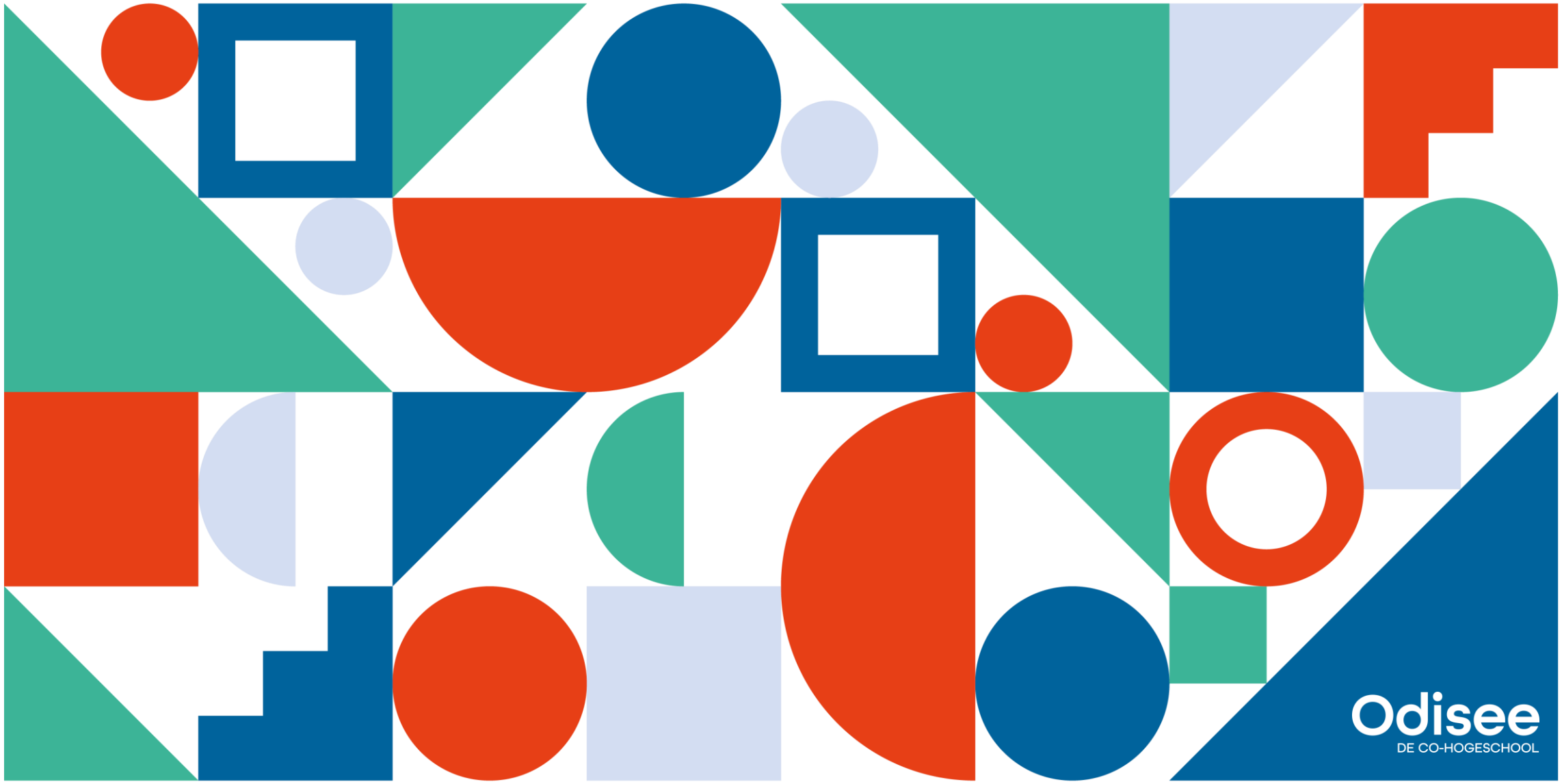

Odisee
DE CO-HOGESCHOOL

Application Development

.NET 6: Web API (Security)

Sam Van Buggenhout

Academiejaar 2022-2023

Odisee
DE CO-HOGESCHOOL

Odisee
DE CO-HOGESCHOOL

Inleiding

Web API: Security

- Ook bij de ontwikkeling van Web API's is **authenticatie** en **autorisatie** zeer belangrijk!
- Er bestaan verschillende mogelijkheden (**protocols**) om authenticatie en autorisatie te implementeren:
  - OAuth (open authorization)
  - SAML (SOAP-based)
  - JSON Web Tokens (JWT)

