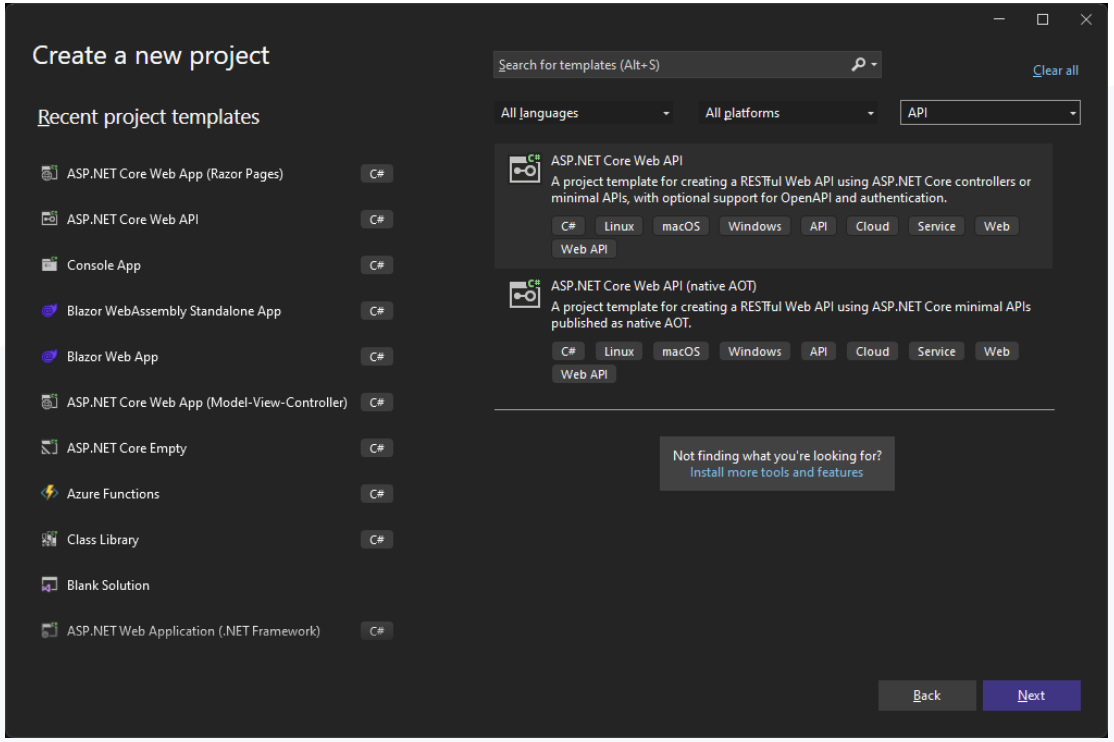
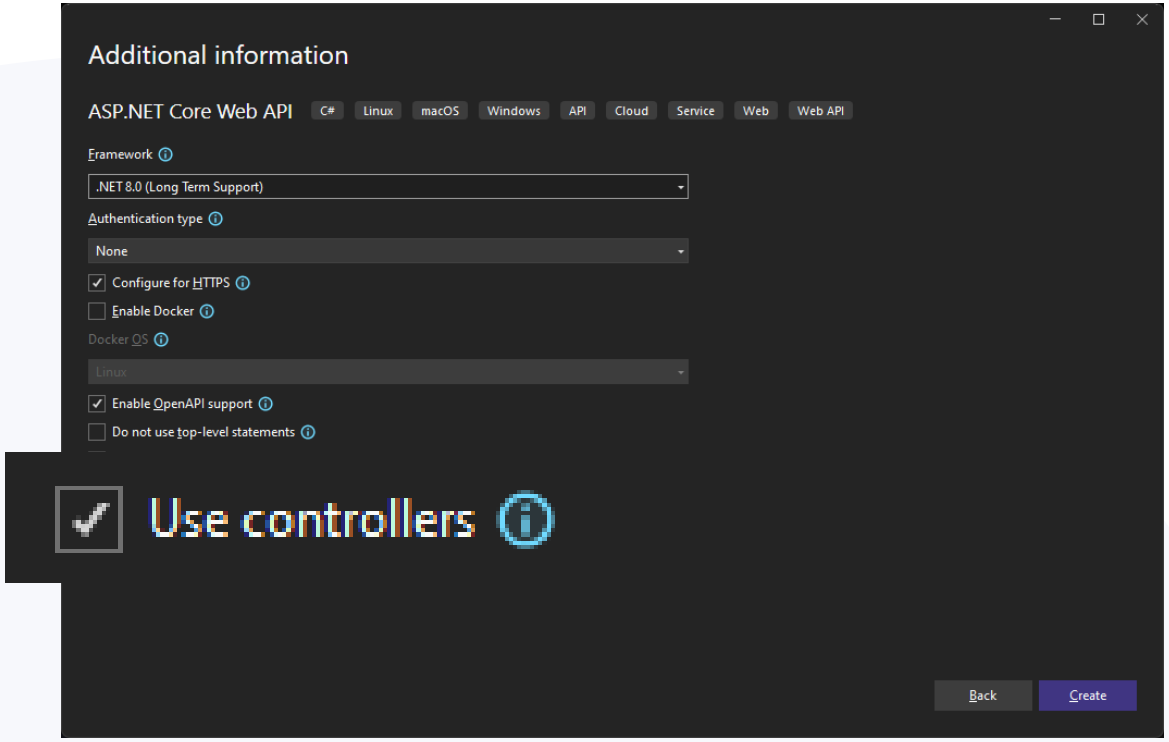
Xebia

Core principles

ASP.NET Core WebAPI

File → New Project...





File → New Project…

Program.cs

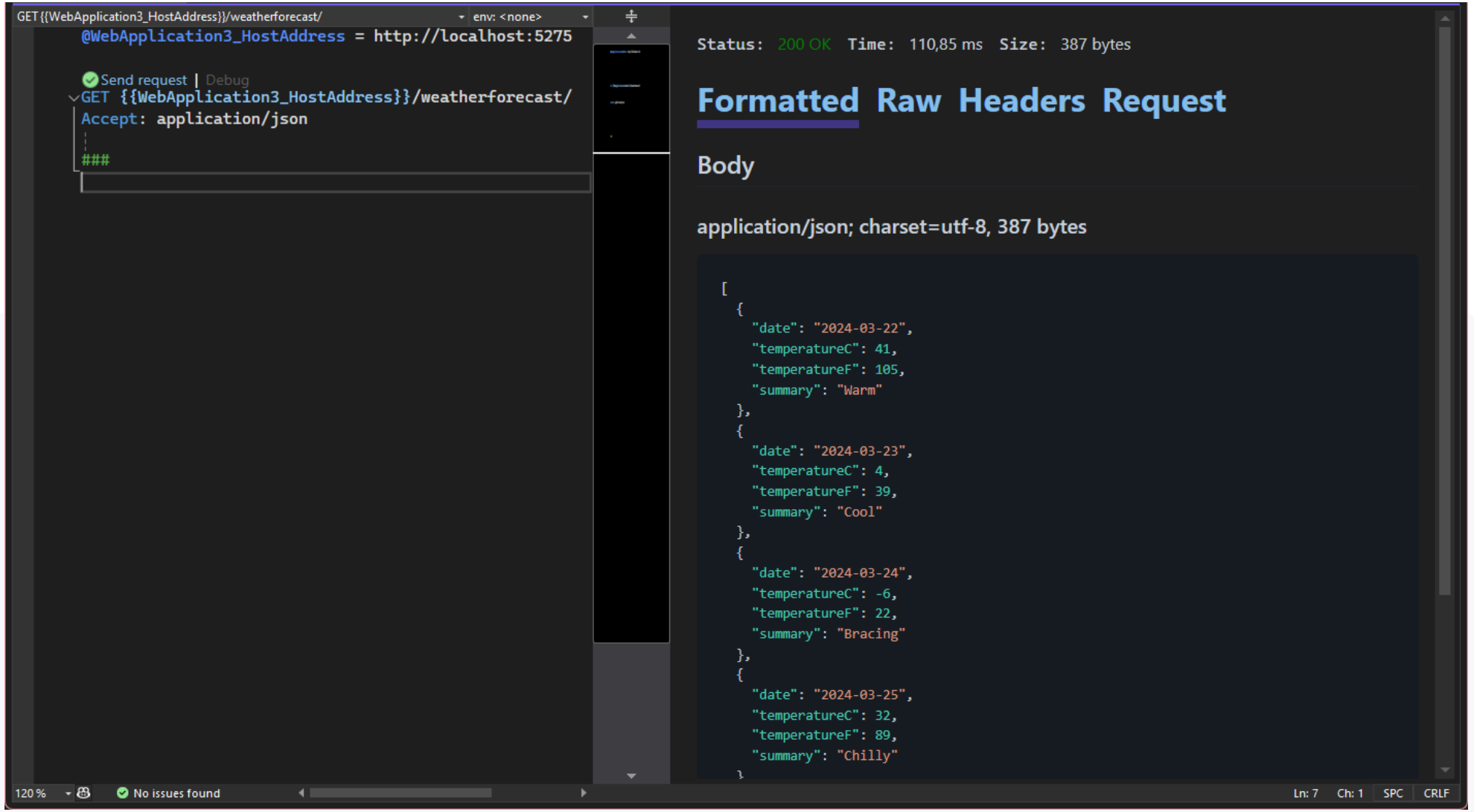
```
var builder = WebApplication.CreateBuilder(args);

// 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();

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();
```

