FHV
Vorarlberg University
of Applied Sciences

Application Integration and Security

Philipp Scambor
Valmir Bekiri

Learning outcomes and Methodology

- Learning outcomes
  - Difference of Authentication & Authorization
  - Basics of Authorization variants
  - Common Authentication methods
  - Basics of OAuth
  - Implementing and using JWT

- Methodology
  - Lecture
  - Exercises

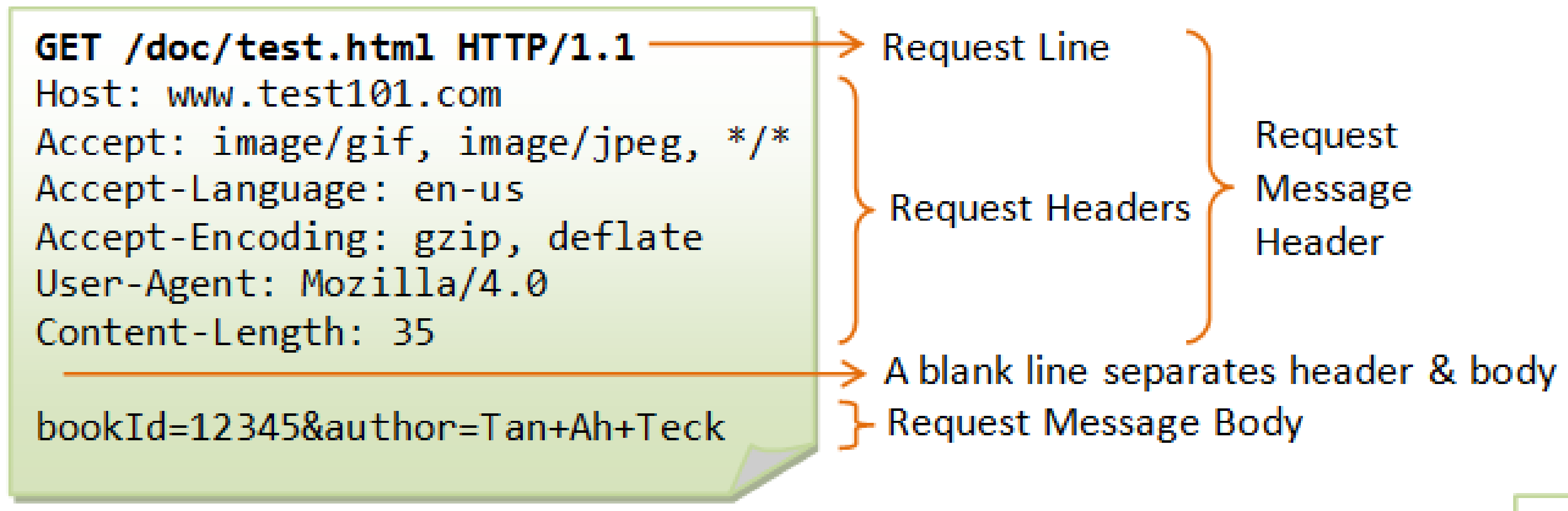
Agenda

- Recap HTTP
- Authentication vs. Authorization
- Authorization Methods
- Authentication Methods
- OAuth
- JWT + Example

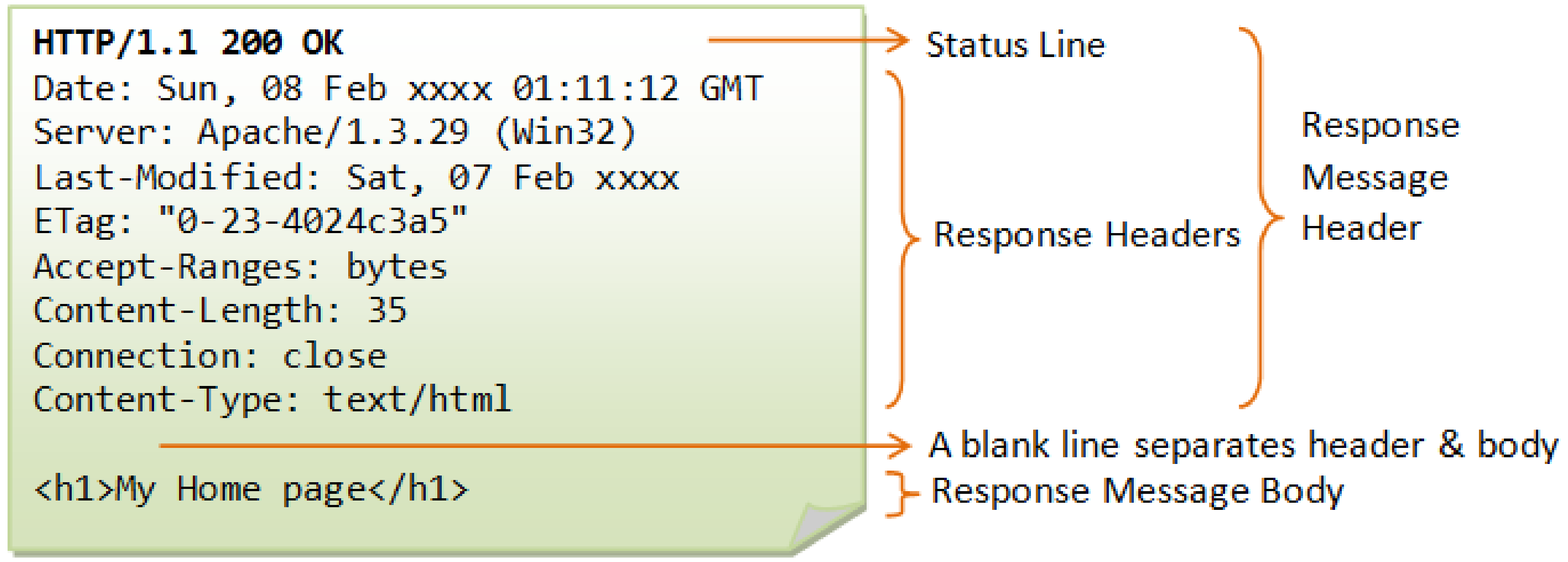
HTTP recap

[https://documentation.help/DogeTool-HTTP-Requests-vt/http_request.htm]

- **H**yper**T**ext **T**ransfer **P**rotocol is ***stateless***
- Different methods – GET, POST, PUT, …
- Response can have different Codes e.g. 200, 404 …



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Authentication and Authorization

Authentication vs Authorization

Authentication ➔ Who are you?

Authorization ➔ Are you allowed to do?

- Authentication - process of verifying a user/device accessing a system!
- Authorization - process of verifying whether a user/device is allowed to do a operation on a system!

[https://www.imperva.com/learn/data-security/role-based-access-control-rbac/]

[https://dinolai.com/notes/others/authorization-models-acl-dac-mac-rbac-abac.html]

[https://dzone.com/articles/acl-rbac-abac-pbac-radac-and-a-dash-of-cbac]

Authorization

There are different ways to implement an authorization system:

- ♦ ACL – Access Control List
  - DAC – Discretionary Access Control
  - MAC – Mandatory Access Control
- ♦ RBAC – Role Based Access Control
- ♦ ABAC/PBAC – Attribute/Policy Based Access Control

Authorization - ACL

[https://www.ekransystem.com/en/blog/mac-vs-dac]

With ACL we basically have a list of who is allowed to do something:

- `Alice: read, write, delete`
- `Bob: read`

Classic ACLs are defined for the whole system. There are variations for a more granular approach:

DAC extends classic ACL functionality

- `Alice can now grant her rights also to Bob`
- `Bob now can also grant those rights to other users`

MAC is more restrictive

- Only the Administrator can define which resources can be accessed by who.

Authorization - RBAC

A Role-Based Access Control authorization system is very common in content management systems – WordPress, Joomla, Sulu, …

- A role can have a set of permissions
  - `Admin: read, write, update, delete`
  - `Guest: read`
  - `Supervisor: read, update`
- A user can have one (or multiple) roles:
  - `Alice: Supervisor`
  - `Bob: Guest`
- Easier to manage but still effective

Authorization - ABAC/PBAC

- Attribute/Policy-Based Access Control is complex
- A User can have different types of access based on their attributes
- Therefore, it is dynamic and offers a high level of security
- Basic Example:
  - `User had write rights to post from 01.02.2021 to 28.02.2021`
  - `After that the user can only read the post`

Authentication

[https://testdriven.io/blog/web-authentication-methods/]

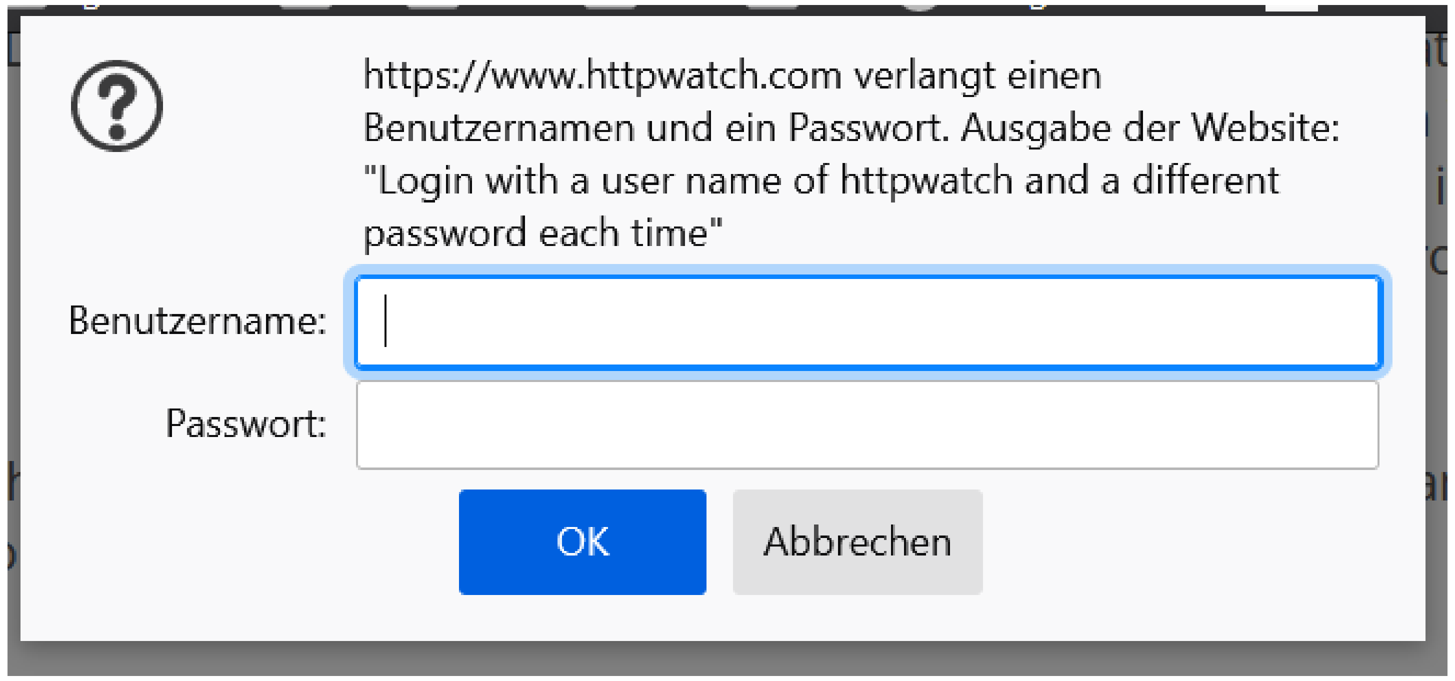
- Different authentication methods:
  - Basic Authentication
  - Session-Cookie Based Authentication
  - Token Based Authentication (e.g. JWT)
  - One Time Passwords (often used for 2FA)
  - Federated Identity Authentication (OAuth/OpenID)

