.NET Core: Developing Cross-Platform Web Apps with ASP.NET Core – Workshop*PLUS*

< Engineer Name >

Customer Engineer

v3.1

Module 6: Web API

Module Overview

Module 6: Web API

Section 1: RESTful Services

Lesson: REST Fundamentals

What is Representational State Transfer?

- REST defines a set of *design principles and architectural styles* for highly scalable distributed systems
- REST is coined by Roy Fielding in his PhD thesis in 2000
- Roy is also the principle author of the HTTP specification and the co-author of the Uniform Resource Identifier (URI) specification
- The World Wide Web is an existing proof of a massively scalable distributed system that follows REST principles

REST Principles

- Identifiable resources
    - Give everything an ID
- Resource representations
    - Resources with multiple representations
- Hypermedia as the Engine of Application State (HATEOAS)
    - Link resources together
- Uniform Interface
    - Use standard methods
- Stateless communication
    - Communicate in a stateless fashion

Resource Representations

- Resource can have multiple representations, for example, HTML, XHTML, XML, JSON, plain text, multimedia, Atom, etc.
- Content negotiation allows client to specify the type of content it can accept and preference: “I want XML”
- MIME defines some standard content types:
    - “text/plain”
    - “text/html”
    - “image/jpeg”
    - “audio/mp3” and so on.

Uniform Interface

- Every resource supports the same interface
- In HTTP, this interface is
  - HTTP verbs: GET, PUT, POST, DELETE, HEAD, OPTIONS
  - HTTP status codes
- Safe operations: Idempotence
  - If an operation is executed twice the results are the same e.g. two Put requests with same data
- Advantages
  - Simplicity
    - Stateless with respect to client and server communication
  - Extensibility
    - Mashups, MIME types, etc.
  - Visibility
    - URIs, Verbs, headers provide for services like caching

Why REST?

- Move towards a resource-based application model
- Interoperability between cloud, on-premises and cross-domains
- Take advantage of Web’s infrastructure
- Easy composition of resources via Uniform Interface
- A “must have” if your APIs will be consumed by third-party services

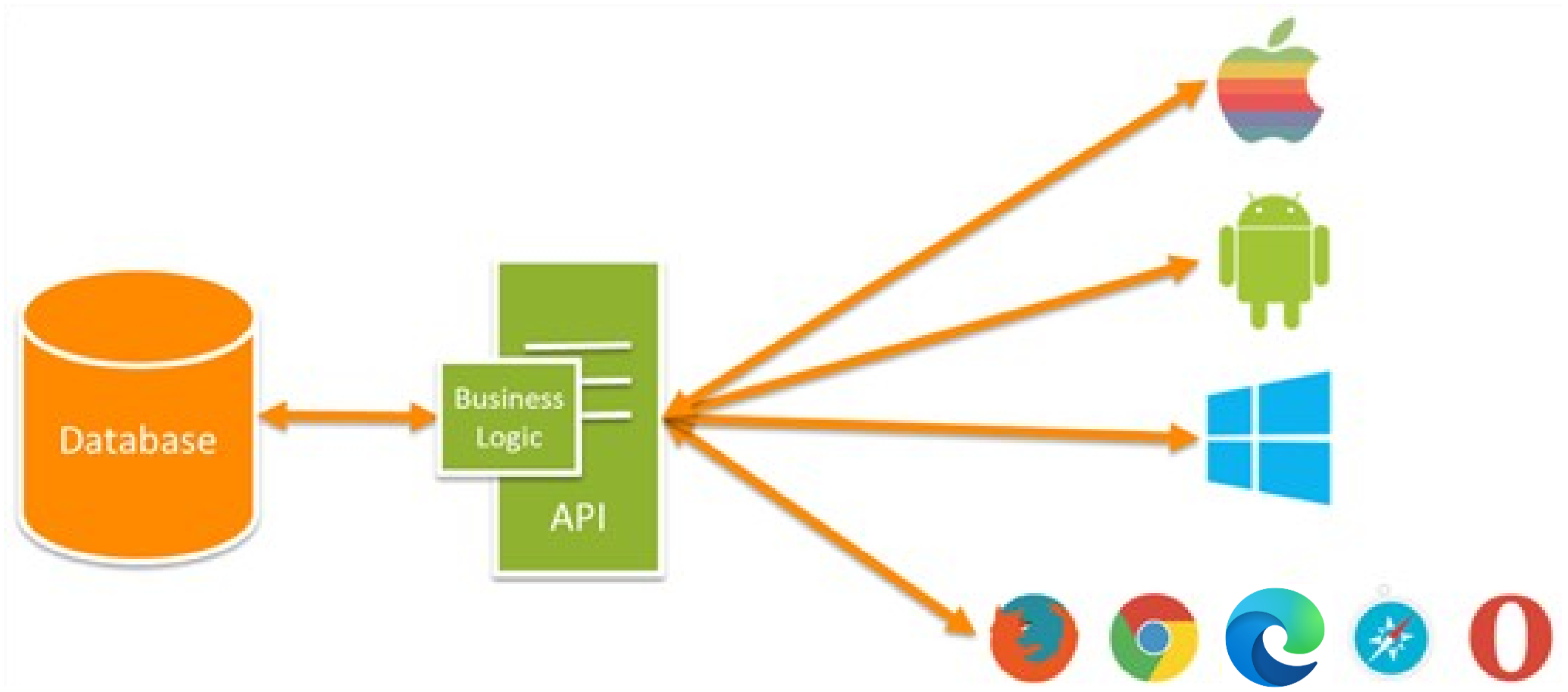
Demo: Online REST Web Service

Module 6: Web API

Section 2: ASP.NET Core Web API

Lesson: Web API Fundamentals

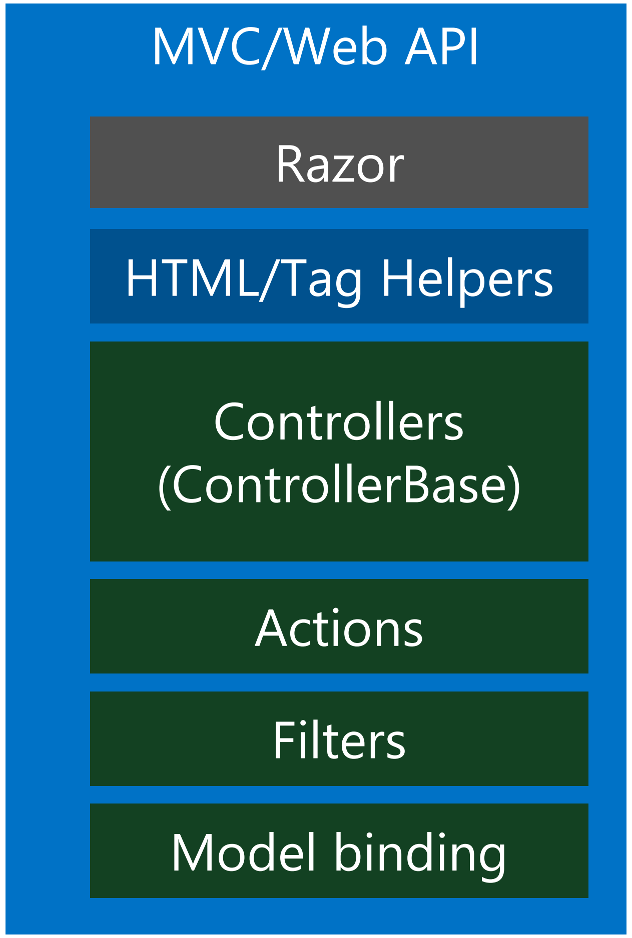
Web API



ASP.NET Web API

- ASP.NET Web API is a framework that makes it easy to build HTTP services.
  - Full support for ASP.NET Routing
  - Content negotiation and custom formatters
  - Model binding and validation
  - Filters
  - Query composition
  - Easy to unit test
  - Inversion of Control (IoC)
  - Code-based configuration
  - Self hosting

