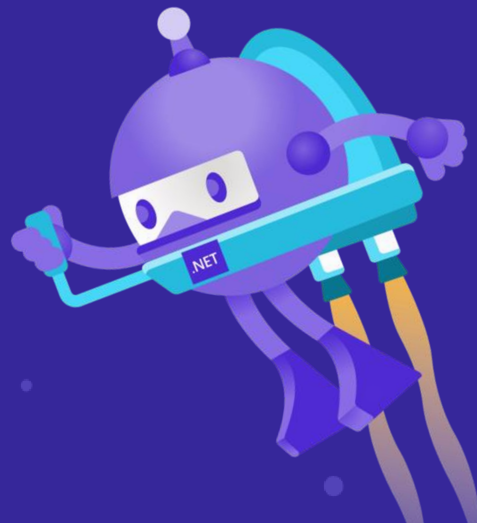
Day 18: API Versioning

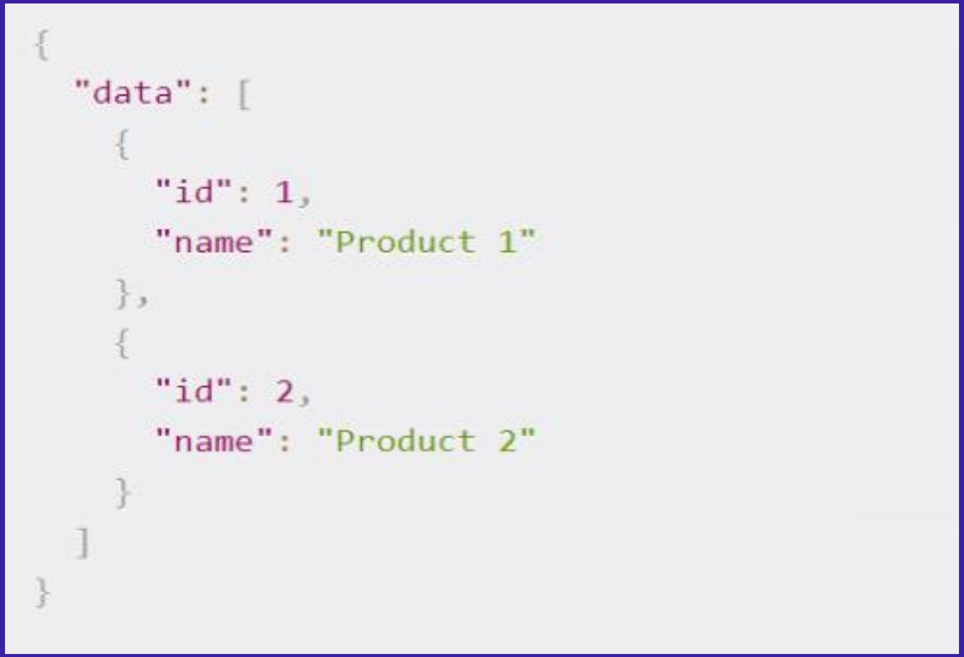


API Versioning

- Is the practice of transparently managing changes to your API
- Ultimately its about managing data contracts and breaking changes
- Backward compatibility (BC) of the APIs should be maintained whenever possible
- If a change that may break BC is necessary, you should introduce it in new version of the API

Web API – Data Contracts

- Is an agreement on the shape and general content of request and/or response data
- Example – Response Data Contract



```
{
  "data": [
    {
      "id": 1,
      "name": "Product 1"
    },
    {
      "id": 2,
      "name": "Product 2"
    }
  ]
}
```

Web API – Breaking Changes

- Is any change to your API contract that forces the consumer to also make a change

- Includes
  - Changing the request/response format
  - Changing a property name or data type on a property
  - Adding a required field on the request
  - Removing a property on the response

Web API – Versioning Strategies

- A versioning strategy allows clients to continue using the existing REST API and migrate their applications to the newer API when they are ready

- Four Approaches
    - Through URI Path
    - Through query parameters
    - Through custom headers
    - Through content negotiation

Versioning Through URI Path

- Adds a query parameter to the request that indicates the version
- Is often done with the prefix "v".
- Example
  - http://www.example.com/api/1/products
  - http://api.example.com/v1/products
- Pros:
  - Clients can cache resources easily
- Cons:
  - Duplication will be there in the codebase since multiple branches will be needed for maintaining different versions

Versioning Through Query Parameters

- Involves putting the version number in the path of the URI

- Example
  - http://www.example.com/api/products?version=1
  - http://api.example.com/products?ver=v1

- Pros:
  - Straightforward way to version an API, and it's easy to default to the latest version

- Cons:
  - Query parameters are more difficult to use for routing requests to the proper API version

Versioning Through Custom Headers

- Involves putting the version number in the Request Header collection
- Main difference with other two approach is that there will be no change in the URI
- Example
  - curl -H “Accepts-version: 1.0”   http://www.example.com/api/products
  - curl -H “Accepts-version: 2.0”   http://www.example.com/api/products
- Pros:
  - doesn’t clutter the URI with versioning information
- Cons:
  - requires custom headers

