Module "Web"

Submodule " Web API "

Part 3

UA Resource Development Unit

2021

AGENDA

1. Filters. Action and result filters. Exception filters
2. Authentication and Authorization
3. ASP.NET Identity
4. Authentication and Authorization in Web API
5. Authentication Filters in Web API
6. SSL

Filters.
Action and result filters.
Exception filters.

Filters

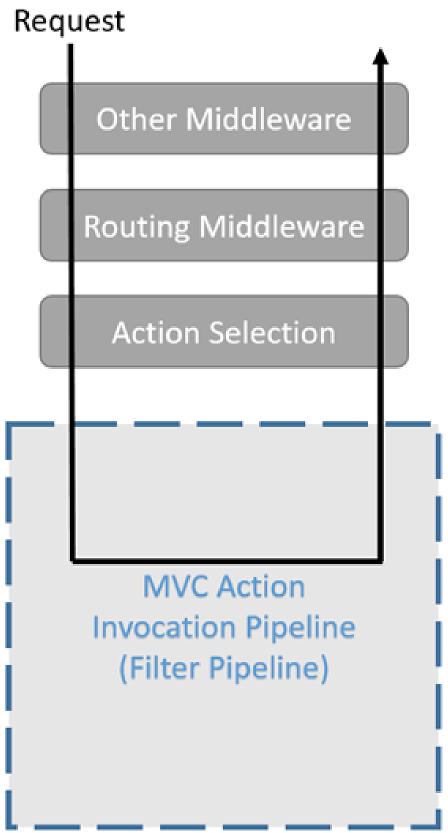
Web API includes filters to add extra logic before or after action method executes. Filters can be used to provide cross-cutting features such as logging, exception handling, performance measurement, authentication and authorization.

Every filter attribute class must implement **IFilter** interface included in ***System.Web.Http.Filters*** namespace.

Filter Type	Interface	Class	Description
Simple Filter	IFilter	-	Defines the methods that are used in a filter
Action Filter	IActionFilter	ActionFilterAttribute	Used to add extra logic before or after action methods execute.
Authentication Filter	IAuthenticationFilter	-	Used to force users or clients to be authenticated before action methods execute.
Authorization Filter	IAuthorizationFilter	AuthorizationFilterAttribute	Used to restrict access to action methods to specific users or groups.
Exception Filter	IExceptionFilter	ExceptionFilterAttribute	Used to handle all unhandled exception in Web API.
Override Filter	IOverrideFilter	-	Used to customize the behaviour of other filter for individual action method.

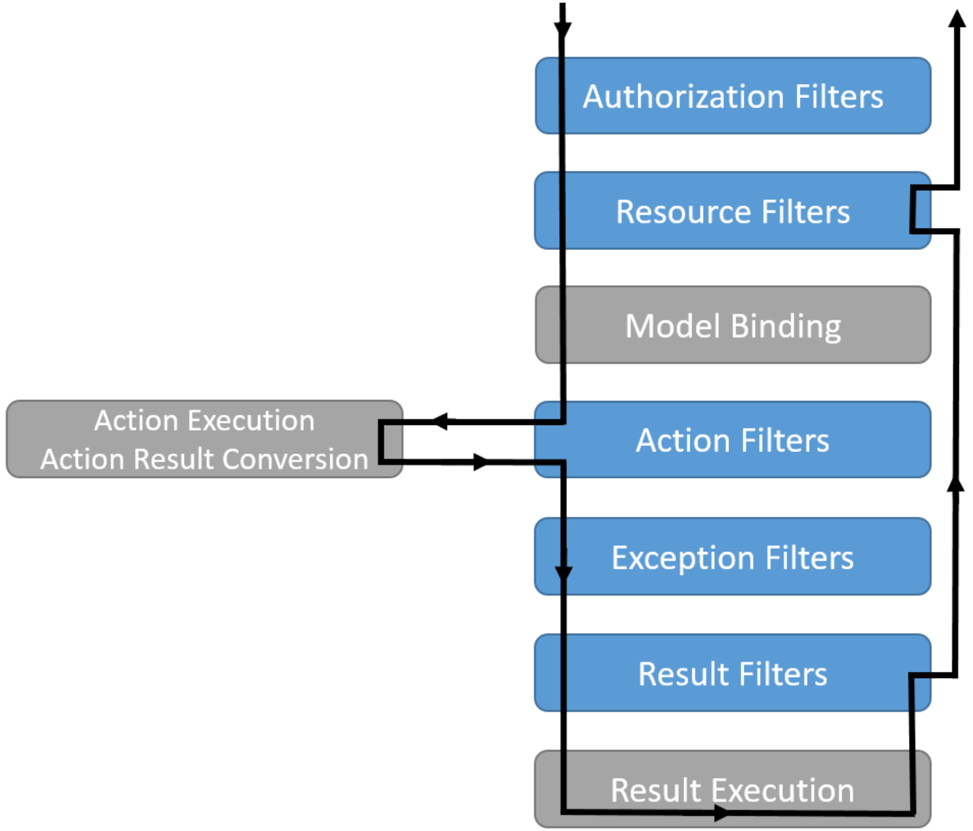
Filters: How filters work

Filters run within the ASP.NET Core action ***invocation pipeline***, sometimes referred to as the ***filter pipeline***.



Filters: Filter types

- **Authorization filters** – They run first to determine whether a user is authorized for the current request
- **Resource filters** – They run right after the authorization filters and are very useful for caching and performance
- **Action filters** – They run right before and after the action method execution
- **Exception filters** – They are used to handle exceptions before the response body is populated
- **Result filters** – They run before and after the execution of the action methods result.



Filters: Filter types

Filter types	Synchronous Interface	Asynchronous Interface
Authorization filters	IAuthorizationFilter	IAsyncAuthorizationFilter
Resource filters	IResourceFilter	IAsyncResourceFilter
Action filters	IActionFilter	IAsyncActionFilter
Exception filters	IExceptionFilter	IAsyncExceptionFilter
Result filters	IResultFilter	IAsyncResultFilter

Filters: Multiple filter stages

Interfaces for multiple filter stages can be implemented in a single class.

- Synchronous
- Asynchronous
- IOrderedFilter

Synchronous:

I[Stage]Filter

- On[Stage]Executing
- On[Stage]Executed

Asynchronous:

IAsync[Stage]Filter

- On[Stage]ExecutionAsync

Filters: Synchronous VS Asynchronous

Synchronous: I[Stage]Filter

- On[Stage]Executing
- On[Stage]Executed

```
using Microsoft.AspNetCore.Mvc.Filters;

namespace FiltersApp.Filters
{
    public class SimpleResourceFilter : IActionFilter
    {
        public void OnActionExecuting(ActionExecutingContext context)
        {
            // ...
        }

        public void OnActionExecuted(ActionExecutedContext context)
        {
            // ...
        }
    }
}
```

Asynchronous: IAsync[Stage]Filter

- On[Stage]ExecutionAsync

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.Filters;

namespace FiltersApp.Filters
{

    public class SimpleAsynActionFilter : IAsyncActionFilter
    {
        public async Task
OnActionExecutionAsync(ActionExecutingContext context,
                         ActionExecutionDelegate next)
        {
            // ...
            await next();
        }
    }
}
```

Filters: Filter scopes and order of execution

A filter can be added to the pipeline at one of three scopes:

- Using an attribute on a controller action. Filter attributes cannot be applied to Razor Pages handler methods.
- Using an attribute on a controller or Razor Page.
- Globally for all controllers, actions, and Razor Pages

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews(options =>
    {
        options.Filters.Add(typeof(MySampleActionFilter));
    });
}
```

```
[ServiceFilter(typeof(MyActionFilterAttribute))]
```

Filters: Default order of execution

The filter sequence:

- The before code of global filters.
    - The before code of controller and Razor Page filters.
        - The before code of action method filters.
        - The after code of action method filters.
    - The after code of controller and Razor Page filters.
- The after code of global filters.

Sequence	Filter scope	Filter method
1	Global	OnActionExecuting
2	Controller or Razor Page	OnActionExecuting
3	Method	OnActionExecuting
4	Method	OnActionExecuted
5	Controller or Razor Page	OnActionExecuted
6	Global	OnActionExecuted

Filters: Controller level filters

Every controller that inherits from the **Controller base** class includes **Controller.OnActionExecuting**, **Controller.OnActionExecutionAsync**, and **Controller.OnActionExecuted OnActionExecuted** methods.

These methods:

- Wrap the filters that run for a given action.
- **OnActionExecuting** is called before any of the action's filters.
- **OnActionExecuted** is called after all of the action filters.
- **OnActionExecutionAsync** is called before any of the action's filters. Code in the filter after next runs after the action method.

Filters: Overriding the default order

The default sequence of execution can be overridden by implementing **IOrderedFilter**. **IOrderedFilter** exposes the **Order** property that takes precedence over scope to determine the order of execution.

A filter with a lower Order value:

- Runs the before code before that of a filter with a higher value of Order.
- Runs the after code after that of a filter with a higher Order value.

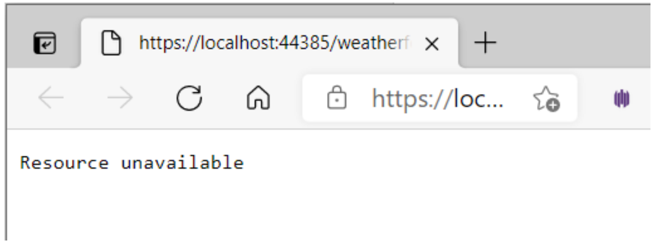
The Order property is set with a constructor parameter:

```
[SampleActionFilter(Order = int.MinValue)]
```

Filters: Cancellation and short-circuiting

```
[FakeNotFoundResourceFilter]
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using System;

namespace FiltersApp.Filters
{
    public class FakeNotFoundResourceFilter : Attribute, IResourceFilter
    {
        public void OnResourceExecuted(ResourceExecutedContext context)
        {

        }

        public void OnResourceExecuting(ResourceExecutingContext context)
        {
            context.Result = new ContentResult { Content = "Resource unavailable" };
        }
    }
}
```



```
Resource unavailable
```

Filters: Action filters

The Action Filters are executed after the Authorization Filters. They are called just before and just after an Action method is called.

They are derived either from the **IActionFilter** or asynchronous **IAsyncActionFilter** interface.

