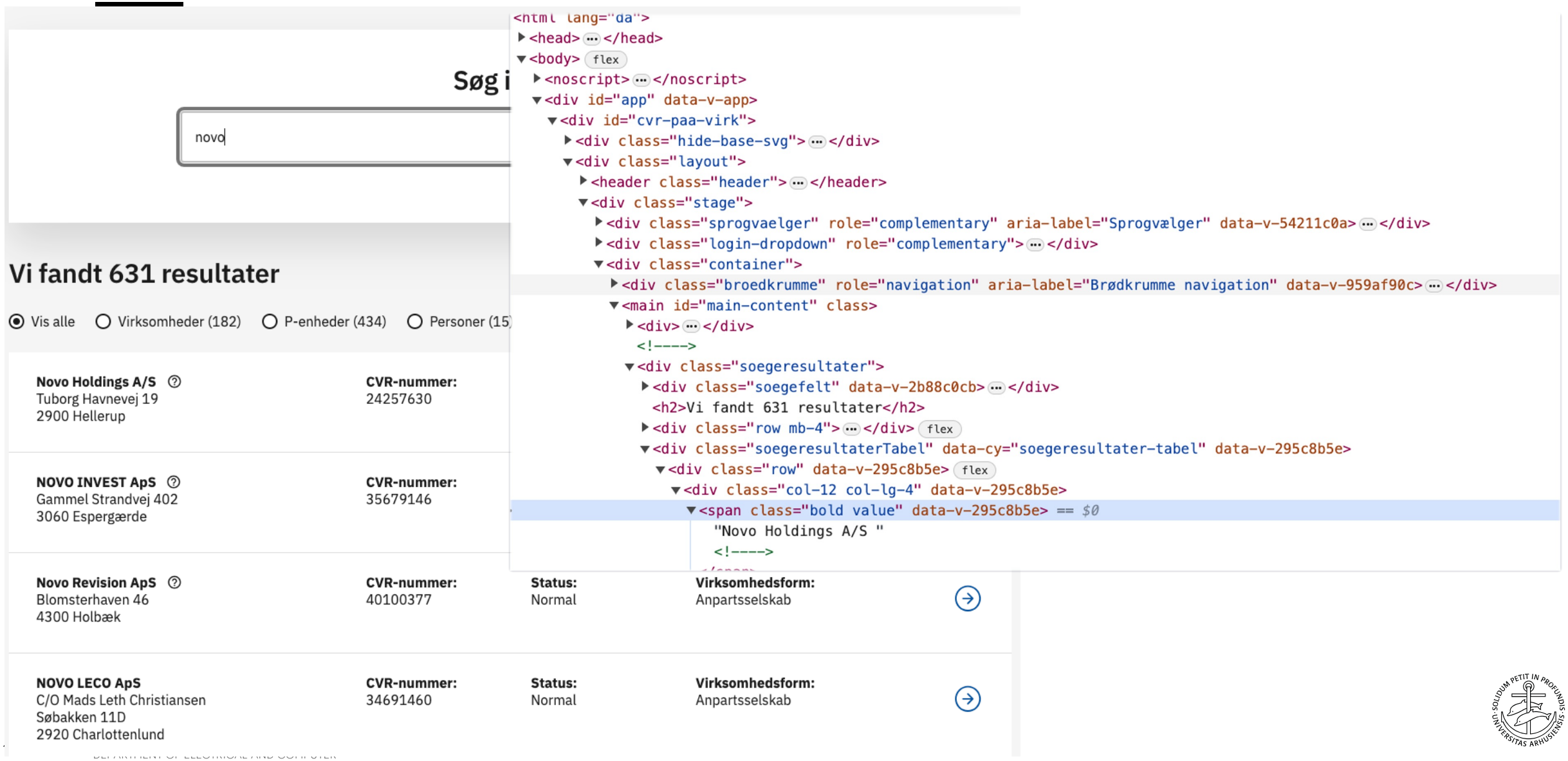
# WHERE DO DATA COME FROM

Any one system can't contain all the needed data

So how does an application get data from *another* application?