The World Today with ASP.NET Core



Module 6: Web API

Section 2: ASP.NET Core Web API

Lesson: Routing

Routing - Controllers

- ASP.NET Web API controllers derive from **ControllerBase**

```
[Route("api/[controller]")]
public class ValuesController : ControllerBase
```

- Do not create **Web Api** controller deriving from **Controller** class
- **MVC Controller** also derive from **ControllerBase**, but has code to deal with views
- You can use RESTful style routing with both MVC and Web API controllers

- Same routing framework for both MVC and the Web API
- Convention-based routes or attribute routes
- Default route template for Web API is "**api/[controller]**"

IActionResult for UI and API

- Controllers return IActionResult
  - For MVC Controllers, it might be view or data
  - For Web API controllers, it might be data

UI	API
PartialViewResult	BadRequestResult
RedirectResult	ContentResult
ViewResult	CreatedAtRouteResult
JsonResult	HttpStatusCodeResult
	JsonResult
	ObjectResult
ChallengeResult	
HttpNotFoundResult	
FileContentResult	

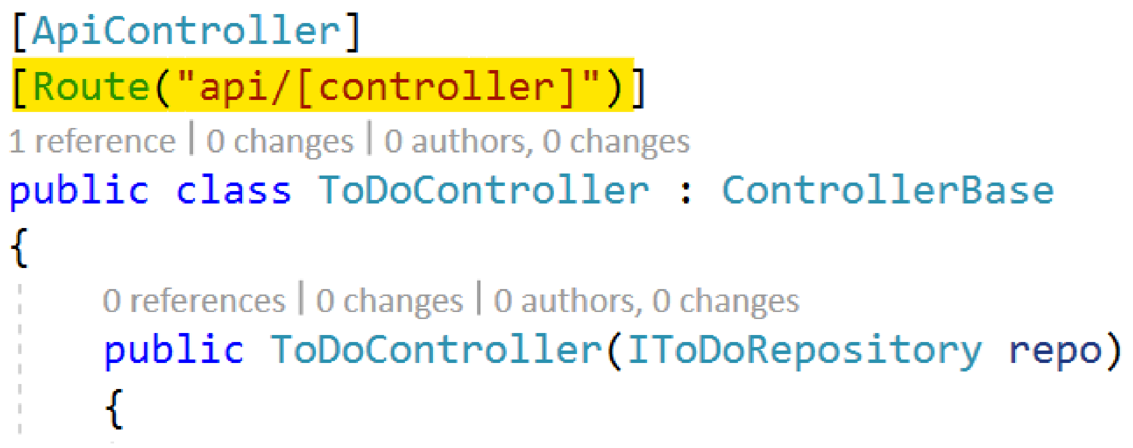
IActionResult for UI and API

- Controllers return IActionResult
  - For MVC Controllers, it might be view or data
  - For Web API controllers, it might be data

```csharp
[HttpGet("{id:int}")]
public IActionResult GetById(int id)
{
     var item = _items.FirstOrDefault(x => x.Id == id);
     if (item == null)
     {
            return NotFound();
     }

     return new ObjectResult(item);
}
```

Routing - Controllers

- To find the controller, Web API adds "Controller" to the value of the *[controller]* variable.
- Once a matching route is found, Web API selects the controller
- Web API uses the HTTP method, not the URI path, to select the action
- If no route matches, the client receives a 404 error



```
[ApiController]
[Route("api/[controller]")]
1 reference | 0 changes | 0 authors, 0 changes
public class ToDoController : ControllerBase
{
    0 references | 0 changes | 0 authors, 0 changes
    public ToDoController(IToDoRepository repo)
    {
```

Routing - Actions

- Web API looks at the HTTP method, and then looks for an action whose HTTP attribute is the same as the method
- This convention applies to GET, POST, PUT, and DELETE methods
- Other placeholder variables in the route template, such as {id} and query strings, are mapped to action parameters.

```
[HttpGet("{id}")]
public string Select(int id)
```

- [**FromBody**] attribute tells the framework that the parameter **should** be deserialized from the request body and not from URI

```
[HttpPut("{id:int}")]
public void Update(int id, [FromBody]string value)
```

HTTP Verbs

	Collection URI (http://api.example.com/v1/resources/)	**Element URI (http://api.example.com/v1/resources/item17)**
GET	List the URIs and perhaps other details of the collection's members.	Retrieve a representation of the addressed member of the collection.
PUT	Replace the entire collection with another collection.	Replace the addressed member of the collection, or if it does not exist, create it.
POST	Create a new entry in the collection.	Treat the addressed member as a collection in its own right and create a new entry in it.
DELETE	Delete the entire collection.	Delete the addressed member of the collection.

Routing - Actions

Verb	URL	Action
GET	api/values	[HttpGet] public IEnumerable<string> Get()
GET	api/values/5	[HttpGet("{id}")] public string Select(int id)
PUT	api/values/5	[HttpPut("{id}")] public void Update(int id, [FromBody]string value)
POST	api/values	[HttpPost] public void Create([FromBody]string value)
DELETE	api/values/5	[HttpDelete("{id}")] public void Delete(int id)

- To prevent a method from getting invoked as an action, use the **NonAction** attribute
- In case multiple Actions match, you get runtime exception:
  - 500 Internal Server Error - Microsoft.AspNetCore.Mvc.AmbiguousActionException

Test and Debug Web APIs

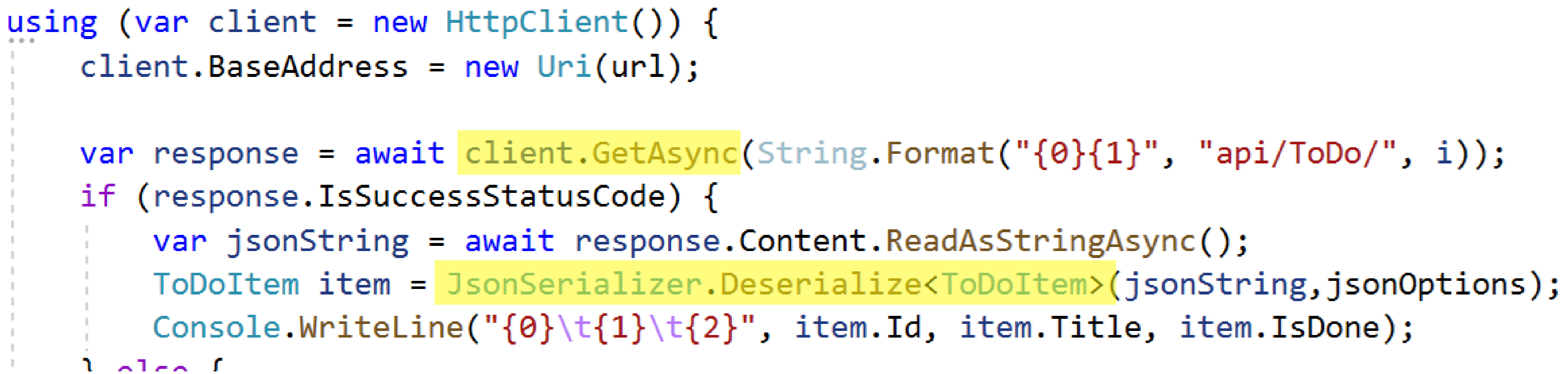
- Developer tools in browser are not enough to debug Web APIs
- Need tools that offer more control to issue HTTP requests and check responses
- UI Tools:
  - Telerik's Fiddler
    - Swiss knife for HTTP debugging
    - Capture and replay requests
    - Inspect requests/responses
    - Compose requests
  - Postman
    - Compose requests
    - Inspect responses
    - More user friendly
    - Organize requests in collections
- Command Line Tools
  - HTTPRepl
    - Multiplatform .Net Tool
    - Support for Swagger