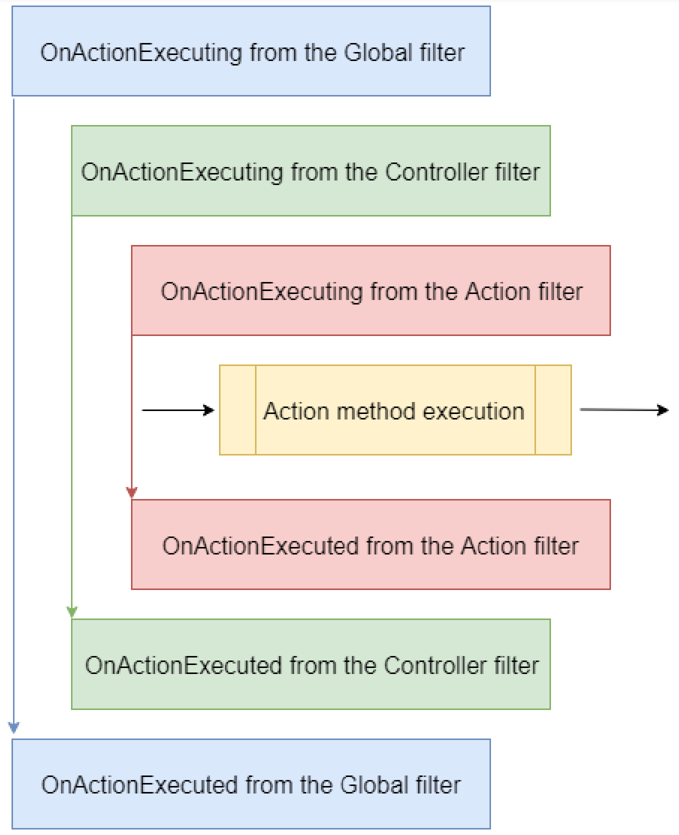
Filters: Action filters

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddControllers(config =>
  {
    config.Filters.Add(new GlobalFilterExample());
  });
}
```

```
services.AddScoped<ActionFilterExample>();
services.AddScoped<ControllerFilterExample>();
```

```
[Route("api/[controller]")]
[ApiController]
public class TestController : ControllerBase
{
    [HttpGet]
    [ServiceFilter(typeof(ActionFilterExample))]
    public IEnumerable<string> Get()
    {
        return new string[] { "example", "data" };
    }
}
```

Filters: Action filters - Order of Invocation


OnActionExecuting from the Global filter
OnActionExecuting from the Controller filter
OnActionExecuting from the Action filter
Action method execution
OnActionExecuted from the Action filter
OnActionExecuted from the Controller filter
OnActionExecuted from the Global filter

Filters: Action filters

```
[ServiceFilter(typeof(ControllerFilterExample), Order = 2)]
[Route("api/[controller]")]
[ApiController]
public class TestController : ControllerBase
{
        [HttpGet]
        [ServiceFilter(typeof(ActionFilterExample), Order = 1)]
        public IEnumerable<string> Get()
        {
            return new string[] { "example", "data" };
        }
}
```

```
[HttpGet]
[ServiceFilter(typeof(ActionFilterExample), Order = 2)]
[ServiceFilter(typeof(ActionFilterExample2), Order = 1)]
public IEnumerable<string> Get()
{
        return new string[] { "example", "data" };
}
```

Filters: Validation with Action Filters

```
[Table("Movie")]
public class Movie : IEntity
{
        [Key]
        public Guid Id { get; set; }
        [Required(ErrorMessage = "Name is required")]
        public string Name { get; set; }
        [Required(ErrorMessage = "Genre is required")]
        public string Genre { get; set; }
        [Required(ErrorMessage = "Director is required")]
        public string Director { get; set; }
}
```

```
if (movie == null)
{
    return BadRequest("Movie object is null");
}
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

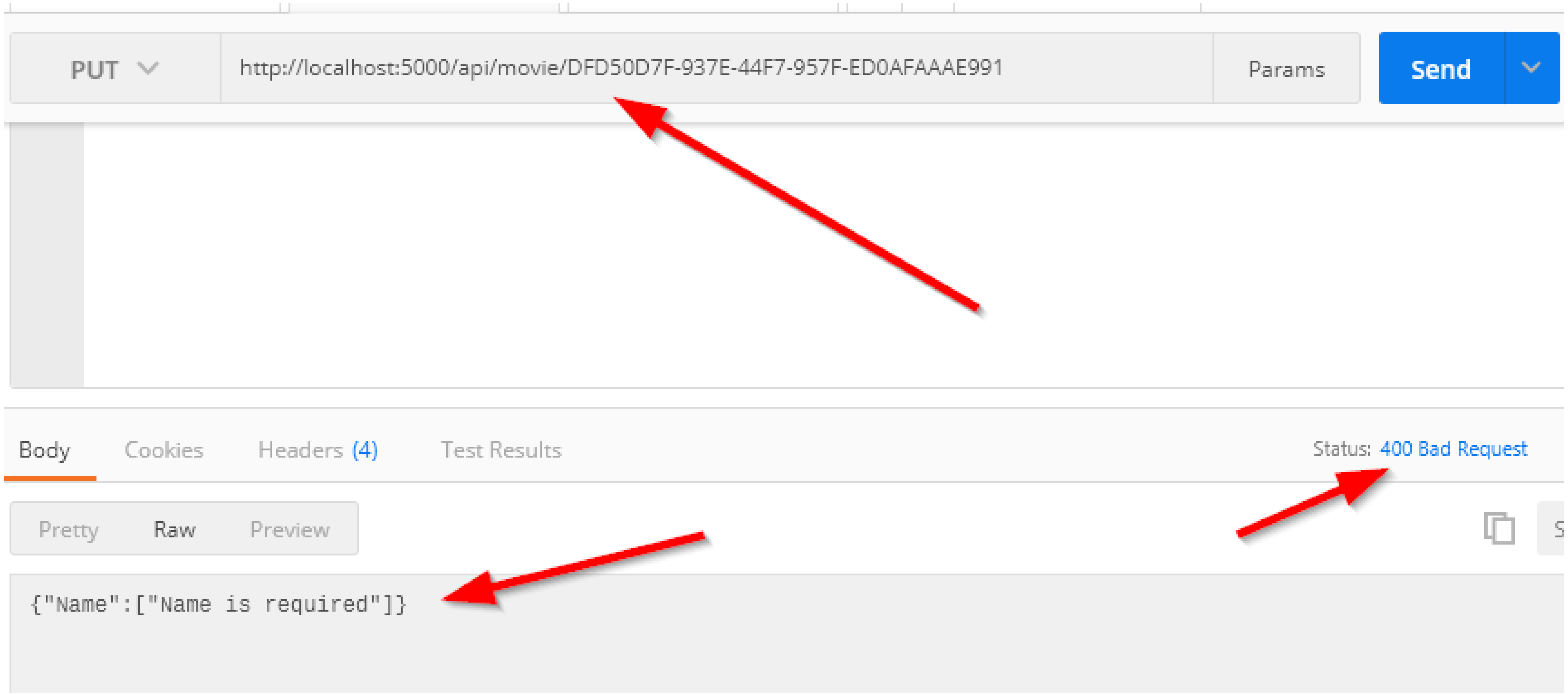
```
public class ValidationFilterAttribute : IActionFilter
{
        public void OnActionExecuting(ActionExecutingContext context)
        {
          var param = context.ActionArguments.SingleOrDefault(p => p.Value is IEntity);
          if (param.Value == null)
          {
            context.Result = new BadRequestObjectResult("Object is null");
            return;
          }

          if (!context.ModelState.IsValid)
          {
            context.Result = new BadRequestObjectResult(context.ModelState);
          }
        }
        public void OnActionExecuted(ActionExecutedContext context)
        {

        }
}
```

Filters: Validation with Action Filters

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<MovieContext>(options =>
        options.UseSqlServer(Configuration
            .GetConnectionString("sqlConString")));
    services.AddScoped<ValidationFilterAttribute>();
    services.AddControllers();
}
```

```
[HttpPost]
[ServiceFilter(typeof(ValidationFilterAttribute))]
public IActionResult Post([FromBody] Movie movie)
{
    _context.Movies.Add(movie);
    _context.SaveChanges();
    return CreatedAtRoute("MovieById", new { id = movie.Id }, movie);
}
[HttpPut("{id}")]
[ServiceFilter(typeof(ValidationFilterAttribute))]
public IActionResult Put(Guid id, [FromBody] Movie movie)
{
    var dbMovie = _context.Movies.SingleOrDefault(x=>x.Id.Equals(id));
    if (dbMovie == null)
    {
        return NotFound();
    }
    dbMovie.Map(movie);
    _context.Movies.Update(dbMovie);
    _context.SaveChanges();
    return NoContent();
}
```

Filters. Action and result filters. Exception filters



```
http://localhost:5000/api/movie/DFD50D7F-937E-44F7-957F-ED0AFAAAE991
```

Status: 400 Bad Request

```
{"Name":["Name is required"]}
```

Filters: Dependency Injection in Action Filters

```
var dbMovie = _context.Movies.SingleOrDefault(x => x.Id.Equals(id));
if (dbMovie == null)
{
    return NotFound();
}
```

```
public class ValidateEntityExistsAttribute<T> : IActionFilter where T : class, IEntity
  {
      private readonly MovieContext _context;
      public ValidateEntityExistsAttribute(MovieContext context)
      {
        _context = context;
      }
      public void OnActionExecuting(ActionExecutingContext context)
      {
        Guid id = Guid.Empty;
        if (context.ActionArguments.ContainsKey("id"))
        {
          id = (Guid)context.ActionArguments["id"];
        }
        else
        {
          context.Result = new BadRequestObjectResult("Bad id parameter");
          return;
        }
        var entity = _context.Set<T>().SingleOrDefault(x => x.Id.Equals(id));
        if (entity == null)
        {
          context.Result = new NotFoundResult();
        }
        else
        {
          context.HttpContext.Items.Add("entity", entity);
        }
      }
      public void OnActionExecuted(ActionExecutedContext context)
      {
      }
  }
```

Filters: Dependency Injection in Action Filters