OAuth

Bron: https://auth0.com/intro-to-iam/what-is-oauth-2/

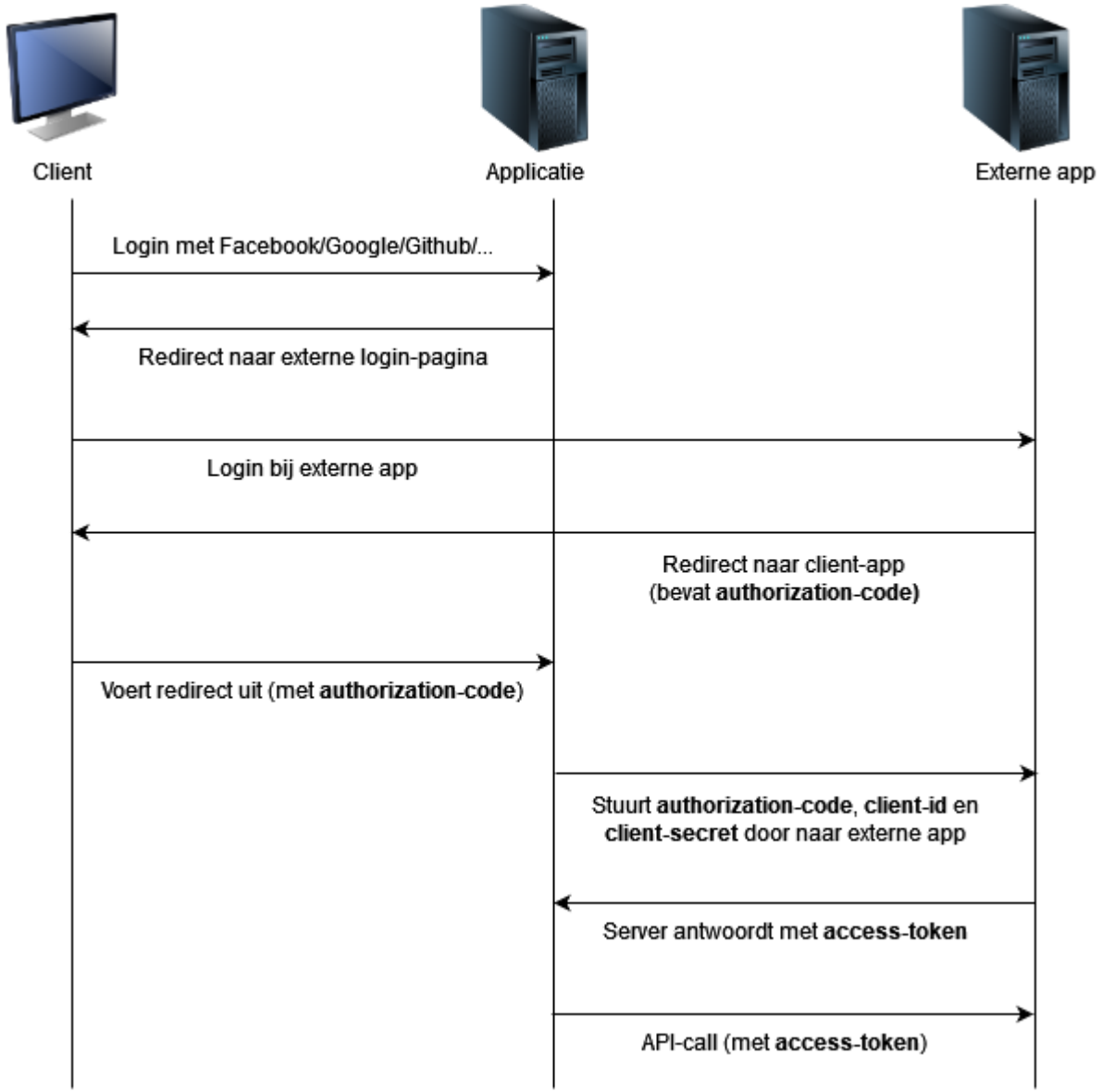
OAuth

- OAuth staat voor **Open Authorization**
- Standaard om applicaties/websites (in naam van een gebruiker) **toegang** te geven tot **resources** van **andere web apps**
- Is de industriestandaard voor online autorisatie
- Via OAuth kan toegang tot resources (data)/acties geregeld worden
- OAuth is een **autorisatie-protocol** (**géén authenticatie-protocol**!)
- Credentials (gebruikersnaam + wachtwoord) worden nóóit gedeeld (alles verloopt via **tokens**)

OAuth

1. Gebruiker krijgt URL die er zo kan uitzien:
   .../authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read
2. Vervolgens krijgt gebruiker een login-scherm te zien met de vraag of hij/zij de app toegang wil geven tot de gevraagde informatie/acties
3. Wanneer de gebruiker zich authentiseert en akkoord gaat, wordt een redirect uitgevoerd naar de callback-url. Hierbij wordt **een authorization-code** meegestuurd (bv.: als GET-parameter)
4. Vervolgens stuurt applicatie een nieuwe request naar de externe applicatie om een **access-token** te bekomen:
   .../token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&grant_type=authorization_code&code=AUTHORIZATION_CODE&redirect_uri=CALLBACK_URL
5. Indien de parameters geldig zijn, stuurt de externe applicatie een request naar de callback-url met het **access-token** van de gebruiker
6. De applicatie kan nu requests versturen naar de externe applicatie, door telkens het token van de gebruiker mee te sturen

OAuth (flow)



Odisee
DE CO-HOGESCHOOL

JWT

JWT

- JWT = JSON Web Token
- Standaard voor creëren van data met een optionele signatuur
  - Payload bevat aantal “claims” in JSON-formaat (bv.: Ingelogd Als Admin)

- JWT-token bestaat typisch uit drie onderdelen:
  - **Header** (bevat informatie in verband met gebruikte signatuur-algoritme)
  - **Payload** (bevat de claims en typisch ook een iat-veld (Issued At Time)
  - **Signatuur** (signatuur is soort “digitale handtekening” die verifieert of token gegenereerd werd door de server en inhoud van het token niet werd aangepast)