    Dotnet tool install –g Microsoft.dotnet-httprepl
  - Curl
  - Powershell: Invoke-Webrequest

Demo: Web API Controllers

Call Web API from .Net Core application

- Make a request using **System.Net.Http.HttpClient**
- Use **System.Text.Json.JsonSerializer** to serialize and deserialize objects



```
using (var client = new HttpClient()) {
    client.BaseAddress = new Uri(url);

    var response = await client.GetAsync(String.Format("{0}{1}", "api/ToDo/", i));
    if (response.IsSuccessStatusCode) {
        var jsonString = await response.Content.ReadAsStringAsync();
        ToDoItem item = JsonSerializer.Deserialize<ToDoItem>(jsonString,jsonOptions);
        Console.WriteLine("{0}\t{1}\t{2}", item.Id, item.Title, item.IsDone);
    } else {
```

Demo: Call a Web API

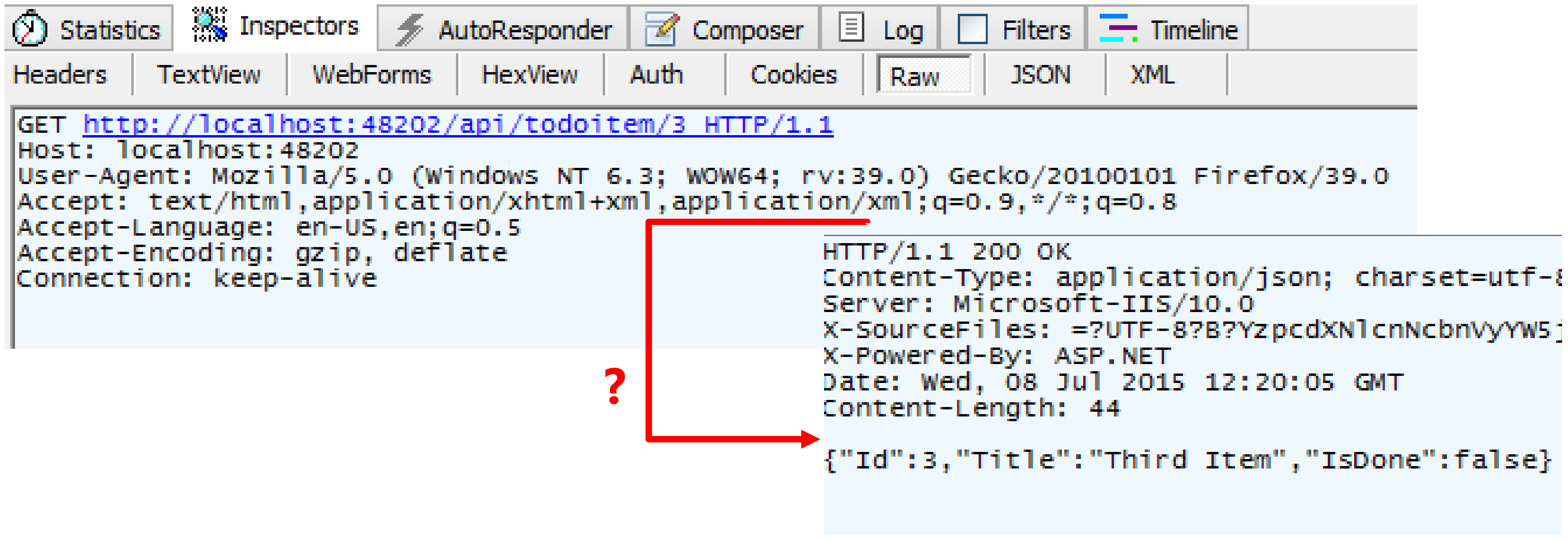
Module 6: Web API

Section 2: ASP.NET Core Web API

Lesson: Content Negotiation

Content Negotiation

- Apps consume content in different ways
- JSON and Extensible Markup Language (XML) are the most popular formats
- A good client will request the favored type

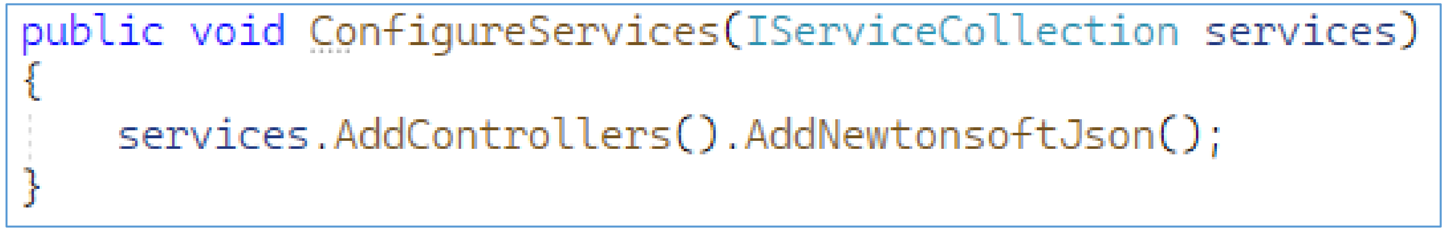


```
GET http://localhost:48202/api/todoitem/3 HTTP/1.1
Host: localhost:48202
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-
Server: Microsoft-IIS/10.0
X-SourceFiles: =?UTF-8?B?YzpcdXNlcnNcbnVyYW5
X-Powered-By: ASP.NET
Date: Wed, 08 Jul 2015 12:20:05 GMT
Content-Length: 44

{"Id":3,"Title":"Third Item","IsDone":false}
```

JSON Only By Default

- XML formatters (input and output) are now removed by default
Input formatters	Output formatters
SystemTextJsonInputFormatter	SystemTextJsonOutputFormatter
NewtonsoftJsonInputFormatter	StringOutputFormatter
NewtonsoftJsonPatchInputFormatter	StreamOutputFormatter
	HttpNoContentOutputFormatter
	NewtonsoftJsonOutputFormatter
- Custom formatters can be developed
- To use **`Newtonsoft*`** formatters you need to add the support to Newtonsoft Json:
  1. Install the **`Microsoft.AspNetCore.Mvc.NewtonsoftJson`** package.
  2. Update **`ConfigureServices`**



```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddNewtonsoftJson();
}
```

If You Need XML…

- XML formatters are found in "Microsoft.AspNetCore.Mvc.Formatters.Xml" package
- Manually add it to the formatters collection in Startup.cs

```
services.Configure<MvcOptions>(options =>
            {
             options.OutputFormatters.Add(new XmlSerializerOutputFormatter());

              //options.InputFormatters.Add(new XmlSerializerInputFormatter());

            });
```

- Use the [Produces] attribute in the action method

```
[Produces("application/xml")]
public IActionResult GetAll(int id)
```

