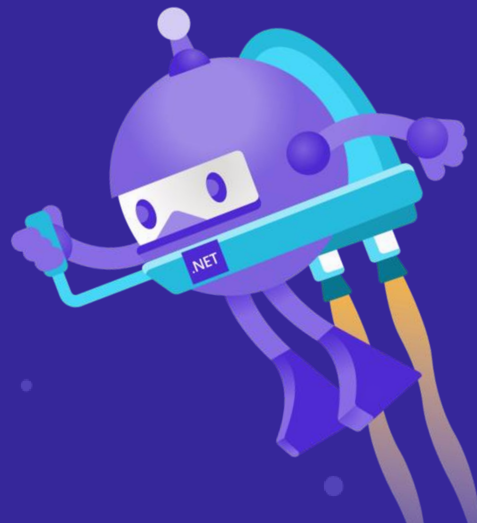
Day 14 : Securing Web API



Authentication

- Is the process of determining the user’s identity

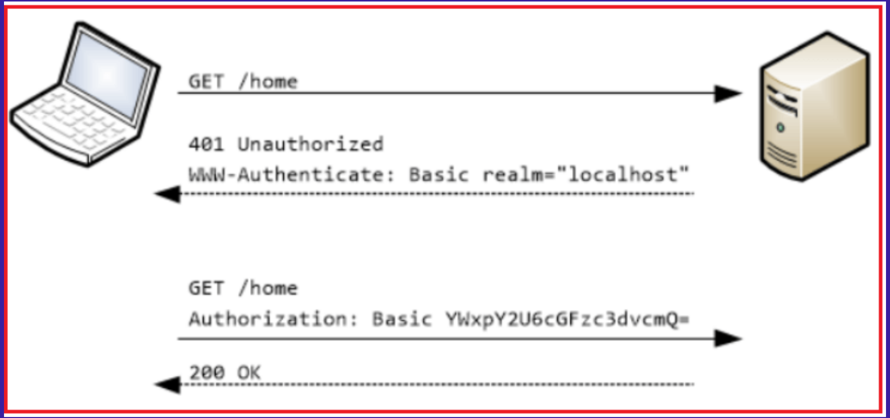
- In ASP.NET Core, authentication is handled by *IAuthenticationService*

- Authentication Service uses registered handlers to complete authentication related actions such as
  - Authenticating the user
  - Responding when an unauthenticated user tries to access a restricted resource

- Registered Authentication handlers and their configuration options are called ***Schemes***

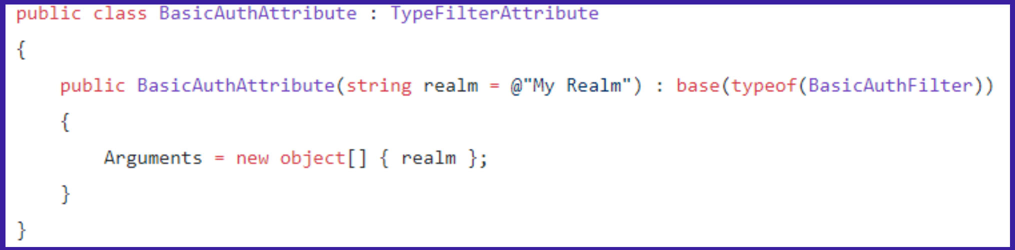
Basic Authentication

- Simplest technique for enforcing access
- Built into HTTP Protocol
- Allows browsers or other agents to request access using credentials consisting of username and password
- Doesn't require cookies, session identifiers or login pages
- Authentication scheme checks the Authorization header in HTTP requests

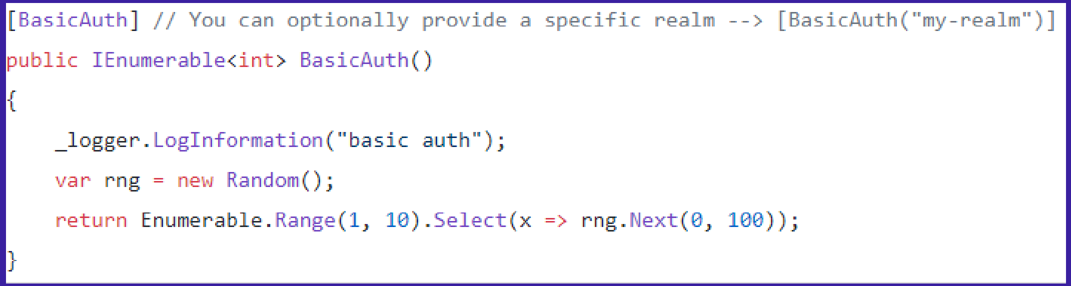

GET /home
401 Unauthorized
WWW-Authenticate: Basic realm="localhost"
GET /home
Authorization: Basic YWxpY2U6cGFzc3dvcmQ=
200 OK