```
[HttpGet("{id}", Name = "MovieById")]
[ServiceFilter(typeof(ValidateEntityExistsAttribute<Movie>))]
public IActionResult Get(Guid id)
{
    var dbMovie = HttpContext.Items["entity"] as Movie;
    return Ok(dbMovie);
}
[HttpPut("{id}")]
[ServiceFilter(typeof(ValidationFilterAttribute))]
[ServiceFilter(typeof(ValidateEntityExistsAttribute<Movie>))]
public IActionResult Put(Guid id, [FromBody] Movie movie)
{
    var dbMovie = HttpContext.Items["entity"] as Movie;
    dbMovie.Map(movie);
    _context.Movies.Update(dbMovie);
    _context.SaveChanges();
    return NoContent();
}
[HttpDelete("{id}")]
[ServiceFilter(typeof(ValidateEntityExistsAttribute<Movie>))]
public IActionResult Delete(Guid id)
{
    var dbMovie = HttpContext.Items["entity"] as Movie;
    _context.Movies.Remove(dbMovie);
    _context.SaveChanges();
    return NoContent();
}
```

Filters: Result filters

- **Implement an interface:**
  - IResultFilter or IAsyncResultFilter
  - IAlwaysRunResultFilter or IAsyncAlwaysRunResultFilter
- Their execution surrounds the execution of action results.

Result filters are only executed when an action or action filter produces an action result.

Result filters are not executed when:

- An authorization filter or resource filter short-circuits the pipeline.
- An exception filter handles an exception by producing an action result.

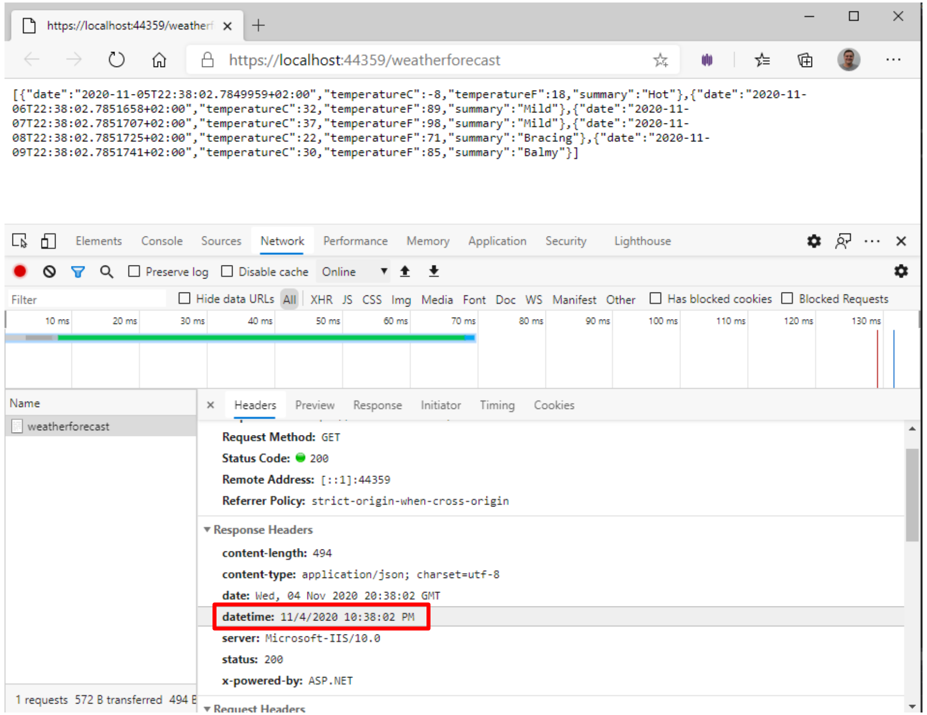
Filters: Result filters - DEMO

```
// Add folder ~/Filters
namespace WebApplicationWebAPIFilters01.Filters
{
   public class DateTimeExecutionFilterAttribute : Attribute, IResultFilter
   {
      public void OnResultExecuting(ResultExecutingContext context)
      {
         context.HttpContext.Response.Headers.Add("DateTime", DateTime.Now.ToString());
         context.Cancel = true;
      }
      public void OnResultExecuted(ResultExecutedContext context)
      {

      }
   }
}
```

Filters: Result filters - DEMO

```
[HttpGet]
[DateTimeExecutionFilter]
public IEnumerable<WeatherForecast> Get()
{
    var rng = new Random();
    return Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = rng.Next(-20, 55),
        Summary = Summaries[rng.Next(Summaries.Length)]
    })
    .ToArray();
}
```

Filters: Result filters - DEMO



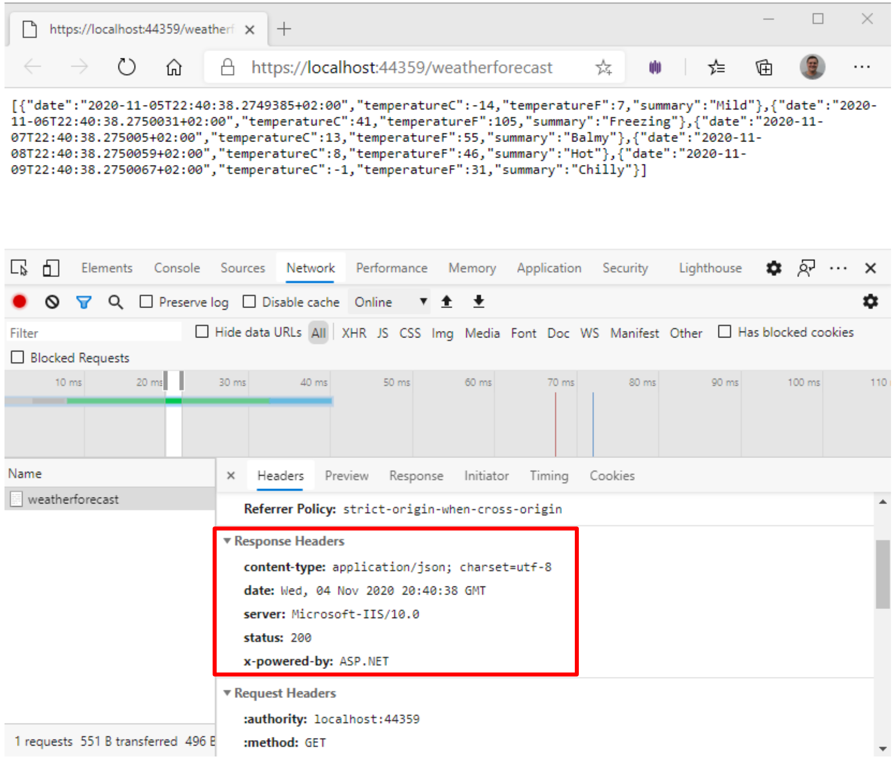
https://localhost:44359/weatherforecast

```
[{"date":"2020-11-05T22:38:02.7849959+02:00","temperatureC":-8,"temperatureF":18,"summary":"Hot"},{"date":"2020-11-
06T22:38:02.7851658+02:00","temperatureC":32,"temperatureF":89,"summary":"Mild"},{"date":"2020-11-
07T22:38:02.7851707+02:00","temperatureC":37,"temperatureF":98,"summary":"Mild"},{"date":"2020-11-
08T22:38:02.7851725+02:00","temperatureC":22,"temperatureF":71,"summary":"Bracing"},{"date":"2020-11-
09T22:38:02.7851741+02:00","temperatureC":30,"temperatureF":85,"summary":"Balmy"}]
```

weatherforecast

```
Request Method: GET
Status Code: 200
Remote Address: [::1]:44359
Referrer Policy: strict-origin-when-cross-origin

▼ Response Headers
content-length: 494
content-type: application/json; charset=utf-8
date: Wed, 04 Nov 2020 20:38:02 GMT
datetime: 11/4/2020 10:38:02 PM
server: Microsoft-IIS/10.0
status: 200
x-powered-by: ASP.NET
▼ Request Headers
```

1 requests 572 B transferred 494 B



https://localhost:44359/weatherforecast

```
[{"date":"2020-11-05T22:40:38.2749385+02:00","temperatureC":-14,"temperatureF":7,"summary":"Mild"},{"date":"2020-
11-06T22:40:38.2750031+02:00","temperatureC":41,"temperatureF":105,"summary":"Freezing"},{"date":"2020-11-
07T22:40:38.275005+02:00","temperatureC":13,"temperatureF":55,"summary":"Balmy"},{"date":"2020-11-
08T22:40:38.2750059+02:00","temperatureC":8,"temperatureF":46,"summary":"Hot"},{"date":"2020-11-
09T22:40:38.2750067+02:00","temperatureC":-1,"temperatureF":31,"summary":"Chilly"}]
```

weatherforecast

```
Referrer Policy: strict-origin-when-cross-origin

▼ Response Headers
content-type: application/json; charset=utf-8
date: Wed, 04 Nov 2020 20:40:38 GMT
server: Microsoft-IIS/10.0
status: 200
x-powered-by: ASP.NET
▼ Request Headers
:authority: localhost:44359
:method: GET
```

1 requests 551 B transferred 496 B

Filters: Result filters - DEMO

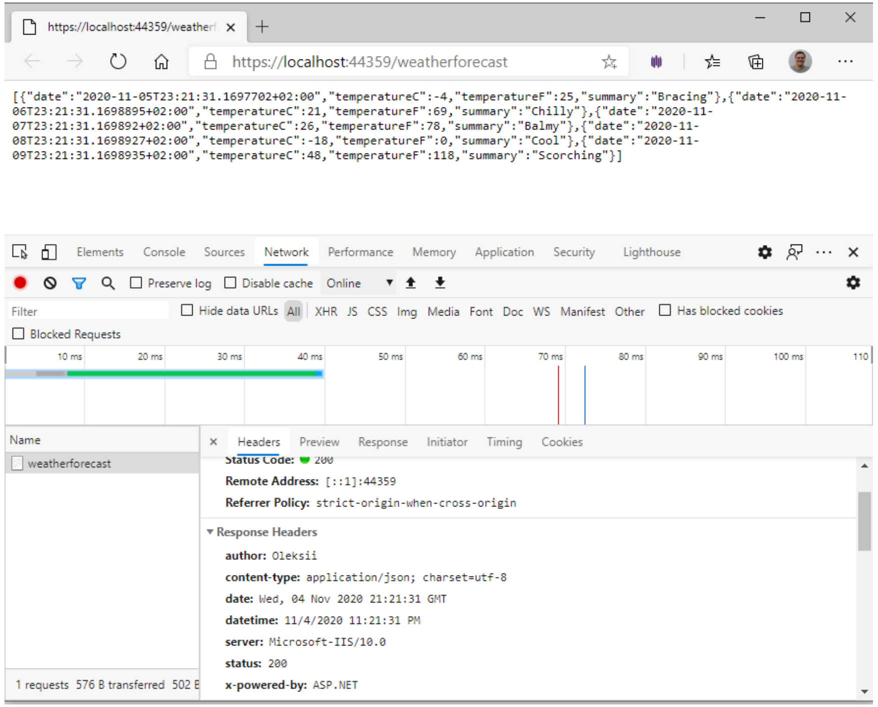
```
public class ResultFilterAttribute : Attribute, IAsyncResultFilter
{
    public async Task OnResultExecutionAsync(ResultExecutingContext context,
                                             ResultExecutionDelegate next)
    {
        context.HttpContext.Response.Headers.Add("DateTime", DateTime.Now.ToString());
        await next();
    }
}
```

```
if (!(context.Result is EmptyResult))
{
    await next();
}
else
{
    context.Cancel = true;
}
```

Filters: Result filters - DEMO