# WEBPAGES AND HTML



# JSON OR XML

Data format better suited for applications

## JSON

- E.g. https://cvrapi.dk/api?search=novo&country=dk
- JavaScript Object Notation
- Lightweight
- Simple to parse with https://www.newtonsoft.com/json

## XML

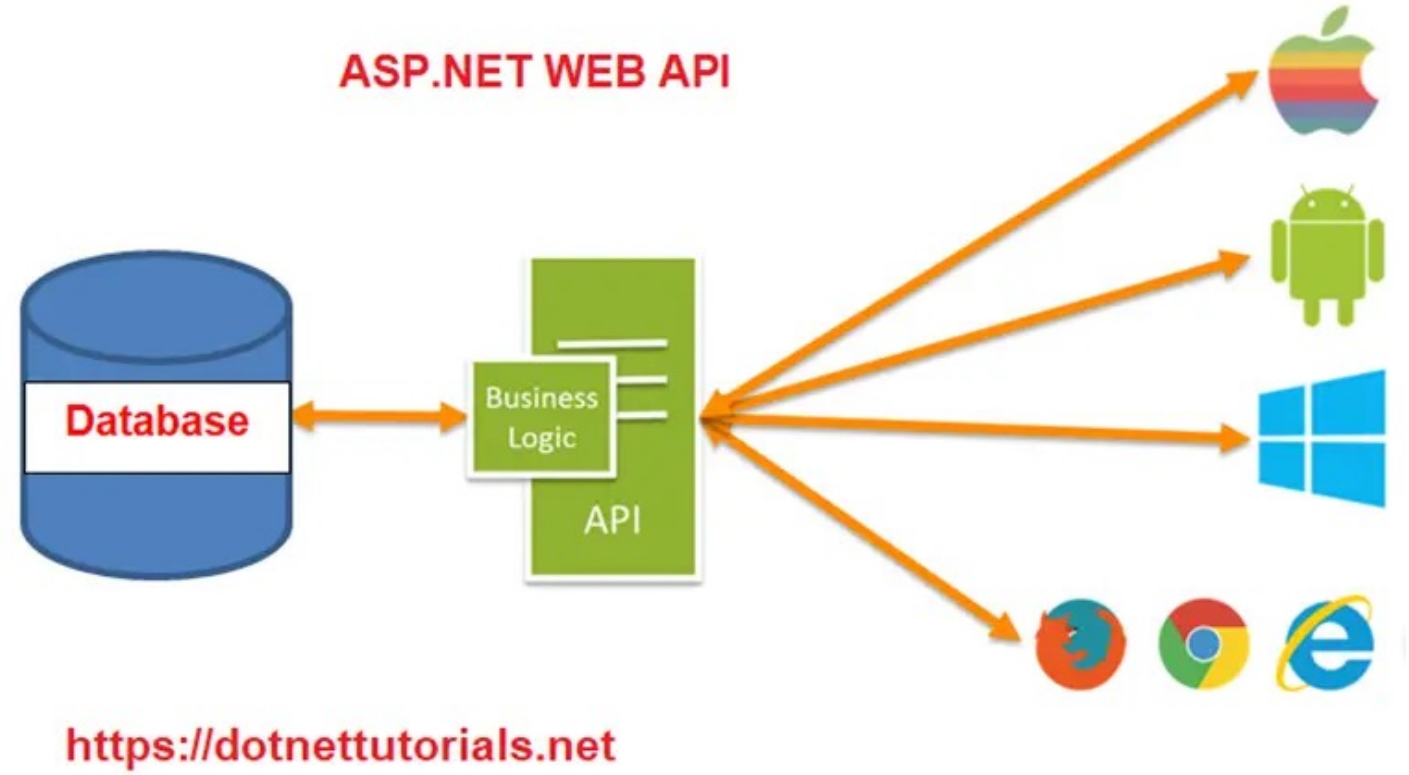
- E.g. https://cvrapi.dk/api?search=novo&country=dk&format=xml
- Extensible Markup language
- Verbose
- Parse with https://learn.microsoft.com/en-us/dotnet/api/system.xml.xmldocument?view=net-6.0