Header

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

Payload

```
{
  "loggedInAs": "admin",
  "iat": 1422779638
}
```

Signature

```
HMAC_SHA256(
  secret,
  base64urlEncoding(header) + '.' +
  base64urlEncoding(payload)
)
```

Bron: https://en.wikipedia.org/wiki/JSON_Web_Token

JWT

- Elk onderdeel wordt geconverteerd naar een Base64-string
- Nadien worden alle onderdelen achter elkaar geplakt, gescheiden d.m.v. een punt
  Een token ziet er dan bijvoorbeeld als volgt uit:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0MjI3Nzk2Mzh9.gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjLiuyu5CSpyHI

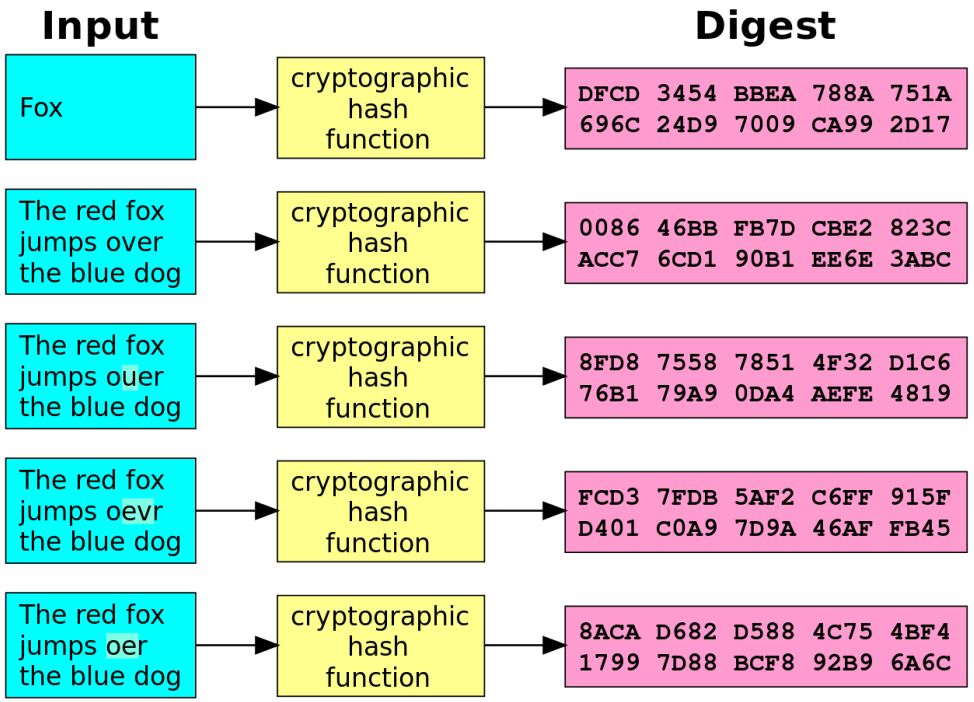
header | payload | signatuur

- Tokens kunnen eenvoudig gedecodeerd/gegenereerd worden
- Tóch is het gebruik van JWT-tokens veilig! ... MAAR waarom?

Een beetje cryptografie

- Om een antwoord te kunnen bieden op deze vraag, hebben we een kleine achtergrond nodig in een aantal principes van de cryptografie...

- Een **cryptografische hash-functie** is een wiskundig algoritme dat data van een **willekeurige grootte** (="message") mapt naar een array van bits van een **vaste grootte** (= "digest")
- Een hash-functie is een **one-way-functie** (eenvoudig uit te voeren in één richting, maar moeilijk/onmogelijk in de andere richting)
  - Van message naar digest is heel eenvoudig, maar van digest kan oorspronkelijke message niet bepaald worden
- Eenzelfde input levert steeds eenzelfde hash op