Authentication – Basic Authentication

- Basic Authentication is build into HTTP
- Very simple form of authentication
- As HTTP is stateless every request sends the user credentials
- Credentials are encoded with base64
- Credentials are put in a simple string and username and password are separated by a : ➔ “`username:password`”

Authentication – Basic Authentication

- When you request a restricted resource the server response with a **401 Unauthorized HTTP Code** and a header **WWW-Authenticate: Basic**
- The browser opens up the credentials pop-up and requests the username and password of the user
- Browser sends second request but now with the **Authorization** header and the encoded credentials ➔ `Authorization: Basic` *`dcdvcmQ=`*

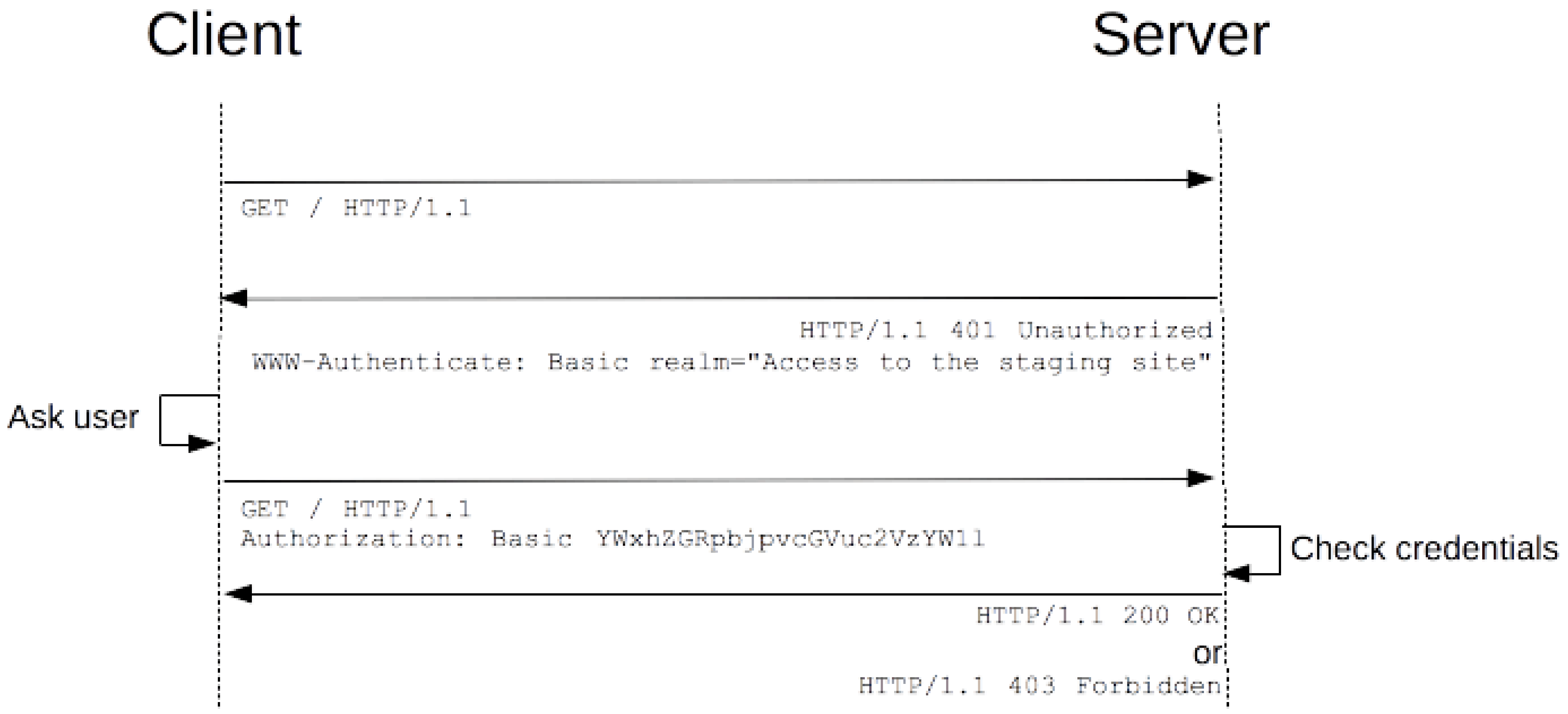


https://www.httpwatch.com verlangt einen Benutzernamen und ein Passwort. Ausgabe der Website: "Login with a user name of httpwatch and a different password each time"

Benutzername:	
Passwort:	

https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

Authentication – Basic Authentication


Client
Server
GET / HTTP/1.1
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to the staging site"
Ask user
GET / HTTP/1.1
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
Check credentials
HTTP/1.1 200 OK
or
HTTP/1.1 403 Forbidden

Authentication – Basic Authentication

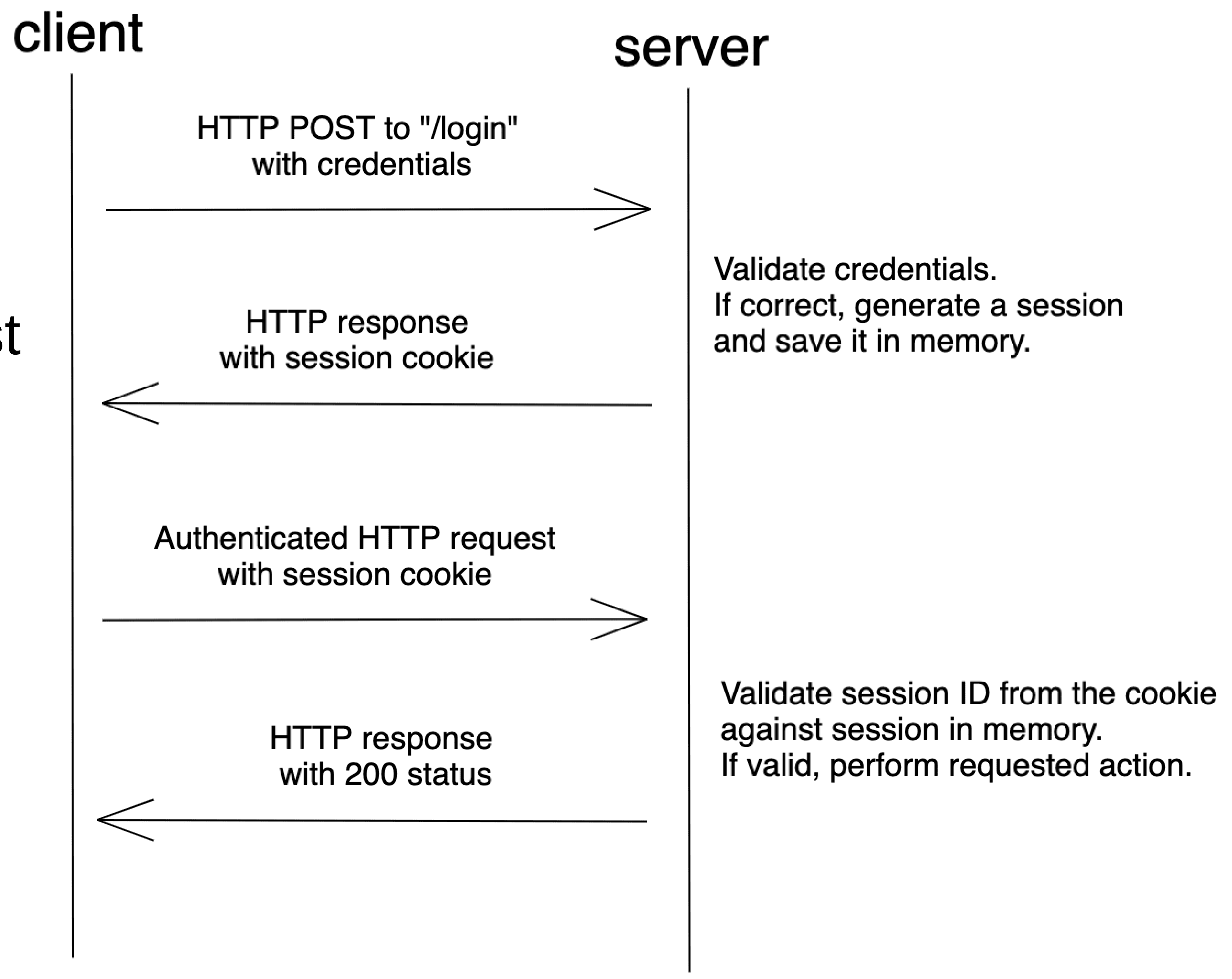
- Problems?
  - Insecure if used without TLS connection
  - Credentials are only encoded and not hashed or encrypted
- There is an extension/variant/enhancement for the Basic Authentication called **HTTP Digest Authentication:**
  - Also built in HTTP
  - Uses MD5 hashing and nonce for the credentials
  - Problem? – still vulnerable if not used through TLS connection

Authentication – Session Based Authentication

- Make use of cookies
- Server stores the login status of the user
- Credentials have only to be send one time
- Server validates credentials and creates cookie with a **session-ID** which is sent to the client.
- Every request has this cookie with the session-ID
- Server can validate the session-ID with its internal sessions
- If valid return 200 OK else 403 Forbidden

Authentication – Session Based Authentication

- Problems?
- Session-Cookie could be stolen
- Vulnerable to CSRF
- Cookie is sent with every request although it might not be necessary
- Not usable when using multiple services because sessions are only saved on one server
- Should only be used with TLS connection



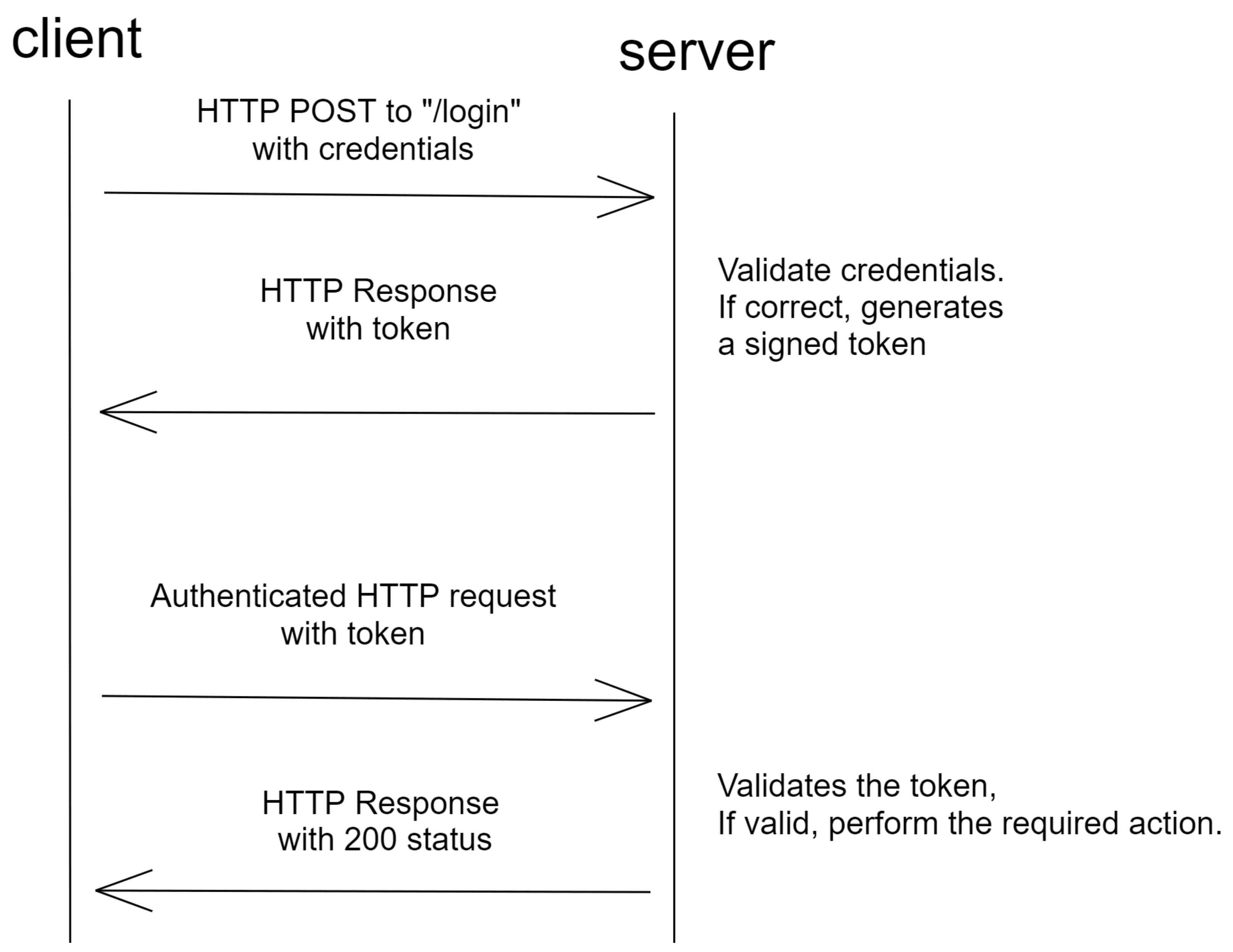
https://testdriven.io/blog/web-authentication-methods/#session-based-aut