# URL

Uniform resource locator

# ASP.NET WEB API

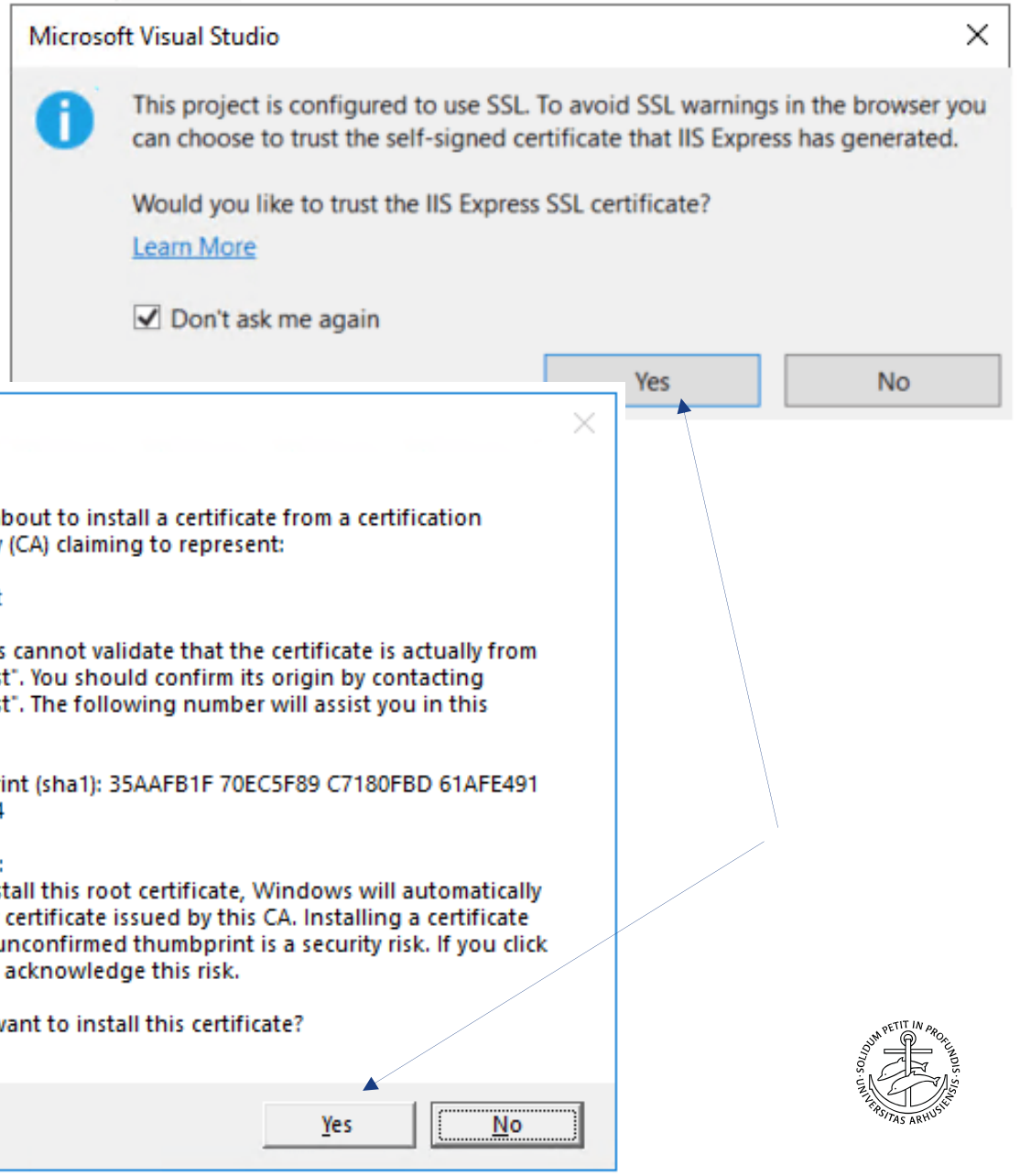
- This is Microsoft's recommended way of creating an API
  - Application Programming Interface