Demo: Content Negotiation

Module 6: Web API

Section 2: ASP.NET Core Web API

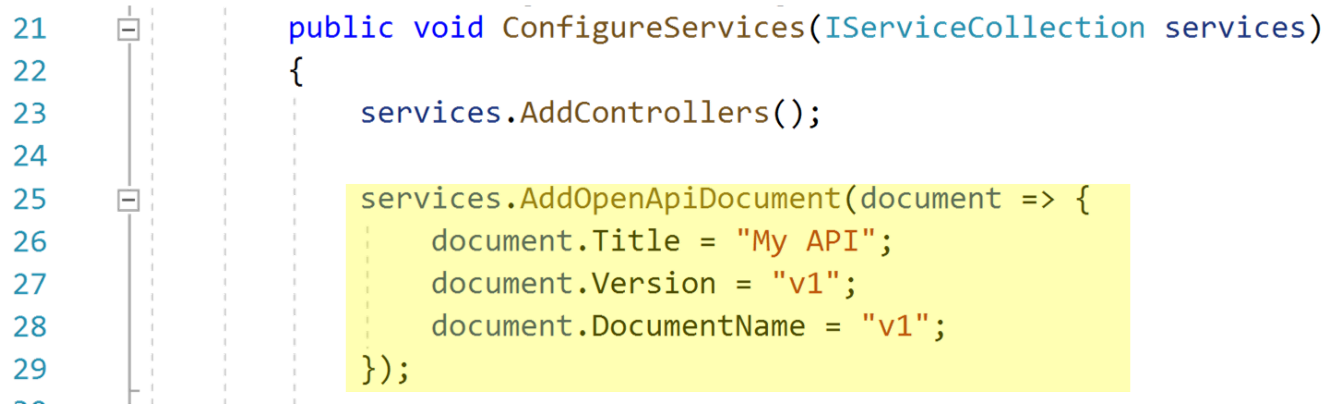
Lesson: Open API (Swagger) Documentation

OpenAPI (Swagger) Documentation

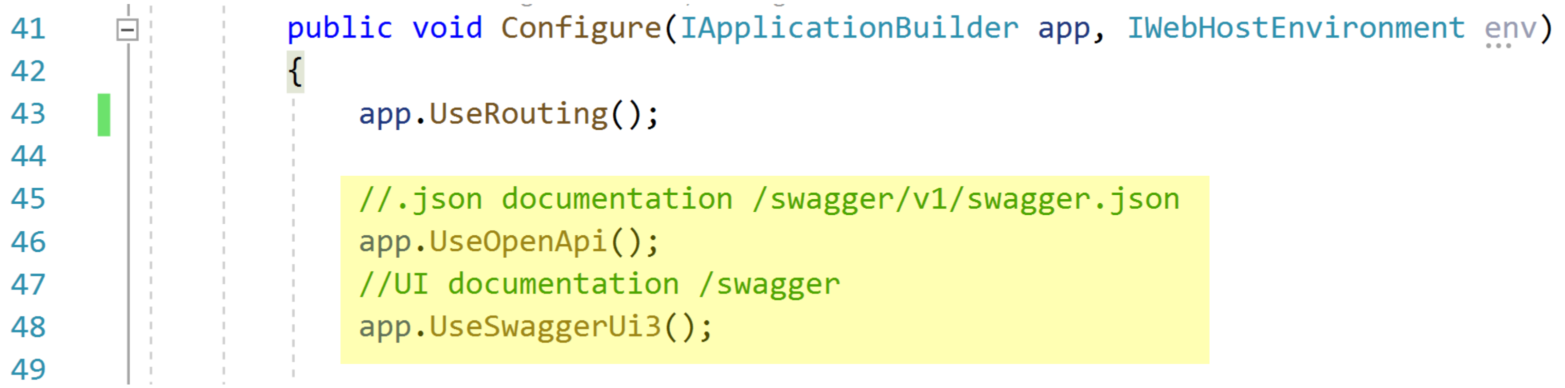
- OpenAPI is a simple yet powerful representation of your RESTful API.
- OpenAPI is the **most popular** representation of RESTful API.
- Swagger is the tooling for OpenAPI specification
- Swagger generates good documentation and help pages as a part of your Web API
- Documentation is customizable:
    - XML comment from you code will be included
    - Data Annotation Attributes will be applied

Open API (Swagger) configuration

- Sample code using NSwag.AspNetCore

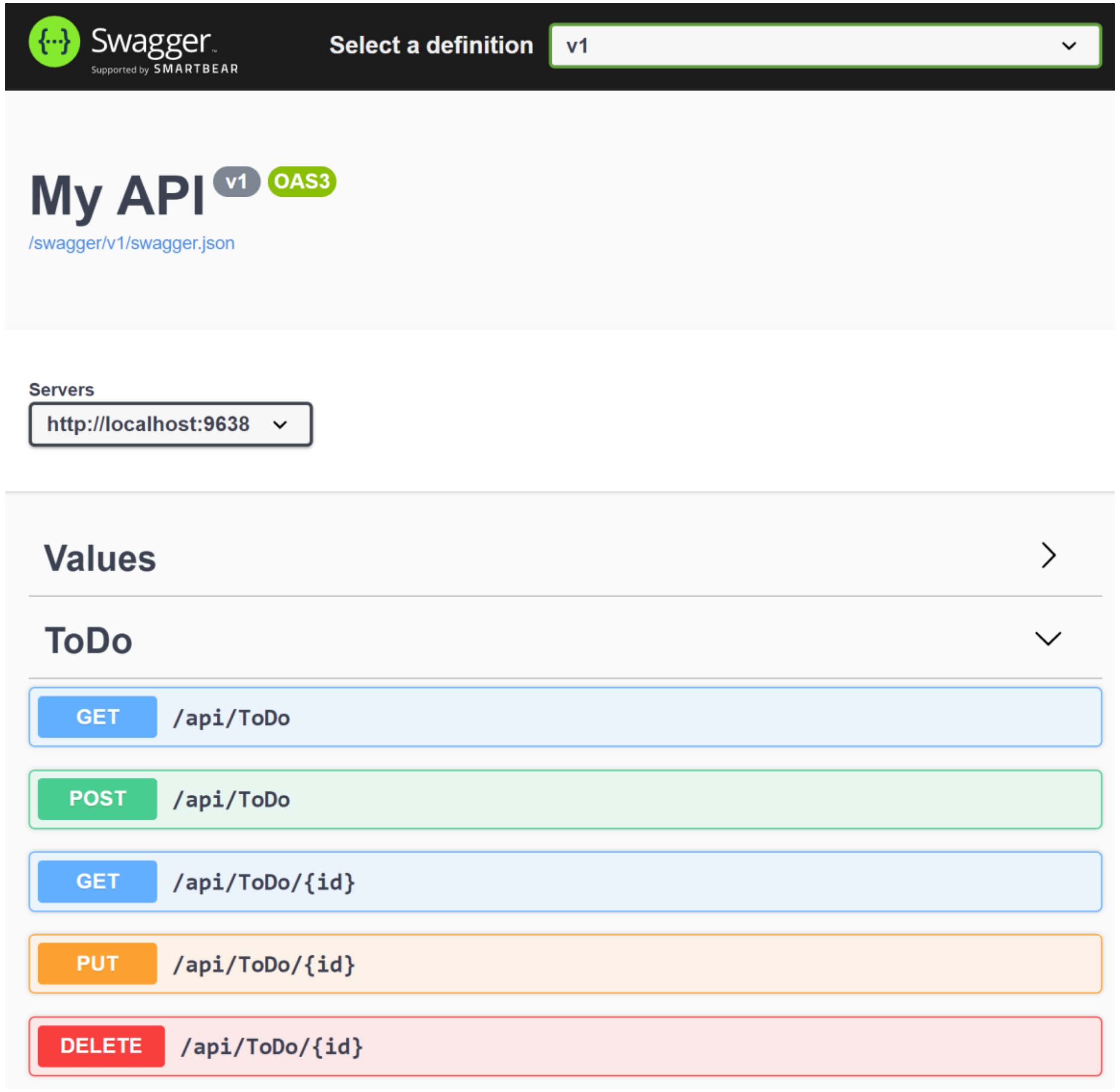


```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    services.AddOpenApiDocument(document => {
        document.Title = "My API";
        document.Version = "v1";
        document.DocumentName = "v1";
    });
```



```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseRouting();

    //.json documentation /swagger/v1/swagger.json
    app.UseOpenApi();
    //UI documentation /swagger
    app.UseSwaggerUi3();
```

