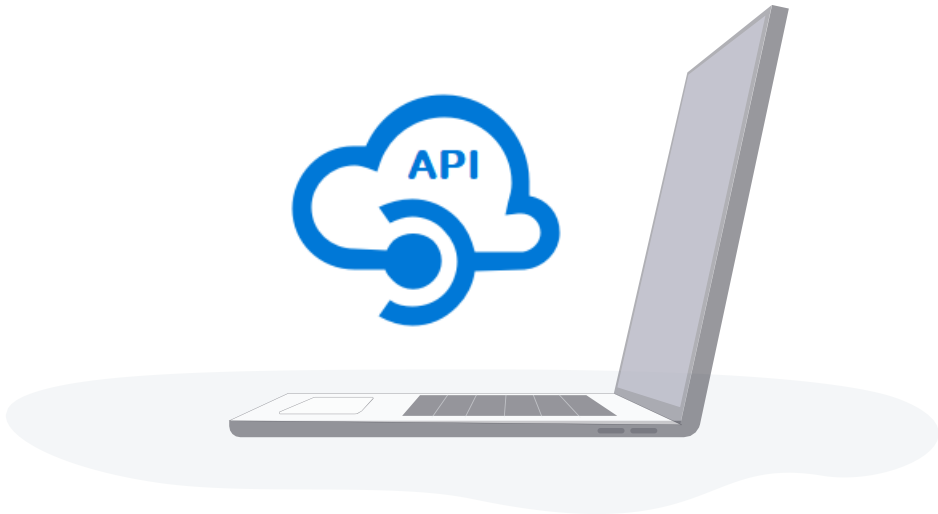
Harmony IT Solution

Web Application Programming Interface (API)

Tahaluf Training Center 2022

Harmony IT Solution

1. Authentication VS Authorization
2. JSON Web Token (JWT)
3. Create LOGIN using JWT Token



Authentication VS Authorization

Authentication VS Authorization

Authentication verifies the user before allowing them access, and **authorization** determines what they can do once the system has granted them access.

Verifying that someone or anything is who they claim they are is done through the **authentication** procedure. To secure access to a program or its data, technology systems normally require some type of authentication. For example, you often need to enter your login and password in order to access a website or service online. Then, in the background, it checks your entered login and password to a record in its database. The system decides you are a valid user and provides you access if the data you provided matches.

The security procedure known as **authorization** establishes a user's or service's level of access. In technology, authorization is used to provide users or services permission to access certain data or perform specific tasks.

JSON Web Token (JWT)

What is JSON Web Token?

JSON Web Token (JWT) is an open standard (RFC 7519) that specifies a condensed and independent method for sending information securely between parties as a JSON object. Due to its digital signature, this information can be verified and trusted.

Uses of JSON Web Tokens:

1. **Authorization:** The most typical application of JWT is for **authorization**. The JWT will be included in each request once the user logs in, enabling access to the routes, services, and resources that are authorized with that token

Uses of JSON Web Tokens:

2. Information Exchange: Sending **information securely** between parties is made possible by JSON Web Tokens. You can be certain that the senders are who they claim to be since JWTs can be signed. You may also confirm that the content hasn't been altered because the signature is created using the header and the payload.

JSON Web Token structure

1. Header
2. Payload
3. Signature

The three components of a JSON Web Token are separated by dots (.) like the following:
xxxxx.yyyyy.zzzzz

Header

The type of the token, which is JWT, and the signature algorithm being used, such as HMAC SHA256 or RSA, are both typically component included in the header.

HEADER: ALGORITHM & TOKEN TYPE

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

Payload

The payload is the second part of the token, which contains the claims. Claims are statements about a subject (usually the user) and additional information.

PAYLOAD: DATA

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

Signature

The encoded payload, encoded header, a secret, and the algorithm mentioned in the header must all be combined to generate the signature portion.

When a token is signed with a private key, it may also confirm that the sender of the JWT is who they claim to be. The signature is used to ensure that the message wasn't altered along the way.