Authentication – Token Based Authentication - JWT

- Instead of using cookies with session-ids token-based systems are using tokens generated by the server/service to check if credentials are valid
- Tokens do not have to be saved as cookies on the client side – localStorage/sessionStorage are possible places to save tokens
- Server does not save generated tokens (normally)
- Token contains all necessary information for the server/service to identify the user
- JSON Web Tokens (JWT) are a very common and standardized method for Token Based Authentication

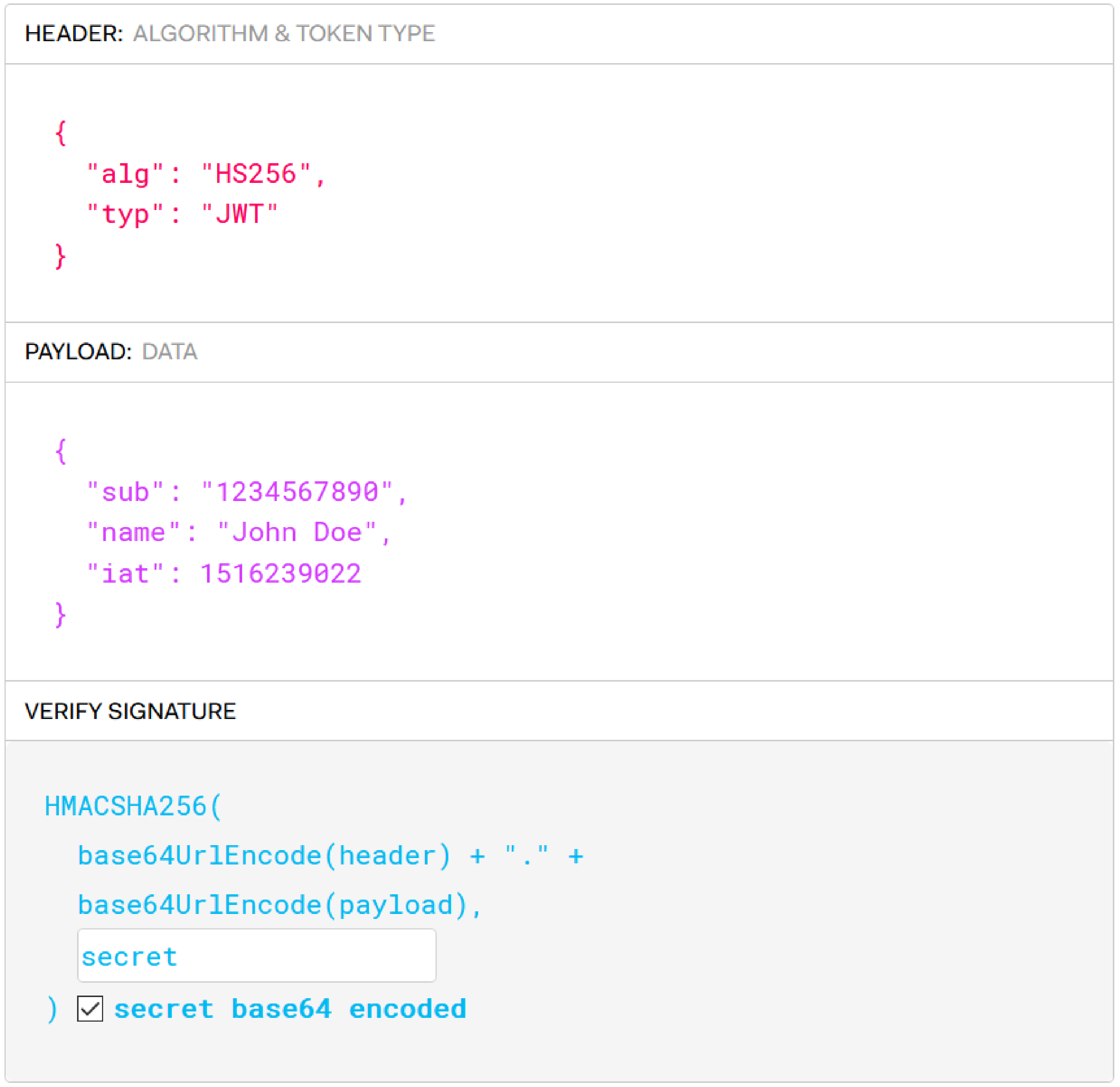
Authentication – Token Based Authentication - JWT

- Client sends user-credentials to login operation
- Server validates credentials and creates token which is sent back to client
- Every operation that needs the token gets it from the client in the payload of the request, via header or via cookie


client
server
HTTP POST to "/login" with credentials
HTTP Response with token
Validate credentials. If correct, generates a signed token
Authenticated HTTP request with token
HTTP Response with 200 status
Validates the token, If valid, perform the required action.

Authentication – Token Based Authentication - JWT

- JWT are specified under [RFC 7519]
- Token contains 3 parts
  - **Header** – information about token and used algorithm
  - **Payload** – the data that is necessary for the service to identify the user
  - **Signature** – used by service to validate if token is correct or manipulated – be aware that you have to have your own password entered to make it secure


HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

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

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☑ secret base64 encoded
```

https://jwt.io/

Authentication – Token Based Authentication - JWT

- Every part is separated by a "."
- Every part is base64 encoded ➔ can be decoded by everyone
- Signature is hashed together with header, payload and the secret before encoded ➔ therefore if someone tries to manipulate the content the signature will not be the same

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.PcmVI
PbcZl9j7qFzXRAeSyhtuBnHQNMuLHsaG5l804A
```

https://jwt.io/

Authentication – Token Based Authentication - JWT

- For JWT it is crucial that the password for the signature is really secure!
- Otherwise, the security of the application/service is compromised!
- The tokens can be reused if another service uses the same password ➔ one login but multiple services useable
- As the server is normally not saving any tokens or data the whole token-based authentication via JWT is **stateless**

Authentication – Token Based Authentication - JWT

- The **payload** of the token is filled with **claims**
  - Some claims are reserved by the standard
  - iss ➔ Issuer of the token
  - sub ➔ Subject for which those tokens are valid
  - aud ➔ Audience – the target domain for which the token is valid
  - exp ➔ Expiration Time – unix timestamp
  - nbf ➔ Not Before – unix timestamp of when the token will be valid
  - iat ➔ Issued At – unix timestamp of when the token is issued
  - jti ➔ JWT ID – unique id of the token
- Service can add own claims to further identify the user

Authentication – Token Based Authentication - JWT

- The token can be send to the service in multiple ways:
  - Payload of the HTTP Request:
    - https://example.com/api/restrictedoperation?jwt_token=..........
    - In the body of the request for POST,…, request
  - Saved in a cookie:
    - Sent in with every request in the header
    - `Cookie: token=…………….`
  - Send via Authorization Header:
    - `Authorization: Bearer ……….. `

Authentication – Token Based Authentication - JWT

- ♦ One main security issue
  - An issued JWT lasts forever!
- ♦ Solution would be to set **expiration time** claim or **issued at** claim
  - How would we then go ahead after the token is expired?
    - Require user to reauthenticate – bad user experience
    - Extend the expiration without credentials
  - Is that a good idea?

Authentication – Token Based Authentication - JWT

- ♦ No!
  - Thief would have unlimited access if the token gets stolen because he can automatically extend the jwt
  - If the expiration time is set very high e.g. 1 week the thief could use the token for the whole week
  - If the token is very short-lived e.g. 15 minutes the token might be useless for the thief, but the user has to login every 15 minutes!

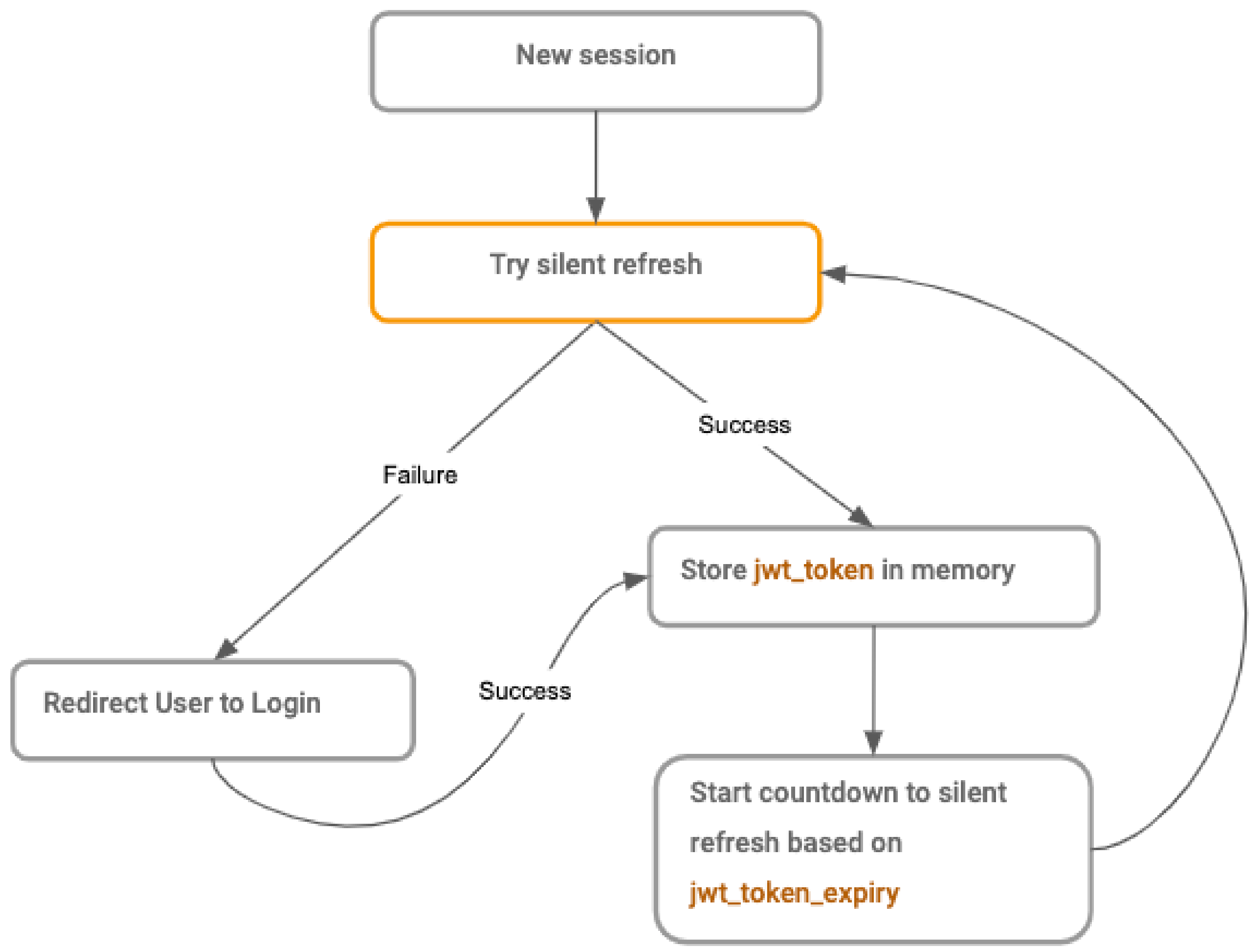
- ♦ Solution is to make use of a refresh token

