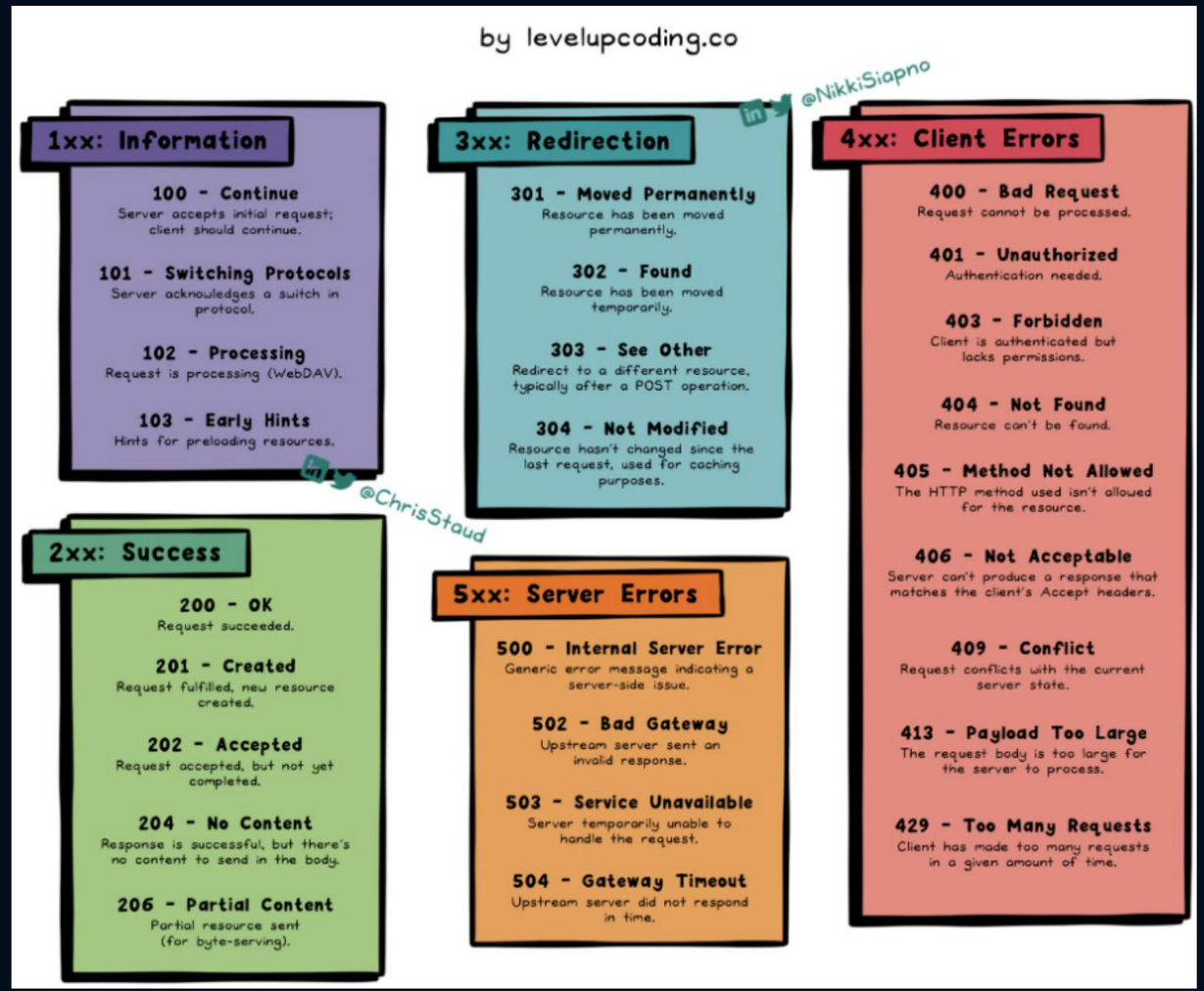
Status codes

HTTP status codes indicate the result of a request. They provide information about whether the request was successful, encountered an error, or needs further action.