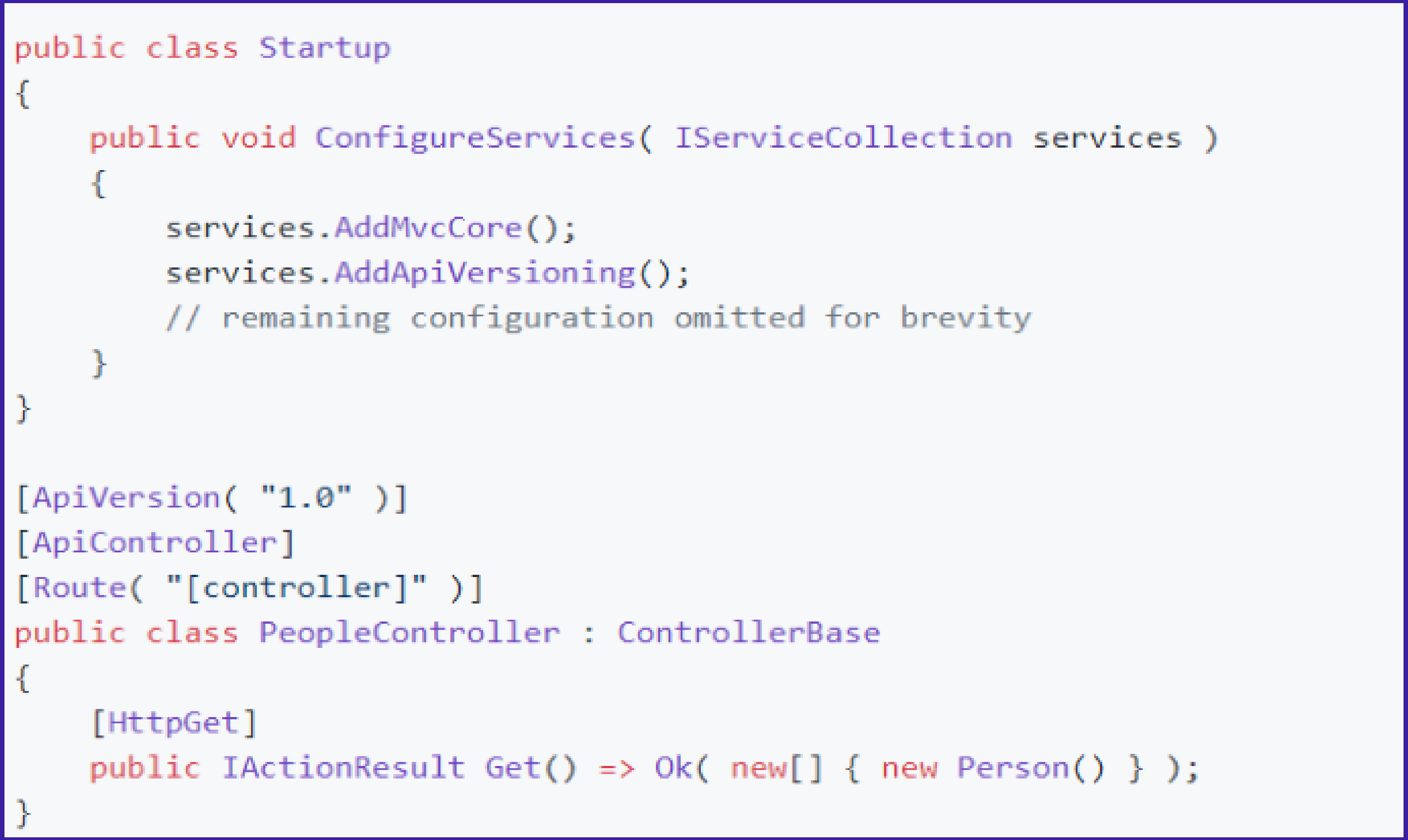
Versioning Through Content Negotiation

- Passes version information without the adding custom header or by changing the URI
- It versions the resource representations rather than versioning the entire API
- Example
  - curl -H “Accept: application/vnd.xm.device+json; version=2” http://www.example.com/api/products
- Pros:
  - Allows to version a single resource representation
  - Gives a more granular control over versioning
  - Creates only a smaller code base
- Cons:
  - Requiring HTTP headers with media types makes it more difficult to test and explore the API using a browser

Adding Versioning in ASP.NET Core

- ASP.NET API versioning gives you a powerful, but easy-to-use method for adding API versioning semantics
- Is compliant with the versioning semantics outlined by the Microsoft REST Guidelines
- API versioning extensions define simple metadata attributes and conventions that you use to describe which API versions are implemented by your services
- Pacakge to be added is Microsoft.AspNetCore.Mvc.Versioning