Open API (Swagger) UI example:



Swagger
Supported by SMARTBEAR
Select a definition v1

My API v1 OAS3
/swagger/v1/swagger.json

Servers
http://localhost:9638

Values

ToDo

GET /api/ToDo
POST /api/ToDo
GET /api/ToDo/{id}
PUT /api/ToDo/{id}
DELETE /api/ToDo/{id}

Demo: Open API (Swagger)

Module 6: Web API

Section 3: Web API Usage Patterns

Lesson: Web API Design

Good API Design

- There is no real standard or method for designing APIs
- Keep your base URL simple and intuitive
- Do not change your URLs over time

Nouns, No Verbs

- Use nouns for resource names
- Use plural rather than singular nouns, and concrete rather than abstract names
- Keep verbs out of your base URLs
- Use HTTP verbs to operate on the collections and elements

Tip: Nouns are good, verbs are bad

- Actions that do not fit into the world of CRUD operations can be endpoint
  - GitHub's API
    - star a gist with PUT /gists/:id/star
    - unstar with DELETE /gists/:id/star

Simplify Associations

- Nested resources(Association)
- User URI navigation (/resource/identifier/resource)
    - GET /api/Customers/123/Invoices
    - GET /api/Customers/123/Payments
    - GET /api/Customers/123/Shipments
- Keep the base resource URLs lean by implementing query parameters
    - GET /api/Customers?state=GA
    - GET /api/Customers?hasOpenOrders=true

Use Meaningful HTTP Status Codes

- Use HTTP status codes and map them cleanly to relevant standard-based codes

Group	Comment
1xx	Informational (100, 101)
2xx	Success (200, 201)
3xx	Redirect (301, 304)
4xx	Client Error (401, 404, 405)
5xx	Server Error (500, 503)

Versioning

- Never release an API without a version and make the version mandatory
- Maintain at least one version back
- Versioning with Content Negotiation and custom headers are popular now
  - Content-Type: application/json;v=2
  - X-MyApp-Version = 2
- Versioning with URI components are more common
  - Easier to implement
  - GET /api/v2/Customers/123/Invoices
- Check the **Microsoft.AspNetCore.MvcVersionig** Nuget package

Demo: API Versioning

Caching

- Implement caching carefully
- Time-based (Last-Modified)
  - When generating a response include a HTTP header Last-Modified
    - **Last-Modified: Thu, 29 Aug 2019 16:24:26 GMT**
  - When generating a request, include a HTTP header If-Modified-Since
    - **If-Modified-Since: Thu, 29 Aug 2019 16:24:26 GMT**
  - If an inbound HTTP requests contains a If-Modified-Since header, the API should return a 304 Not Modified status code instead of the output representation of the resource
- Content-based (ETag)
  - Hash based on the content sent as HTTP header in a response
    - **ETag: "5d67fc3c-4a5f“**
  - If-none-match HTTP header in request with same hash
    - **If-None-Match: " 5d67fc3c-4a5f“**
  - Useful when the last modified date is difficult to determine

Demo: ETag

Security

- Do you need to secure your API?

Are you?	Secure?
Using private or personalized data	Yes
Sending sensitive data across the wire?	Yes
Using credentials of any kind	Yes
Trying to protect overuse of your servers?	Yes

Security

- Do not expose your domain model
- Always use SSL
- Secure the API itself
    - HTTP Basic Authentication
    - API token keys
    - Custom authentication mechanism with cookies or token
    - JWT tokens
    - OAuth

Demo: Web API Security using simple Key

Security

- JWT tokens
  - Open standard (RFC 7519)
  - JSON
  - Compact and self-contained way for securilly transmit informations between parties
  - Signed using secrets (HMAC) or public/private key pair (RSA)
  - Base64 encoded
  - Can be sent through URL, POST or HTTP header
  - Used by authentication flows like OAuth and OpenId Connect
  - Normally are only signed and not encrypted. Always use over HTTPS/SSL.

Security

- Configure application to authorize with JWT tokens:

```
public void ConfigureServices(IServiceCollection services){
   …
    var key = Encoding.ASCII.GetBytes(appSettings.Secret);
    services.AddAuthentication(aut => {
        aut.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        aut.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(o => {
        o.RequireHttpsMetadata = false;
        o.SaveToken = true;
        o.TokenValidationParameters = new TokenValidationParameters {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key)
        };
    });
```

Security

- How to generate JWT tokens:

```
…
var tokenHandler = new JwtSecurityTokenHandler();
var key = Encoding.ASCII.GetBytes(_appSettings.Secret);
var tokenDescriptor = new SecurityTokenDescriptor{
    Subject = new ClaimsIdentity(new Claim[]{
        new Claim(ClaimTypes.Name, user.UserName)
    }),
    Expires = DateTime.UtcNow.AddDays(2),
    SigningCredentials =
       new SigningCredentials(new SymmetricSecurityKey(key),
                              SecurityAlgorithms.HmacSha256Signature)
};
var token = tokenHandler.CreateToken(tokenDescriptor);
user.Token = tokenHandler.WriteToken(token)
…
```

Demo: Web API Authorization with Jwt Token

Module 6: Web Api

Section 4: .NET gRPC Support

Lesson: Introduction to gRPC

What is gRPC

- Modern, open source remote procedure call (RPC) framework that can run anywhere
- Enables client and server applications to communicate transparently
- Makes it easier to build connected systems
- Point-to-point real-time services that need to handle streaming requests or responses.

https://grpc.io/

gRPC use cases

- Efficiently connecting polyglot services in microservices style architecture
  - Support for more than 10 languages:

    C/C++, C#, Dart, Go, Java, Kotlin/JVM, Node.js, Objective-C, PHP, Python, Ruby
- Connect mobile devices to backend services
- Generate efficient client libraries