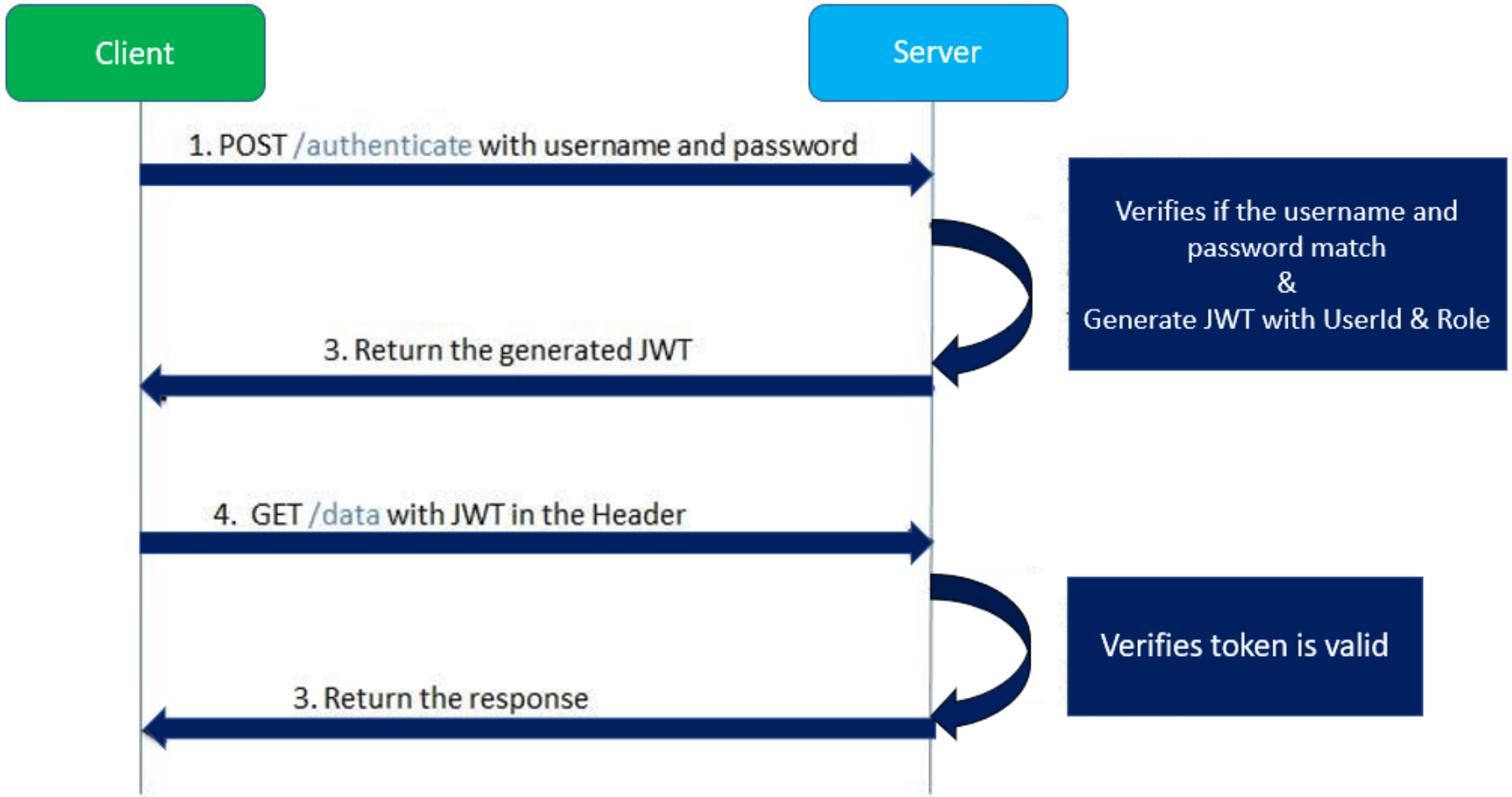
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☑ secret base64 encoded
```

JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThII
oDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ
```