Authentication – Token Based Authentication - JWT

- Different variations of implementation
  - Basic idea is to have two tokens **access_token** and **refresh_token**
  - **access_token** is still short-lived 5-20 minutes
  - **refresh_token** is long-living 1 day to X months
  - As soon as the **access_token** is invalid the client should get a new one with the **refresh_token**
  - **refresh_token** can typically be blacklisted by the server if malicious access is detected
  - **refresh_token** are normally saved as a cookie

Authentication – Token Based Authentication - JWT

- Flow of a refresh


New session
Try silent refresh
Failure
Success
Store jwt_token in memory
Redirect User to Login
Success
Start countdown to silent refresh based on jwt_token_expiry

https://hasura.io/blog/best-practices-of-using-jwt-with-graphql/

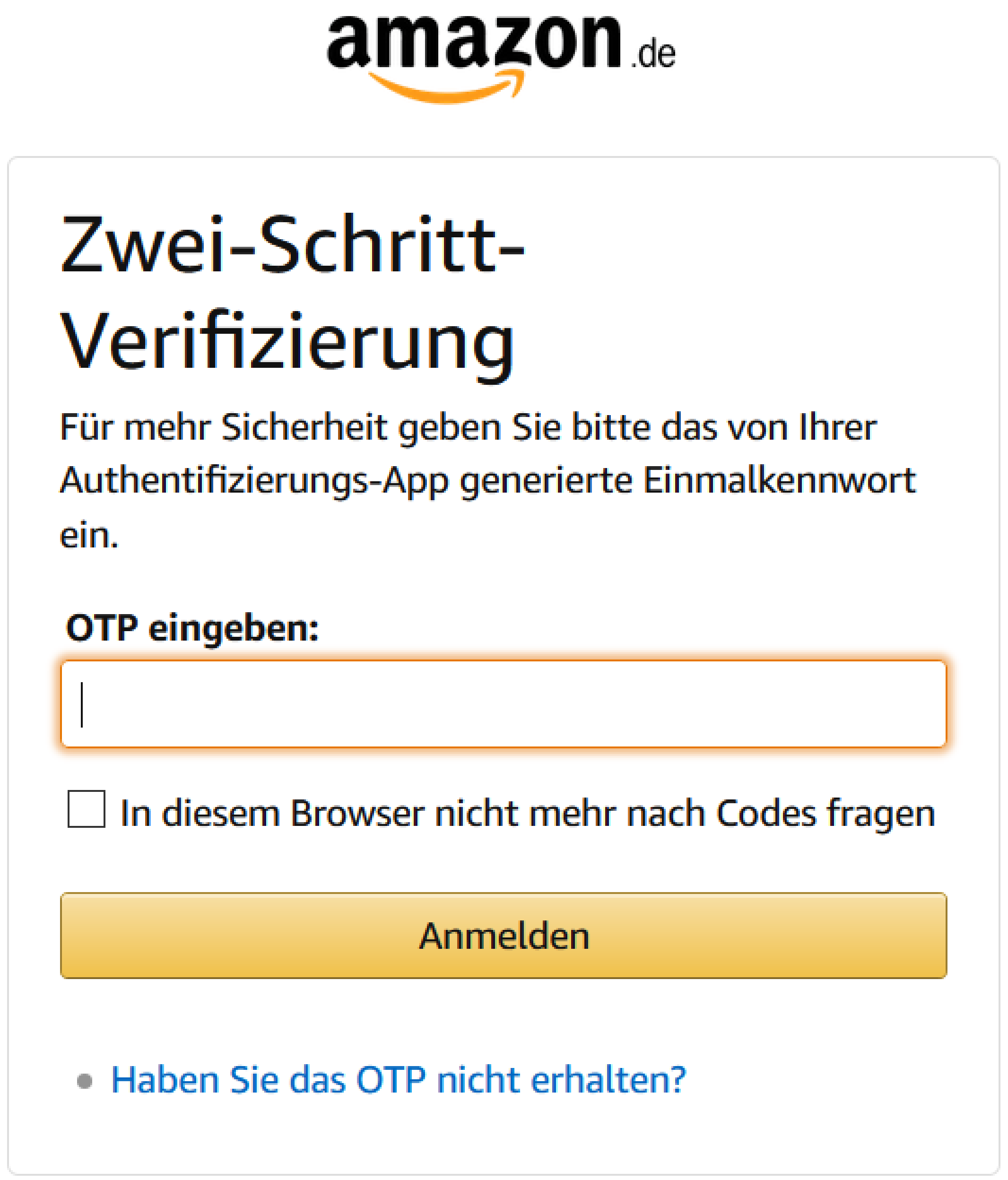
Authentication – OTP

https://testdriven.io/blog/web-authentication-methods/#one-time-passwords

- One-time passwords are randomly generated single use codes to verify a user or a service
- Typically used for two factor authentication (2FA)
- Special type of OTPs are the Time-based OTP (TOTP)
- Example:
  - After a user tries to sign in a service
  - Service is sending OTP to E-Mail, telephone number via SMS or any other trusted system
  - User has to enter this OTP (mostly 6 figures) into the service to finalize the login

Authentication – OTP

- Example of OTP ➔ TAN for eBanking
- Modern 2FA implementations are using (T)OTP agents like
  Google Authenticator, Microsoft Authenticator App
- These kind of 2FA is very secure as long as the generation of the codes is also secured


amazon.de

Zwei-Schritt-Verifizierung

Für mehr Sicherheit geben Sie bitte das von Ihrer Authentifizierungs-App generierte Einmalkennwort ein.

OTP eingeben:

☐ In diesem Browser nicht mehr nach Codes fragen

- Haben Sie das OTP nicht erhalten?

Authentication – OAuth

https://oauth.net/2/

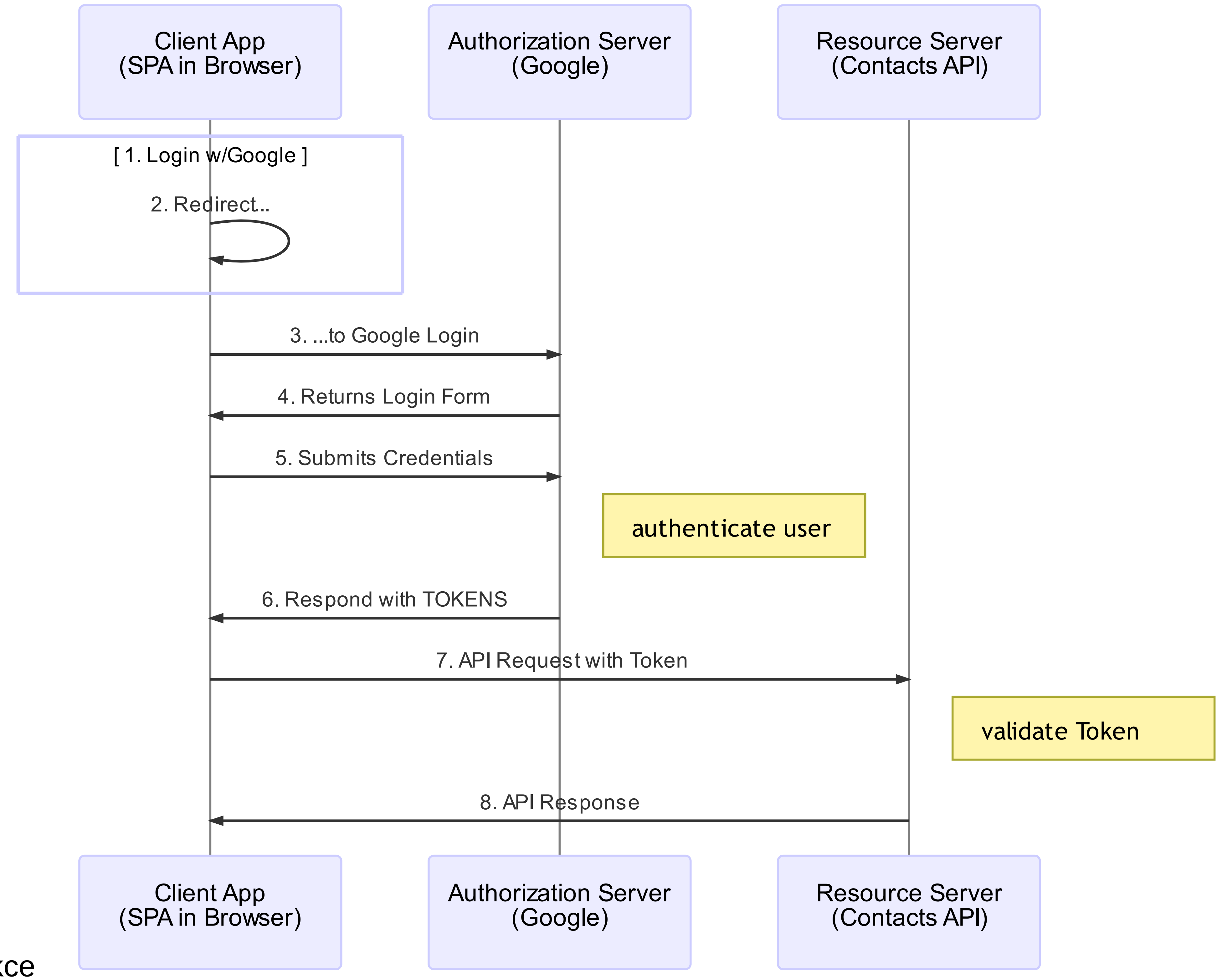
https://goteleport.com/blog/how-oauth-authentication-works/

- OAuth (Open Authentication) is standardized protocol for Authentication & Authorization of services/apps/devices
- OAuth 2.0 based on RFC6749 & RFC6750
- Often used to login through third parties ➔ Federated Authentication
- Involves 4 parties:
  - **Resource Owner** (RO) ➔ User that wants to sign in to app and can grant access through another app (e.g. Google/Twitter/Facebook/….)
  - **App/Client** ➔ needs data from third party service
  - **Authorization Server** (AS) ➔ Authenticates RO and generates access token for app/client
  - **Resource Server** (RS) ➔ holds the data

Authentication – OAuth

https://oauth.net/2/
https://developer.okta.com/blog/2019/08/22/okta-authjs-pkce

- Implicit flow:



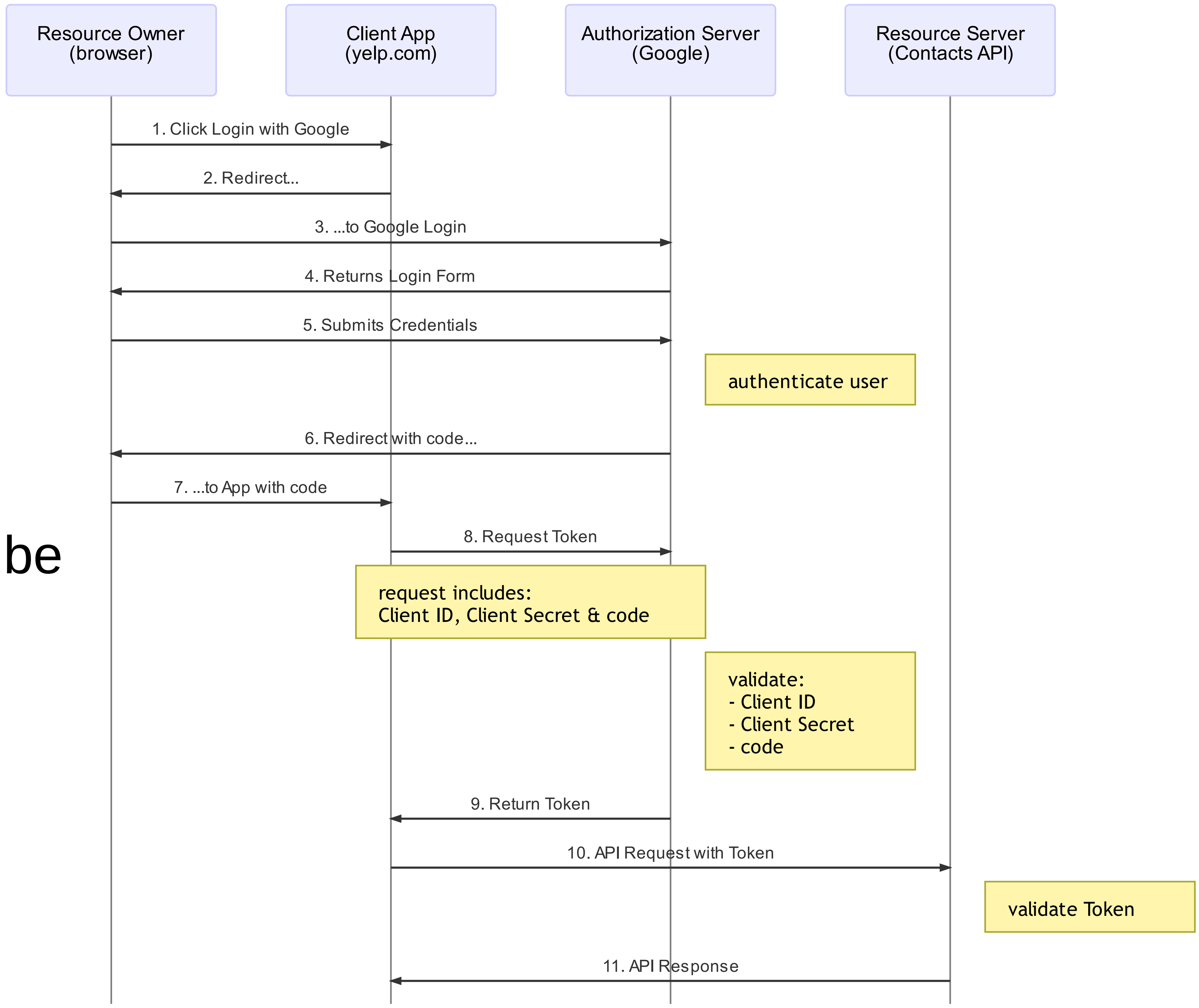
https://developer.okta.com/blog/2019/08/22/okta-authjs-pkce

https://oauth.net/2/

https://developer.okta.com/blog/2019/08/22/okta-authjs-pkce

Authentication – OAuth

- Auth Code flow:
- To add an additional layer of security – every app needs to obtain a **client id & client secret** from the AS/RS before requesting data….
- The client id & client secrets can be withdrawn/blacklisted
- Problem?
  - Client Secret can get stolen



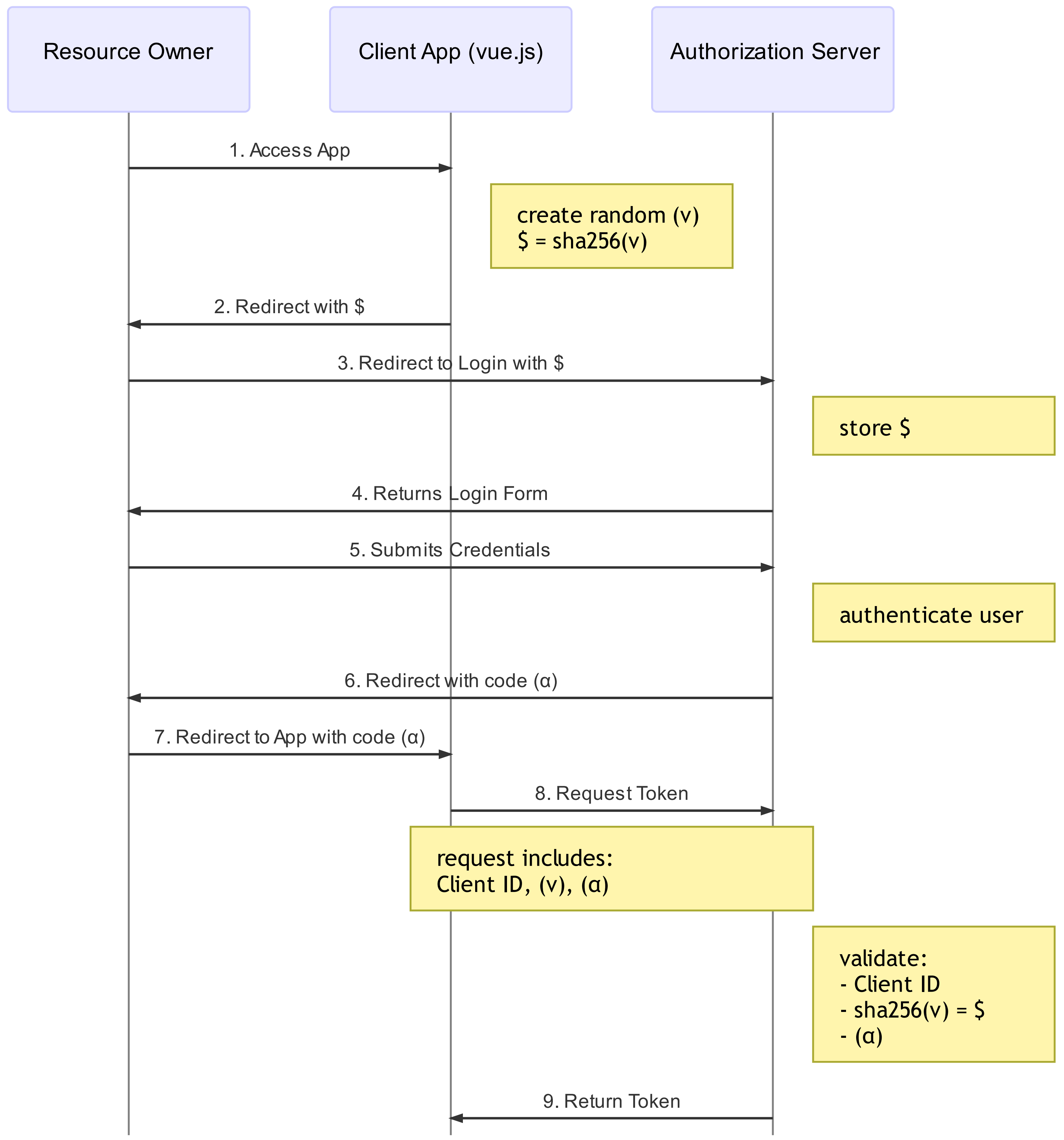
https://developer.okta.com/blog/2019/08/22/okta-authjs-pkce

Authentication – OAuth

https://oauth.net/2/

https://developer.okta.com/blog/2019/08/22/okta-authjs-pkce

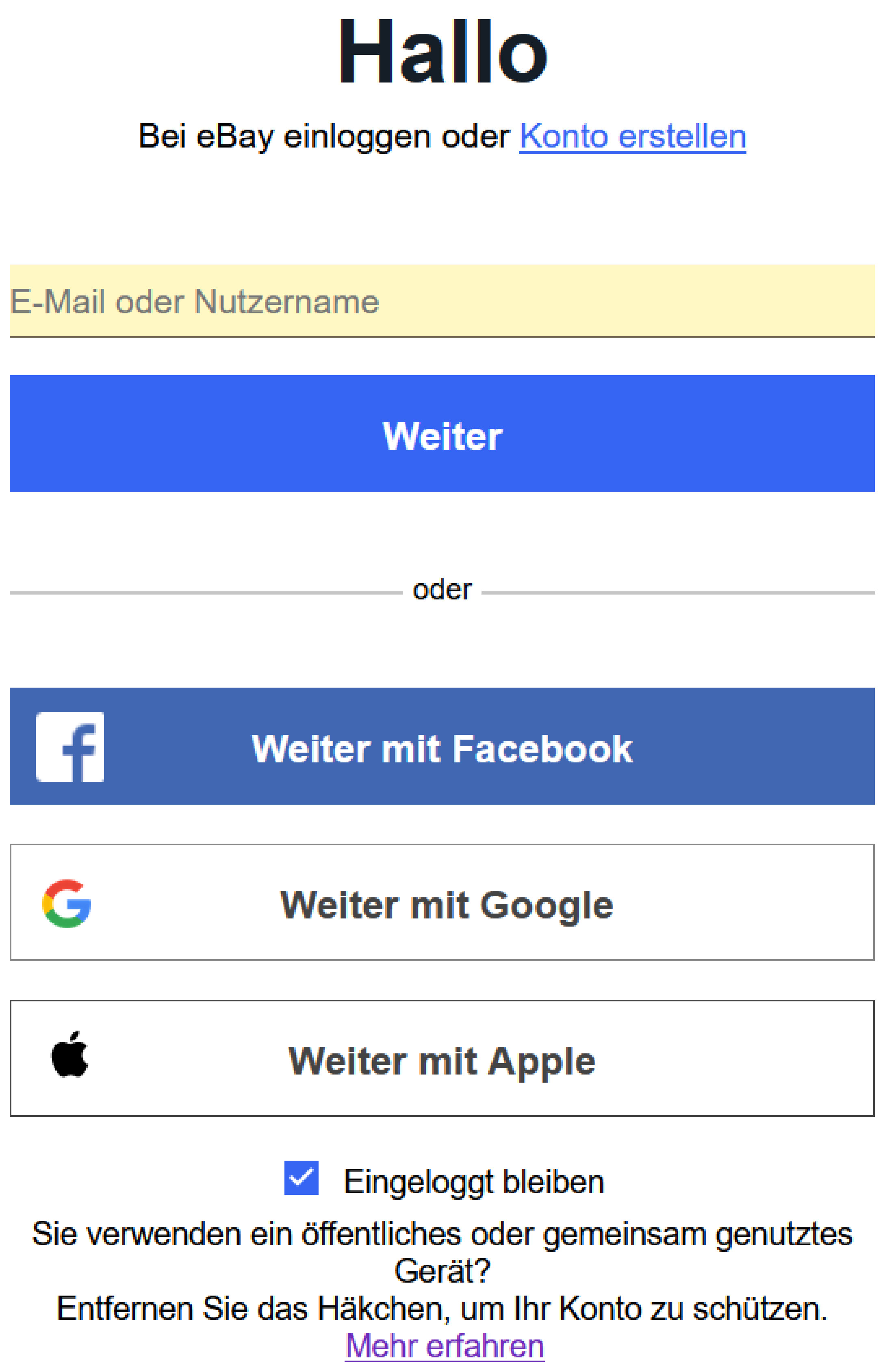
- PKCE flow:
- Method especially for apps and SPA
- Instead of using a fixed **client secret** the client app is generating random tokens called **Code Verifier**



https://developer.okta.com/blog/2019/08/22/okta-authjs-pkce

Authentication – OAuth

- Especially to implement SSO – Single Sign On into apps
- Many implementations are using OpenID Connect