Controllers/WeatherForecastController

```
[ApiController]
[Route("[controller]")] 👈
public class WeatherForecastController : ControllerBase 👈
{
    [HttpGet(Name = "GetWeatherForecast")] 👈
    public IEnumerable<WeatherForecast> Get()
    {
        // return WeatherForecast
    }
}
```

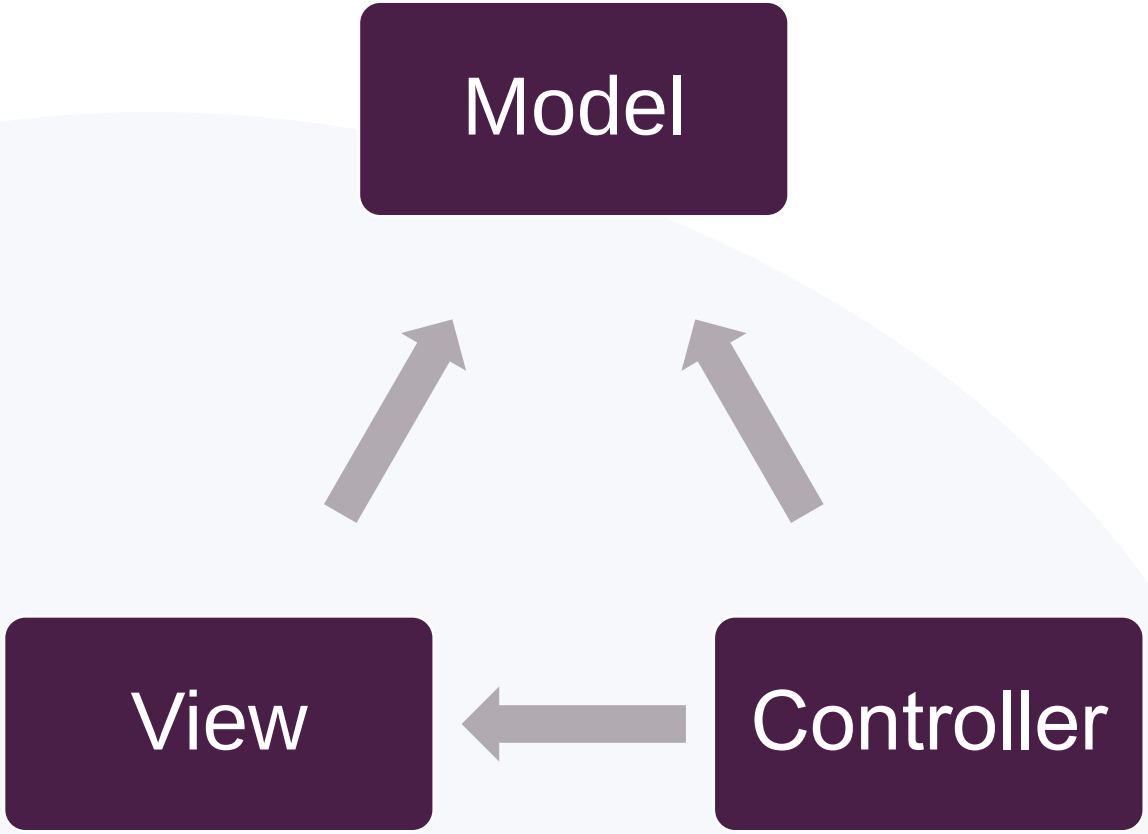
File → New Project…

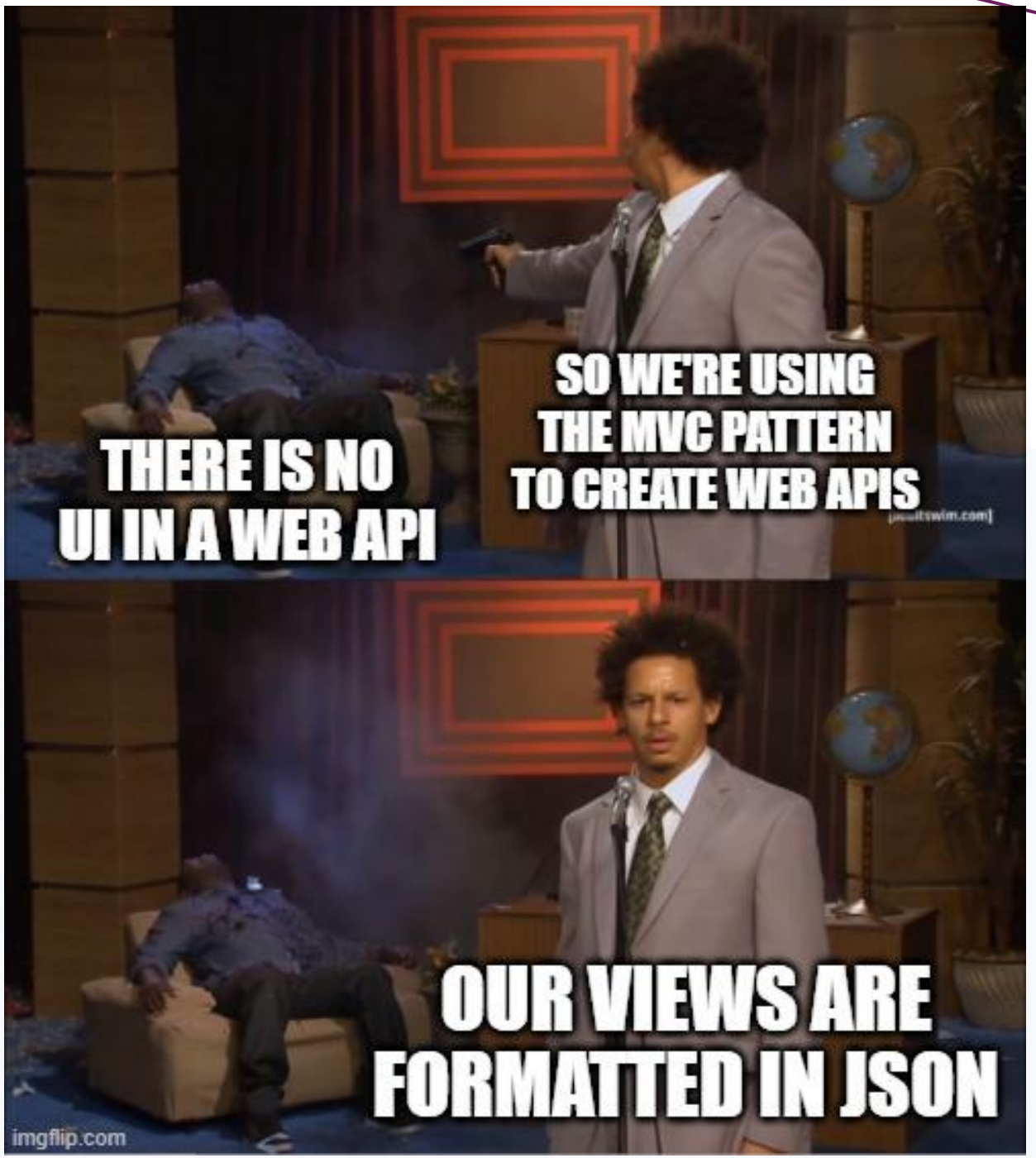
WebApplication.http



The MVC pattern

- MVC: Model-View-Controller
  - Model: data and business logic
  - View: presentation of the data to the end user
  - Controller: interaction between view and model
- Language agnostic **UI** pattern, that tackles:
  - Separation of concerns
  - Testability
  - Reuse
- 2009: release of ASP.NET MVC 1.0
  as an alternative to ASP.NET WebForms
  to build **Server Side Web Applications**
- 2012: release of ASP.NET Web API 1.0
  as an alternative to WCF
  to build **REST APIs**