Implementing Basic Authentication

- Can be implemented using
  - Middleware
  - Attribute



```
public class BasicAuthAttribute : TypeFilterAttribute
{
    public BasicAuthAttribute(string realm = @"My Realm") : base(typeof(BasicAuthFilter))
    {
        Arguments = new object[] { realm };
    }
}
```



```
[BasicAuth] // You can optionally provide a specific realm --> [BasicAuth("my-realm")]
public IEnumerable<int> BasicAuth()
{
    _logger.LogInformation("basic auth");
    var rng = new Random();
    return Enumerable.Range(1, 10).Select(x => rng.Next(0, 100));
}
```

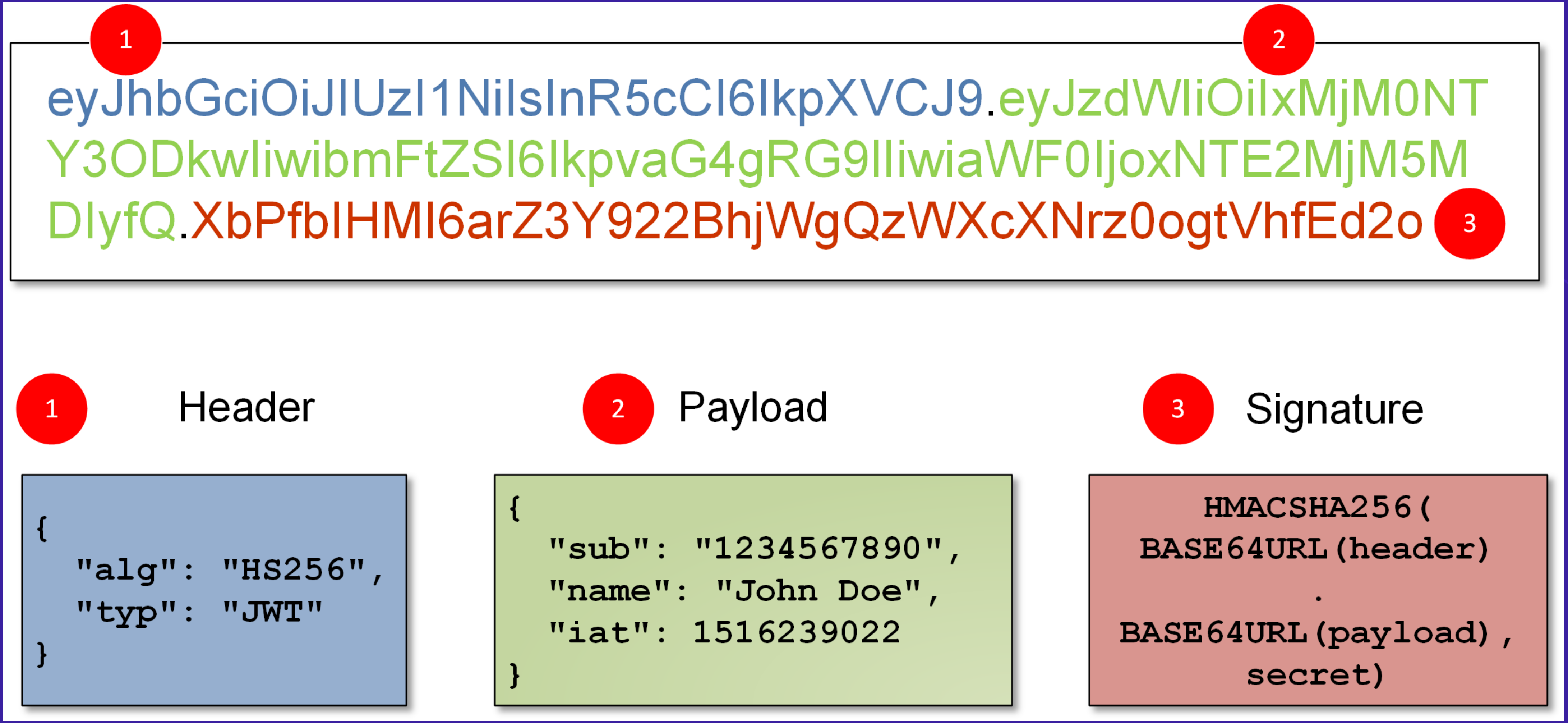
Bearer Authentication

- Is an HTTP authentication scheme that involves security tokens called ***Bearer Tokens***
- Is also known as token authentication
- Bearer token is a cryptic string usually generated by the server in response to the login request