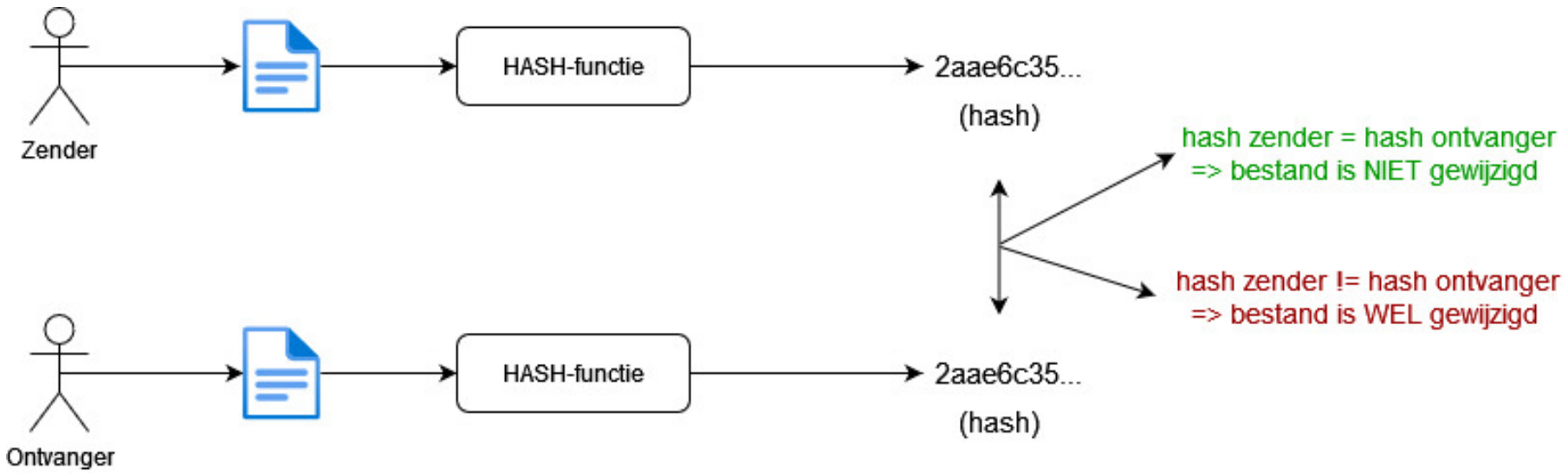
Odisee
DE CO-HOGESCHOOL

Een beetje cryptografie

- Voorbeelden van cryptografische hashing-algoritmen zijn: MD5, SHA-1, SHA-3, SHA-256, SHA-512, ...
- Het gebruik van hashing kent verschillende toepassingen:
  - Basis voor opslaan van wachtwoorden
  - Als id voor file (bv.: in version-control systemen)
  - Om integriteit van data te verifiëren
- Integriteit van data = nagaan dat data volledig en ongewijzigd is
  - Maakt gebruik van principe dat zelfde input zelfde hash oplevert
  - Zelfde file levert zelfde hash op → mogelijkheid om te detecteren of bestand hetzelfde is als ander bestand (zonder bit-per-bit te controleren)

Een beetje cryptografie

- Zender van bestand berekent oorspronkelijke hash van bestand dat hij/zij wenst te versturen en publiceert de bekomen hash-waarde
- Ontvanger downloadt het bestand en berekent nadien ook hash-waarde van gedownloade bestand
- Indien hashes van zender en ontvanger overeenkomen, betekent dit dat bestanden waarop hash berekend werden (waarschijnlijk) identiek zijn => het bestand werd ondertussen niet aangepast Indien hashes NIET overeenkomen, werd bestand onderweg wél gewijzigd (bv. door corrupte netwerk-stream)!