THERE IS NO
UI IN A WEB API
SO WE'RE USING
THE MVC PATTERN
TO CREATE WEB APIS
OUR VIEWS ARE
FORMATTED IN JSON
imgflip.com

Lab
Setting up the endpoint

1. ASP.NET Core Empty template
2. Add Controller support
3. Create the Controller
4. Create the Model
5. Use the Model in our Controller
6. Testing our API with a .http file

Lab Review

- Web Application Builder
- Middleware pipeline
- Model
- ControllerBase
- APIController attribute
- Route attribute
- HttpGet

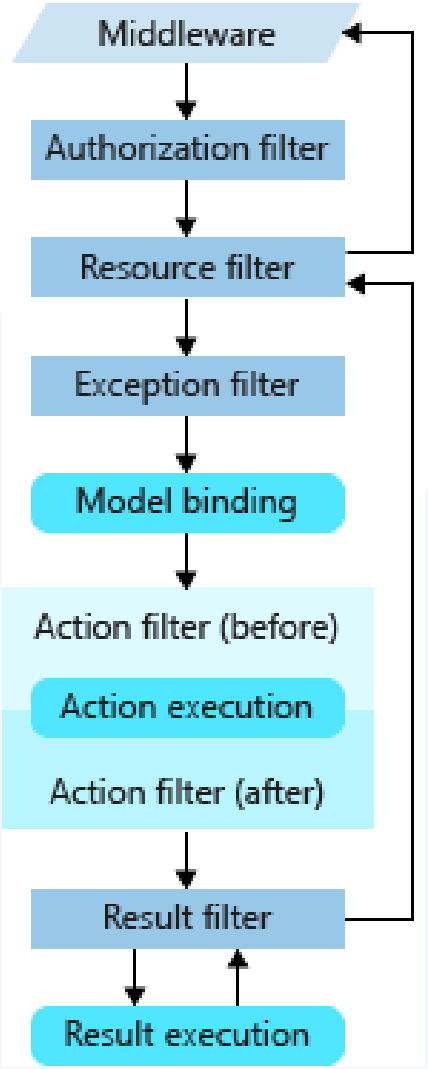
Hosting our app

During startup, a **Host** is built by ASP.NET Core, encapsulating:

- The HTTP server
- Middleware Components
- Logging
- Dependency Injection
- Configuration

Different host options:

- ASP.NET Core WebApplication 👈
- .NET Generic Host (+ ConfigureWebHostDefaults)
- ASP.NET Core WebHost



Hosting our app

ASP.NET Core WebApplication (aka Minimal Host):

- Recommended for Web APIs in ASP.NET Core
- Builds on .NET Generic host
- Default options:
  - Uses Kestrel as the web server (and enable IIS)
  - Loads Configuration (appsettings.json, environment variables, command line args, …)
  - Sends logging to console and debug providers

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

// Add services to the container.
builder.Services.AddControllers();

var app = builder.Build();

// Configure the HTTP request pipeline.
app.MapControllers();

app.Run();
```

Exposing an Endpoint

- Route attribute
  - Controller level: Base route for our endpoint
  - Method level: Sub part of the route for the action
  - Route(“[controller]”): convention-based routing
- HttpGet attribute
  - Tells the endpoint middleware that the action supports HTTP Get
  - Convention-based routing or Route attribute on action
- ControllerBase
  - Base class for MVC controllers without view support
  - View support needed? Use Controller as a base class
- APIController attribute
  - Indicates that the class will serve HTTP API responses
  - Enables opioniotated API behaviors
  - Goal: improve developer experience

```
[Route("pies")]
public class PiesController
{
    [HttpGet]
    public JsonResult GetPies()
    {
        return new JsonResult (
          new List<Pie>
        {
            // pies
        });
    }
}
```

```
[ApiController]
[Route("api/[controller]")]
public class PiesController: ControllerBase
{
    [HttpGet]
    public JsonResult GetPies()
    {
        return new JsonResult (
          new List<Pie>
        {
            // pies
        });
    }
}
```

Lab
Adding more methods

1. Add an inline data store for our pies
2. Add the GetById method
3. Add the SearchPie method

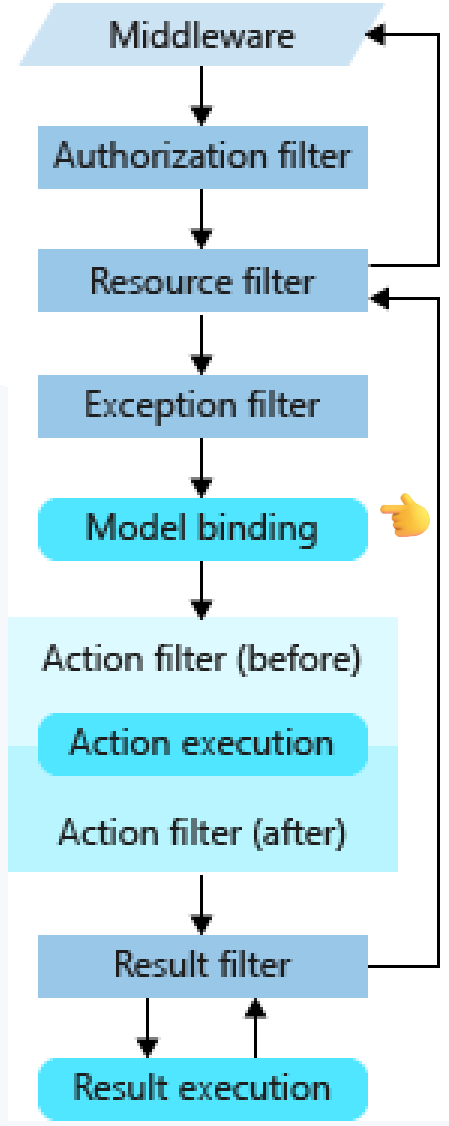
Lab Review

- Route parameter (int id)
- Route constraints
- Query string parameter
- Http status codes

Model binding

Controllers work with data from the HTTP requests:

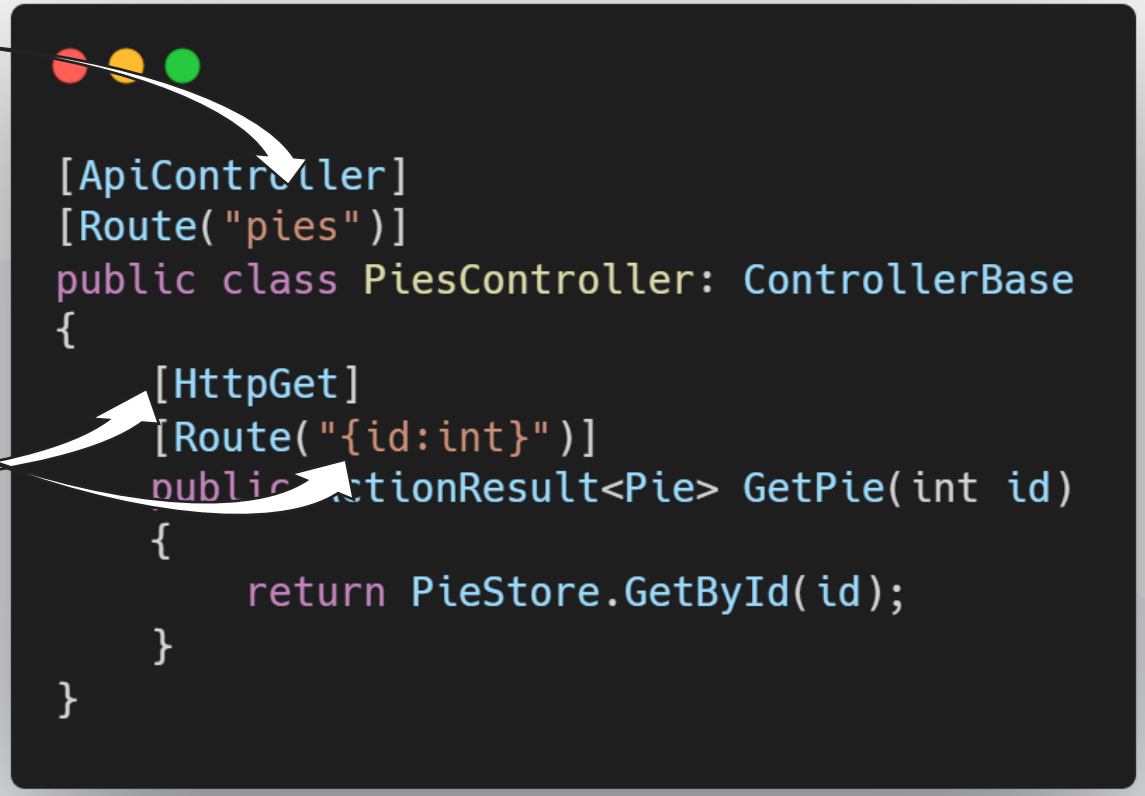
- Route data 👈
- Query string 👈
- Request body
- Form fields
- Headers
- Uploaded files



Model Binding
Route data

🌐 GET https://api.pieshop.com/pies/1

Method Domain Route



❗ Only simple types are supported
➔ types that can be converted to string

Model Binding
Route data

🌐 GET https://api.pieshop.com/pies/1

🛑 Route constraints:

- Checks the route parameter before binding
  - Type constraints (e.g. {id:int}, {price:decimal})
  - Logical constraint (e.g. {age:min(18)}, {name}
- Returns a HTTP-404 – Not Found response
- Don’t use for validation (HTTP-400 Bad Request)
- Full list of constraints