# CREATING A PROJECT

In Visual Studio

1. In the *File* menu choose *New* -> *Project*
2. Search for Web API and select '**ASP.NET Core Web API**'
3. In the Additionals information menu
    1. Make sure you uses .NET 6.0 or higher
    2. Make sure you *Use Controllers* are selected
4. Run the project.



# GENERATED CODE

WeatherForecast.cs

Controllers/

    WeatherForecastController.cs

Program.cs

# GENERATED CODE - MODEL

WeatherForecast.cs

Controllers/

    WeatherForecastController.cs

Program.cs

```
public class WeatherForecast
{
    public DateOnly Date { get; set; }
    public int TemperatureC { get; set; }
    public int TemperatureF => 32 +
(int)(TemperatureC / 0.5556);
    public string? Summary { get; set; }
}
```

# GENERATED CODE

WeatherForecast.cs

Controllers/

    WeatherForecastController.cs

Program.cs

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController :
ControllerBase {
  [HttpGet(Name = "GetWeatherForecast")]
  public IEnumerable<WeatherForecast> Get() {
    return Enumerable.Range(1, 5).Select(index => new
                                    WeatherForecast{
      Date = DateOnly.FromDateTime(
                          DateTime.Now.AddDays(index)),
      TemperatureC = Random.Shared.Next(-20, 55),
      Summary = Summaries[
                  Random.Shared.Next(Summaries.Length)]
    })
   .ToArray();
  }
}
```

# GENERATED CODE

WeatherForecast.cs

Controllers/

    WeatherForecastController.cs

Program.cs

```
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment()) {
  app.UseSwagger();
  app.UseSwaggerUI();
}
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

# EXERCISES

Solve **1** and **2**

Jump to Bonus when done

# REQUESTS

- HTTP Verbs
- **GET**
- POST
- PUT
- PATCH
- DELETE

```
[HttpGet("{id}")]
public async Task<ActionResult<TodoItem>> GetTodoItem(long
id) {
    // GET todoItem
    // TodoItem todoItem = …
    if (todoItem == null) {
       return NotFound();
    }
    return todoItem;
}
```

Used to return resources from a web server API (one or many)

# TASK + ASYNC/AWAIT

You will see this in much greater details in SW4SWD

- `Task` is a lightwait thread.
  - "All" methods that is contains 'async' in name returns a task.
- `Task<T>` can contain data.

- `await` unwraps (among other things) inner data
  - `string stringData = await httpClient.GetStringAsync(URL);`
- If you uses await in a method body, the body needs to be decorated with `async` and return a `Task`

# REQUESTS

- HTTP Verbs
- GET
- **POST**
- PUT
- PATCH
- DELETE

```
[HttpPost]
public async Task<ActionResult<TodoItem>>
PostTodoItem(TodoItem todoItem)
{
    return Created(“todoitem”, todoItem);

    //return CreatedAtAction(nameof(GetTodoItem), new { id =
todoItem.Id }, todoItem);
}
```

Used to create new resources on the web server

# REQUESTS

- HTTP Verbs
- GET
- POST
- **PUT**
- PATCH
- DELETE

```
[HttpPut("{id}")]
public async Task<IActionResult> PutTodoItem(long id,
TodoItem todoItem)
{
    if (id != todoItem.Id) {
       return BadRequest();
    }
    return NoContent();
}
```

Used to update existing resources on the server

# REQUESTS

- HTTP Verbs
- GET
- POST
- PUT
- PATCH
- **DELETE**

```
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteTodoItem(long id)
{
    var todoItem = null; // Find resources somewhere
    if (todoItem == null) {
       return NotFound();
    }
    return NoContent();
}
```