Errors.- Consumer passes invalid data to the API, and the API correctly rejects this. Errors are commonly related with level 400 status codes and do not contribute to API availability.

Faults.- API fails to return a response to a valid request. Are commonly related with level 500 status codes and do not contribute to API availability.

Status codes that contribute to API availability are those that indicate a successful response or a temporary redirection. These status codes indicate that the API is operational and can handle incoming requests.

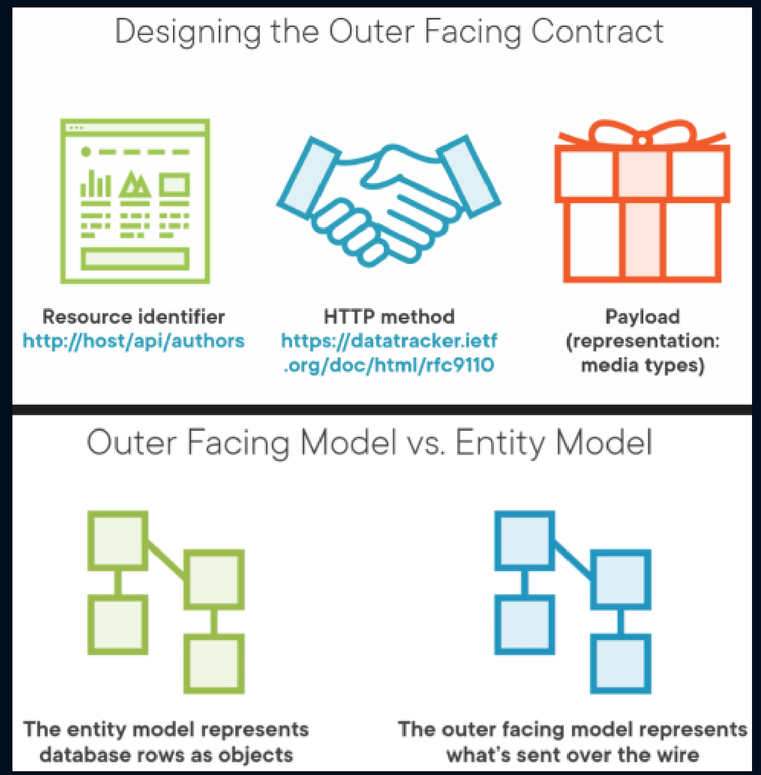


Status Codes

HTTP Method Overview by Use case

Web APIs Models

API Models must be : Well-Named, Well-Factored, Well-Sized and Stable
Resource Names that clients understand, that don't reveal internal architecture, reveal intention.
Don't combine multiple concepts into a single model
Large models with many sub-records are resource intensive for both client and server, provide tools like paging and filtering.
Your most used-resources and models need to be the most stable, make changes in an additive fashion



Outer facing model (AuthorDto)

```
Guid Id
string Name
int Age
```

Entity model (Author)

```
Guid Id
string FirstName
string LastName
DateTimeOffset DateOfBirth
```