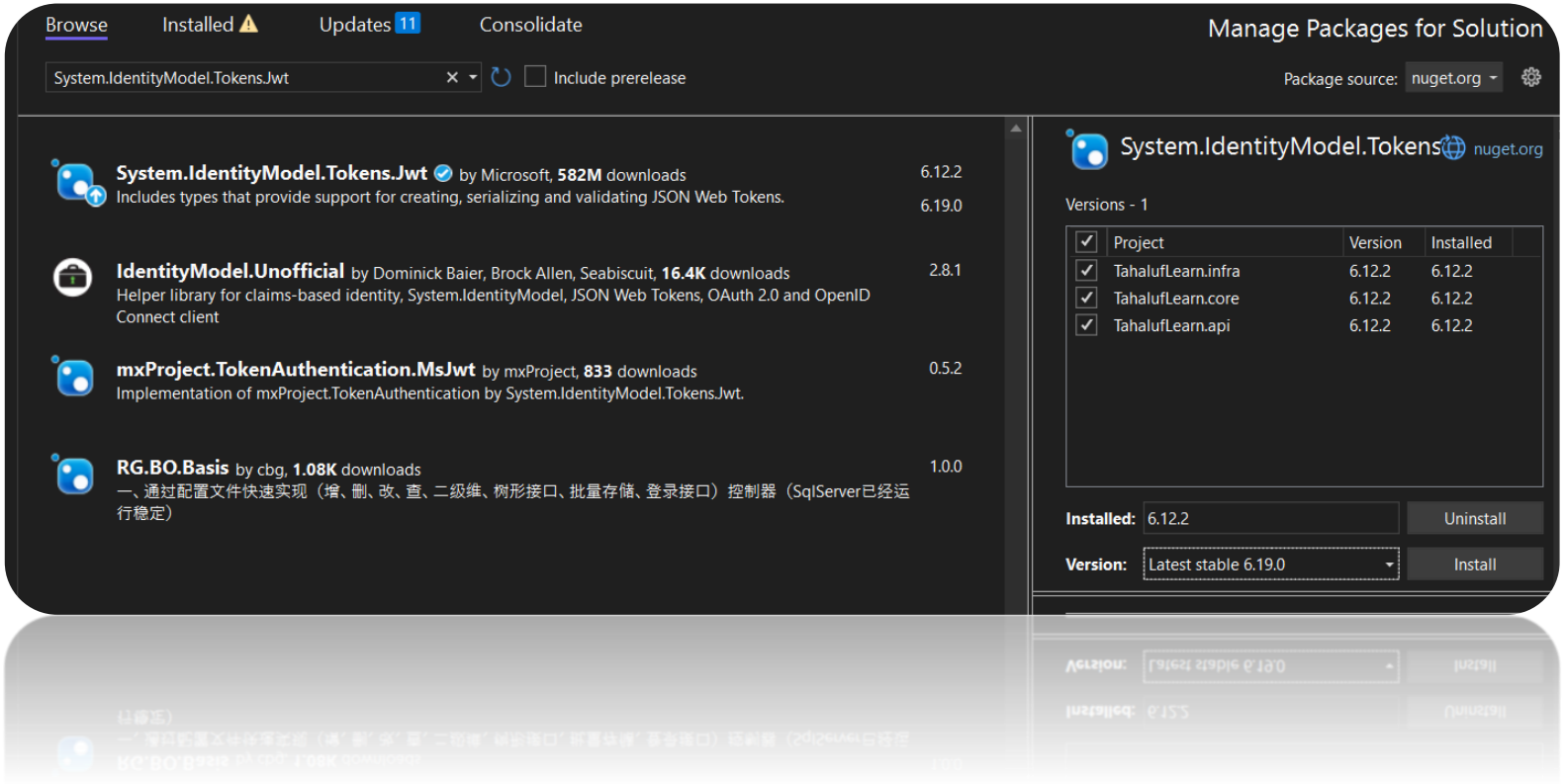
How do JSON Web Tokens work



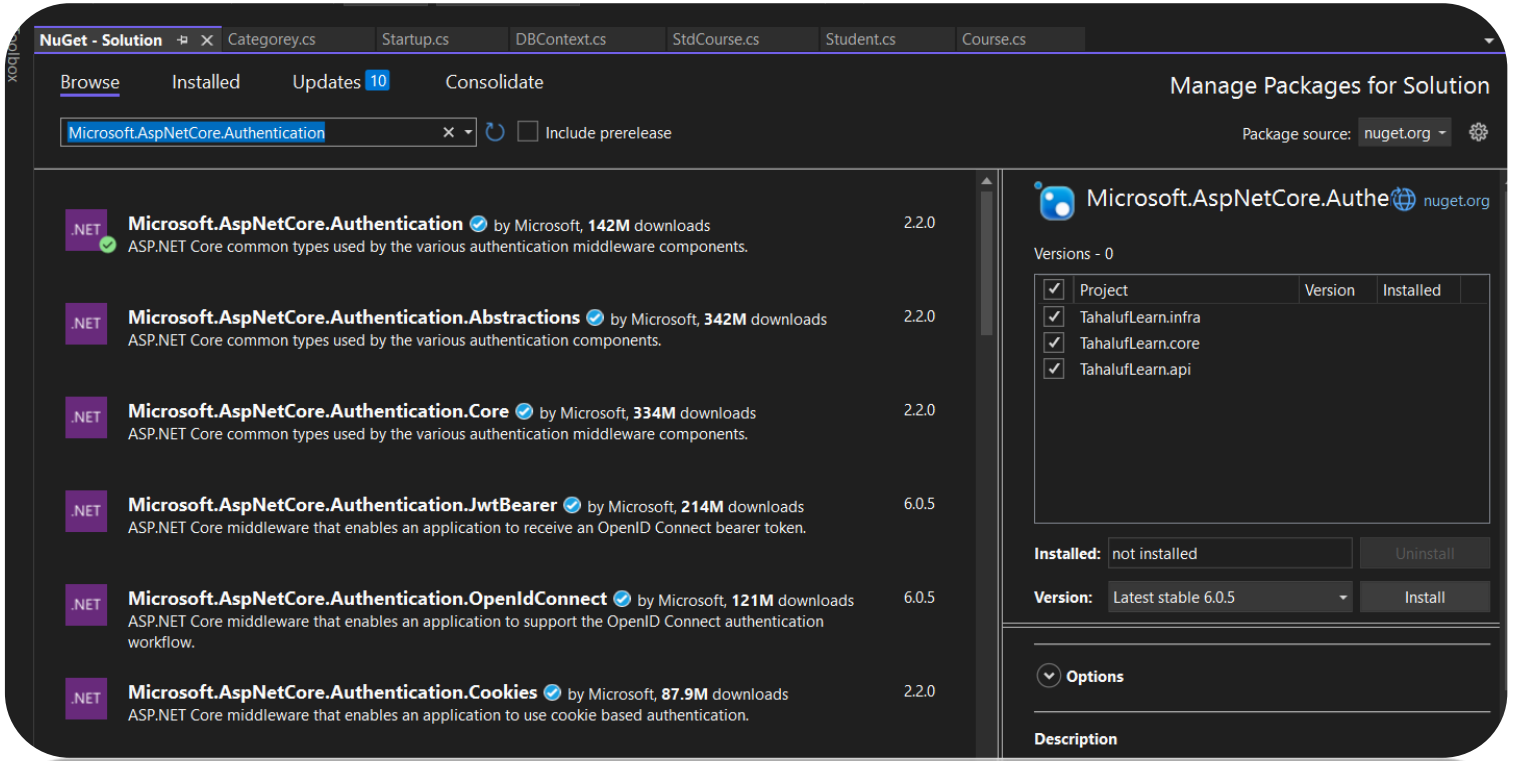
Harmony IT Solution

Create LOGIN using JWT Token

Tools => NuGet Package Manager => Manage NuGet Packages for Solution => System.IdentityModel.Tokens.Jwt