- Client must send this token in the authorization header in the format
  - `Authorization: Bearer <token>`

JWT (JSON Web Token)

- Is an open standard, self defined and compact mechanism for securely transmitting information using JSON Object
- The information can be verified and trusted because its digitally signed
- Can be signed using a secret or a public-private key pair
- Usage Scenarios
    - Authorization
    - Information Exchange

JWT Structure



1 2

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NT
Y3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5M
DIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

3

1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

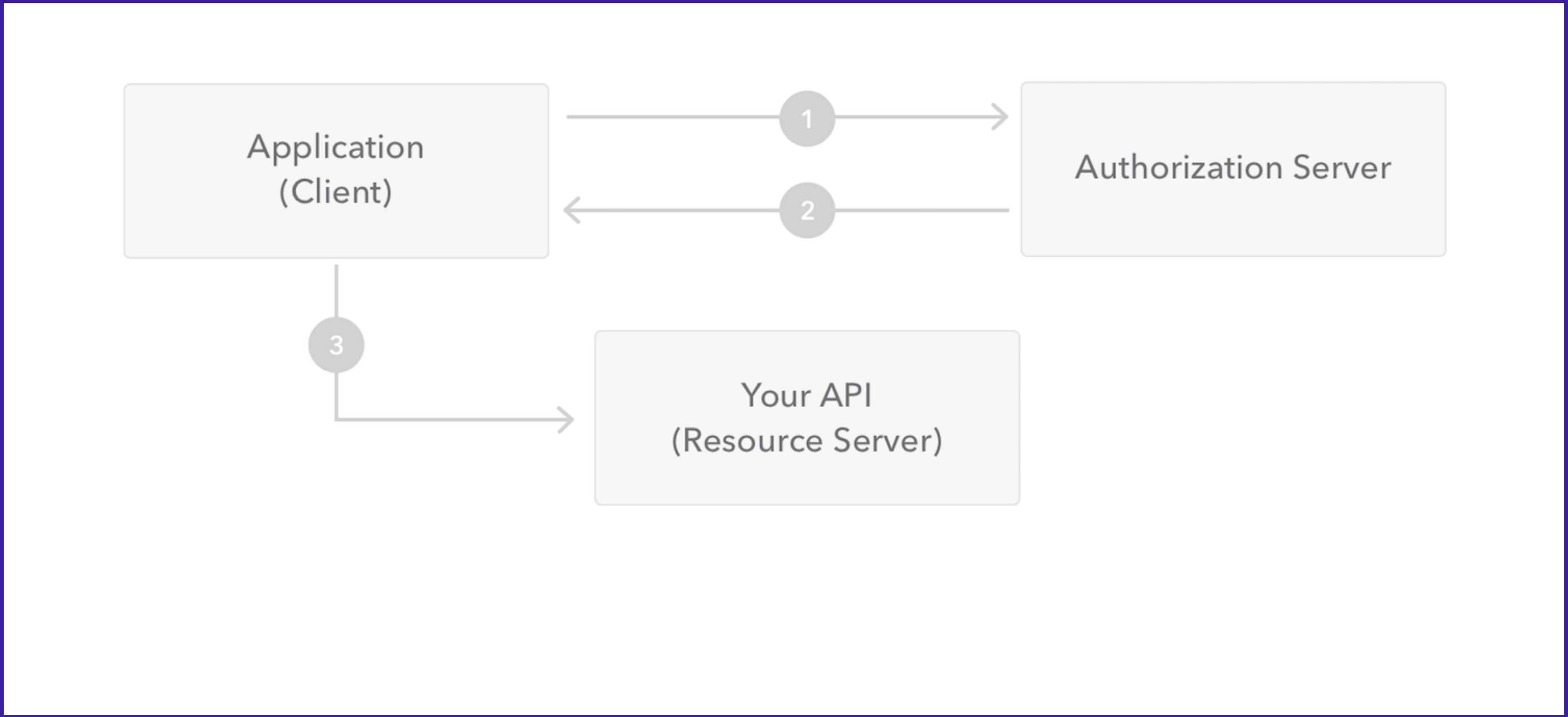
2 Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