Hallo

Bei eBay einloggen oder Konto erstellen

E-Mail oder Nutzername

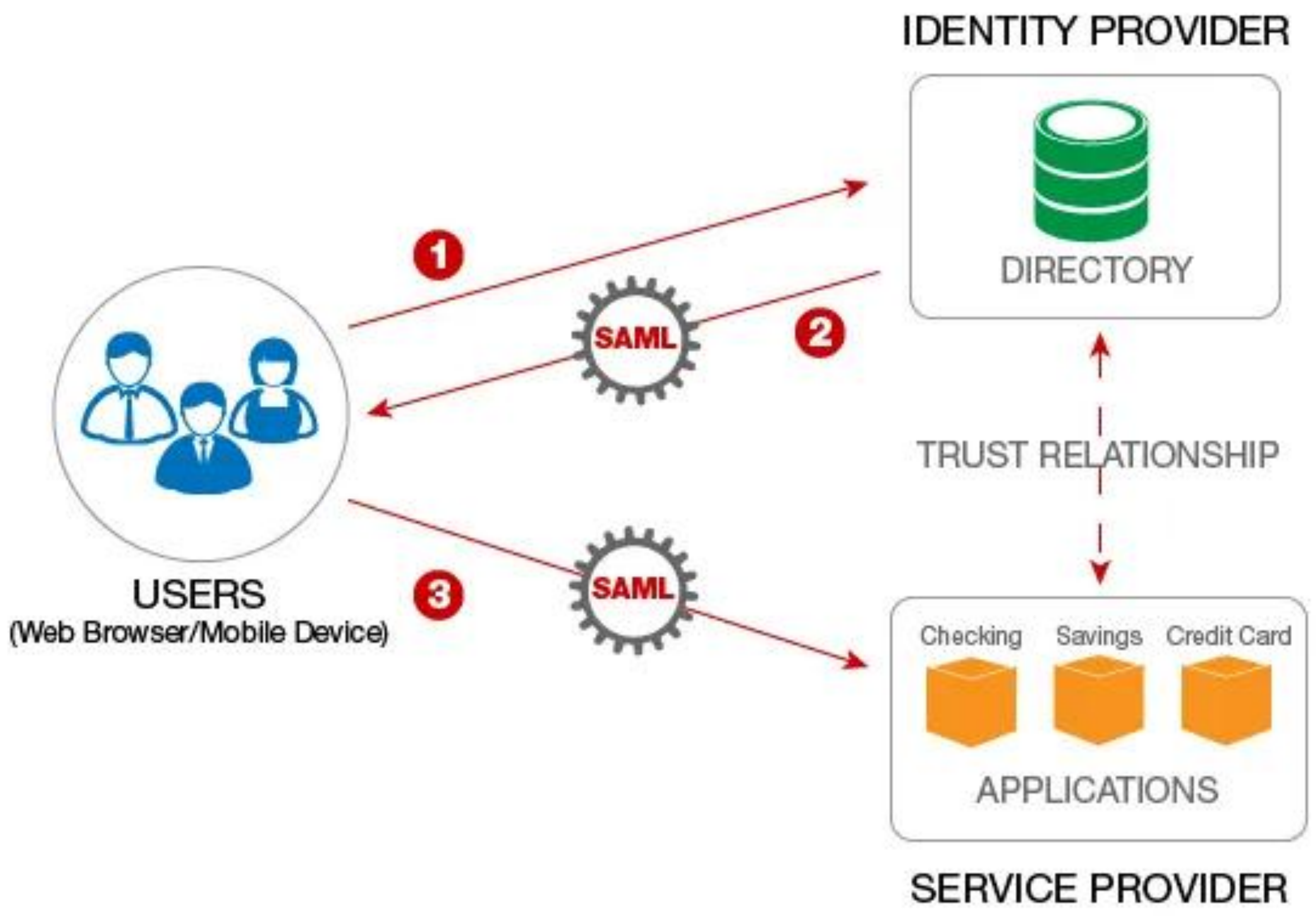
[✓] Eingeloggt bleiben

Sie verwenden ein öffentliches oder gemeinsam genutztes Gerät?
Entfernen Sie das Häkchen, um Ihr Konto zu schützen.
Mehr erfahren

Authentication – SAML

https://medium.com/brightlab-techblog/single-sign-on-sso-saml-authentication-explained-1e463b9168cb

- **S**ecurity **A**ssertion **M**arkup **L**anguage is an XML-based single sign-on standard
- There are three parties involved in the login procedure:
  - Principle/Subject (**User/Client**) => wants to access services from SP
  - Service Provider (**SP**) => offers services after the authentication
  - Identity Provider (**IP**) => has the login information of the User
- Shibboleth at the FHV is an extended variation of SAML



Identity Framework
JWT & RBAC

Authentication – JWT & RBAC Example

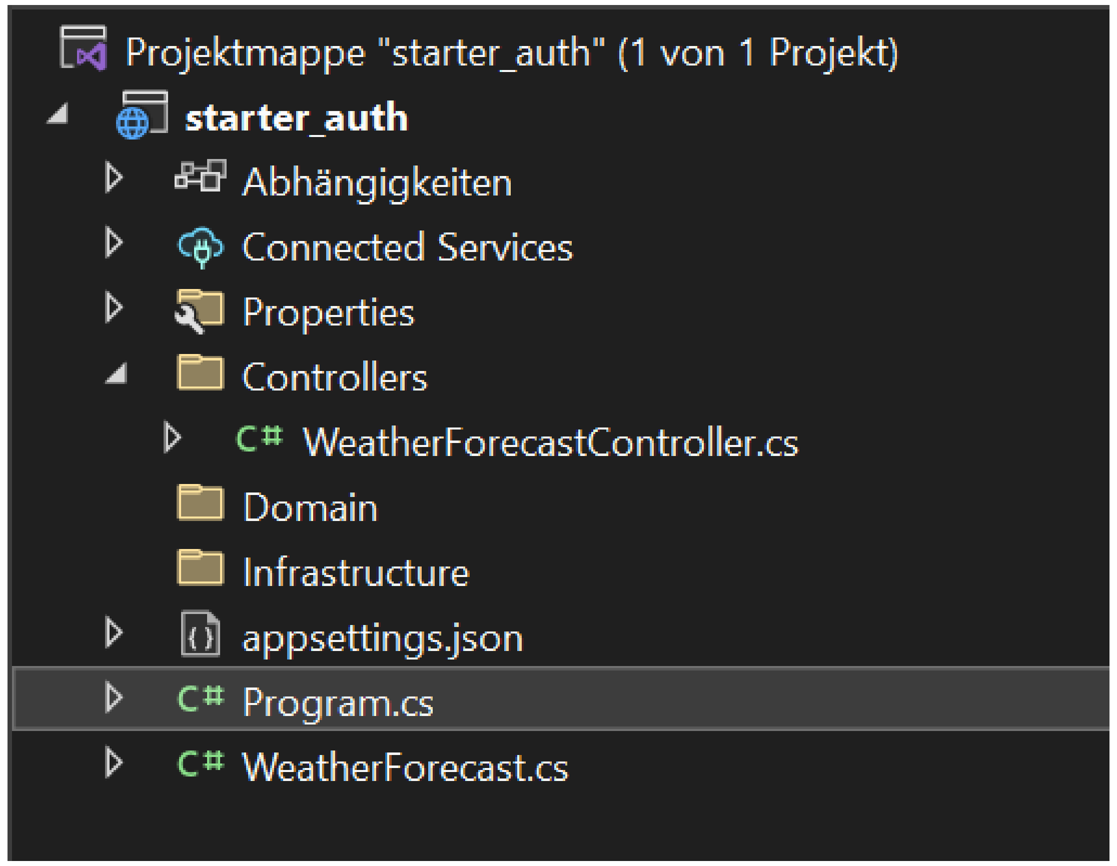
- To make use of simple JWT Authentication & RBAC you need to setup the WebApi project correctly!
- Create Asp.Net Core Web application and install necessary NuGet packages:

```
mkdir starter_auth
cd starter_auth
dotnet new webapi –f net8.0 --no-https --use-controllers
dotnet new sln
dotnet sln add starter_auth.csproj
dotnet add package Microsoft.EntityFrameworkCore.Design –v 8.0.13
dotnet add package Pomelo.EntityFrameworkCore.MySql –v 8.0.2
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer –v 8.0.13
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore –v 8.0.13
dotnet add package Swashbuckle.AspNetCore -v 7.2.0
```

Authentication – JWT & RBAC Example

- First model needs to be the User Model which holds the credentials of the user.
- The model contains all necessary properties and methods by just extending/inheriting from the IdentityUser class
- The rest of the model can stay empty

```
public class ApplicationUser : IdentityUser
{
}
```



```
Projektmappe "starter_auth" (1 von 1 Projekt)
  starter_auth
    Abhängigkeiten
    Connected Services
    Properties
    Controllers
      WeatherForecastController.cs
    Domain
    Infrastructure
    appsettings.json
    Program.cs
    WeatherForecast.cs
```

Authentication – JWT & RBAC Example

- The DBContext is now a special **IdentityDbContext** to automatically include the necessary entities for authentication & authorization

```
public class StarterDbContext : IdentityDbContext<ApplicationUser>
{
    public StarterDbContext(DbContextOptions<StarterDbContext> options) : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
    }
}
```

Authentication – JWT & RBAC Example

- Setup the **appsettings.json** with **connectionstring** and the **jwt-secret**

```json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "dbconnection": "DataSource=localhost;DataBase=starterauth;UserID=root;"
  },
  "JwtSettings": {
    "Secret": "RandomButSecureStringThatYouShouldNotShareWithAnyone"
  },
  "AllowedHosts": "*"
}
```

Authentication – JWT & RBAC Example

Services & App need to be configured correctly to make use of the IdentityModel and to accept JWT

```
var connectionString = builder.Configuration.GetConnectionString("dbconnection");

builder.Services.AddDbContext<StarterDbContext>(
    options => options.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString))
);

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
    {
        options.SignIn.RequireConfirmedAccount = false;
        options.User.RequireUniqueEmail = true;
        options.Password.RequireDigit = false;
        options.Password.RequiredLength = 6;
        options.Password.RequireNonAlphanumeric = false;
        options.Password.RequireUppercase = false;
        options.Password.RequireLowercase = false;
    }).AddEntityFrameworkStores<StarterDbContext>();
```

Authentication – JWT & RBAC Example

Services & App need to be configured correctly to make use of the IdentityModel and to accept JWT

```
builder.Services.AddAuthentication(a =>
{
    a.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    a.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    a.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(opt =>
{
    opt.TokenValidationParameters = new TokenValidationParameters
    {
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.ASCII.GetBytes(
                builder.Configuration.GetSection("JwtSettings")["Secret"]!
                )),
        ValidateIssuer = false,
        ValidateAudience = false,
        RequireExpirationTime = false,
        ValidateLifetime = true
    };
});
```

Authentication – JWT & RBAC Example

To Integrate the Authentication functionality into Swagger UI:

```
builder.Services.AddSwaggerGen(opt =>
{
    opt.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme() {
        Name = "Authorization",
        Type = SecuritySchemeType.ApiKey,
        Scheme = "Bearer",
        BearerFormat = "JWT",
        In = ParameterLocation.Header,
        Description = "Enter 'Bearer' [space] and then your valid token in the text input below.\r\n\r\nExample: \"Bearer 
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9\""
    });
    opt.AddSecurityRequirement(new OpenApiSecurityRequirement
                {
                    {
                          new OpenApiSecurityScheme
                            {
                                Reference = new OpenApiReference
                                {
                                    Type = ReferenceType.SecurityScheme,
                                    Id = "Bearer"
                                }
                            }, new string[] {}
                    }
                });
});
```

Authentication – JWT & RBAC Example

You need to tell the app to use first Authentication and afterwards Authorization

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

Authentication – JWT & RBAC Example

Now we can create three DTOs which contain information about login, register and response

```
public class RegisterDTO
    {
        [Required(ErrorMessage = "User Name is required")]
        public string Username { get; set; }

        [Required(ErrorMessage = "Email is required")]
        public string Email { get; set; }

        [Required(ErrorMessage = "Password is required")]
        public string Password { get; set; }
    }
```

```
public class LoginDTO
    {
        [Required(ErrorMessage = "User Name is required")]
        public string Username { get; set; }

        [Required(ErrorMessage = "Password is required")]
        public string Password { get; set; }
    }
```

```
public class ResponseDTO
    {
        public string Status { get; set; }
        public string Message { get; set; }
    }
```