```
[ApiController]
[Route("pies")]
public class PiesController: ControllerBase
{
    [HttpGet]
    [Route("{id:int}")]
    public ActionResult<Pie> GetPie(int id)
    {
        return PieStore.GetById(id);
    }
}
```

Model Binding
Query string

🌐 GET https://api.pieshop.com/pies/search?name=cherry

Method | Domain | Route | Query String

- ?: precedes the query string
- Key/Value pairs
  - Key: name
  - Value: cherry
- &: multiple key/value pairs
  - name=cherry&page=1&skip=10
- %XX: special characters are encoded
  - %20: space ➔ name=cherry%20pie

```
[ApiController]
[Route("pies")]
public class PiesController: ControllerBase
{
    [HttpGet]
    [Route("search")]
    public ActionResult<Pie> SearchPie(string name)
    {
        return PieStore.GetByPartialName(name);
    }
}
```

HTTP Status codes

2xx
Success

- **200** - OK
- **201** - Created
- **204** - No Content

4xx
Client Error

- **400** - Bad Request
- **401** - Unauthorized
- **403** - Forbidden
- **404** - Not Found
- **405** - Method Not Allowed

5xx
Server Error

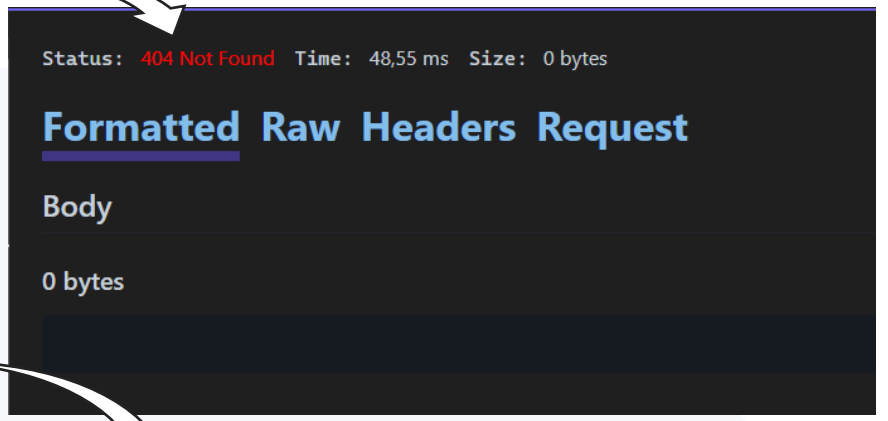
- **500** - Internal Server Error

HTTP Status codes

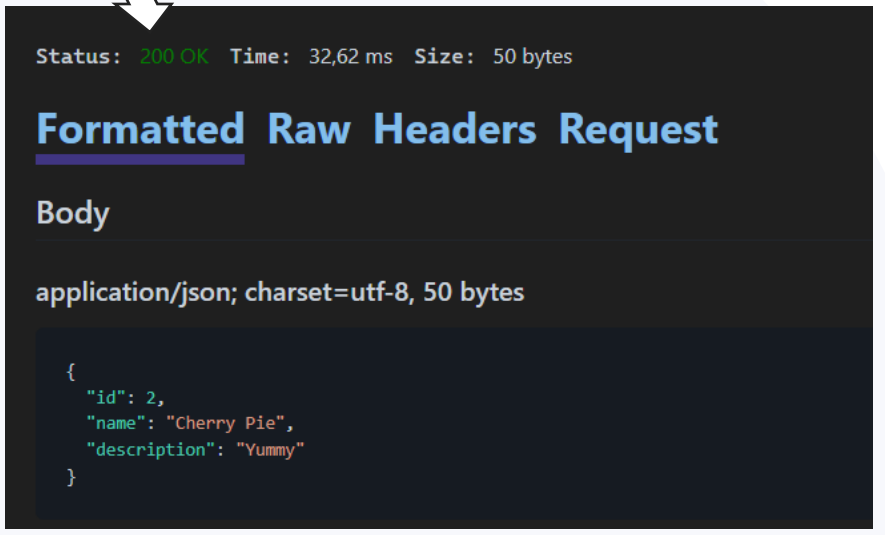
Globally accepted standard: RFC 7231

➔Applying the right status code helps the consumers of your API

🌐 GET https://api.pieshop.com/pies/cherry ❌



🌐 GET https://api.pieshop.com/pies/search/name=cherry ✔



Lab
Connecting the database

1. Add Entity Framework Core support
2. Add a Repository class
3. Use the Repository in the Controller
4. Implement (simple) paging
5. Handling Exceptions
6. Return the correct HTTP status codes
7. Adding Problem Details

Lab Review

- Service Registration
- Dependency Injection
- Deferred execution
- ObjectResults
- ProblemDetails
- Developer Exceptions

Service Registration & Dependency Injection

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

// Add services to the container.
builder.Services.AddDbContext<PieShopDbContext>(options =>
    options.UseInMemoryDatabase("PieShopDb"));

builder.Services.AddScoped(typeof(IAsyncRepository<>),
typeof(RepositoryBase<>));
builder.Services.AddScoped<IPieRepository, PieRepository>();

builder.Services.AddControllers();

var app = builder.Build();

// Configure the HTTP request pipeline.
app.MapControllers();

app.Run();
```

```
[ApiController]
[Route("pies")]
public class PiesController : ControllerBase
{
    private readonly IPieRepository _pieRepository;

    public PiesController(IPieRepository pieRepository)
    {
        _pieRepository = pieRepository;
    }

    [HttpGet]
    public async Task<ActionResult<IEnumerable<Pie>>> GetPies(int? size, int?
page)
    {
        return Ok(await _pieRepository.ListAllAsync());
    }

    [HttpGet]
    [Route("{id:int}")]
    public async Task<ActionResult<Pie>> GetPie(int id)
    {
        var pie = await _pieRepository.GetByIdAsync(id);

        return Ok(pie);
    }
}
```

Service Registration & Dependency Injection

Lifetimes

- Transient
  - Services are created each time they are requested
  - Use for lightweight and stateless services (e.g. string encoder)
- Scoped
  - Services are created once per client request
  - Use when state needs to be maintained within a request , but not across requests (e.g. EF DbContext)
- Singleton
  - Services are created on the first request and reused in subsequent request
  - Use for stateful services that can be reused across requests (e.g. memory cache)
  - ❗ Don’t resolve a singleton from scoped or transient