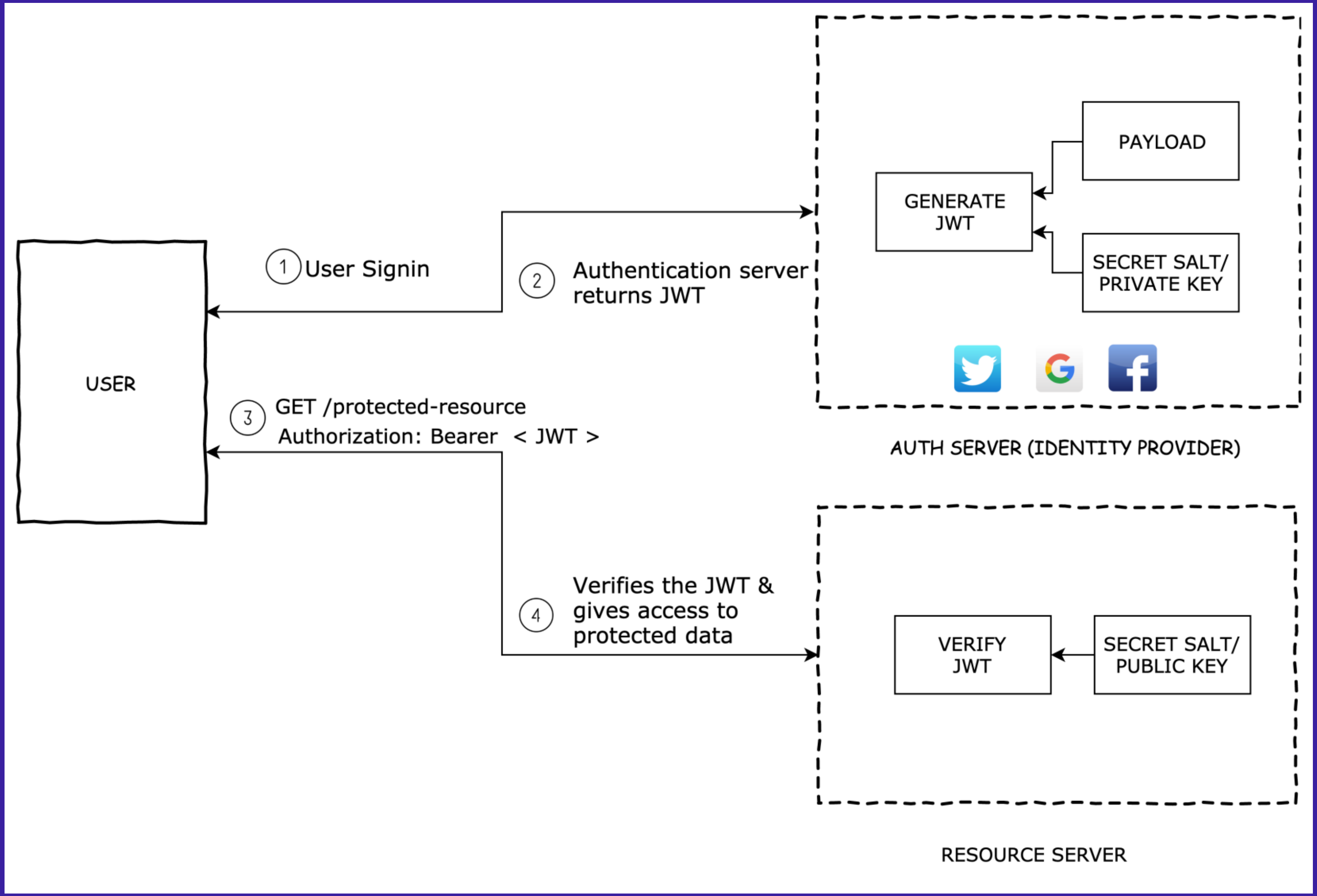
3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