```
public class AddHeaderAttribute : ResultFilterAttribute
{
    private readonly string _name;
    private readonly string _value;

    public AddHeaderAttribute(string name, string value)
    {
        _name = name;
        _value = value;
    }

    public override void OnResultExecuting(ResultExecutingContext context)
    {
        context.HttpContext.Response.Headers.Add(_name, new string[] { _value });
        base.OnResultExecuting(context);
    }
}
```

```
[HttpGet]
[DateTimeExecutionFilter]
[AddHeader("Author", "Oleksii")]
public IEnumerable<WeatherForecast> Get()
{
    // ...
}
```



```
[{"date":"2020-11-05T23:21:31.1697702+02:00","temperatureC":-4,"temperatureF":25,"summary":"Bracing"},{"date":"2020-11-06T23:21:31.1698895+02:00","temperatureC":21,"temperatureF":69,"summary":"Chilly"},{"date":"2020-11-07T23:21:31.169892+02:00","temperatureC":26,"temperatureF":78,"summary":"Balmy"},{"date":"2020-11-08T23:21:31.1698927+02:00","temperatureC":-18,"temperatureF":0,"summary":"Cool"},{"date":"2020-11-09T23:21:31.1698935+02:00","temperatureC":48,"temperatureF":118,"summary":"Scorching"}]
```

```
Status Code: ● 200
Remote Address: [::1]:44359
Referrer Policy: strict-origin-when-cross-origin
▾ Response Headers
author: Oleksii
content-type: application/json; charset=utf-8
date: Wed, 04 Nov 2020 21:21:31 GMT
datetime: 11/4/2020 11:21:31 PM
server: Microsoft-IIS/10.0
status: 200
x-powered-by: ASP.NET
```

Filters: Exception filters

Exception Filters allow catching exceptions without having to write try & catch block. They implement the **IExceptionFilter** or **IAsyncExceptionFilter** interface. The **IAsyncExceptionFilter** interface is used for creating Asynchronous Exception Filters.

For both interfaces, context data is provided through the **ExceptionContext** class, which is a parameter to the methods – **OnException** & **OnExceptionAsync**.

The properties of the **ExceptionContext** class are:

Name	Description
Exception	The property contains the Exceptions that are thrown
ExceptionDispatchInfo	It contains the stack trace details of the exception
ExceptionHandled	A read-only property that tells if the exception is handled
Result	This property sets the IActionResult that will be used to generate the response

Filters: Exception filters

```
public class CustomExceptionFilterAttribute : Attribute, IExceptionFilter
 {
    public void OnException(ExceptionContext context)
    {
       string actionName = context.ActionDescriptor.DisplayName;
       string exceptionStack = context.Exception.StackTrace;
       string exceptionMessage = context.Exception.Message;
       context.Result = new ContentResult
       {
          Content = $"An exception was thrown in the {actionName} method: \n {exceptionMessage} \n {exceptionStack}"
       };
       context.ExceptionHandled = true;
    }
 }
```

```
[CustomExceptionFilter]
```

Filters: Using middleware in the filter pipeline

```
public class LocalizationPipeline
 {
    public void Configure(IApplicationBuilder applicationBuilder)
    {
      var supportedCultures = new[]
      {
      new CultureInfo("en-US"),
      new CultureInfo("fr")
    };

      var options = new RequestLocalizationOptions
      {
        DefaultRequestCulture = new RequestCulture(
                      culture: "en-US",
                      uiCulture: "en-US"),
        SupportedCultures = supportedCultures,
        SupportedUICultures = supportedCultures
      };
      options.RequestCultureProviders = new[]
        { new RouteDataRequestCultureProvider() {
        Options = options } };

      applicationBuilder.UseRequestLocalization(options);
    }
 }
```

Resource filters work like middleware in that they surround the execution of everything that comes later in the pipeline. But filters differ from middleware in that they're part of the runtime, which means that they have access to context and constructs.

```
[HttpGet]
[DateTimeExecutionFilter]
[AddHeader("Author", "Oleksii")]
[MiddlewareFilter(typeof(LocalizationPipeline))]
public IEnumerable<WeatherForecast> Get()
{
    // ...
}
```

Authentication & authorization

Authentication & Authorization: JSON Web Tokens (JWT)

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

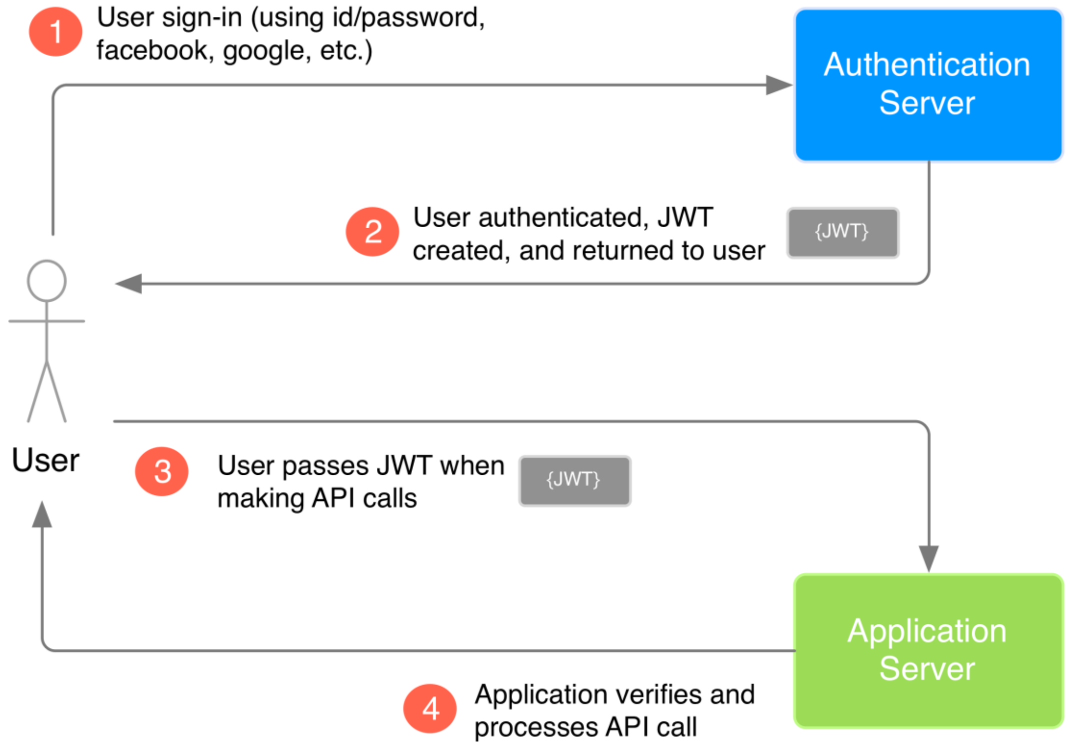
What is the JSON Web Token structure?
In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

- Header
- Payload
- Signature

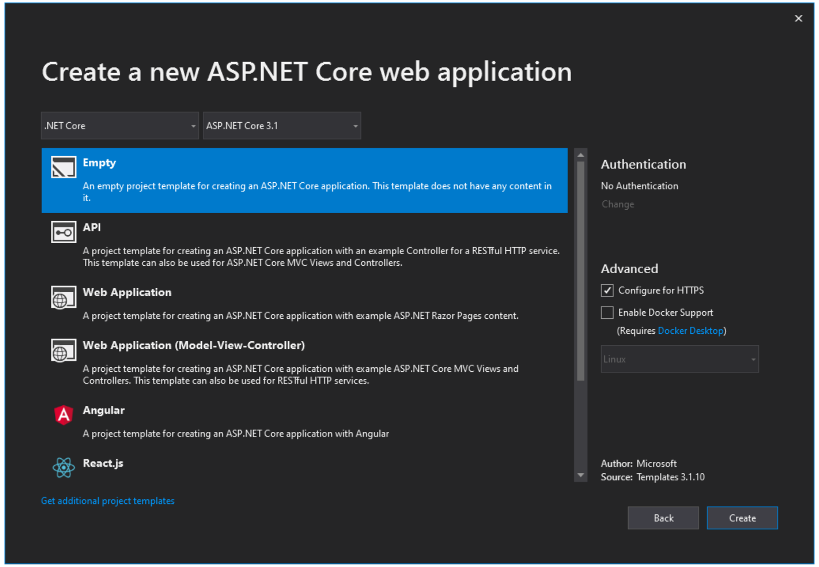
Therefore, a JWT typically looks like the following: ***header.payload.signature***

https://jwt.io/introduction/

Authentication & Authorization: JSON Web Tokens (JWT)


1
User sign-in (using id/password, facebook, google, etc.)
Authentication Server
2
User authenticated, JWT created, and returned to user
{JWT}
User
3
User passes JWT when making API calls
{JWT}
Application Server
4
Application verifies and processes API call

Authentication & authorization: JWT DEMO



Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular
A project template for creating an ASP.NET Core application with Angular

React.js

Get additional project templates

Authentication
No Authentication
Change

Advanced
[✓] Configure for HTTPS
[ ] Enable Docker Support
(Requires Docker Desktop)
Linux

Author: Microsoft
Source: Templates 3.1.10

Back | Create

Install Nuget package: Microsoft.AspNetCore.Authentication.JwtBearer

```
public class AuthOptions
{
    // token publisher
    public const string ISSUER = "WebApplicationwWebAPI_JWT_demoServer";

    // token user
    public const string AUDIENCE = "WebApplicationwWebAPI_JWT_demoClient";

    const string KEY = "mysupersecret_secretkey!123";    // key for encrypting

    public const int LIFETIME = 1; // token  ttl - 1 minute
    public static SymmetricSecurityKey  GetSymmetricSecurityKey()
    {
        return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(KEY));
    }
}
```

```
// ~/Models/Person
  public class Person
  {
     public string Login { get; set; }
     public string Password { get; set; }
     public string Role { get; set; }
  }
```

Authentication & authorization: JWT DEMO

```
public void ConfigureServices(IServiceCollection services)
    {
        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
          .AddJwtBearer(options =>
        {
          options.RequireHttpsMetadata = false;
          options.TokenValidationParameters = new TokenValidationParameters
          {
              ValidateIssuer = true,
              ValidIssuer = AuthOptions.ISSUER,

              ValidateAudience = true,
              ValidAudience = AuthOptions.AUDIENCE,
              ValidateLifetime = true,

              IssuerSigningKey = AuthOptions.GetSymmetricSecurityKey(),
              ValidateIssuerSigningKey = true,
          };
        });
        services.AddControllersWithViews();
    }
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseDefaultFiles();
        app.UseStaticFiles();
        app.UseRouting();

        app.UseAuthentication();
        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        });

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapDefaultControllerRoute();
        });
    }
```

Authentication & authorization: JWT DEMO