Tools => NuGet Package Manager => Manage NuGet Packages for Solution => Microsoft.AspNetCore.Authentication.JwtBearer



Create Login package on SQL developer :

```
create or replace PACKAGE Login_Package
AS
PROCEDURE User_Login(User_NAME IN VARCHAR,PASS IN VARCHAR);
END Login_Package;
```

```
create or replace PACKAGE body Login_Package
AS
PROCEDURE User_Login(User_NAME IN VARCHAR,PASS IN VARCHAR)
AS
c_all SYS_REFCURSOR;
BEGIN
open c_all for
SELECT USERNAME,roleid FROM LOGIN WHERE USERNAME=User_NAME
AND PASSWORD=PASS;
DBMS_SQL.RETURN_RESULT(c_all);
end User_Login;
END Login_Package;
```

Startup => ConfigureServices => Add the following:

```
services.AddAuthentication(opt => {
                opt.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
                opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            })
    .AddJwtBearer(options =>
    {
```

```
 options.TokenValidationParameters = new TokenValidationParameters
            {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSecretKey@345"))
            };
    });
```

Startup => Configure => Add the following:

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

TahalufLearn.core => Repository => Create IJWTRepository.cs

```
public interface IJWTRepository
        {
        Login Auth(Login login);

        }
```

TahalufLearn.core => Service => Create IJWTService.cs

```
public interface IJWTService
        {
        string Auth(Login login);

        }
```

TahalufLearn.Infra => Repositry => Create JWTRepository.cs

```
public class JWTRepository : IJWTRepository
        {
private readonly IDBContext dBContext;

        public JWTRepository(IDBContext dBContext)
            {
            this.dBContext = dBContext;
            }
```

```
 public Login Auth(Login login)
            {

            var p = new DynamicParameters();
            p.Add("User_NAME", login.Username, dbType: DbType.String, direction:
ParameterDirection.Input);
            p.Add("PASS", login.Password, dbType: DbType.String, direction:
ParameterDirection.Input);
            IEnumerable<Login> result =
dBContext.Connection.Query<Login>("Login_Package.User_Login", p, commandType:
CommandType.StoredProcedure);
            return result.FirstOrDefault();
            }
```

TahalufLearn.Infra => Service => Create JWTService.cs

```
private readonly IJWTRepository reposiytory;
    public JWTService(IJWTRepository reposiytory)
        {
        this.reposiytory = reposiytory;
        }
```

```
public string Auth(Login login)
            {
            var result = reposiytory.Auth(login);
            if (result == null)
                {
                return null;
                }
            else
                {
                var secretKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSecretKey@345"));
                var signinCredentials = new SigningCredentials(secretKey,
SecurityAlgorithms.HmacSha256);
                var claims = new List<Claim>
```

```
{
        new Claim(ClaimTypes.Name, result.Username),
        new Claim(ClaimTypes.Role, result.Roleid.ToString())
    };
                var tokeOptions = new JwtSecurityToken(
                    claims: claims,
                    expires: DateTime.Now.AddSeconds(10),
                    signingCredentials: signinCredentials
                );
                var tokenString = new
JwtSecurityTokenHandler().WriteToken(tokeOptions);
                return tokenString;

                }
```

In Startup:

```
services.AddScoped<IJWTRepository, JWTRepository>();
services.AddScoped<IJWTService, JWTService>();
```

TahalufLearn.API => Controller => Create JWTController.cs

```
private readonly IJWTService jwtservice;
        public JWTController(IJWTService jwtservice)
            {
            this.jwtservice = jwtservice;
            }
```

```
[HttpPost]

public IActionResult Auth([FromBody] Login login)
            {
            var token = jwtservice.Auth(login);
            if (token == null)
                {
                return Unauthorized();
                }
            else
                {
                return Ok(token);
                }

            }
```

Harmony IT Solution