Used to delete resources from the

# REQUESTS

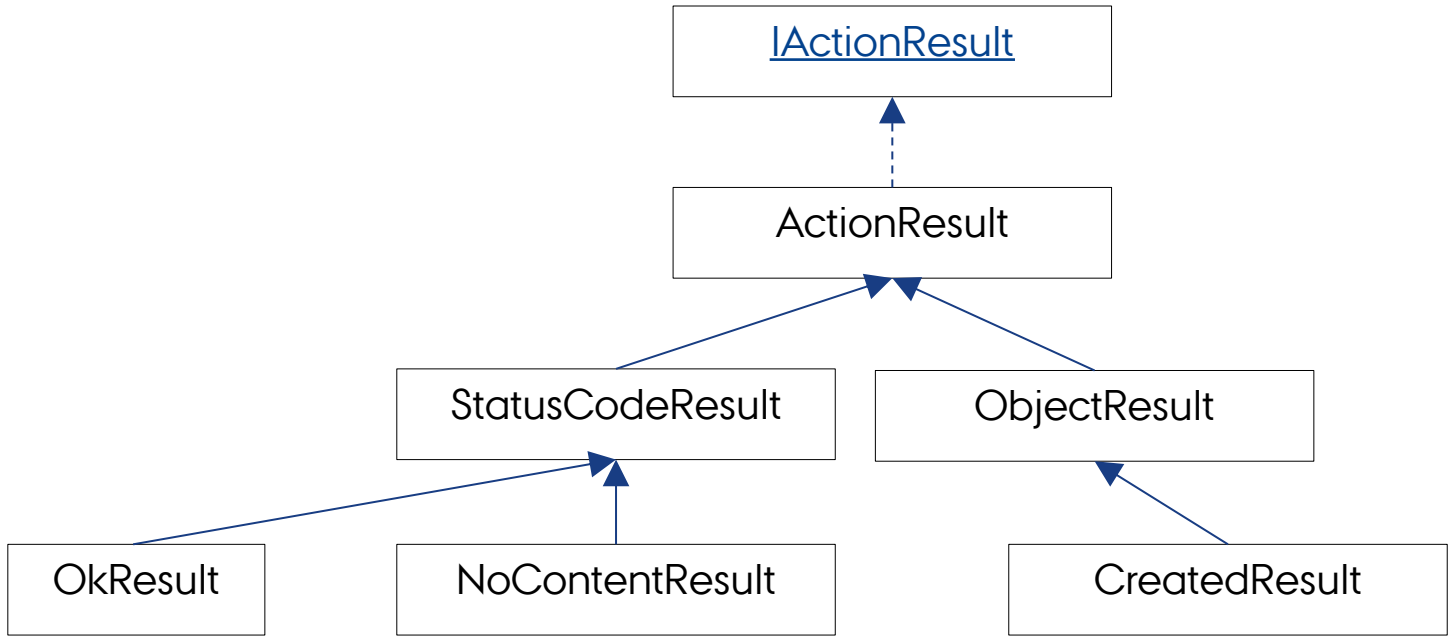
- HTTP Verbs
- GET
- POST
- PUT
- **PATCH**
- DELETE

Used to update resources from partial documents

# HTTP RESPONSE CODES

1. Informational responses (100 – 199)
2. Successful responses (200 – 299)
    1. 200 OK
    2. 201 CREATED
    3. 204 NO CONTENT
3. Redirection messages (300 – 399)
    1. NOT MODIFIED
4. Client error responses (400 – 499)
    1. 403 FORBIDDEN
    2. 404 NOT FOUND
5. Server error responses (500 – 599)
    1. 500 INTERNAL SERVER ERROR

# IACTIONRESULT




Helper methods for creating Results

Ok(), NoContent(), etc...

# EXERCISES

Continue on **1** and **2**

Solve **3->**

Jump to Bonus when done

AARHUS
UNIVERSITY