Authentication – JWT & RBAC Example

Now we need a Controller to handle logins (and registrations)

```
private readonly UserManager<ApplicationUser> userManager;
private readonly IConfiguration configuration;

public AuthenticateController(UserManager<ApplicationUser>
userManager, IConfiguration configuration)
{
       this.userManager = userManager;
       this.configuration = configuration;
}
```

Authentication – JWT & RBAC Example

```
[HttpPost]
[Route("login")]
public async Task<IActionResult> Login([FromBody] LoginDTO 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(JwtRegisteredClaimNames.Sub, user.Id),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
            };

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

Authentication – JWT & RBAC Example

```
            var authSigningKey = new SymmetricSecurityKey(
            Encoding.ASCII.GetBytes(
                this.configuration.GetSection("JwtSettings")["Secret"]
            ));
            var token = new JwtSecurityToken(
                expires: DateTime.Now.AddMinutes(15),
                claims: authClaims,
                signingCredentials: new SigningCredentials(
                      authSigningKey, SecurityAlgorithms.HmacSha256)
                );
            return Ok(new
            {
                token = new JwtSecurityTokenHandler().WriteToken(token),
                expiration = token.ValidTo
            });
        }
        return Unauthorized();
    }
```

Authentication – JWT & RBAC Example

```
[HttpPost]
[Route("register")]
public async Task<IActionResult> Register([FromBody] RegisterDTO model)
{
   var userExists = await userManager.FindByNameAsync(model.Username);
   if (userExists != null)
       return StatusCode(StatusCodes.Status409Conflict, new ResponseDTO { Status = "Error",
Message = "User already exists!" });

   ApplicationUser user = new ApplicationUser()
   {
       Email = model.Email,
       SecurityStamp = Guid.NewGuid().ToString(),
       UserName = model.Username
   }; //no role assigned
    IdentityResult result = await userManager.CreateAsync(user, model.Password);
    if (!result.Succeeded)
        return StatusCode(StatusCodes.Status500InternalServerError, new ResponseDTO { Status =
"Error", Message = "User creation failed! Please check user details and try again." });

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

Authentication – JWT & RBAC Example

- ♦ We created the IdentityUser model
- ♦ We made a specific DBContext to handle the Identity Entity
- ♦ We setup the appsettings to hold the connectionstring & JWT secret
- ♦ We added and configured the according services to cope with JWT authentication & authorization
- ♦ New models to handle login, registration and responses
- ♦ Controller to allow clients to register and login

Next Steps:

- Setup migrations and update DB
- Seed the DB with core data
- Define controller-methods that use authorization!

Authentication – JWT & RBAC Example

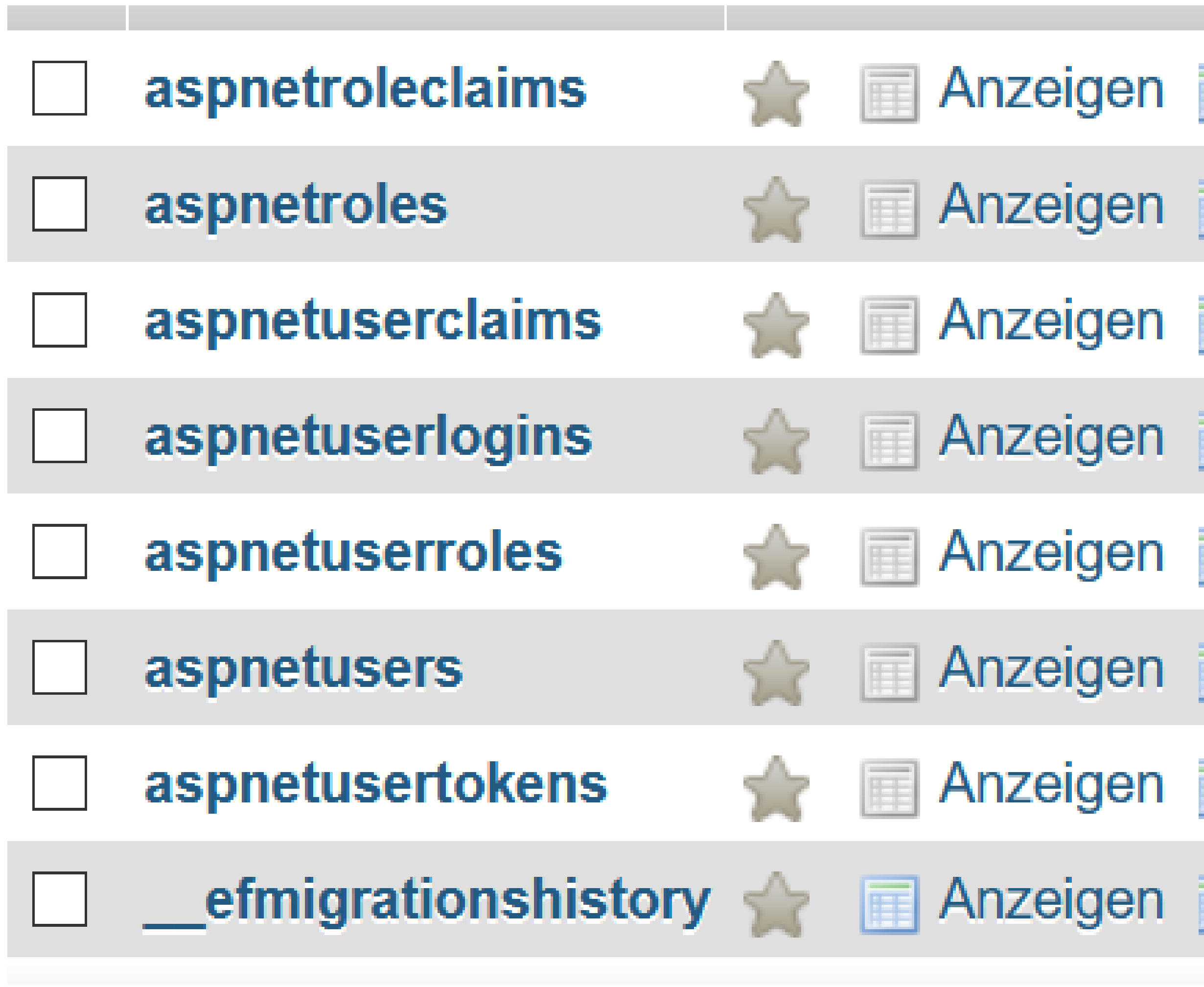
Using the "internal" Visual Studio tools:

```
dotnet add package Microsoft.EntityFrameworkCore.Tools
Add-Migration InitialCreate
Update-Database
```

Using the new cross platform tools:

```
dotnet new tool-manifest
dotnet tool install
--local dotnet-ef --version 8.0.3
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Make sure that DB already exists!



☐	aspnetroleclaims	Anzeigen
☐	aspnetroles	Anzeigen
☐	aspnetuserclaims	Anzeigen
☐	aspnetuserlogins	Anzeigen
☐	aspnetuserroles	Anzeigen
☐	aspnetusers	Anzeigen
☐	aspnetusertokens	Anzeigen
☐	__efmigrationshistory	Anzeigen

Authentication – JWT & RBAC Example

- ♦ Seeder fills the Database with either fake data or necessary core data for the application
- ♦ We want to create two users
  - Admin User (all roles)
  - Normal User (User role)
- ♦ We also want to add three roles “SuperAdmin”, “Admin” and “User” and assign them to the users

- ♦ Create new SeedDB class in Infrastructure folder

Authentication – JWT & RBAC Example

```
public class SeedDB
   {
       public static async void Initialize(IServiceProvider serviceProvider)
       {
           var context = serviceProvider.GetRequiredService<StarterDbContext>();
           var userManager = serviceProvider.GetRequiredService<UserManager<ApplicationUser>>();

           string[] roles = new string[] { "SuperAdmin", "Admin", "User" };

           foreach (string role in roles)
           {
               var roleStore = new RoleStore<IdentityRole>(context);


               if (!context.Roles.Any(r => r.Name == role))
               {
                   await roleStore.CreateAsync(new IdentityRole()
                   {
                       Name = role,
                       NormalizedName = role.ToUpper()
                   });
               }
           }
```

Authentication – JWT & RBAC Example

```
        if (!context.Users.Any())//Accounts should be checked here directly
        {
            ApplicationUser userAdmin = new ApplicationUser()
            {
                Email = "admin@fhv.at",
                UserName = "admin",
                SecurityStamp = Guid.NewGuid().ToString() ,
                NormalizedUserName = "ADMIN"
            };
            await userManager.CreateAsync(userAdmin, "1234567Aa!");
            await userManager.AddToRolesAsync(userAdmin, roles);

            ApplicationUser userSimple = new ApplicationUser()
            {
                Email = "user@fhv.at",
                UserName = "user",
                SecurityStamp = Guid.NewGuid().ToString(),
                NormalizedUserName = "user".ToUpper()
            };

            await userManager.CreateAsync(userSimple, "1234567Aa!");
            await userManager.AddToRoleAsync(userSimple, "User");
            await context.SaveChangesAsync();
        }
    }
}
```

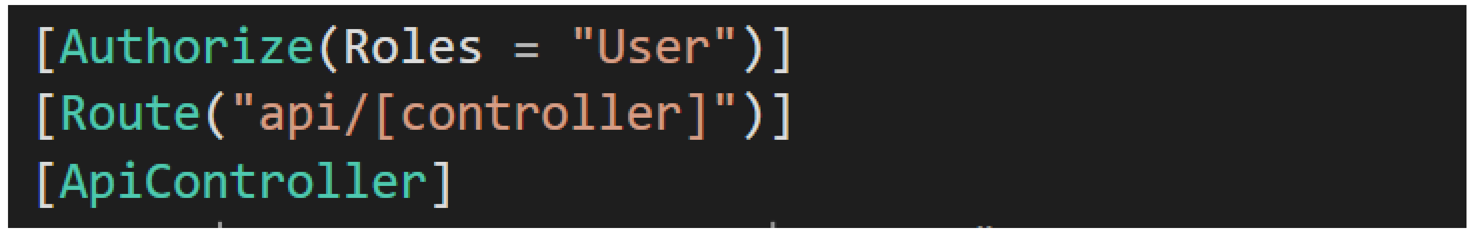
Authentication – JWT & RBAC Example

- Last step is to activate the Seeder when running the application:

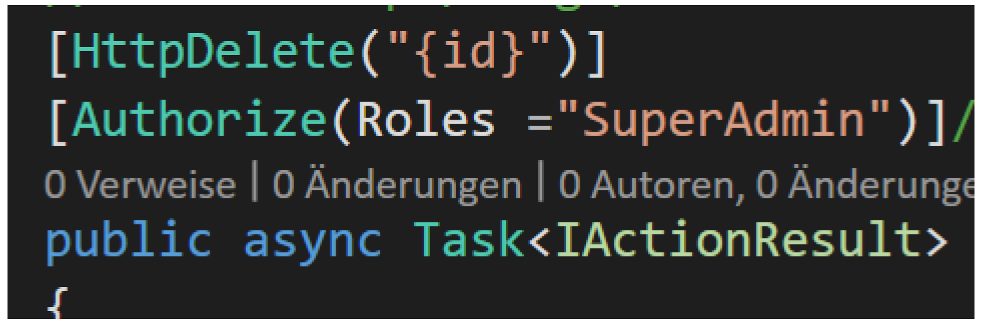
```
SeedDB.Initialize(
    app.Services.GetRequiredService<IServiceProvider>().CreateScope().ServiceProvider
);
```

Authentication – JWT & RBAC Example

- Now you can use the “Authorize” annotation in controller to annotate all methods that need any kind of authorization
- You can also define the roles that the authenticated user has to have to make that api call.
- Multiple roles can be defined via comma separation.

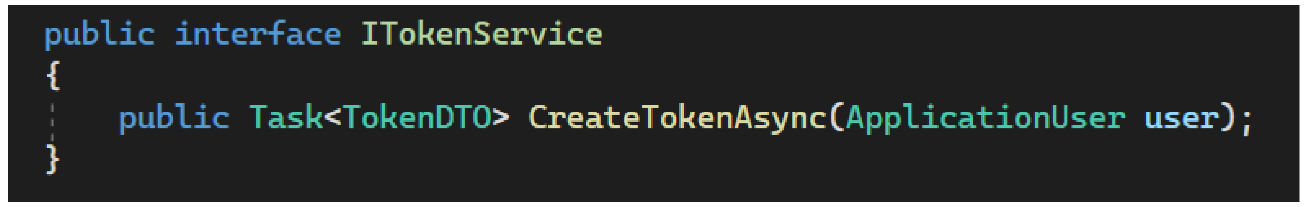