```
public class AccountController : Controller
{
        private List<Person> people = new List<Person>
        {
                new Person {Login="admin@gmail.com", Password="12345", Role = "admin" },
                new Person { Login="user1@gmail.com", Password="54321", Role = "user" }
        };

        [HttpPost("/token")]
        public IActionResult Token(string username, string password)
        {
                var identity = GetIdentity(username, password);
                if (identity == null)
                {
                        return BadRequest(new { errorText = "Invalid username or password." });
                }
                var now = DateTime.UtcNow;
                var jwt = new JwtSecurityToken(
                        issuer: AuthOptions.ISSUER,
                        audience: AuthOptions.AUDIENCE,
                        notBefore: now,
                        claims: identity.Claims,
                        expires: now.Add(TimeSpan.FromMinutes(AuthOptions.LIFETIME)),
                        signingCredentials: new SigningCredentials(AuthOptions.GetSymmetricSecurityKey(),
                        SecurityAlgorithms.HmacSha256));
                var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);
                var response = new
                {
                        access_token = encodedJwt,
                        username = identity.Name
                };
                return Json(response);
        }
```

```
private ClaimsIdentity GetIdentity(string username, string password)
{
        Person person = people.FirstOrDefault(x => x.Login == username && x.Password == password);
        if (person != null)
        {
                var claims = new List<Claim>
                {
                        new Claim(ClaimsIdentity.DefaultNameClaimType, person.Login),
                        new Claim(ClaimsIdentity.DefaultRoleClaimType, person.Role)
                };
                ClaimsIdentity claimsIdentity =
                        new ClaimsIdentity(claims, "Token", ClaimsIdentity.DefaultNameClaimType,
                ClaimsIdentity.DefaultRoleClaimType);
                return claimsIdentity;
        }
        // if user not found
        return null;
}
```

```
}
```

Authentication & authorization: JWT DEMO

```
[ApiController]
[Route("api/[controller]")]
public class ValuesController : Controller
{
    [Authorize]
    [Route("getlogin")]
    public IActionResult GetLogin()
    {
        return Ok($"Your login: {User.Identity.Name}");
    }

    [Authorize(Roles = "admin")]
    [Route("getrole")]
    public IActionResult GetRole()
    {
        return Ok("Your role is: administrator");
    }
}
```

Authentication & authorization: JWT DEMO

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>JWT в ASP.NET Core Web API</title>
</head>
<body>
  <div id="userInfo" style="display:none;">
    <p>You have entered as: <span id="userName"></span></p>
    <input type="button" value="Exit" id="logOut" />
  </div>
  <div id="loginForm">
    <h3>Login on site</h3>
    <label>Enter email</label><br />
    <input type="email" id="emailLogin" /> <br /><br />
    <label>Enter password</label><br />
    <input type="password" id="passwordLogin" /><br /><br />
    <input type="submit" id="submitLogin" value="Login" />
  </div>
  <div>
    <input type="submit" id="getDataByLogin" value="Data about the login"
/>
  </div>
  <div>
    <input type="submit" id="getDataByRole" value="Data about the role" />
  </div>

  <script>
    var tokenKey = "accessToken";

    // send request to controller AccountController for getting the token
    async function getTokenAsync() {
```

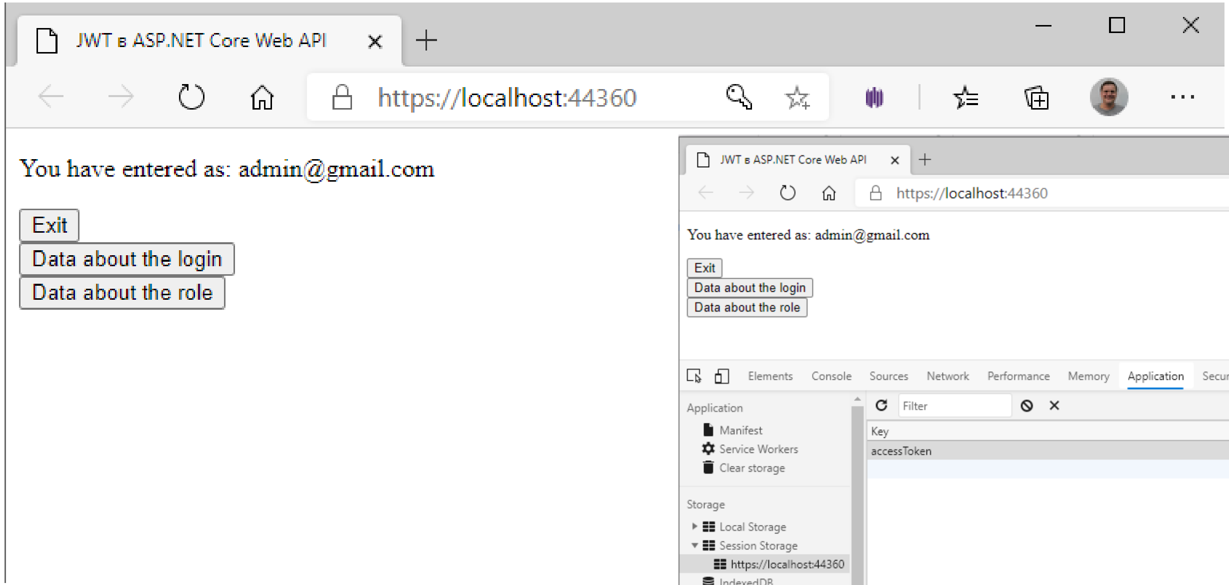
```
// get forms data and make object for sending
const formData = new FormData();
formData.append("grant_type", "password");
formData.append("username", document.getElementById("emailLogin").value);
formData.append("password", document.getElementById("passwordLogin").value);

// send request and get response
const response = await fetch("/token", {
  method: "POST",
  headers: {"Accept": "application/json"},
  body: formData
});
// get data
const data = await response.json();

// if request OK
if (response.ok === true) {
  // change bloks content and visialisation in the page
  document.getElementById("userName").innerText = data.username;
  document.getElementById("userInfo").style.display = "block";
  document.getElementById("loginForm").style.display = "none";
  // save sessionStorage access token in storage
  sessionStorage.setItem(tokenKey, data.access_token);
  console.log(data.access_token);
 }
else {
  // if error is presented, get error message fron errorText
  console.log("Error: ", response.status, data.errorText);
}
};
```

Authentication & authorization: JWT DEMO

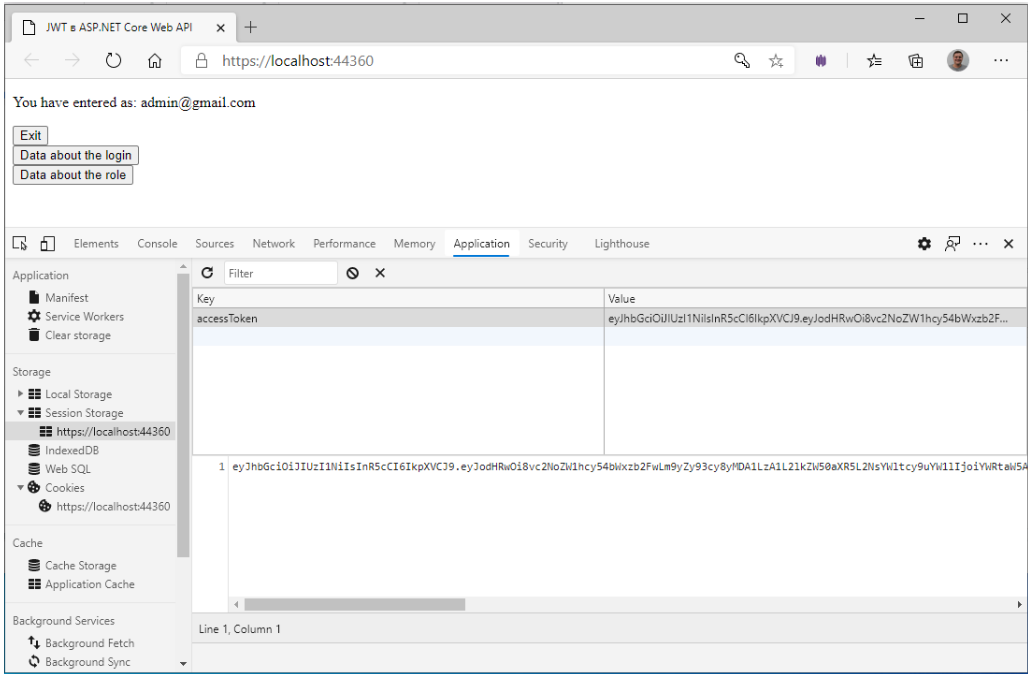
```
// send response to ValuesController
    async function getData(url) {
      const token = sessionStorage.getItem(tokenKey);

      const response = await fetch(url, {
        method: "GET",
        headers: {
          "Accept": "application/json",
          "Authorization": "Bearer " + token  // send token in header
        }
      });
      if (response.ok === true) {

        const data = await response.json();
        alert(data)
      }
      else
        console.log("Status: ", response.status);
    };

    // get token
    document.getElementById("submitLogin").addEventListener("click", e => {
      e.preventDefault();
      getTokenAsync();
    });
    // condition's exit - delete token and change blocks visibility
    document.getElementById("logOut").addEventListener("click", e => {
      e.preventDefault();
      document.getElementById("userName").innerText = "";
      document.getElementById("userInfo").style.display = "none";
      document.getElementById("loginForm").style.display = "block";
      sessionStorage.removeItem(tokenKey);
    });
```

```
    // button getting user name - /api/values/getlogin
    document.getElementById("getDataByLogin").addEventListener("click", e => {
      e.preventDefault();
      getData("/api/values/getlogin");
    });

    // button getting user role - /api/values/getrole
    document.getElementById("getDataByRole").addEventListener("click", e => {
      e.preventDefault();
      getData("/api/values/getrole");
    });
  </script>
</body>
</html>
```

Authentication & authorization



You have entered as: admin@gmail.com



You have entered as: admin@gmail.com

Key	Value
accessToken	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2F...

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiYWRtaW5A
```

Line 1, Column 1

ASP.NET Identity

ASP.NET Identity

ASP.NET Identity

Create new project based on template for .NET Core Web Application (API), then you will get a WeatherForcast model and WeatherForecastController.

Add NuGet Packages

Please make sure you add below NuGet packages to the Web API Project:

- ➢ Microsoft.AspNetCore.Identity.EntityFrameworkCore
- ➢ Microsoft.EntityFrameworkCore
- ➢ Microsoft.EntityFrameworkCore.Design
- ➢ Microsoft.EntityFrameworkCore.SqlServer
- ➢ Microsoft.EntityFrameworkCore.Tools

ASP.NET Identity

IdentityDbContext

The important classes are:

- **IdentityDbContext**, represents the DbContext for Identity. It has definitions for all the tables required to enable ASP .NET Core Identity.
- **IdentityUser**, which represents a user in Identity database
- **IdentityRole**, which represents a role in Identity database

```
public class ApplicationDbContext : IdentityDbContext<IdentityUser>
{
    public ApplicationDbContext(DbContextOptions options) : base(options)
    {
    }
}
```

ASP.NET Identity

Configure Identity

AddIdentity(IServiceCollection) adds the default identity system configuration for the specified User and Role types.