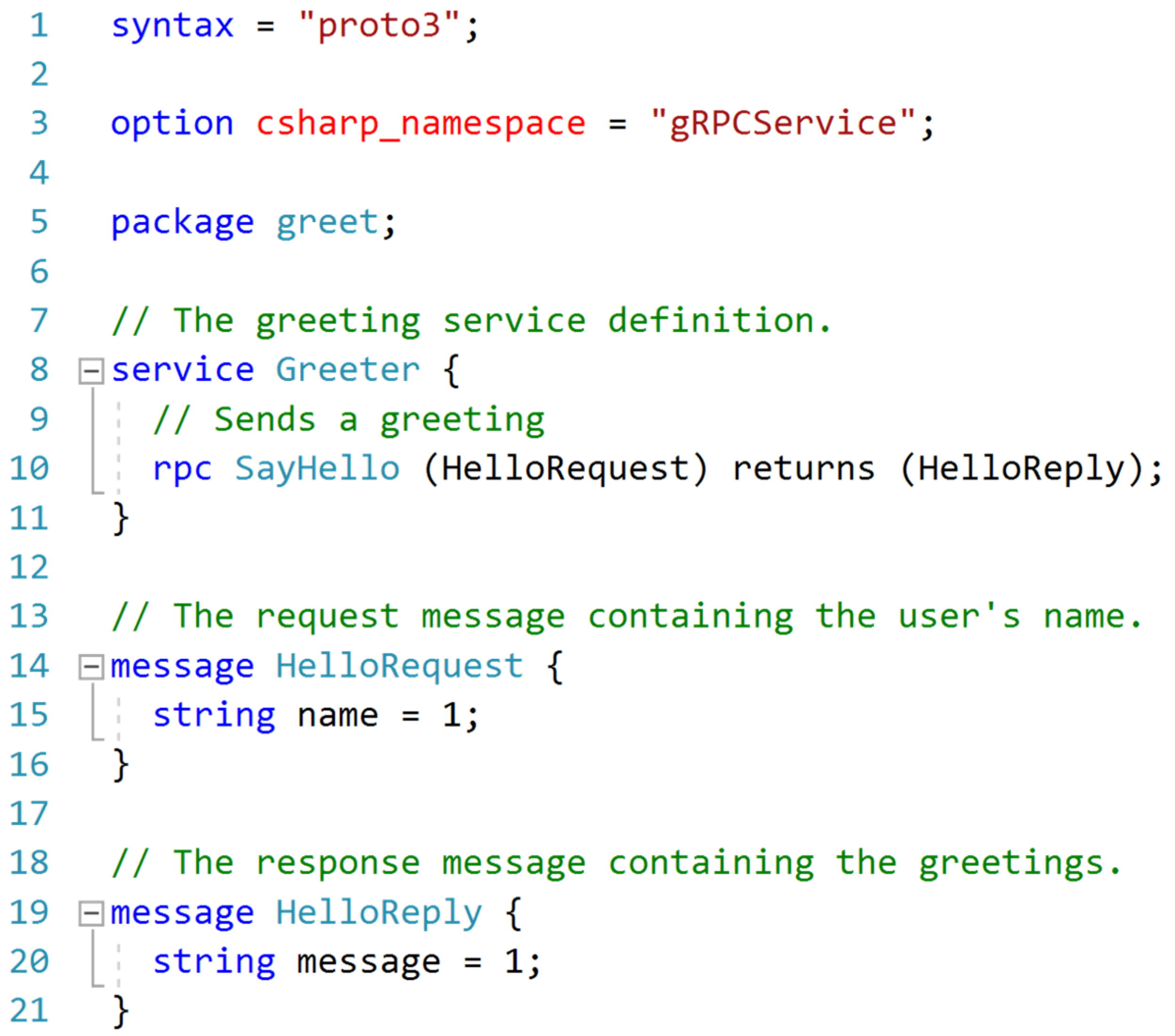
HTTP/2

- gRPC is built on top of HTTP/2
- HTTP/2 is a replacement for HTTP representation “on the wire”
- Keeps the methods, status codes and semantics of HTTP
- Focus on performance and low latency
- Key, high level, differences to HTTP/1.x?
    - Binary, instead of textual
    - Fully multiplexed, instead of ordered and blocking
    - Can use single connection for parallelism
    - Header compression to reduce overhead
    - Allow servers to “push” responses proactively into client caches

.NET Core support for gRPC

- C# Tooling Support for .proto files
- Project template for gRPC services
- C# client development
- Currently (May 2020) not supported on **Azure App Service** or **IIS**

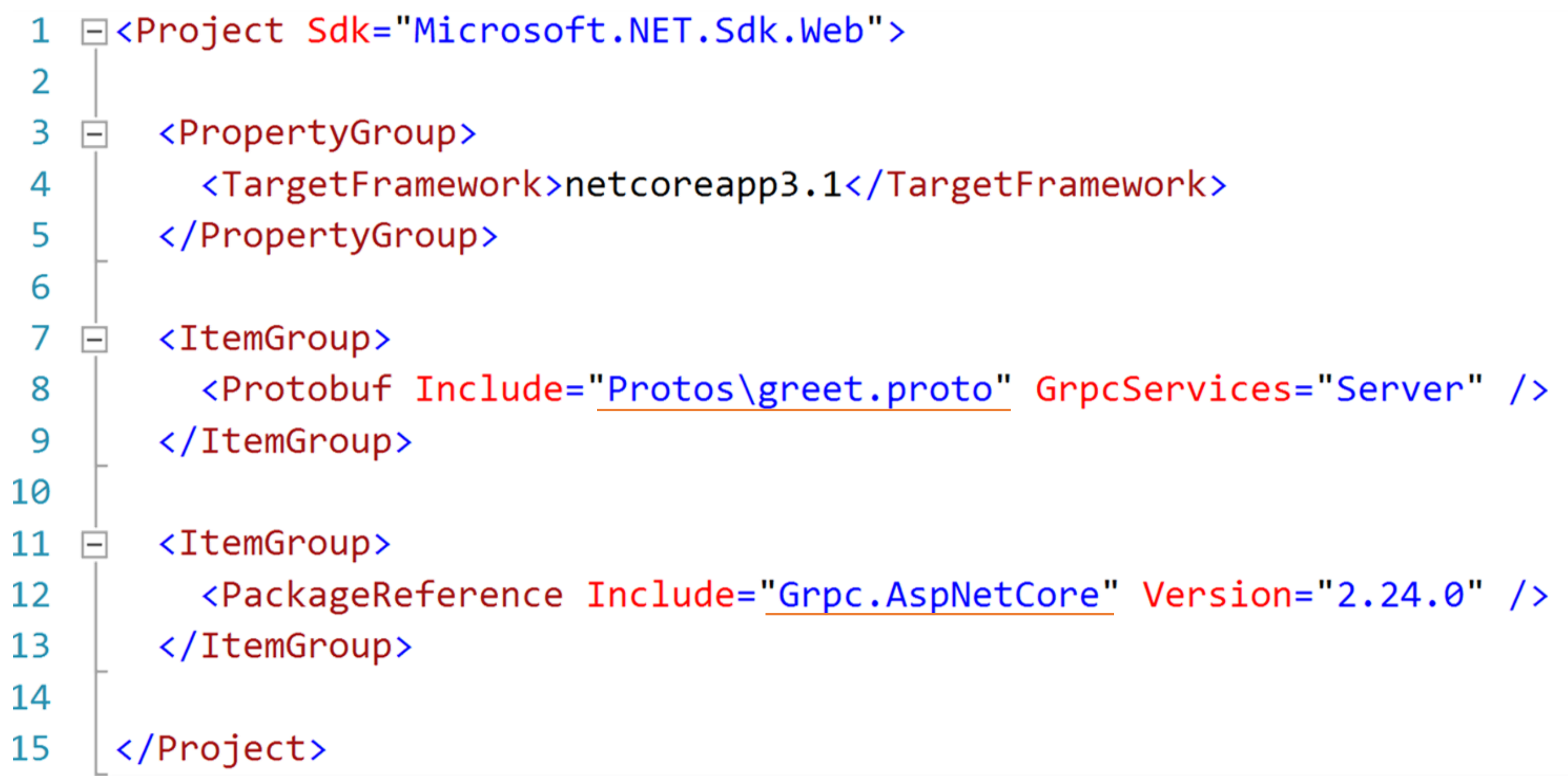
Protocol Buffers (.proto files)



```
syntax = "proto3";

option csharp_namespace = "gRPCService";

package greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply);
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings.
message HelloReply {
  string message = 1;
}
```