```
[Authorize(Roles = "User")]
[Route("api/[controller]")]
[ApiController]
```

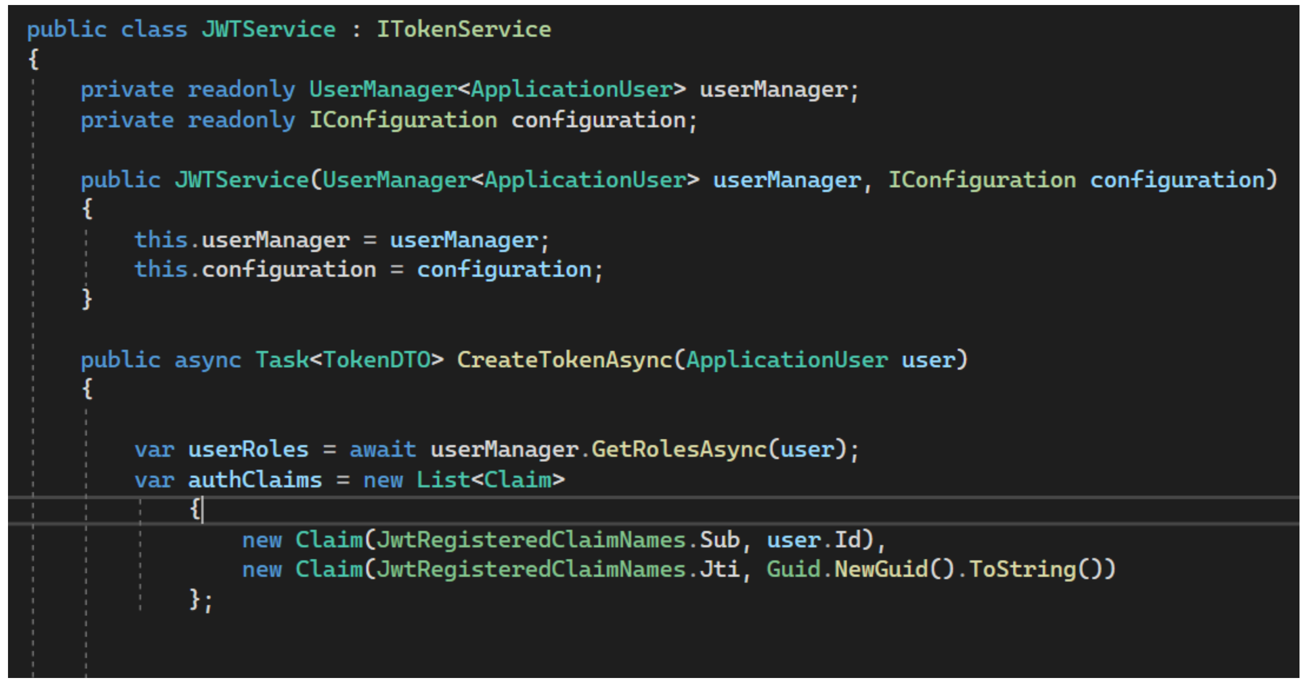


```
[HttpDelete("{id}")]
[Authorize(Roles ="SuperAdmin")]/
0 Verweise | 0 Änderungen | 0 Autoren, 0 Änderunge
public async Task<IActionResult>
{
```

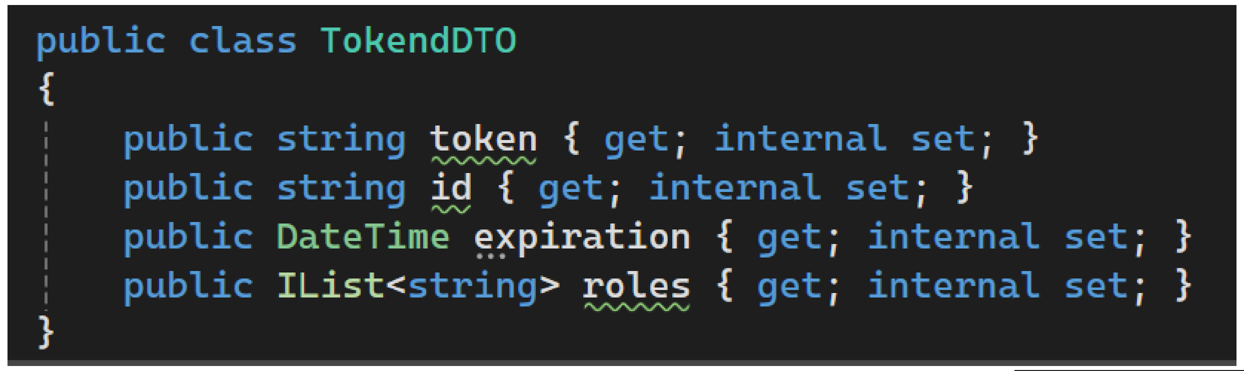
Optimization: Make use of Services



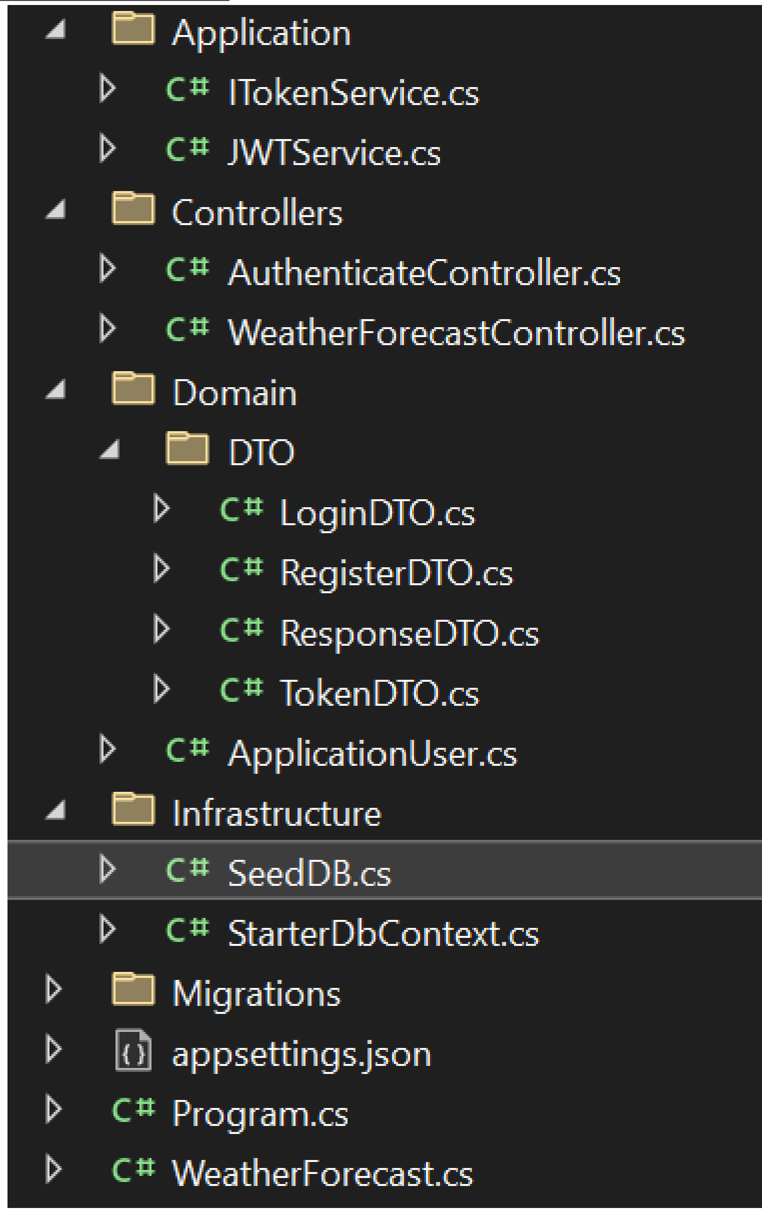
```
public interface ITokenService
{
    public Task<TokenDTO> CreateTokenAsync(ApplicationUser user);
}
```



```
public class JWTService : ITokenService
{
    private readonly UserManager<ApplicationUser> userManager;
    private readonly IConfiguration configuration;

    public JWTService(UserManager<ApplicationUser> userManager, IConfiguration configuration)
    {
        this.userManager = userManager;
        this.configuration = configuration;
    }

    public async Task<TokenDTO> CreateTokenAsync(ApplicationUser user)
    {

        var userRoles = await userManager.GetRolesAsync(user);
        var authClaims = new List<Claim>
            {
                new Claim(JwtRegisteredClaimNames.Sub, user.Id),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
            };
```

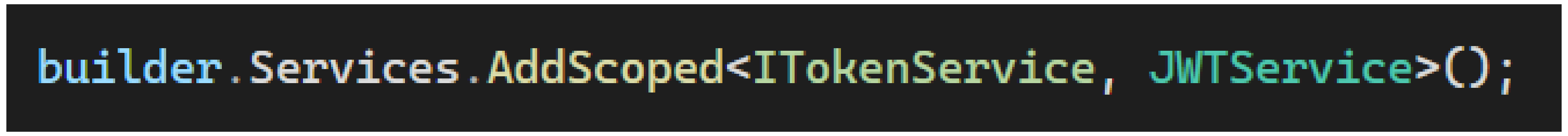


```
public class TokendDTO
{
    public string token { get; internal set; }
    public string id { get; internal set; }
    public DateTime expiration { get; internal set; }
    public IList<string> roles { get; internal set; }
}
```



```
◢ Application
    ▷ C# ITokenService.cs
    ▷ C# JWTService.cs
◢ Controllers
    ▷ C# AuthenticateController.cs
    ▷ C# WeatherForecastController.cs
◢ Domain
    ◢ DTO
        ▷ C# LoginDTO.cs
        ▷ C# RegisterDTO.cs
        ▷ C# ResponseDTO.cs
        ▷ C# TokenDTO.cs
    ▷ C# ApplicationUser.cs
◢ Infrastructure
    ▷ C# SeedDB.cs
    ▷ C# StarterDbContext.cs
▷ Migrations
▷ appsettings.json
▷ C# Program.cs
▷ C# WeatherForecast.cs
```

Create an **TokenService** Interface and implement the Tokencreation based on the JWT-Principle! Add the service in the WebApplicationBuilder in Program.cs



```
builder.Services.AddScoped<ITokenService, JWTService>();
```

Authentication – JWT & RBAC Example – Additional Infos

- Sample implementation of refresh token:
- https://www.c-sharpcorner.com/article/jwt-authentication-with-refresh-tokens-in-net-6-0/
- https://code-maze.com/using-refresh-tokens-in-asp-net-core-authentication/
- https://learn.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-7.0
- https://auth0.com/blog/whats-new-in-dotnet-7-for-authentication-and-authorization/
- https://medium.com/geekculture/how-to-add-jwt-authentication-to-an-asp-net-core-api-84e469e9f019
- https://markjames.dev/blog/jwt-authorization-asp-net-core