Example

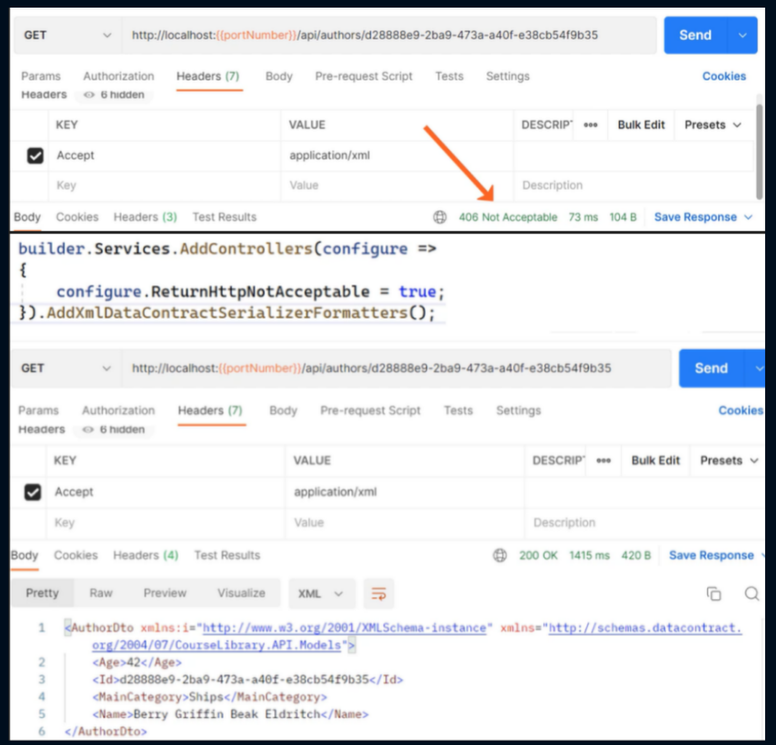
Content negotiation and formatters

Content Negotiation in ASP.NET Core is the process of selecting the appropriate response format (media type) based on the client's preferences and the available formats supported by the server. It enables the server to return responses in various data formats, such as JSON, XML, or custom formats, based on the client's requested format.

```
builder.Services.AddControllers(configure =>
{
    configure.ReturnHttpNotAcceptable = true;
})
    .AddNewtonsoftJson(setupAction =>
    {
        setupAction.SerializerSettings.ContractResolver =
        new CamelCasePropertyNamesContractResolver();
    })
    .AddXmlDataContractSerializerFormatters();


builder.Services.AddControllers(configure =>
{
    configure.ReturnHttpNotAcceptable = true;
})
    .AddXmlDataContractSerializerFormatters()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.PropertyNamingPolicy =
JsonNamingPolicy.CamelCase;
    });
```

Add content negotiation and formatters



Content negotiation and formatters example

Add support for Head

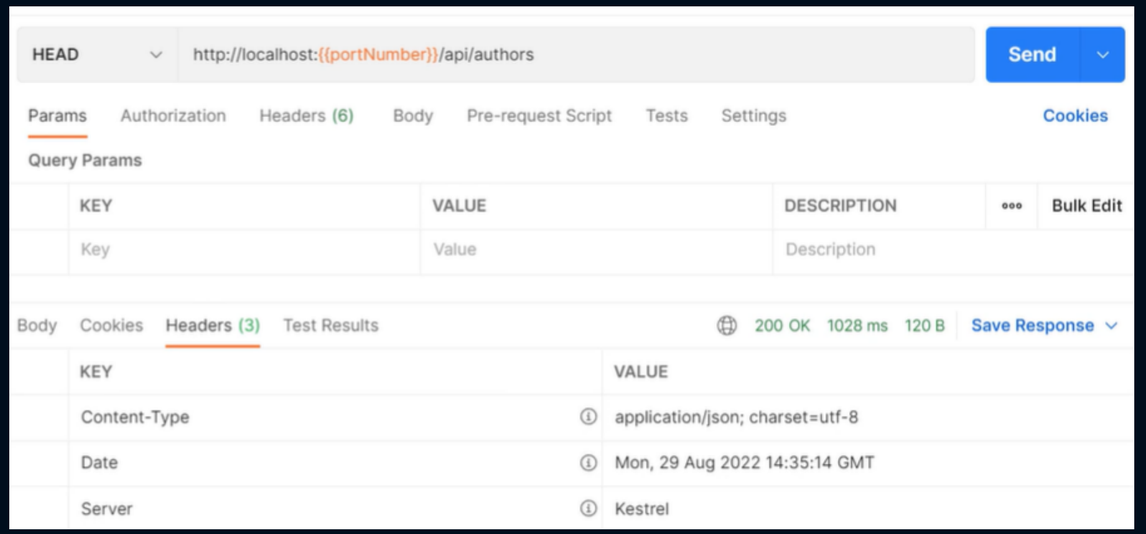
The HEAD request method is similar to the GET request but only requests the response headers and not the actual content. It is useful when you want to retrieve metadata or check the existence of a resource without downloading the entire payload. The server responds with the same headers that would be returned for a corresponding GET request, but the response body is empty.

```
[HttpGet]
[HttpHead]
0 references
public async Task<ActionResult<IEnumerable<AuthorDto>>> GetAuthors()
{
    // throw new Exception("Test exception");

    // get authors from repo
    var authorsFromRepo = await _courseLibraryRepository
        .GetAuthorsAsync();

    // return them
    return Ok(_mapper.Map<IEnumerable<AuthorDto>>(authorsFromRepo));
}
```