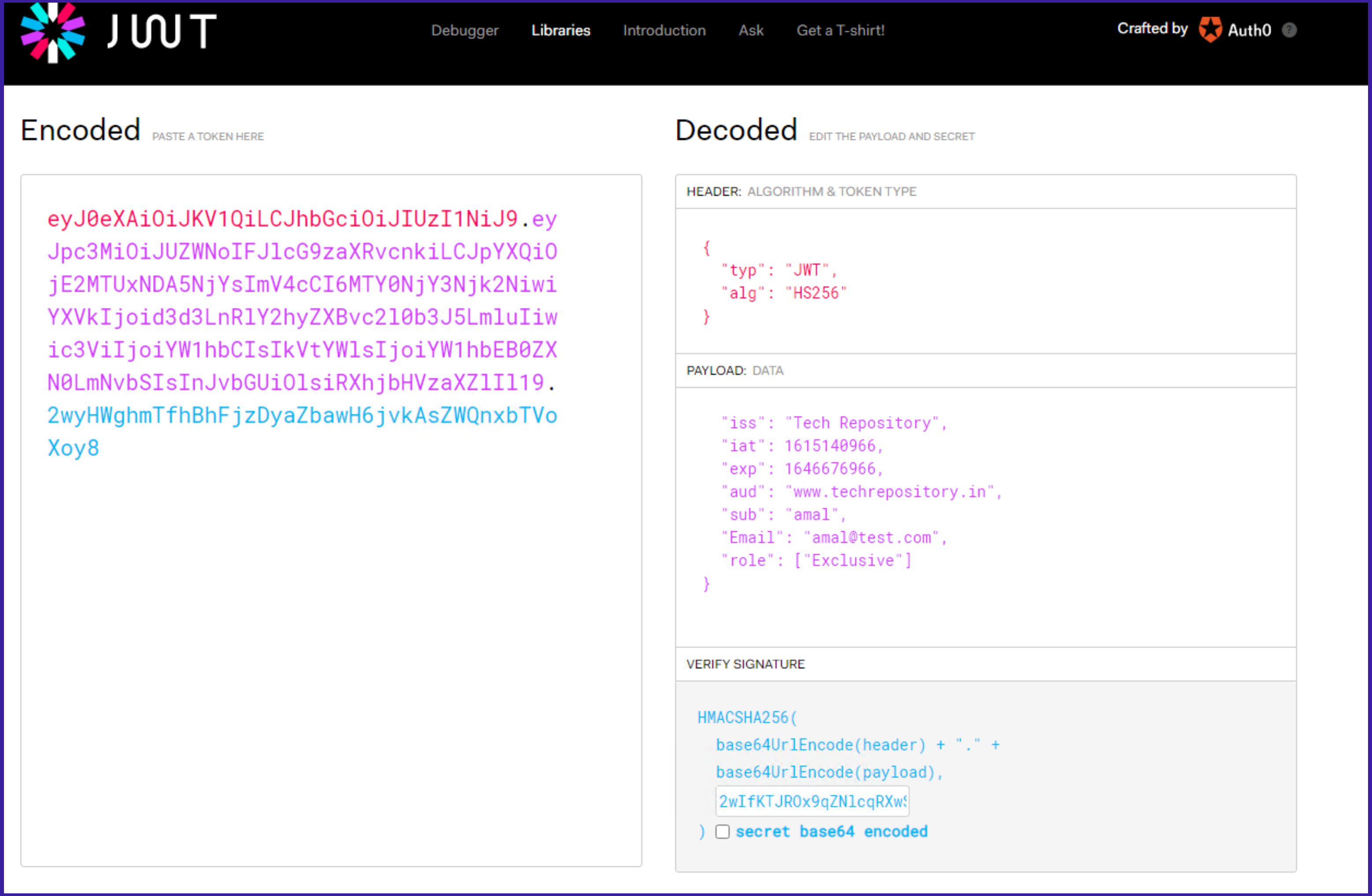
JWT Authentication Flow



JWT Authentication Flow


USER
1 User Signin
2 Authentication server returns JWT
3 GET /protected-resource
Authorization: Bearer < JWT >
4 Verifies the JWT & gives access to protected data
GENERATE JWT
PAYLOAD
SECRET SALT/ PRIVATE KEY
AUTH SERVER (IDENTITY PROVIDER)
VERIFY JWT
SECRET SALT/ PUBLIC KEY
RESOURCE SERVER