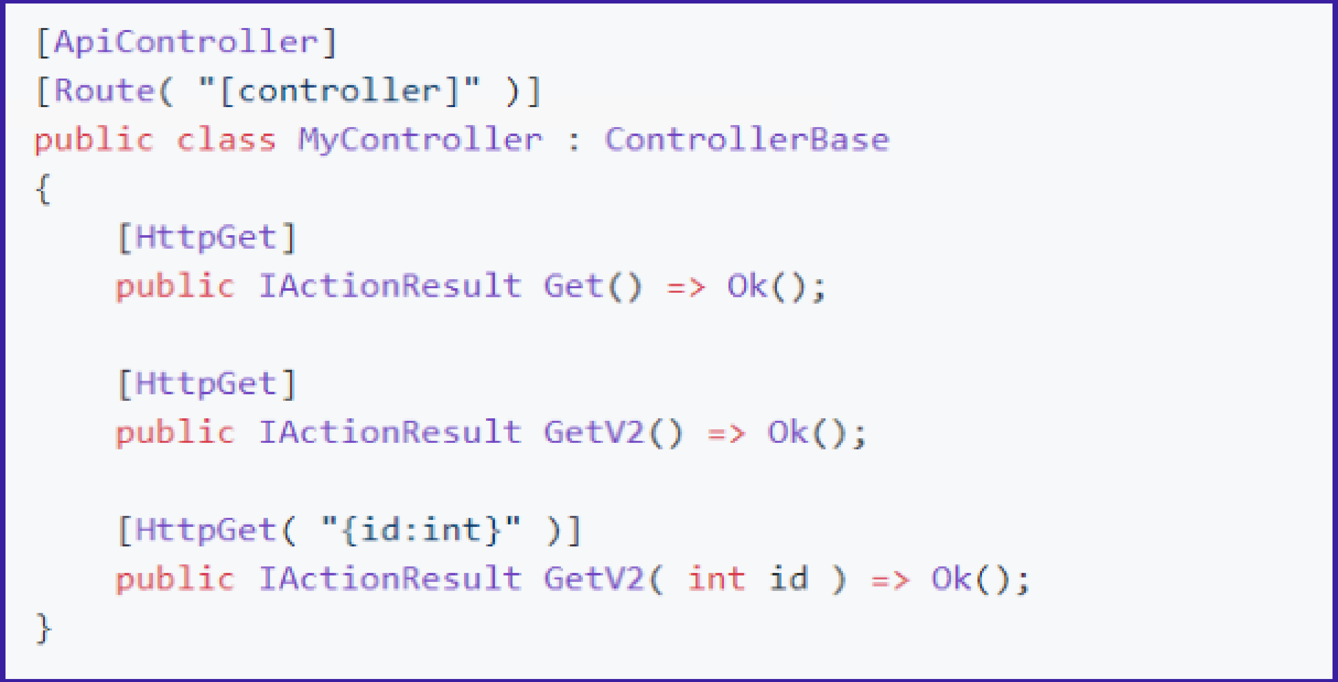
Adding Versioning in ASP.NET Core



```
public class Startup
{
    public void ConfigureServices( IServiceCollection services )
    {
        services.AddMvcCore();
        services.AddApiVersioning();
        // remaining configuration omitted for brevity
    }
}

[ApiVersion( "1.0" )]
[ApiController]
[Route( "[controller]" )]
public class PeopleController : ControllerBase
{
    [HttpGet]
    public IActionResult Get() => Ok( new[] { new Person() } );
}
```

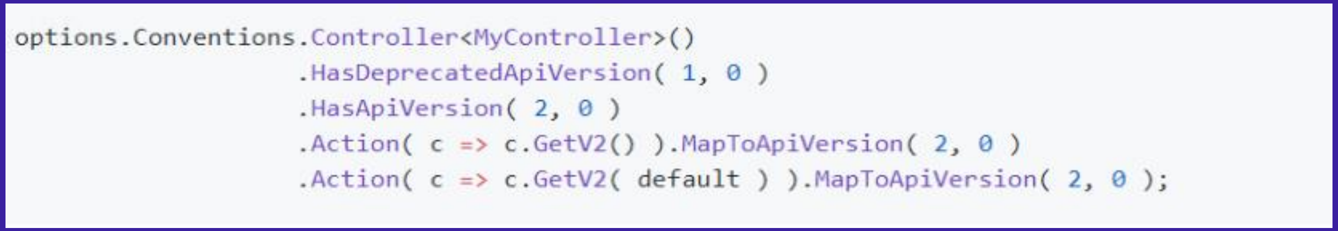
Adding Versioning in ASP.NET Core



```
[ApiController]
[Route( "[controller]" )]
public class MyController : ControllerBase
{
    [HttpGet]
    public IActionResult Get() => Ok();

    [HttpGet]
    public IActionResult GetV2() => Ok();

    [HttpGet( "{id:int}" )]
    public IActionResult GetV2( int id ) => Ok();
}
```



```
options.Conventions.Controller<MyController>()
                   .HasDeprecatedApiVersion( 1, 0 )
                   .HasApiVersion( 2, 0 )
                   .Action( c => c.GetV2() ).MapToApiVersion( 2, 0 )
                   .Action( c => c.GetV2( default ) ).MapToApiVersion( 2, 0 );
```

Thanks for joining!