```
services.AddDbContext<ApplicationDbContext>(options =>
     options.UseSqlServer(Configuration.GetConnectionString("SqlCo
     nnection")));

services.AddIdentity<IdentityUser, IdentityRole>(options =>
     options.SignIn.RequireConfirmedAccount =
     true).AddEntityFrameworkStores<ApplicationDbContext>();
```

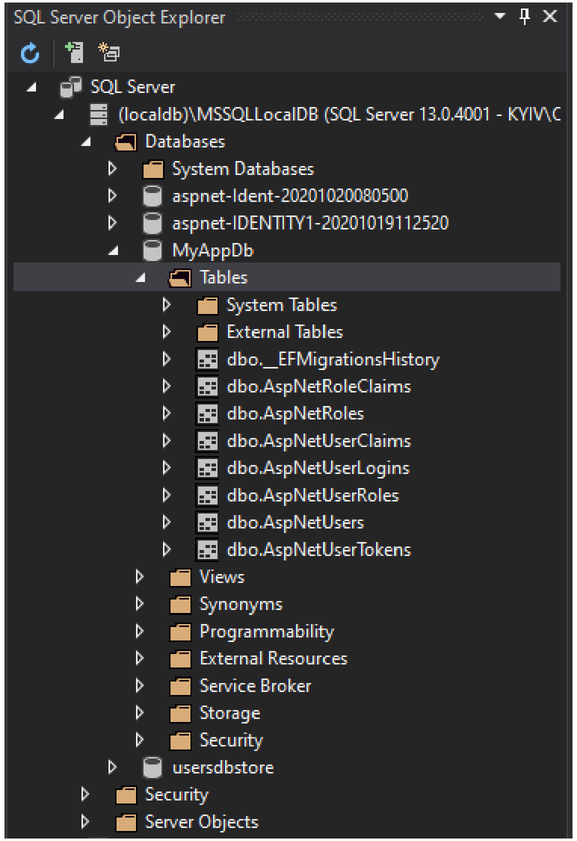
ASP.NET Identity

```
{
 "Logging": {
  "LogLevel": {
   "Default": "Information",
   "Microsoft": "Warning",
   "Microsoft.Hosting.Lifetime": "Information"
  }
 },
 "ConnectionStrings": {
  "SqlConnection": "Server=(localdb)\\mssqllocaldb; Initial Catalog=MyAppDb; Integrated Security=true;"
 },
 "AllowedHosts": "*"
}
```

ASP.NET Identity

Migrations and Create Database

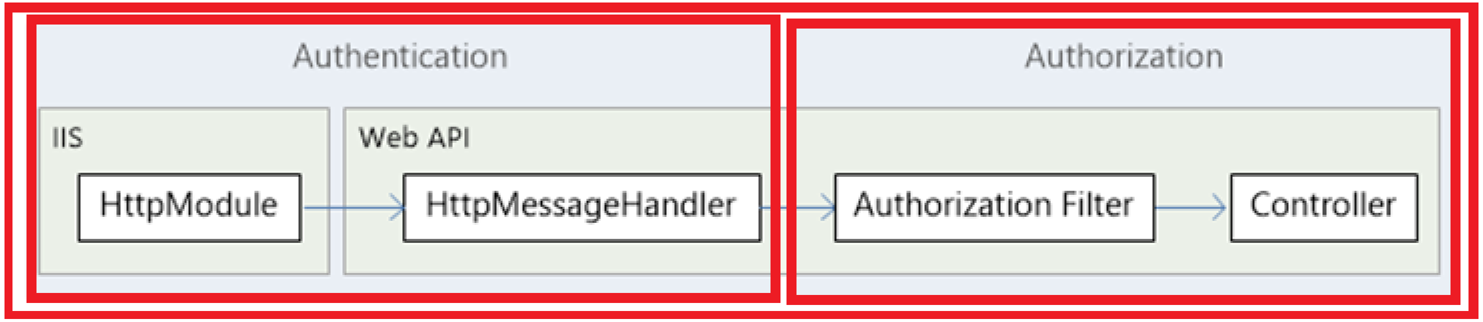
```
dotnet tool install --global dotnet-ef

dotnet-ef migrations add First --project CookieAuthSampleAPI

dotnet-ef database update --project CookieAuthSampleAPI
```



```
SQL Server Object Explorer
SQL Server
  (localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - KYIV\C
    Databases
      System Databases
      aspnet-Ident-20201020080500
      aspnet-IDENTITY1-20201019112520
      MyAppDb
        Tables
          System Tables
          External Tables
          dbo.__EFMigrationsHistory
          dbo.AspNetRoleClaims
          dbo.AspNetRoles
          dbo.AspNetUserClaims
          dbo.AspNetUserLogins
          dbo.AspNetUserRoles
          dbo.AspNetUsers
          dbo.AspNetUserTokens
        Views
        Synonyms
        Programmability
        External Resources
        Service Broker
        Storage
        Security
      usersdbstore
    Security
    Server Objects
```

Authentication and Authorization in Web API

Authentication and Authorization in Web API



Authentication and Authorization in Web API

Authentication Filters in Web API

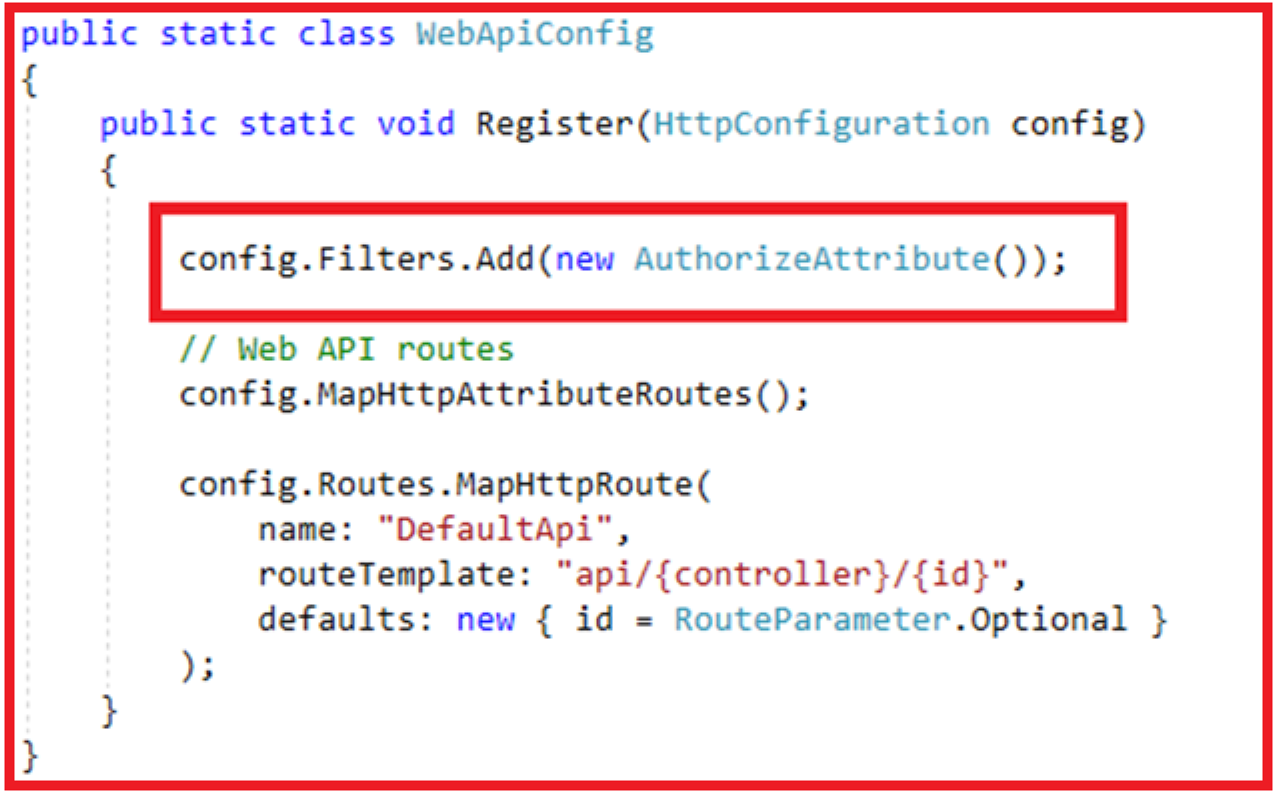
Authentication Filters in Web API

Using the [Authorize] Attribute

The ASP.NET Web API Framework provides a built-in authorization filter attribute i.e. AuthorizeAttribute and you can use this built-in filter attribute to checks whether the user is authenticated or not. If not, then it simply returns the HTTP status code 401 Unauthorized, without invoking the controller action method.

Authentication Filters in Web API

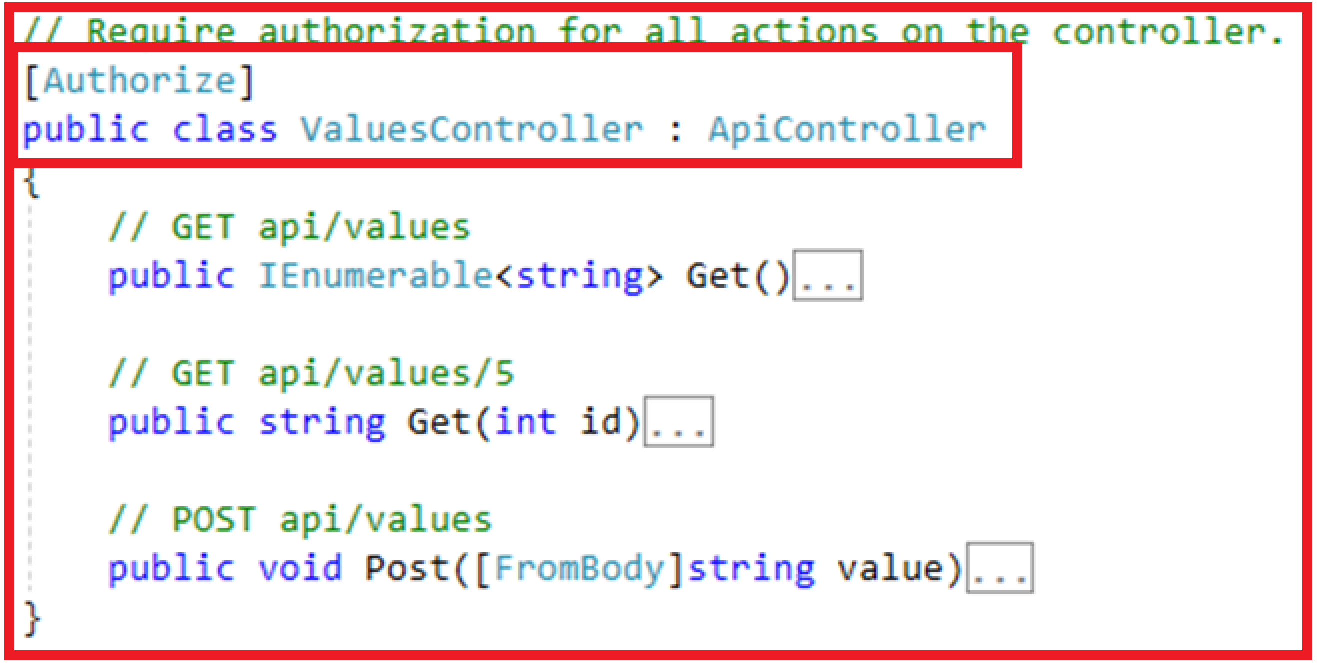
At Globally:



```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {

        config.Filters.Add(new AuthorizeAttribute());

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

Authentication Filters in Web API

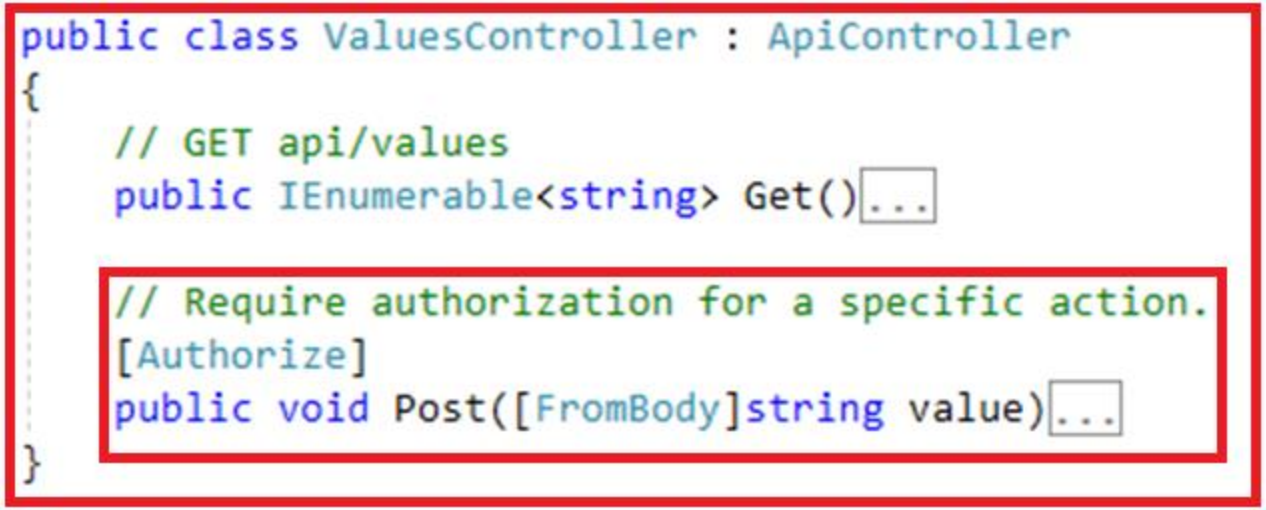
At Controller Level:



```
// Require authorization for all actions on the controller.
[Authorize]
public class ValuesController : ApiController
{
    // GET api/values
    public IEnumerable<string> Get()...

    // GET api/values/5
    public string Get(int id)...

    // POST api/values
    public void Post([FromBody]string value)...
}
```

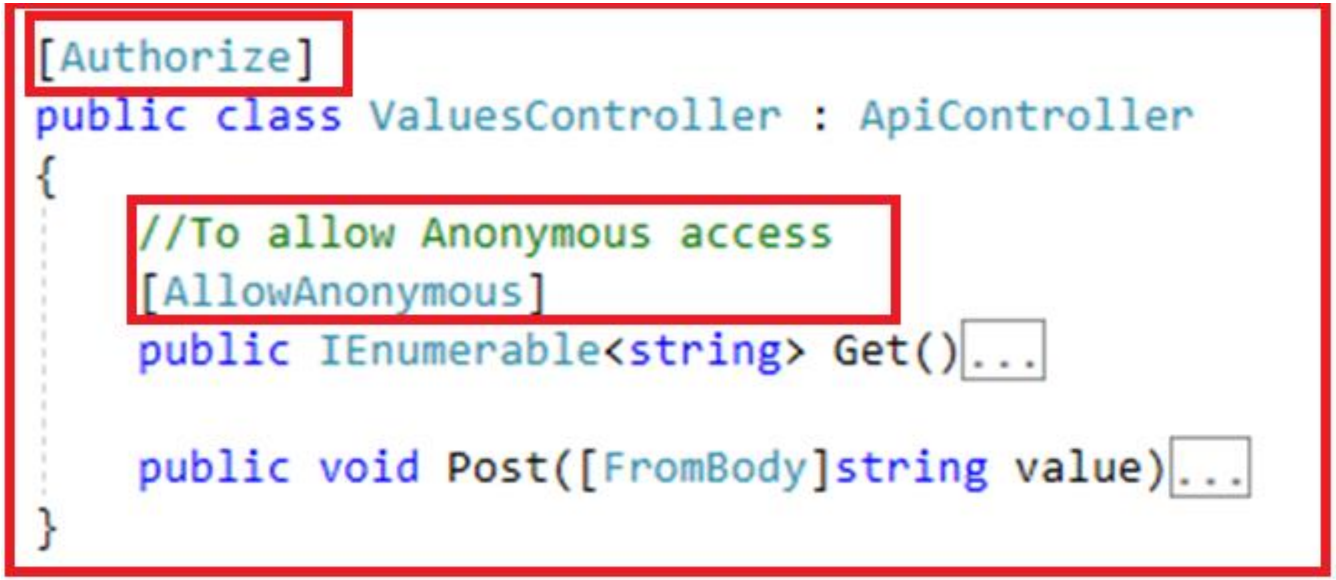
Authentication Filters in Web API

At Action Level:



```
public class ValuesController : ApiController
{
    // GET api/values
    public IEnumerable<string> Get()...

    // Require authorization for a specific action.
    [Authorize]
    public void Post([FromBody]string value)...
}
```

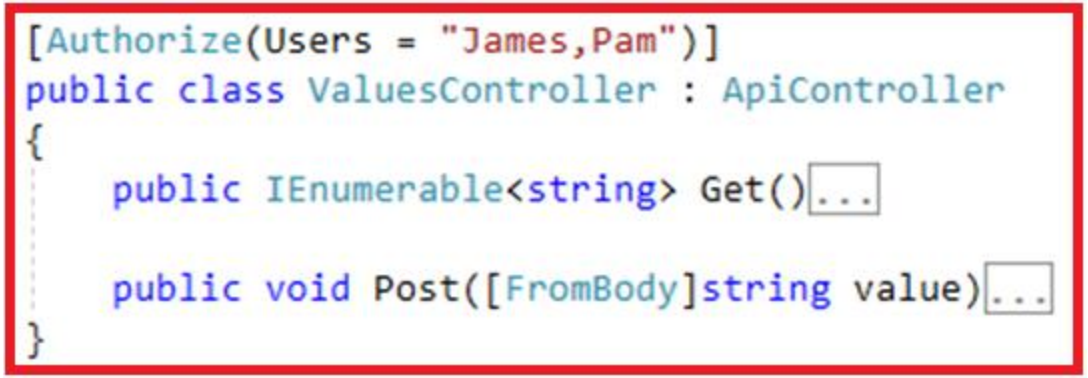
Authentication Filters in Web API



```
[Authorize]
public class ValuesController : ApiController
{
    //To allow Anonymous access
    [AllowAnonymous]
    public IEnumerable<string> Get()...

    public void Post([FromBody]string value)...
}
```

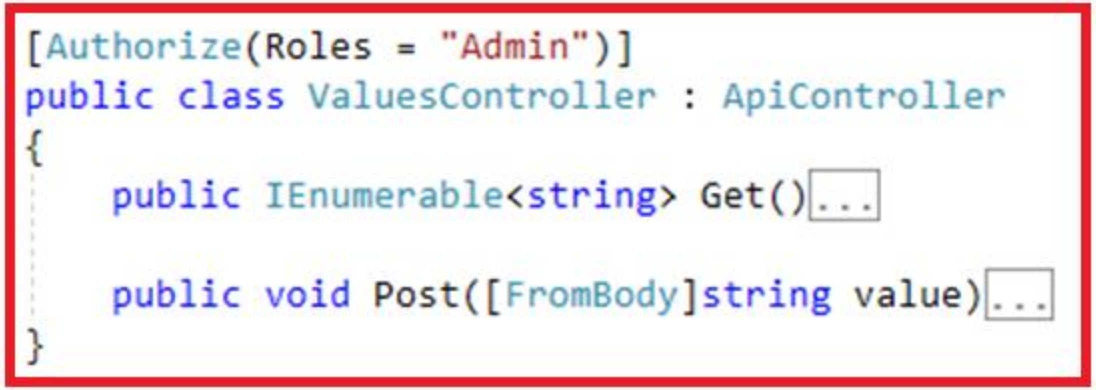
Authentication Filters in Web API

Restrict by Users



```
[Authorize(Users = "James,Pam")]
public class ValuesController : ApiController
{
    public IEnumerable<string> Get()...

    public void Post([FromBody]string value)...
}
```

Restrict by Roles:



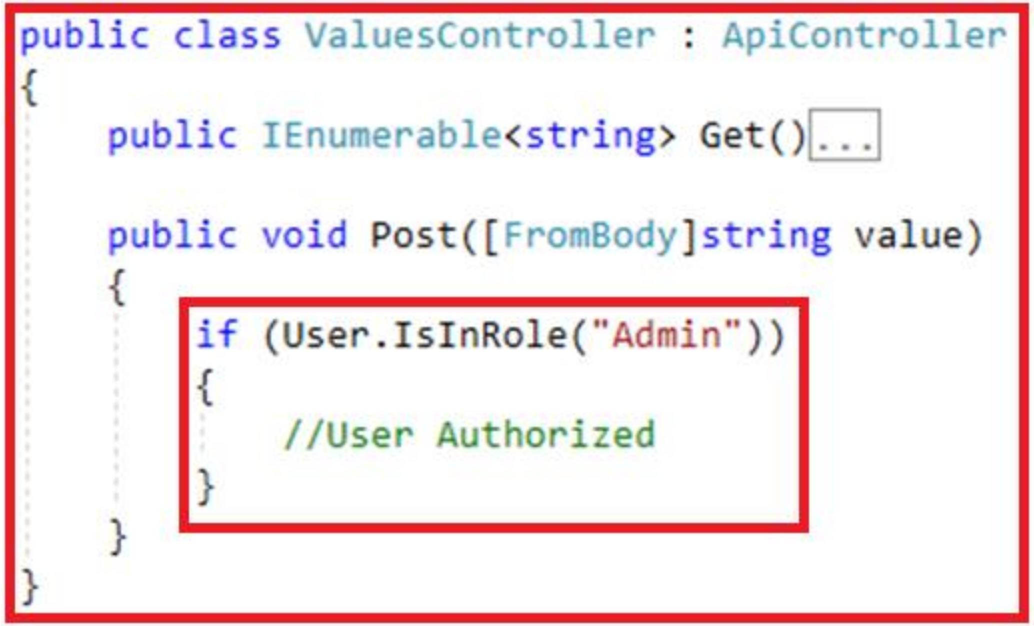
```
[Authorize(Roles = "Admin")]
public class ValuesController : ApiController
{
    public IEnumerable<string> Get()...

    public void Post([FromBody]string value)...
}
```

System.Web.Http

~~System.Web.Mvc~~

Authentication Filters in Web API

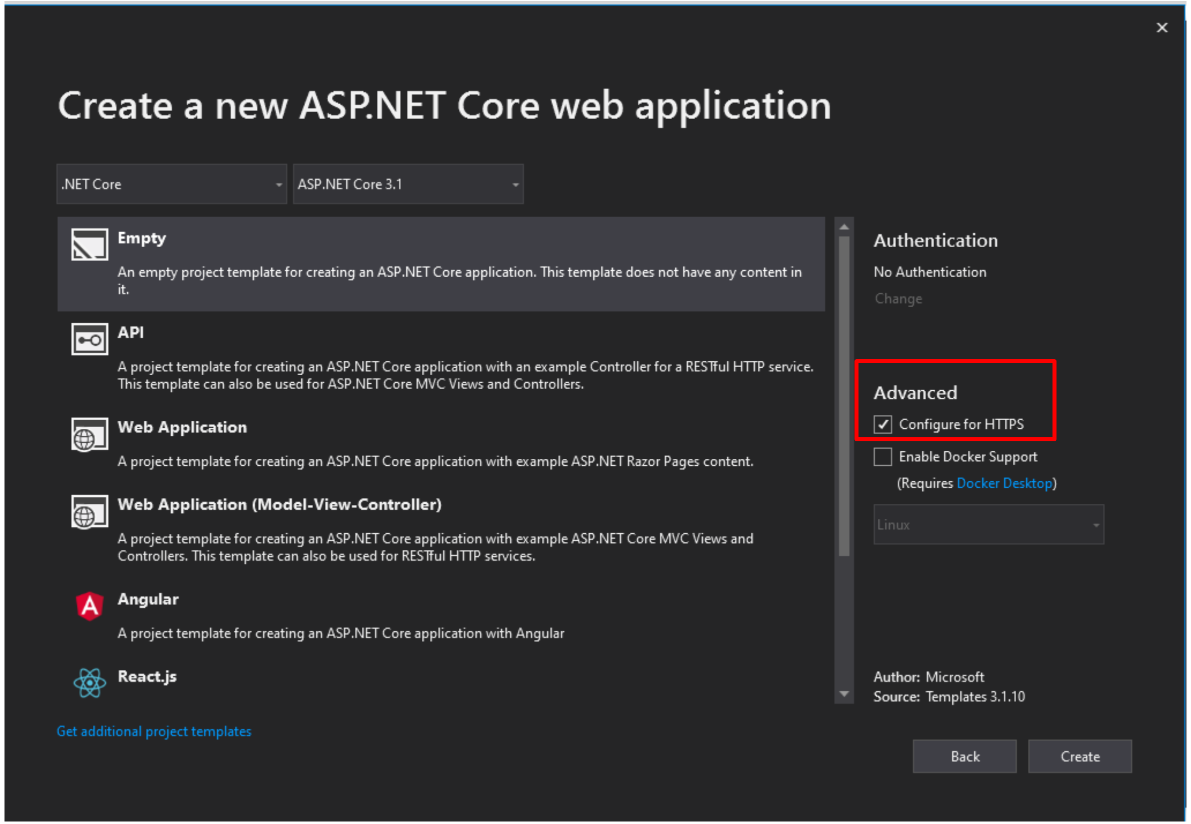
Authorization Inside a Controller Action



```
public class ValuesController : ApiController
{
    public IEnumerable<string> Get()...

    public void Post([FromBody]string value)
    {
        if (User.IsInRole("Admin"))
        {
            //User Authorized
        }
    }
}
```

SSL

SSL



×

Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular
A project template for creating an ASP.NET Core application with Angular

React.js

Get additional project templates

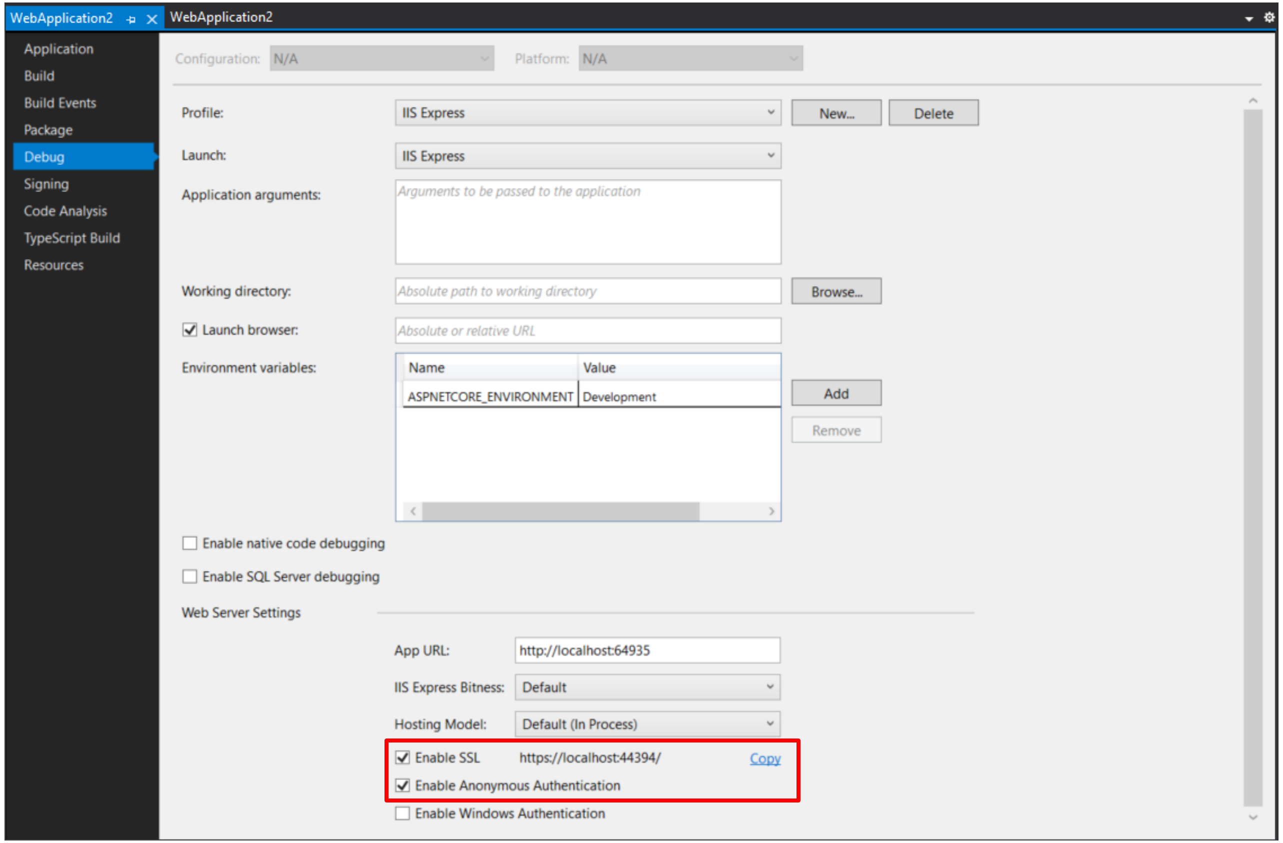
Authentication
No Authentication
Change

Advanced
☑ Configure for HTTPS
☐ Enable Docker Support
(Requires Docker Desktop)
Linux

Author: Microsoft
Source: Templates 3.1.10

Back | Create

SSL



Configuration: N/A   Platform: N/A

Profile: IIS Express

Launch: IIS Express

Application arguments: Arguments to be passed to the application

Working directory: Absolute path to working directory

☑ Launch browser: Absolute or relative URL

Environment variables:

Name	Value
ASPNETCORE_ENVIRONMENT	Development

☐ Enable native code debugging

☐ Enable SQL Server debugging

Web Server Settings

App URL: http://localhost:64935

IIS Express Bitness: Default

Hosting Model: Default (In Process)

☑ Enable SSL https://localhost:44394/

☑ Enable Anonymous Authentication

☐ Enable Windows Authentication

SSL - UseHttpsRedirection

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        });
    }
```

SSL - AddHttpsRedirection()

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddHttpsRedirection(options =>
        {
           options.RedirectStatusCode = StatusCodes.Status307TemporaryRedirect;
           options.HttpsPort = 44344;
        });
}
```

SSL - HTTP Strict Transport Security Protocol (HSTS)

Example of Strict-Transport-Security header:

Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

SSL - HTTP Strict Transport Security Protocol (HSTS)

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseHsts();
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapGet("/", async context =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        });
    }
```

SSL - HTTP Strict Transport Security Protocol (HSTS)

```
public void ConfigureServices(IServiceCollection services)
    {
        services.AddHsts(options =>
        {
            options.Preload = true;
            options.IncludeSubDomains = true;
            options.MaxAge = TimeSpan.FromDays(60);
            options.ExcludedHosts.Add("us.example.com");
            options.ExcludedHosts.Add("www.example.com");
        });
    }
```

.NET Online UA Training Course Feedback

I hope that you will find this material useful.

If you find errors or inaccuracies in this material or know how to improve it, please report on to the electronic address:

Oleksii_Leunenko@epam.com

With the note [.NET Online UA Training Course Feedback]

Thank you.

Q&A

<epam>

DRIVEN CANDID CREATIVE ORIGINAL INTELLIGENT EXPERT

UA .NET Online LAB