Debugging JWT Token using jwt.io



Encoded PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey
Jpc3MiOiJUZWNoIFJlcG9zaXRvcnkiLCJpYXQiO
jE2MTUxNDA5NjYsImV4cCI6MTY0NjY3Njk2Niwi
YXVkIjoid3d3LnRlY2hyZXBvc2l0b3J5LmluIiw
ic3ViIjoiYW1hbCIsIkVtYWlsIjoiYW1hbEB0ZX
N0LmNvbSIsInJvbGUiOlsiRXhjbHVzaXZlIl19.
2wyHWghmTfhBhFjzDyaZbawH6jvkAsZWQnxbTVo
Xoy8
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "iss": "Tech Repository",
  "iat": 1615140966,
  "exp": 1646676966,
  "aud": "www.techrepository.in",
  "sub": "amal",
  "Email": "amal@test.com",
  "role": ["Exclusive"]
}
```

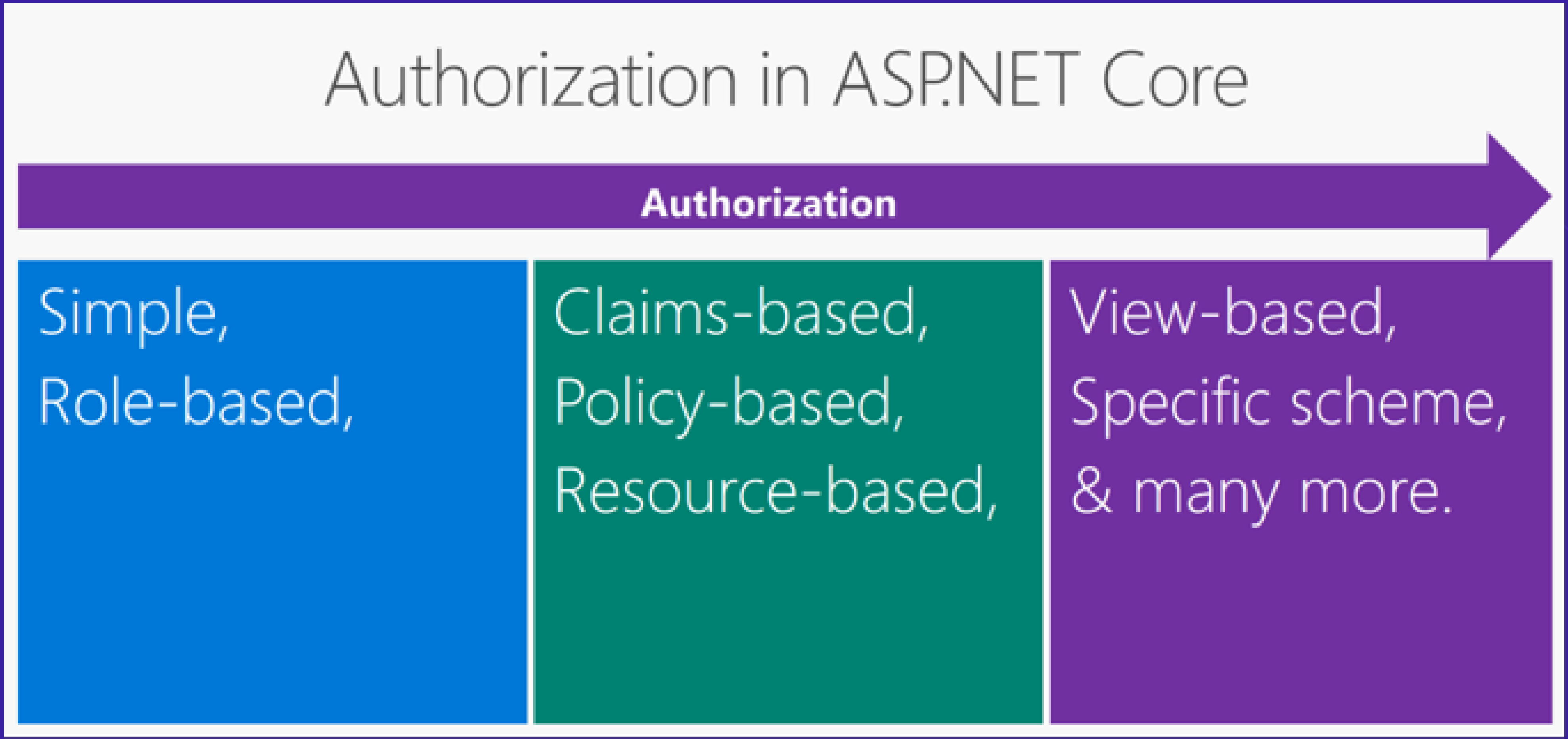
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  2wIfKTJROx9qZNlcqRXw
) ☐ secret base64 encoded
```