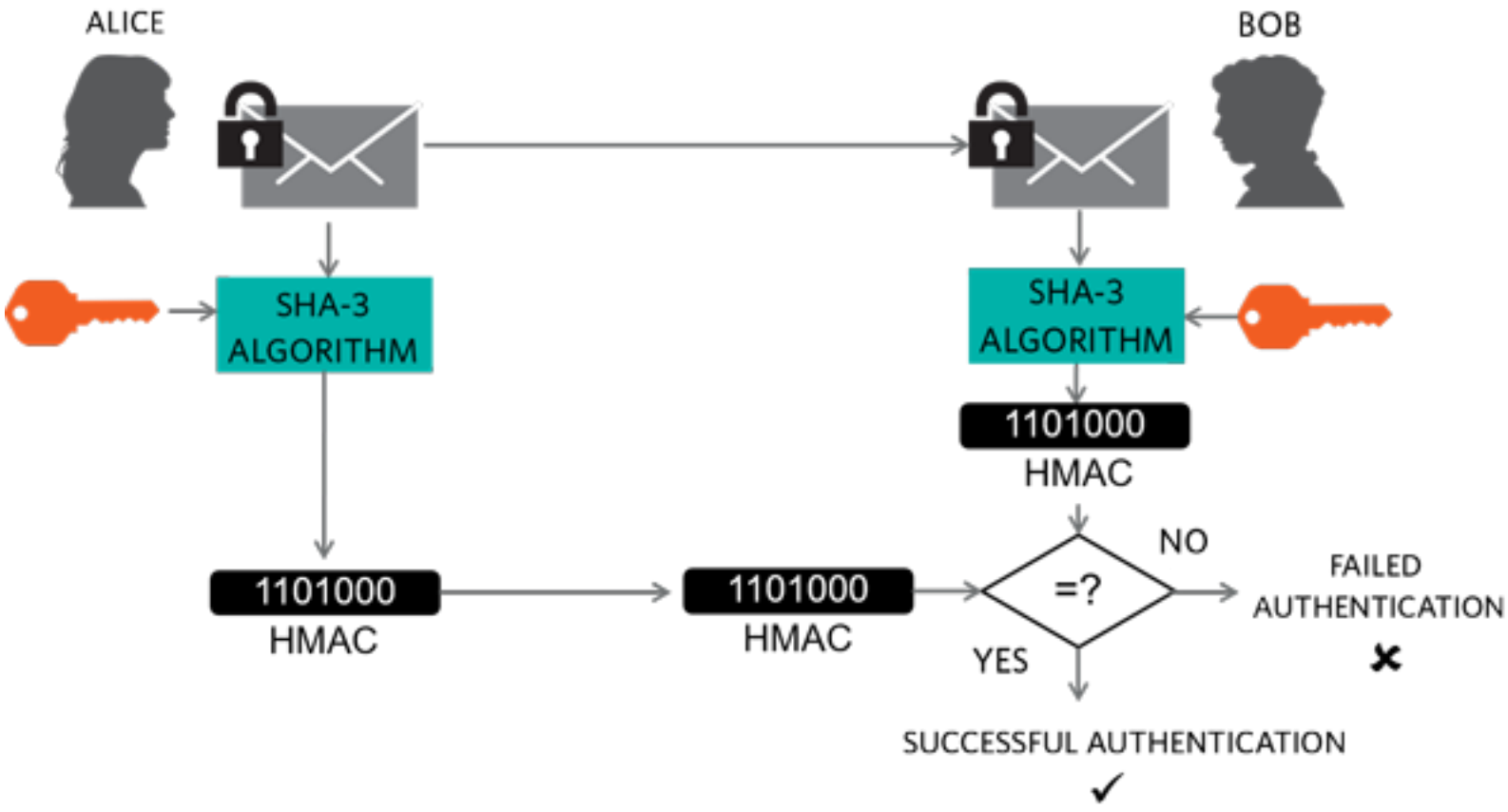


Odisee
DE CO-HOGESCHOOL

Een beetje cryptografie

- Een andere techniek die gebruikt wordt bij het genereren van JWT-tokens is HMAC
  - HMAC = Hash-based Message Authentication Codes
  - Wordt gebruikt om tegelijkertijd **integriteit én authenticiteit** van data te verifiëren
  - **Integriteit** = data is onderweg niet aangepast
  - **Authenticiteit** = data is afkomstig van zender die beweert de zender te zijn
- Hoe wordt dit bereikt?
  1. Verzender en ontvanger spreken “geheime sleutel” af
  2. Bericht + geheime sleutel worden gehasht (via hash-functie naar keuze)
  3. Bericht + HMAC-code worden naar ontvanger gestuurd
  4. Ontvanger berekent zelf ook HMAC van bericht (met zelfde gedeelde sleutel)
  5. Ontvanger controleert of HMAC overeenkomt met HMAC die in bericht werd verstuurd

Een beetje cryptografie



Bron: https://blogs.sap.com/2019/12/25/hmac-sha1-hash-verification-on-api-management/

