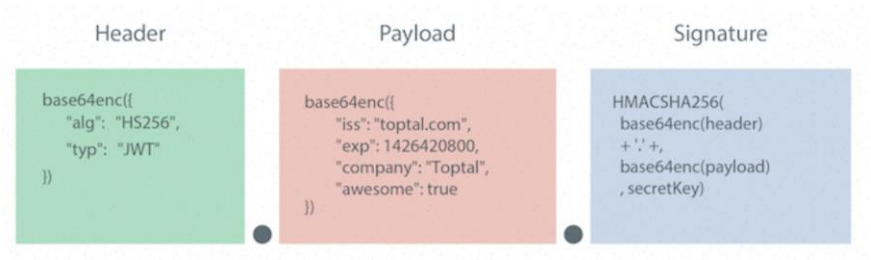
About JWT Token

JWT are an important piece in ensuring trust and security in your application. JWT allow claims, such as user data, to be represented in a secure manner.

A JSON Web Token (JWT) is a JSON object that is defined in RFC 7519 as a safe way to represent a set of information between two parties.

The token is composed of a header, a payload, and a signature.

Format of JWT Token = header.payload.signature



JSON Web Token example:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJpc3MiOiJ0b3B0YWwuY29tIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3RfY2xhaW1zL2lz

X2FkbWluIjp0cnVlLCJjb21wYW55IjoiVG9wdGFsIiwiYXdlc29tZSI6dHJ1ZX0 .

yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK_ZXw

Steps to Create a JWT Token

Step 1. Create the HEADER

The header component of the JWT contains information about how the JWT signature should be computed. The header is a JSON object in the following format:

JWT Header

```
{
 “alg”: “HS256”,
 “typ”: “JWT”
}
```

This JWT Header declares that the encoded object is a JSON Web Token, and that it is signed using the HMAC SHA-256 algorithm.

Once this is base64 encoded, we have the first part of our JWT.

Step 2. Create the PAYLOAD

In the context of JWT, a claim can be defined as a statement about an entity (typically, the user), as well as additional meta data about the token itself. The claim contains the information we want to transmit, and that the server can use to properly handle authentication. There are multiple claims we can provide;

These claims are **not** intended to be **mandatory** but rather to provide a starting point for a set of useful, interoperable claims.

- **iss**: The issuer of the token
- **sub**: The subject of the token
- **aud**: The audience of the token
- **exp**: Token expiration time defined in Unix time
- **nbf**: "Not before" time that identifies the time before which the JWT must not be accepted for processing
- **iat**: "Issued at" time, in Unix time, at which the token was issued
- **jti**: JWT ID claim provides a unique identifier for the JWT

Example Payload

```
{
 "iss": "toptal.com",
 "exp": 1426420800,
 "company": "deccansoft",
 "awesome": true
}
```

Step 3. Create the SIGNATURE

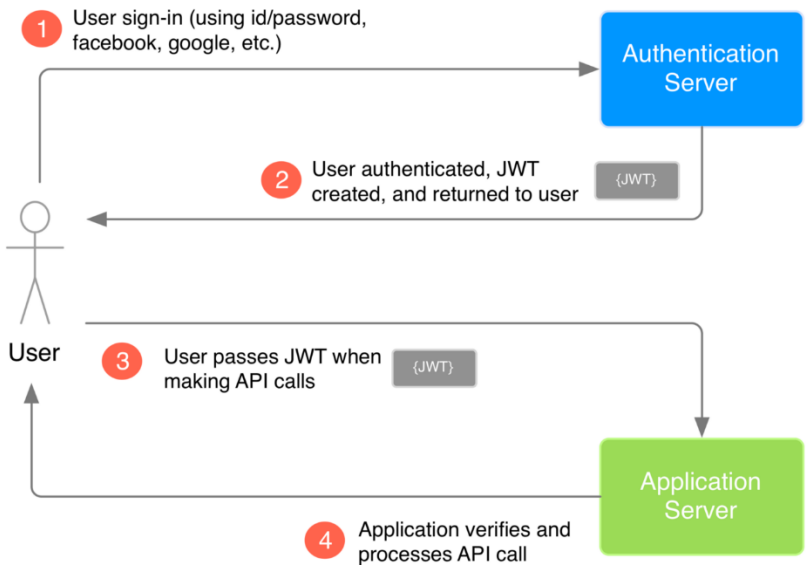
The signature is computed using the following pseudo code:

```
data = base64urlEncode(header) + "." + base64urlEncode( payload )
hashedData = HMACSHA256(data, secret)
```

Step 4: Create JWT Token

```
JWT Token = base64urlEncode( header ) + "." + base64urlEncode( payload ) + base64urlEncode( hashedData )
```

How an application uses JWT to verify the authenticity of a user.



The good news is that authenticating with JWT tokens in ASP.NET Core is straightforward.

Middleware exists in the **Microsoft.AspNetCore.Authentication.JwtBearer** package that does most of the work for us!

```
services.AddAuthentication(options =>
{
   options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme; //Bearer
   options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(x =>
{
   x.RequireHttpsMetadata = false;
   x.SaveToken = true;
   x.TokenValidationParameters = new TokenValidationParameters
   {
      ValidateIssuerSigningKey = true,
      IssuerSigningKey = new SymmetricSecurityKey(key),
      ValidateIssuer = false,
      ValidateAudience = false,
   };
});
```

Example Folder: E:\NET Core\ASP.NET Core\ASP NET Core Course Material\Security\JWT Token

Try this in Postman

1. URL: http://localhost:52629/users/authenticate

   Body = raw / JSON

   ```
   {
     "username": "test",
     "password": "test",
   }
   ```

2. http://localhost:52629/users/getall

   Header: Authorization = Bearer **<Token from Previous Request>**