Implementing JWT Auth in ASP.NET Core

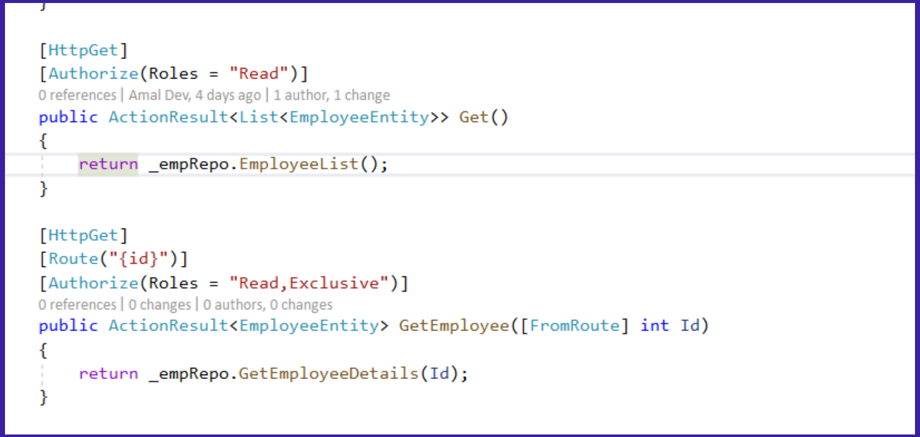
```
services.AddAuthentication(x =>
           {
               x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
               x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
           })
           .AddJwtBearer(x =>
           {
               x.RequireHttpsMetadata = false;
               x.SaveToken = true;
               x.TokenValidationParameters = new TokenValidationParameters
               {
                   ValidateIssuerSigningKey = true,
                   IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(_configuration["Jwt:Key"])),
                   ValidateIssuer = false,
                   ValidateAudience = false,
               };
           });
```

Authorization



Authorization in JWT

- Authorization based on Roles is supported out of the box
- As long as the bearer token contains elements for roles, the middleware infers from it and performs authorization checks
- Access to Controller or Action Methods can be restricted using [Authorize] attribute



```
[HttpGet]
[Authorize(Roles = "Read")]
0 references | Amal Dev, 4 days ago | 1 author, 1 change
public ActionResult<List<EmployeeEntity>> Get()
{
    return _empRepo.EmployeeList();
}

[HttpGet]
[Route("{id}")]
[Authorize(Roles = "Read,Exclusive")]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult<EmployeeEntity> GetEmployee([FromRoute] int Id)
{
    return _empRepo.GetEmployeeDetails(Id);
}
```

Custom Authorization Policies

- Custom Authorization is implemented through custom requirements and handlers

- Building Blocks
    - Requirement
    - Requirement Handler
    - Pre-Configured Policy

Implement a requirement

```
public class MinimumExperienceRequirement : IAuthorizationRequirement
{
    public MinimumExperienceRequirement(int years)
    {
        ExpInYears = years;
    }
    protected int ExpInYears { get; set; }
}
```

Adding a Policy Handler

```
public class MinimumExperienceHandler : AuthorizationHandler<MinimumExperience>
{
    protected override Task HandleRequirementAsync(AuthorizationContext context, MinimumExperience requirement)
    {
        if (!context.User.HasClaim(c => c.Type ==  ClaimTypes.YearsOfExp))
       {
           return Task.CompletedTask;
       }
       var expInYears = Convert.ToDateTime(context.User.FindFirst(c => c.Type == ClaimTypes.YearsOfExp).Value);
       if (expInYears >= 3)
       {
           context.Succeed(requirement);
       }
       return Task.CompletedTask;
    }
}
```

Register Policy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddAuthorization(options =>
    {
        options.AddPolicy(“CustomPolicy”, policy =>
            policy.Requirements.Add(new MinimumExperience(10)));
    });
}
```

Applying Policy

```
[Authorize(Policy="CustomPolicy")]
public class SampleController : Controller
{
    //Write your code here...
}
```

Thanks for joining!