Head operation



Head request example

CORS implementation

CORS is an essential security feature that helps protect web applications from unauthorized access while still allowing legitimate cross-origin requests when required.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();

    services.AddCors(options =>
    {
        options.AddPolicy("MySingleOriginCorsPolicy", builder =>
            builder.WithOrigins("http://localhost:5001"));
    });
}

    services.AddCors(options =>
    {
        options.AddPolicy("MySingleOriginCorsPolicy", builder =>
            builder.WithOrigins("http://localhost:5001")
            .WithMethods("GET"));
    });
```

Configure Cors

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseCors("MySingleOriginCorsPolicy");

    //app.UseCors(builder =>
    //{
    //    builder.WithOrigins("http://localhost:5001");
    //});
```

Use Cors

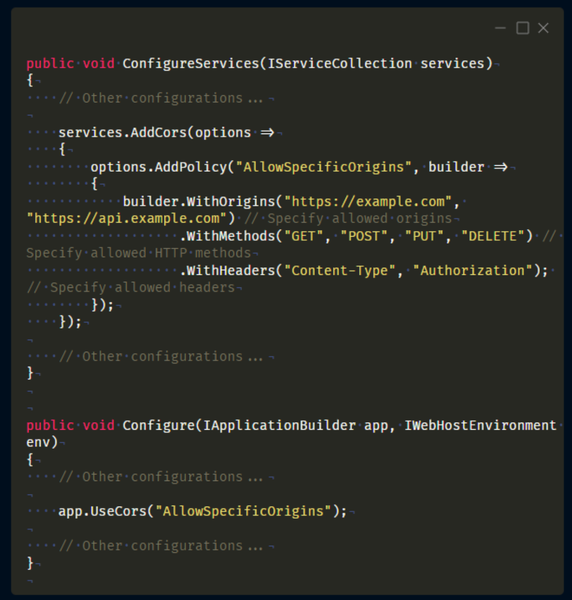
```
[Route("api/data")]
[EnableCors("MySingleOriginCorsPolicy")]
0 references
public class DataController : Controller
{
    [HttpGet]
    [EnableCors("MySingleOriginCorsPolicy")]
    0 references
    public string GetSomeData()
    {
        return "some data here...";
    }
}

[Route("api/data")]
[EnableCors("MySingleOriginCorsPolicy")]
0 references
public class DataController : Controller
{
    [HttpGet]
    [DisableCors]
    0 references
    public string GetSomeData()
    {
        return "some data here...";
    }
}
```

Explicit CORS use

Add support for Options

Is used to retrieve the communication options available for a given resource or server. It allows the client to determine which HTTP methods and headers are allowed on a specific endpoint. The server responds with an ”Allow” header, which lists the HTTP methods that the server supports for the requested resource. It is often used as part of the Cross-Origin Resource Sharing (CORS) mechanism, allowing browsers to check if a particular cross-origin request is allowed before actually making it.



```
public void ConfigureServices(IServiceCollection services)
{
    // Other configurations ...

    services.AddCors(options =>
    {
        options.AddPolicy("AllowSpecificOrigins", builder =>
        {
            builder.WithOrigins("https://example.com",
"https://api.example.com") //Specify allowed origins
                .WithMethods("GET", "POST", "PUT", "DELETE") //
Specify allowed HTTP methods
                .WithHeaders("Content-Type", "Authorization");
//Specify allowed headers
        });
    });

    // Other configurations ...
}


public void Configure(IApplicationBuilder app, IWebHostEnvironment
env)
{
    // Other configurations ...

    app.UseCors("AllowSpecificOrigins");

    // Other configurations ...
}
```