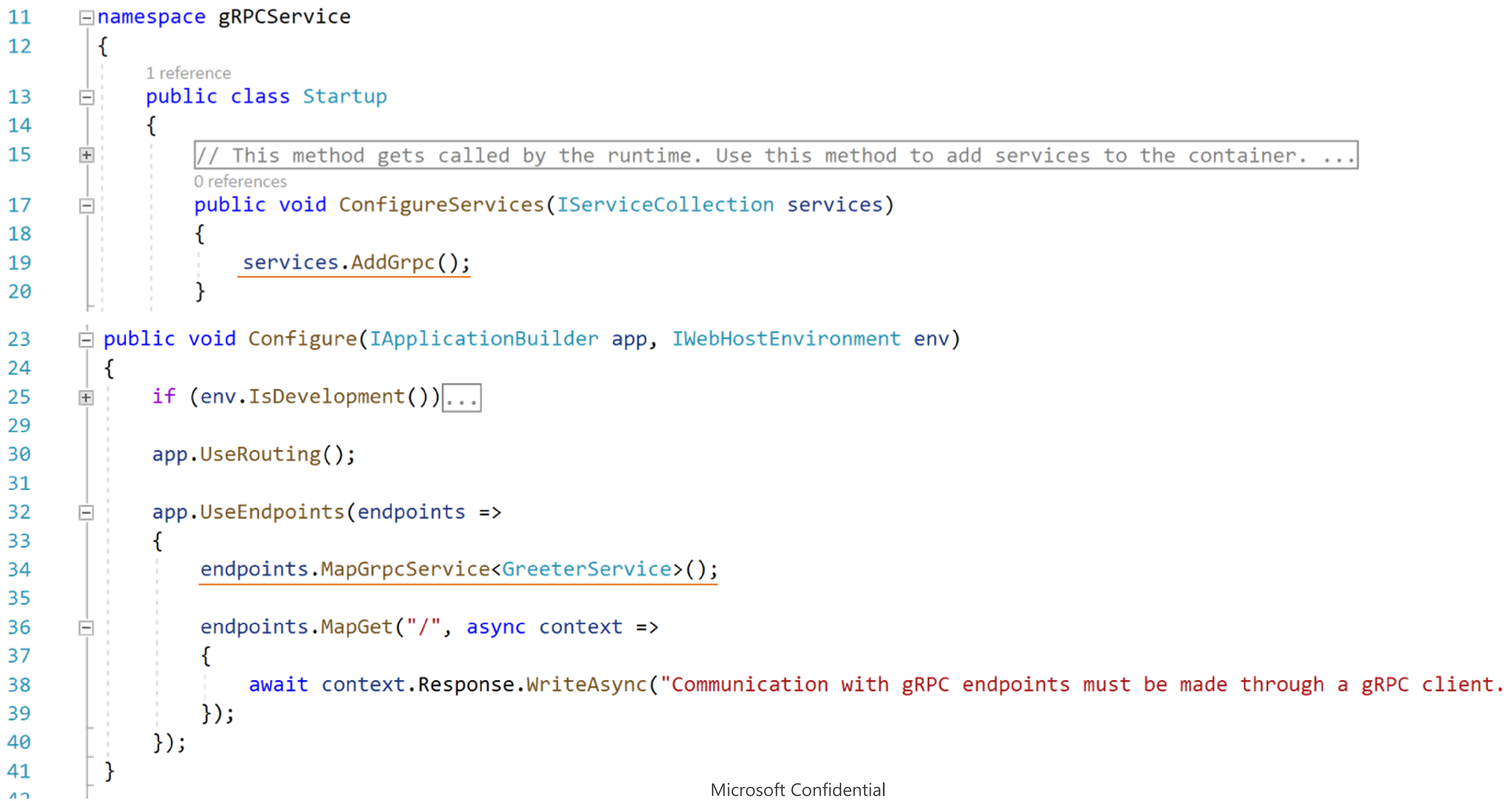
Project file (.csproject)

- References for the .proto files and the Grpc.AspNetCore



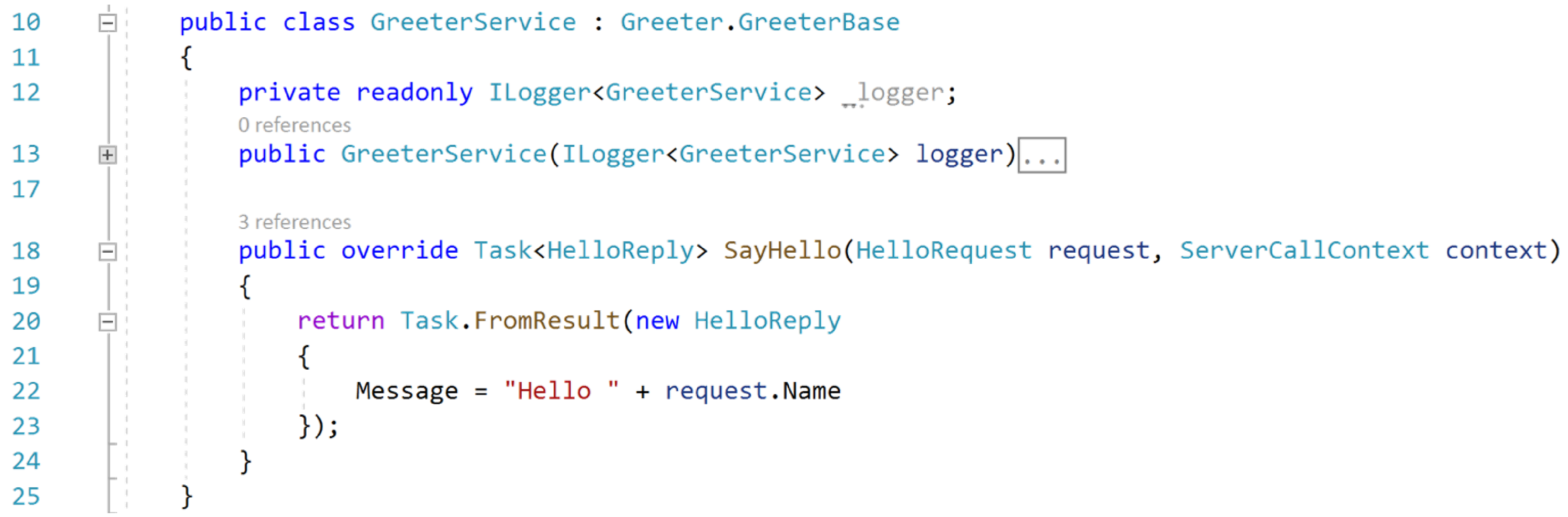
```xml
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Protobuf Include="Protos\greet.proto" GrpcServices="Server" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Grpc.AspNetCore" Version="2.24.0" />
  </ItemGroup>

</Project>
```