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

builder.Services.AddTransient<ITransientService, TransientService>
();
```

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

builder.Services.AddScoped<IScopedService, ScopedService>
();
```

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

builder.Services.AddSingleton<ISingletonService, SingletonService>
();
```

Service Registration & Dependency Injection

Keyed Services

✨ New in .NET 8

Multiple registrations of same type

- All types are in the container
- Last registered implementation is used
- *(Useful for testing)*

Keyed services

- All types are in the container
- Reference by key which implementation to use

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

builder.Services.AddScoped<IScopedService, ScopedService1>();❌
builder.Services.AddScoped<IScopedService, ScopedService2>();👈
```

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

services.AddKeyedSingleton<IMessageWriter, MemoryMessageWriter>("memory");
services.AddKeyedSingleton<IMessageWriter, QueueMessageWriter>("queue");
```

```
public class PiesController
{
    private readonly IMessageWriter _writer;

    public PiesController(
        [FromKeyedServices("queue")] IMessageWriter writer)
    {
        _writer = writer;
```

Service Registration & Dependency Injection

Resolution at startup

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

// Add services to the container.
builder.Services.AddDbContext<PieShopDbContext>(options =>
    options.UseInMemoryDatabase("PieShopDb")
           .LogTo(Console.WriteLine));

builder.Services.AddScoped(typeof(IAsyncRepository<>), typeof(RepositoryBase<>));
builder.Services.AddScoped<IPieRepository, PieRepository>();

builder.Services.AddControllers();

var app = builder.Build();

// Configure the HTTP request pipeline.
app.MapControllers();

using (var context = new PieShopDbContext(builder.Services.BuildServiceProvider()
    .GetRequiredService<DbContextOptions<PieShopDbContext>>()))
{
    context.Database.EnsureDeleted();
    context.Database.EnsureCreated();
}

app.Run();
```

Database integration

- Entity Framework Core
  - DbContext
  - DbSet
- Repository Pattern
  - Creates consistency over all entities
  - Useful in API scenarios
- Deferred Execution
  - Performance: only get the data you need
  - ToList, ToArray, foreach, …

```
public class PieShopDbContext : DbContext
{
    public PieShopDbContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Pie> Pies { get; set; }
}
```

```
public interface IAsyncRepository<T> where T : class
{
    Task<T?> GetByIdAsync(int id);
    Task<IReadOnlyList<T>> ListAllAsync();
    Task<IReadOnlyList<T>> GetPagedReponseAsync(int page, int size);
    Task<T> AddAsync(T entity);
    Task UpdateAsync(T entity);
    Task DeleteAsync(T entity);
}
```

```
public async virtual Task<IReadOnlyList<T>> GetPagedReponseAsync(int page, int size)
{
    return await _dbContext.Set<T>().Skip((page - 1) * size)
                                    .Take(size)
                                    .AsNoTracking()
                                    .ToListAsync();
}
```

👈

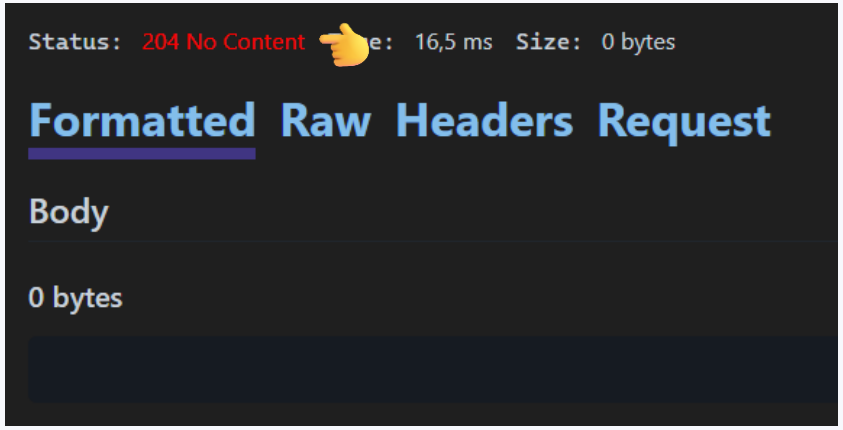
Return Types

Specific types

```
[HttpGet]
[Route("{id:int}")]
public async Task<Pie> GetPie(int id)
{
    var pie = await _pieRepository.GetByIdAsync(id);

    return pie;
}
```

/pies/1

/pies/9000



HTTP Status codes

2xx
Success

- **200** - OK
- **201** - Created
- **204** - No Content

4xx
Client Error

- **400** - Bad Request
- **401** - Unauthorized
- **403** - Forbidden
- **404** - Not Found 👈
- **405** - Method Not Allowed

5xx
Server Error

- **500** - Internal Server Error

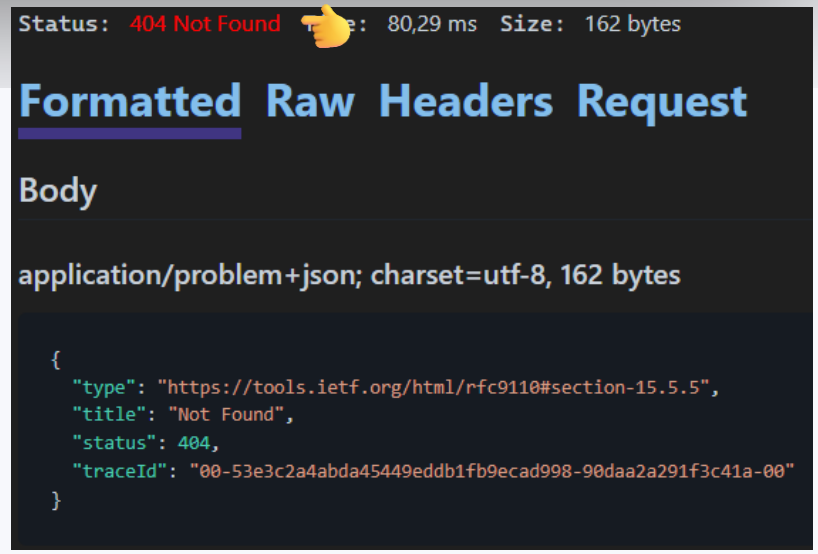
Return Types

ObjectResults

- ControllerBase provides proper HTTP results
  - return Ok(pie) ➔ 200
  - return NotFound() ➔ 404
  - return BadRequest() ➔ 400
  - …
- ActionResult<T>
  - Allows multiple return types (Ok, NotFound, …)
  - Typed results (T = Pie)
  - Untyped ➔ use IActionResult

```
[HttpGet]
[Route("{id:int}")]
public async Task<ActionResult<Pie>> GetPie(int id)
{
    var pie = await _pieRepository.GetByIdAsync(id);

    if (pie == null)
    {
        return NotFound();
    }

    return Ok(pie);
}
```

/pies/9000



Developer Exceptions

Exposing exception information

- 🟢 Non-production environments
  - Not exposed to end-users
  - Local, Dev, Test, Acceptance
  - Exception information helps debugging and bug fixing
- 🔴 Production environment
  - Exposed to end-users
  - Staging, Production
  - Exception information gives inside into how your code works
  - Could be used to find a security weakness and exploit it 🦹
  - Exceptions (including stack trace) can be logged for trouble shooting

Properties/launchsettings.json

```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "profiles": {
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "https://localhost:7165;http://localhost:5116",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development" 👈
      }
    }
  }
}
```

```
[HttpGet]
[Route("filter")]
public async Task<ActionResult<Pie>> FilterPie()
{
    throw new NotImplementedException();
}
```

Production

Status: 500 Internal Server Error Time: 212,29 ms Size: 0 bytes

Formatted Raw Headers Request

Body

0 bytes

Development

Status: 500 Internal Server Error Time: 190,66 ms Size: 2113 bytes

Formatted Raw Headers Request

Body

text/plain; charset=utf-8, 2113 bytes

```
System.NotImplementedException: The method or operation is not implemented.
   at WebApplication5.Controllers.PiesController.FilterPie() in C:\code\WebApplication5\WebAp
   at lambda_method51(Closure, Object)
   at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.AwaitableObjectResultExecu
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeActionMethodAsyn
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeNextActionFilter
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Rethrow(ActionExecutedC
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeInnerFilterAsync(
--- End of stack trace from previous location ---
   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeFilterPipelineAsync>g__A
```