Odisee DE CO-HOGESCHOOL

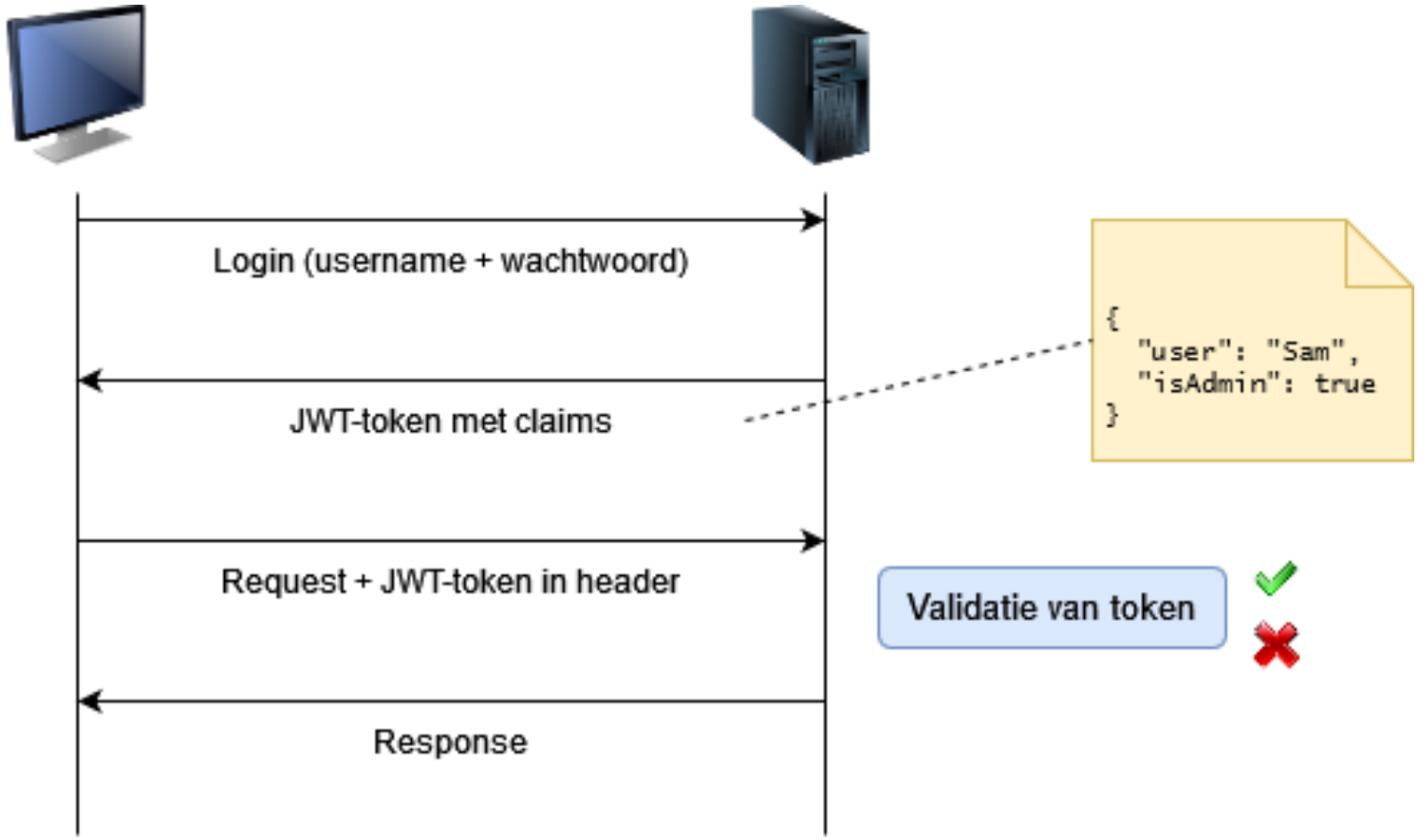
Een beetje cryptografie

- Bericht en sleutel worden samengevoegd in zelfde hash
- Waarom zorgt dit voor **integriteit**?
  - Indien data gewijzigd is, komt HMAC die berekend wordt door ontvanger niet meer overeen met HMAC die meegestuurd wordt door verzender
- Waarom zorgt dit voor **authenticiteit**?
  - Sleutel die gebruikt wordt voor genereren en verifiëren is geheim
  - Indien zender andere sleutel gebruikt dan die normaal verwacht wordt (bv.: aanvaller), komen HMAC-codes niet overeen
- Dus: HMAC-codes komen enkel overeen indien: data niet gewijzigd werd én bericht gestuurd werd door zender die over de geheime sleutel beschikt

Terug naar JWT

- We hebben nu de nodige achtergrondkennis om te begrijpen waarom JWT-tokens veilig zijn! ☺
- Maar: hoe werkt JWT-autorisatie nu?
- De flow ziet er als volgt uit:
  1. De gebruiker authentiseert zich via de API (d.m.v. username + passwoord/OAuth/...)
  2. Indien authenticatie geslaagd is, genereert server JWT-token met username + claims voor gebruiker
  3. Server stuurt token terug als response op authenticatie
  4. Bij elke volgende request, stuurt client het token mee in de header van de request
  5. Server ontvangt request, verifieert token en gebruikt claims in token voor autorisatie

JWT-autorisatie flow



Odisee
DE CO-HOGESCHOOL

Waarom is JWT veilig?

- Bij JWT-autorisatie gaat server uit van claims in token
- MAAR: token wordt door client naar server gestuurd
  - Hoe kunnen we vermijden dat client andere claims mee stuurt dan waar hij/zij recht op heeft...?

Waarom is JWT veilig?

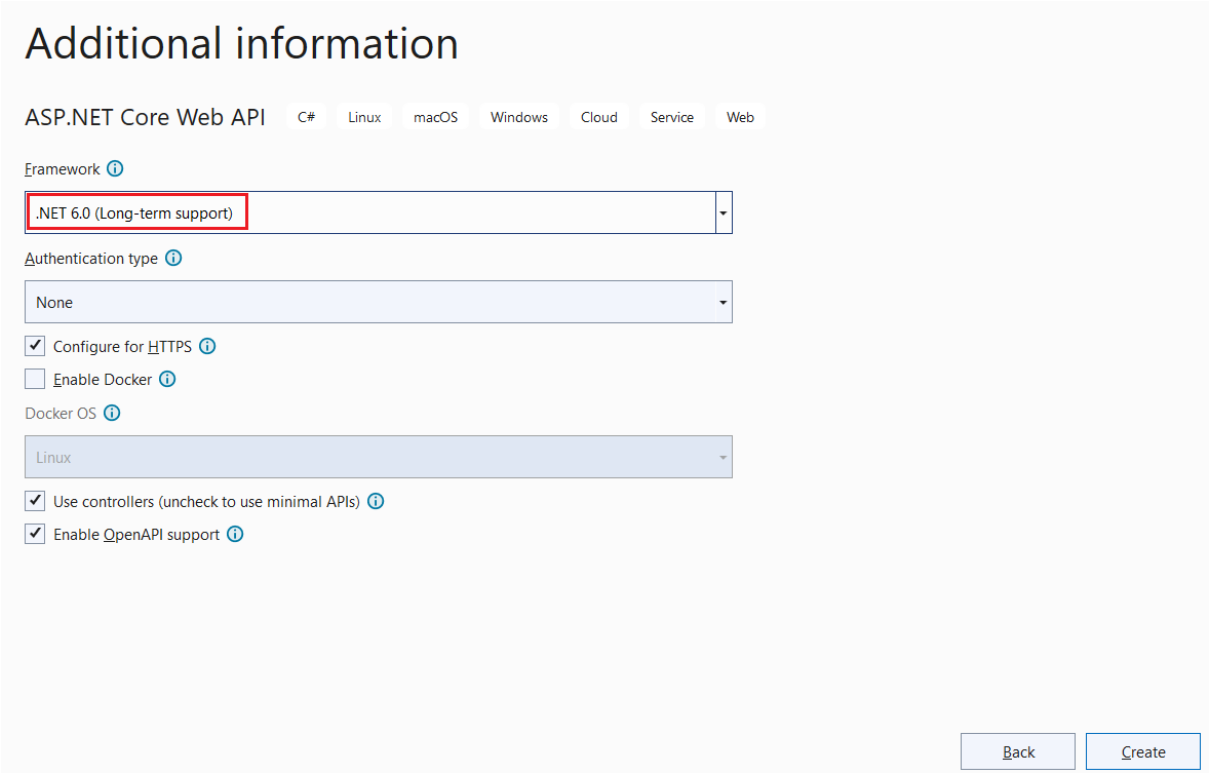
- Server heeft een geheime sleutel (bv.: lang, veilig wachtwoord)
- Na genereren van token, berekent server HMAC van token + geheime sleutel
- HMAC wordt toegevoegd aan token en mee naar client gestuurd
- Wanneer client token mee stuurt met request, berekent server HMAC van token
- Indien berekende HMAC overeenkomt met HMAC in token, is token niet aangepast en uitgegeven door partij die over geheime sleutel beschikt (de server zélf)