Configuration – Startup.cs



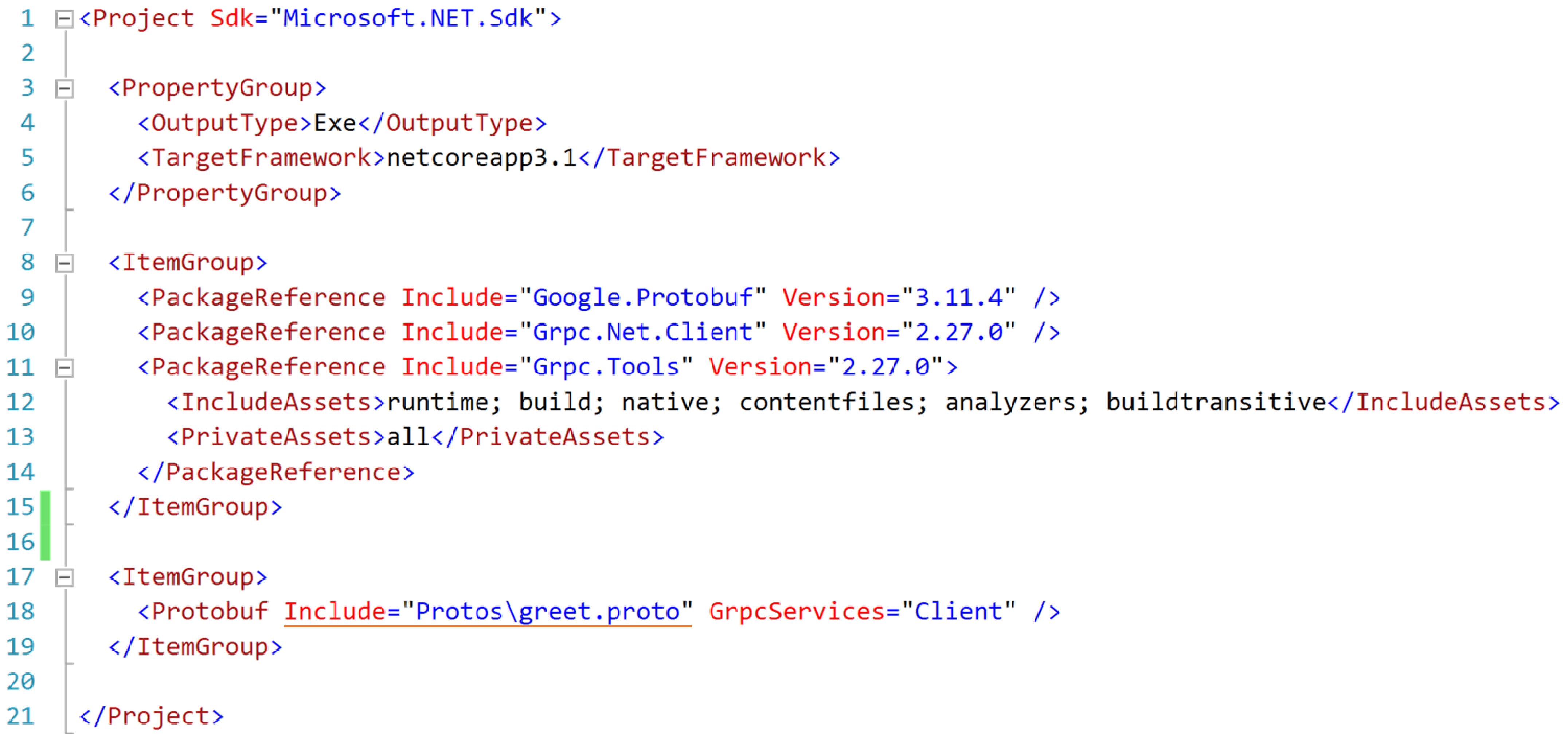
```
namespace gRPCService
{
    1 reference
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the container. ...
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddGrpc();
        }

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())...

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGrpcService<GreeterService>();

        endpoints.MapGet("/", async context =>
        {
            await context.Response.WriteAsync("Communication with gRPC endpoints must be made through a gRPC client.
        });
    });
}
```

gRPC services on ASP.NET Core



```
public class GreeterService : Greeter.GreeterBase
{
    private readonly ILogger<GreeterService> _logger;
    0 references
    public GreeterService(ILogger<GreeterService> logger)...

    3 references
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)
    {
        return Task.FromResult(new HelloReply
        {
            Message = "Hello " + request.Name
        });
    }
}
```

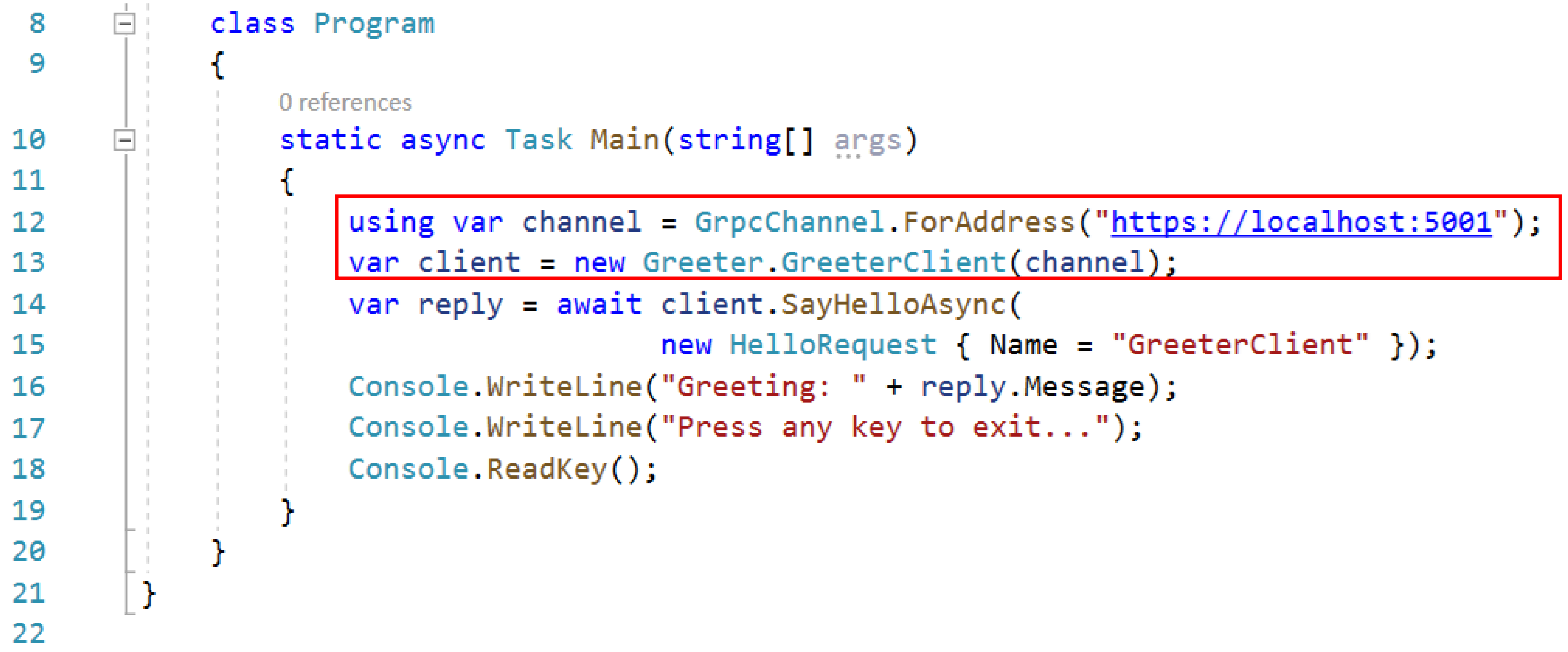
gRPC clients – Project file



```xml
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Google.Protobuf" Version="3.11.4" />
    <PackageReference Include="Grpc.Net.Client" Version="2.27.0" />
    <PackageReference Include="Grpc.Tools" Version="2.27.0">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
  </ItemGroup>

  <ItemGroup>
    <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
  </ItemGroup>

</Project>
```

gRPC clients calling services

- Channel is a long-lived connection to a gRPC service
- The client is created using a channel
- The client is a concrete type generated from .proto files



```
class Program
{
    0 references
    static async Task Main(string[] args)
    {
        using var channel = GrpcChannel.ForAddress("https://localhost:5001");
        var client = new Greeter.GreeterClient(channel);
        var reply = await client.SayHelloAsync(
                          new HelloRequest { Name = "GreeterClient" });
        Console.WriteLine("Greeting: " + reply.Message);
        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }
}
```

Demo: simple gRPC client and server

Module Summary

- In this module, you learned about:
  - REST principles
  - Web API fundamentals
  - Web API routing
  - Content negotiation
  - Web API Design
  - gRPC Introduction

Lab: ASP.NET Web API

Microsoft