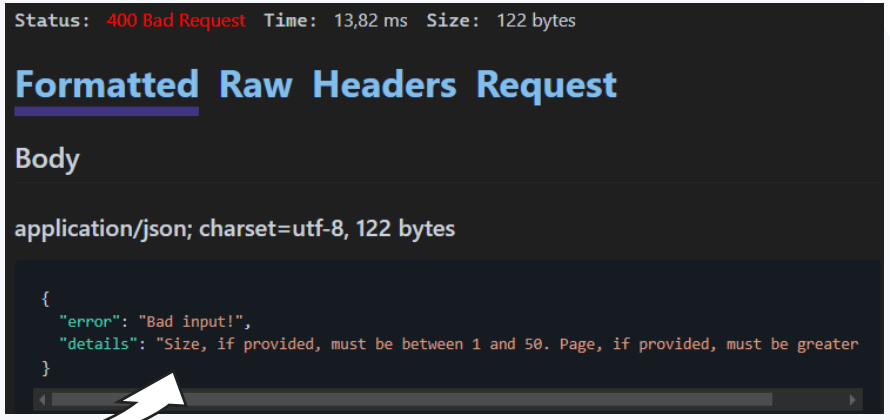
Give Errors extra context

Overloads on ObjectResults

➔ Pass extra info

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Pie>>> GetPies(int? size, int? page)
{
   if ((size.HasValue && (size.Value <= 0 || size.Value > 50)) || (page.HasValue && page.Value <= 0))
   {
       return BadRequest();
   }

   //...
}
```

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Pie>>> GetPies(int? size, int? page)
{
   if ((size.HasValue && (size.Value <= 0 || size.Value > 50)) || (page.HasValue && page.Value <= 0))
   {
      return BadRequest(new {
          error="Bad input!",
          details = "Size, if provided, must be between 1 and 50. Page, if provided, must be greater
than 0."
      });
   }

   //...
}
```



Problem Details

- Standard: RFC 9457
- Machine readable details of errors in HTTP Responses
  ➔ standardized response body
- Allows for addition of extra info

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

// Add services to the container.
builder.Services.AddControllers();

builder.Services.AddProblemDetails(
    options =>
        options.CustomizeProblemDetails = ctx =>
        {
            ctx.ProblemDetails.Extensions.Add("version",
Assembly.GetExecutingAssembly().GetName().Version);
        }
);

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler();
}

// Configure the HTTP request pipeline.
app.MapControllers();

app.Run();
```

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Pie>>> GetPies(int? size, int? page)
{
    if ((size.HasValue && (size.Value <= 0 || size.Value > 50)) || (page.HasValue && page.Value <= 0))
    {
        return Problem(
            title: "Bad Input",
            detail: "Size, if provided, must be between 1 and 50...
            type: "Paging_Error",
            statusCode: StatusCodes.Status400BadRequest
        );

      // ...
    }
```

/pies?page=1&**size=500**

Status: 400 Bad Request **Time:** 174,91 ms **Size:** 243 bytes

Formatted **Raw** **Headers** **Request**

Body

application/problem+json; charset=utf-8, 243 bytes


```
{
  "type": "Paging_Error",
  "title": "Bad Input",
  "status": 400,
  "detail": "Size, if provided, must be between 1 and 50. Page, if provided, must be greater t
  "traceId": "00-9fa148170905335a53ee2b7065aff50a-2fa31b41aa474769-00",
  "version": "1.0.0.0"
}
```

Problem Details & Developer Exceptions

```
[HttpGet]
[Route("filter")]
public async Task<ActionResult<Pie>> FilterPie()
{
    throw new NotImplementedException();
}
```

Development

Status: 500 Internal Server Error Time: 223,56 ms Size: 2594 bytes

Formatted Raw Headers Request

Body

application/problem+json; charset=utf-8, 2594 bytes


```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.6.1",
  "title": "System.NotImplementedException",
  "status": 500,
  "detail": "The method or operation is not implemented.",
  "traceId": "00-9b70f2c2d466edadbf26dee727386ab9-68a57418ed1a49e6-00",
  "version": "1.0.0.0",
  "exception": {
    "details": "System.NotImplementedException: The method or operation is not implemented.
    "headers": {
      "Host": [
        "localhost:7165"
      ],
      "traceparent": [
        "00-9b70f2c2d466edadbf26dee727386ab9-c5bbb17149512d1a-00"
      ]
    },
    "path": "/Pies/Filter",
    "endpoint": "WebApplication5.Controllers.PiesController.FilterPie (WebApplication5)",
    "routeValues": {
      "action": "FilterPie",
      "controller": "Pies"
    }
  }
}
```


Production

Status: 500 Internal Server Error Time: 216,83 ms Size: 221 bytes

Formatted Raw Headers Request

Body

application/problem+json; charset=utf-8, 221 bytes


```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.6.1",
  "title": "An error occurred while processing your request.",
  "status": 500,
  "traceId": "00-04ee69089909c9b25692fa6a43fbf246-1da2aac4a737a63e-00",
  "version": "1.0.0.0"
}
```

Lab
Managing data

1. Adding data
2. Validating the input
3. Updating data
4. Deleting data

Lab Review

- HTTP Verbs
  - POST
  - PUT
  - DELETE
- Request body
- Response header
- Validation

HTTP Verbs

GET	POST	PUT	DELETE
Read a resource	Create a resource	Update a resource	Delete a resource
Idempotent ✓	Idempotent ✗	Idempotent ✓	Idempotent ✓
Safe ✓	Safe ✗	Safe ✗	Safe ✗
Cacheable ✓	Cacheable !	Cacheable ✗	Cacheable ✗

Other HTTP Verbs

- PATCH
  - Support for partial updates
  - HTTP PUT replaces the entire resource
  - Request body contains a collection of patch operations
- HEAD
  - Request the headers that would be returned for a HTTP GET
  - Useful to retrieve metadata, without returning the entire payload
- OPTIONS
  - Request the communication options the API supports
  - E.g. which HTTP verbs are supported

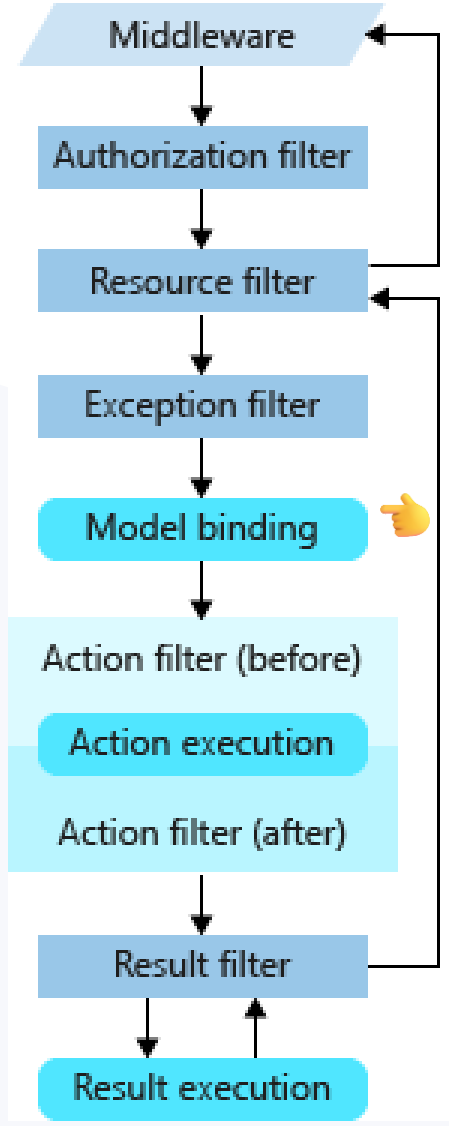
```
[
  { "op": "replace", "path": "/name", "value": "updateName" }
]
```

```
HTTP/1.1 200 OK
Allow: OPTIONS, GET, POST
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: Content-Type, Accept, X-Requested-With
```

Model binding

Controllers work with data from the HTTP requests:

- Route data
- Query string
- Request body 👈
- Form fields
- Headers
- Uploaded files



Adding a new resource

POST https://api.pieshop.com/pies

Method Domain Route

Content-Type: application/json

{

“name” = “Christmas pudding”

“description” = “Festive”

}

Request body

```
[HttpPost]
public async Task<ActionResult<Pie>> CreatePie(Pie pie)
{
    var createdPie = await _pieRepository.AddAsync(pie);

    return CreatedAtAction(nameof(GetPie), new { id = createdPie.Id }, createdPie);
}
```

Status: 201 Created Time: 136,21 ms Size: 68 bytes

Formatted Raw Headers Request

Body

application/json; charset=utf-8, 68 bytes

```
{
  "id": 51,
  "name": "Christmas pudding",
  "description": "New Description"
}
```

Status: 201 Created Time: 136,21 ms Size: 68 bytes

Formatted Raw Headers Request

Response

HTTP/1.1 201 Created (136,21 ms)

Headers

Name	Value
Date	Fri, 22 Mar 2024 16:51:55 GMT
Server	Kestrel
Location	https://localhost:7165/pies/51
Transfer-Encoding	chunked
Content-Type	application/json; charset=utf-8
Content-Length	68

Updating an existing resource

PUT https://api.pieshop.com/pies/51

Method Domain Route

Content-Type: application/json

{

“name” = “Xmas pudding”

“description” = “Festive”

}

Request body

```http
[HttpPut]
[Route("{id:int}")]
public async Task<ActionResult<Pie>> UpdatePie(int id, Pie pie)
{
    var currentPie = await _pieRepository.GetByIdAsync(id);
    if (currentPie == null)
    {
        return NotFound();
    }

    currentPie.Name = pie.Name;
    currentPie.Description = pie.Description;

    await _pieRepository.UpdateAsync(currentPie);

    return NoContent();
}
```

/pies/51

Status: 204 No Content Time: 94,5 ms Size: 0 bytes

Formatted Raw Headers Request

Body

0 bytes

/pies/9000

Status: 404 Not Found Time: 44,78 ms Size: 182 bytes

Formatted Raw Headers Request

Body

application/problem+json; charset=utf-8, 182 bytes

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",
  "title": "Not Found",
```

Deleting an existing resource

🌐 DELETE https://api.pieshop.com/pies/51

Method Domain Route

```http
[HttpDelete]
[Route("{id:int}")]
public async Task<ActionResult> DeletePie(int id)
{
    var pie = await _pieRepository.GetByIdAsync(id);
    if (pie == null)
    {
        return NotFound();
    }

    await _pieRepository.DeleteAsync(pie);

    return NoContent();
}
```

/pies/51

Status: 204 No Content Time: 40,3 ms Size: 0 bytes

Formatted Raw Headers Request

Body

0 bytes

/pies/9000

Status: 404 Not Found Time: 33,15 ms Size: 182 bytes

Formatted Raw Headers Request

Body

application/problem+json; charset=utf-8, 182 bytes


```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",
```

Validating input

Data Annotations

- Validation used by ASP.NET Core during model binding
- Configuration used by Entity Framework Core during migration

```
using System.ComponentModel.DataAnnotations;

public class Pie
{
    public int Id { get; set; }

    [Required]
    [Length(3, 50)]
    public required string Name { get; set; }

    [MaxLength(500)]
    public required string Description { get; set; }
}
```

ModelState

- Implicitly handled by [ApiController] attribute
- Explicitly

```
[HttpPost]
public async Task<ActionResult<Pie>> CreatePie(Pie pie)
{
    if(!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var createdPie = await _pieRepository.AddAsync(pie);

    return CreatedAtAction(nameof(GetPie), new { id = createdPie.Id }, createdPie);
}
```

Validating input

🌐 POST https://api.pieshop.com/pies
Content-Type: application/json
{

“name” = null
“description” = “Festive”

}

❗ Validation Error
➔ validation on the model fails

Status: 400 Bad Request 👈 Time: 33,72 ms Size: 262 bytes

Formatted Raw Headers Request

Body

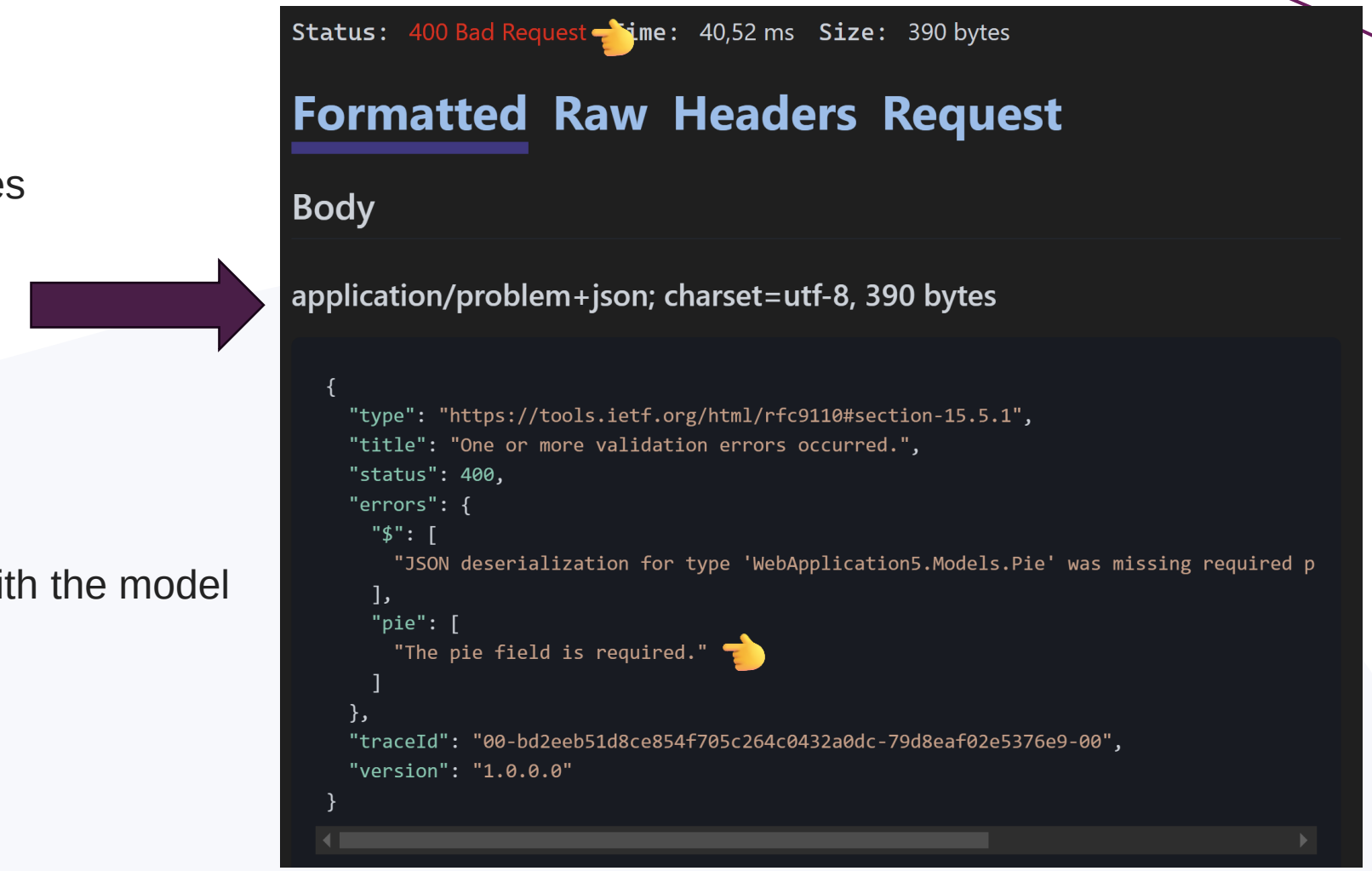
application/problem+json; charset=utf-8, 262 bytes