  ➔ Token kan vertrouwd worden

Implementatie in .NET

JWT implementeren in .NET

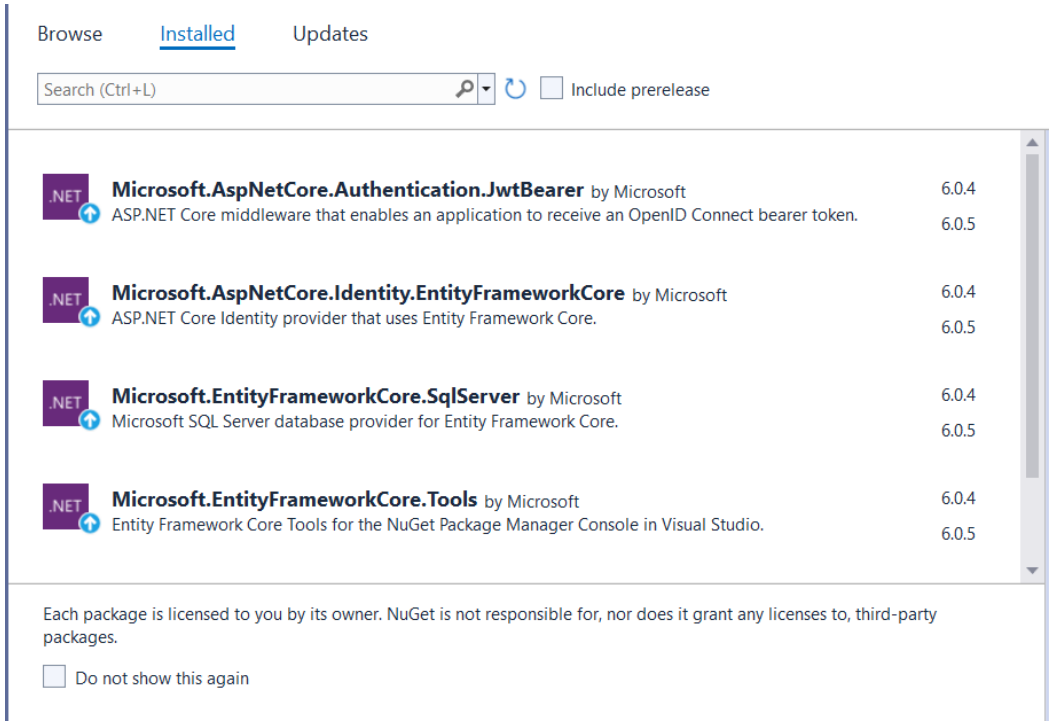
1. Maak een nieuw Web API project
   - Laat *Authentication Type* op “None” staan



Odisee
DE CO-HOGESCHOOL

JWT implementeren in .NET

2. Installeer de volgende NuGet-packages:
   - Microsoft.EntityFrameworkCore.SqlServer
   - Microsoft.EntityFrameworkCore.Tools
   - Microsoft.AspNetCore.Identity.EntityFrameworkCore
   - Microsoft.AspNetCore.Authentication.JwtBearer



Odisee
DE CO-HOGESCHOOL

JWT implementeren in .NET

3. Wijzig configuratie in appsettings.json
   - Voeg ConnectionString toe (voor database-toegang)
   - Voeg configuratie-properties toe voor JWT

```
{
  "ConnectionStrings": {
    "default": "Server=(localdb)\\mssqllocaldb;Database=ApiDB;Trusted_Connection=True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "JWT": {
    "ValidAudience": "http://localhost:4200",
    "ValidIssuer": "http://localhost:7103",
    "Secret": "JWTAuthenticationHIGHsecuredPasswordVVVp1OH7Xzyr"
  }
}
```

JWT implementeren in .NET

4. Maak een DbContext-klasse
   - Erf over van `IdentityDbContext<IdentityUser>`

```
public class DbContext : IdentityDbContext<IdentityUser>
    {
        public DbContext(DbContextOptions<DbContext> options) : base(options)
        {
        }
        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
        }
    }
```

JWT implementeren in .NET

5. Set-up in Program.cs (deel 1)
   - Registreer DbContext voor Dependency Injection
   - Configureer Identity Framework

```
var builder = WebApplication.CreateBuilder(args);

ConfigurationManager configuration = builder.Configuration;

// Add services to the container.

// Registreer DbContext-klasse
builder.Services.AddDbContext<Web_API.Data.DbContext>(options =>
options.UseSqlServer(configuration.GetConnectionString("default")));

// Set-up voor Identity-framework
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<Web_API.Data.DbContext>()
    .AddDefaultTokenProviders();
```

JWT implementeren in .NET

5. Set-up in Program.cs (deel 2)

- Configureer Authenticatie via Jwt-Bearer

```
// Authenticatie via Jwt-Bearers
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
})

//Configuratie van Jwt-Bearer
.AddJwtBearer(options =>
{
    options.SaveToken = true;
    options.RequireHttpsMetadata = false;
    options.TokenValidationParameters = new TokenValidationParameters()
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidAudience = configuration["JWT:ValidAudience"],
        ValidIssuer = configuration["JWT:ValidIssuer"],
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["JWT:Secret"]))
    };
});
```