Options implementation with CORS

```
[HttpOptions()]
0 references
public IActionResult GetAuthorsOptions()
{
    Response.Headers.Add("Allow", "GET,HEAD,POST,OPTIONS");
    return Ok();
}
```

Options endpoint manual implementation

Healthcheck

Health checks play a crucial role in ensuring the availability and reliability of APIs. Health checks are a mechanism that allows applications to periodically check the status and health of their components, including API endpoints, databases, services, and other dependencies. By regularly monitoring the health of these components, you can proactively detect and respond to issues, improving overall API availability.

```
public void ConfigureServices(IServiceCollection services)
{
    // Other configurations ...

    services.AddHealthChecks();

    // Other configurations ...
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment
env)
{
    // Other configurations ...

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
        endpoints.MapHealthChecks("/health"); // Endpoint for
health checks
    });

    // Other configurations ...
}
```

Add Healthcheck

HATEOAS

stands for ”Hypermedia as the Engine of Application State,” is a principle of RESTful API design that emphasizes the use of hypermedia links in API responses to enable clients to navigate and interact with the API dynamically. The idea behind HATEOAS is to make the API self-descriptive, allowing clients to discover available actions and resources without prior knowledge of the API’s URL structure.

```
{ …
  "links": [ …,
      {
        "href": "http://host/api/authors?pageNumber=1&pageSize=10",
        "rel": "previous-page",
        "method": "GET"
      },
      {
        "href": "http://host/api/authors?pageNumber=3&pageSize=10",
        "rel": "next-page",
        "method": "GET"
      }]
}
```

```
{ …
  "links": […,
      {
        "href": "http://host/api/authors/{authorId}/courses/{courseId},
        "rel": "update-course-partial",
        "method": "PATCH"
      },
      {
        "href": "http://host/api/authors/{authorId}/courses/{courseId},
        "rel": "delete-course",
        "method": "DELETE"
      } ,
      {
        "href": "http://host/api/coursereservations,
        "rel": "reserve-course",
        "method": "POST"
      }]}
```

Example

```
{ "id": "5b1c2b4d-48c7-402a-80c3-cc796ad49c6b",
  "title": "Commandeering a ship without getting caught",
  "description": "Commandeering a ship in rough waters ...",
  "authorId": "d28888e9-2ba9-473a-a40f-e38cb54f9b35",
  "numberOfAvailablePlaces": 10,
  "content": "mature" }
```

Issues Without HATEOAS

Intrinsic knowledge of the API contract is required
An additional rule, or a change of a rule, breaks consumers of the API
The API cannot evolve separately of consuming applications

Richardson Maturity model

The Richardson Maturity Model is a model introduced by Leonard Richardson to assess the level of maturity and adherence to RESTful principles in Web API design.

```
POST (info on data)
http://host/myapi

POST (author to create)
http://host/myapi
```

Level 0 (The Swamp of POX)

HTTP protocol is used for remote interaction
... the rest of the protocol isn't used as it should be

RPC-style implementations (SOAP, often seen when using WCF)

```
POST
http://host/api/authors

POST
http://host/api/authors/{id}
```

Level 1 (Resources)

Each resource is mapped to a URI
HTTP methods aren't used as they should be

```
GET
http://host/api/authors
200 Ok (authors)

POST (author representation)
http://host/api/authors
201 Created (author)
```

Level 2 (Verbs)

Correct HTTP verbs are used
Correct status codes are used

```
GET
http://host/api/authors
200 Ok (authors + links that
drive application state)
```

Level 3 (Hypermedia)

The API supports Hypermedia as the Engine of Application State (HATEOAS)

Introduces discoverability

Demo

We will check the following:

What status code are handle it

How to configure Swagger to show different status codes

Head, options and Healthcheck

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

DocumentingASPNetCore6API

You can check the following Repository for some examples:

C# fundamentals

Task

Exercise/ Homework

This is a continuation of the previous exercise. Add validations for the controllers implemented:

- Add support for Healthcheck
- Add support for CORS
- Handle correctly your response codes