```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "errors": {
    "Name": [
      "The Name field is required." 👈
    ]
  },
  "traceId": "00-4cb38b4cf10ac6fc1c35f5a02afe0f8d-30bf5da2ea155ff2-00",
  "version": "1.0.0.0"
}
```

Validating input

🌐 POST https://api.pieshop.com/pies
Content-Type: application/json
{
“description” = “Festive”
}

❌ Model binding error:
➔unable to match the payload with the model



Lab
Working with files

1. Download a file
2. Upload a file

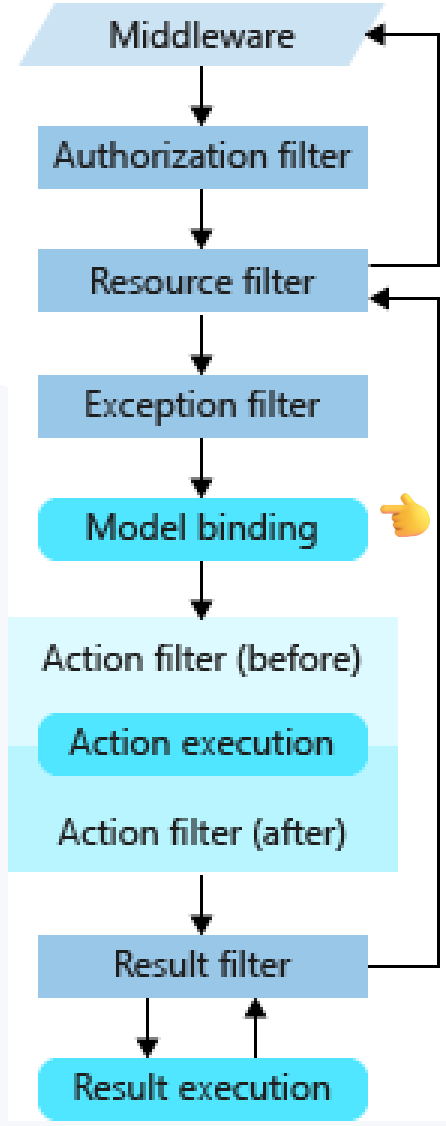
Lab Review

- Content Type
  - application/octet-stream
- FileExtensionContentTypeProvider
- Validating a file

Model binding

Controllers work with data from the HTTP requests:

- Route data
- Query string
- Request body
- Form fields
- Headers
- Uploaded files



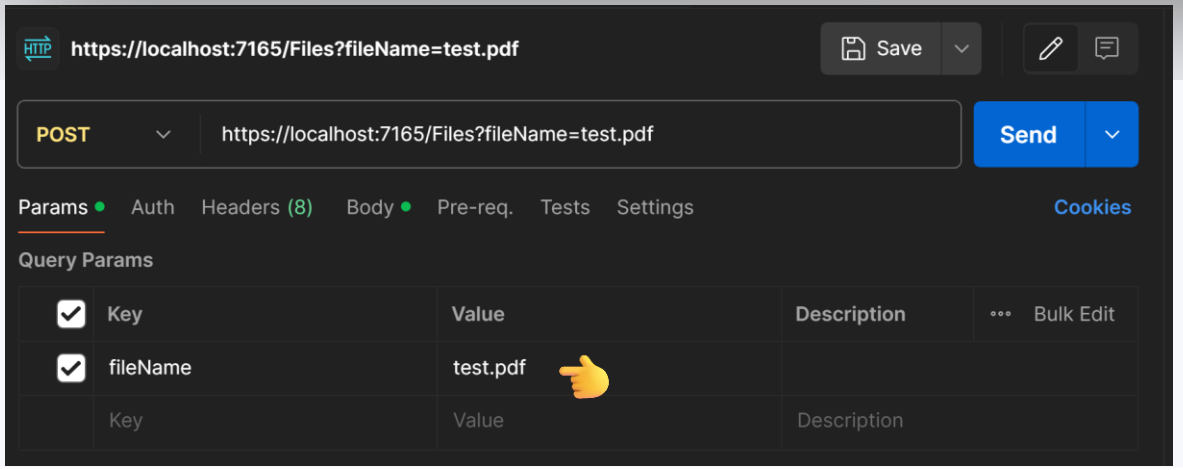
Returning an existing file

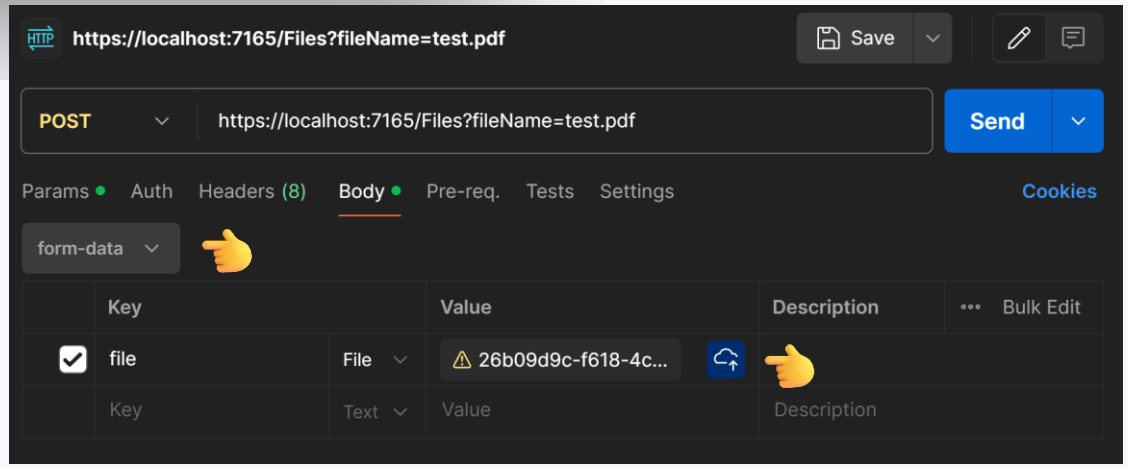
```
[HttpGet]
public async Task<IActionResult> GetFile(string fileName)
{
    var filePath = "Grand tour of Azure API Management.pdf";

    if (string.IsNullOrWhiteSpace(fileName))
    {
        return BadRequest();
    }

    if (!System.IO.File.Exists(filePath))
    {
        return NotFound();
    }

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

    var fileBytes = await
System.IO.File.ReadAllBytesAsync(filePath);

    return File(fileBytes, contentType, fileName);
}
```

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

// Add services to the container.
builder.Services.AddSingleton(new
FileExtensionContentTypeProvider());
builder.Services.AddControllers();

// Configure the HTTP request pipeline.
app.MapControllers();

app.Run();
```

Creating a new file

```
[HttpPost]
public async Task<IActionResult> CreateFile(string fileName, IFormFile file)👈
{
    if(string.IsNullOrWhiteSpace(fileName) || file.Length == 0 || file.Length > 52428800) 👈
        return BadRequest();

    // don't do this in real projects!!!
    var filePath = Path.Combine(Directory.GetCurrentDirectory(), $"{Guid.NewGuid()}-
{fileName}");
    using (var stream = new FileStream(filePath, FileMode.Create))
    {
        await file.CopyToAsync(stream);
    }

    return Created();
}
```





Lab
Connecting a UI

1. Creating a Console Application to interact with our API

Lab Review

- HttpClient
- JSON Serializer

Console App
Getting all pies

```
using var client = new HttpClient();
client.BaseAddress = new Uri("https://localhost:7165");

var response = await client.GetAsync("pies");
response.EnsureSuccessStatusCode();

var pies = await response.Content.ReadAsStringAsync();

// json to object uppercase properties
var piesList = JsonSerializer.Deserialize<List<Pie>>(pies, new JsonSerializerOptions {
PropertyNameCaseInsensitive = true });

foreach (var pie in piesList)
{
    Console.WriteLine($"{pie.Name} - {pie.Description}");
}
```

Console App
Adding a new pie

⚠️ Best practice: use IHttpClientFactory to prevent Socket Exhaustion

```
using var client = new HttpClient(); 👈
client.BaseAddress = new Uri("https://localhost:7165");

var newPie = new Pie { Name = "Apple Pie", Description = "Tasty apple pie" };
var newPieJson = JsonSerializer.Serialize(newPie); 👈
var content = new StringContent(newPieJson, Encoding.UTF8,
"application/json"); 👈
response = await client.PostAsync("pies", content); 👈
response.EnsureSuccessStatusCode();

Console.WriteLine(await response.Content.ReadAsStringAsync());
```

Xebia

xebia.com