JWT implementeren in .NET

5. Set-up in Program.cs (deel 3)
   - Voeg UseAuthentication() en UseAuthorization() toe aan middleware-pipeline

```
app.UseHttpsRedirection();

app.UseAuthentication(); //NIET VERGETEN!
app.UseAuthorization();  //NIET VERGETEN!

app.MapControllers();

app.Run();
```

JWT implementeren in .NET

Endpoint registratie nieuwe user:

```
[Route("api/[controller]")]
[ApiController]
public class AuthenticationController : Controller
    {
        private readonly UserManager<IdentityUser> _userManager;
        private readonly IConfiguration _configuration;

        public AuthenticationController(
            UserManager<IdentityUser> userManager,
            IConfiguration configuration)
        {
            _userManager = userManager;
            _configuration = configuration;
        }


        //vervolg op volgende dia

    }
```

Set-up van dependency's

JWT implementeren in .NET

Endpoint registratie nieuwe user:

```
 [HttpPost]
 [Route("register")]
 public async Task<IActionResult> Register([FromBody] RegisterModel model)
    {
       var userExists = await _userManager.FindByNameAsync(model.Username);
       if (userExists != null)
           return StatusCode(StatusCodes.Status500InternalServerError, new Response { Status = "Error", Message = "User
already exists!" });

       IdentityUser user = new()
       {
          Email = model.Email,
          SecurityStamp = Guid.NewGuid().ToString(),
          UserName = model.Username
       };
       var result = await _userManager.CreateAsync(user, model.Password);
       if (!result.Succeeded)
           return StatusCode(StatusCodes.Status500InternalServerError, new Response { Status = "Error", Message = "User
creation failed! Please check user details and try again." });

        return Ok(new Response { Status = "Success", Message = "User created successfully!" });
     }
```

JWT implementeren in .NET

Endpoint registratie nieuwe user:

https://localhost:7103/api/Authentication/Register

POST https://localhost:7103/api/Authentication/Register

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "username": "Joske",
3     "email": "joske123@gmail.com",
4     "password": "Test123!"
5 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "status": "Success",
3     "message": "User created successfully!"
4 }
```

JWT implementeren in .NET

Endpoint voor login:

```
[HttpPost]
[Route("login")]
public async Task<IActionResult> Login([FromBody] LoginModel model)
{
    var user = await _userManager.FindByNameAsync(model.Username);
    if (user != null && await _userManager.CheckPasswordAsync(user, model.Password))
    {
        var userRoles = await _userManager.GetRolesAsync(user);

        var authClaims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, user.UserName),
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        };

        foreach (var userRole in userRoles)
        {
            authClaims.Add(new Claim(ClaimTypes.Role, userRole));
        }

        var token = GetToken(authClaims);
        return Ok(new
        {
             token = new JwtSecurityTokenHandler().WriteToken(token),
             expiration = token.ValidTo
        });
    }
    return Unauthorized();
}
```

JWT implementeren in .NET

Methode GetToken (voor login)

```
private JwtSecurityToken GetToken(List<Claim> authClaims)
{
    var authSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JWT:Secret"]));

    var token = new JwtSecurityToken(
        issuer: _configuration["JWT:ValidIssuer"],
        audience: _configuration["JWT:ValidAudience"],
        expires: DateTime.Now.AddHours(3),
        claims: authClaims,
        signingCredentials: new SigningCredentials(authSigningKey, SecurityAlgorithms.HmacSha256)
    );

    return token;
}
```

JWT implementeren in .NET

Endpoint voor login:

https://localhost:7103/api/Authentication/Login

Save

POST https://localhost:7103/api/Authentication/Login Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "username": "Joske",
3     "password": "Test123!"
4 }
```

Body Cookies Headers (4) Test Results Status: 200 OK Time: 740 ms Size: 533 B Save Response

Pretty Raw Preview Visualize JSON


```
1 {
2     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
          eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiSm9za2UiLCJqdGkiOiI0ZWRlZTliYi01MTU1LTQzMjgtOTM0Y
          S0yMzY4MTIzNGVjMzIiLCJleHAiOjE2NTIzOTQ1MTEsImlzcyI6Imh0dHA6Ly9sb2NhbGhvc3Q6NzEwMyIsImF1ZCI6Imh0dHA6Ly9sb2NhbGhvc3Q6NDIwMCJ9.
          VEdnw5b_xFZL-rMPs92arSsbS3ktgw5cTL9Z19saBCc",
3     "expiration": "2022-05-12T22:28:31Z"
4 }
```

JWT implementeren in .NET

Kopieer het token van de response en voeg het toe als header bij elke volgende request naar de API

https://localhost:7103/api/WeatherForecast

Save

GET https://localhost:7103/api/WeatherForecast Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Type Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

Body Cookies Headers (4) Test Results Status: 200 OK Time: 129 ms Size: 644 B Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "date": "2022-05-13T21:32:06.8468299+02:00",
        "temperatureC": 27,
        "temperatureF": 80,
        "summary": "Bracing"
    },
```


Odisee DE CO-HOGESCHOOL

JWT implementeren in .NET

Opmerking:

Afdwingen van rechten en authenticatie/autorisatie gebeurt op dezelfde manier als bij MVC-applicaties (mbv.: [Authorize]-attribuut)

```
[ApiController]
[Route("api/[controller]")]
[Authorize]
3 references
public class WeatherForecastController
{
```