JWT

5 Eylül 2021 Pazar 15:57

What is JWT? How to implement?

Jwt provide us transmitting information securely as a JSON object between parties.
The service provider does not need to access the database to validate user roles and permissions for each request; data is extracted from the token.

Why JWT is a stateless authentication?

JSON Web Tokens (JWT) are called stateless because the authorizing server doesn't need to keep track of anything, the token is all that's required to verify a token bearer's authorization. In stateless authentication, no need to store user information in the session.

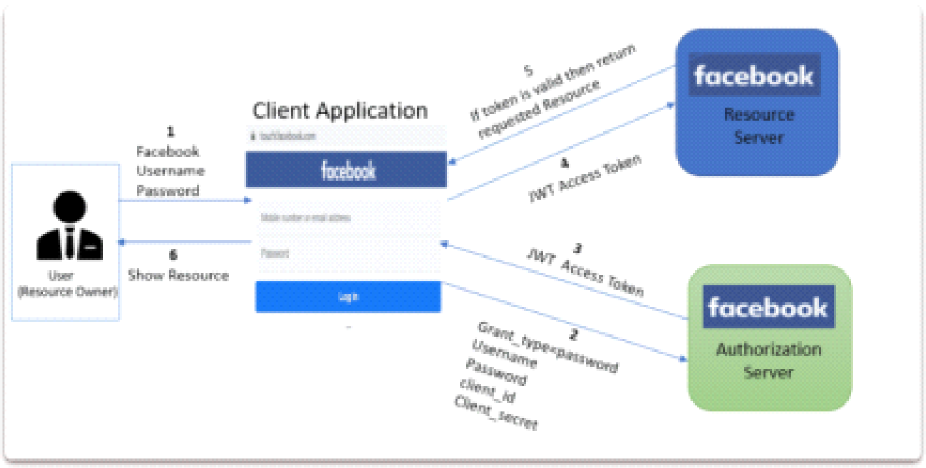
Does JWT token contain password?

The JWT comprises encoded user information as well as a signature that is checked when decoded to confirm that the token has not been tampered with. After the JWT has been confirmed, instead of sending the user their forgotten password, your application can safely allow them to generate a new password.

Advantages of JWT Access Token

We need to call Authorization Server to validate whether provided Access Token is valid or not. However, if we use JWT access token as OAuth token, Resource will not have to call Authorization Server each time to validate the Access Token.

From <https://www.techgeeknext.com/spring-boot-security/web-security-oauth2-jwt-vs-default-access-token>

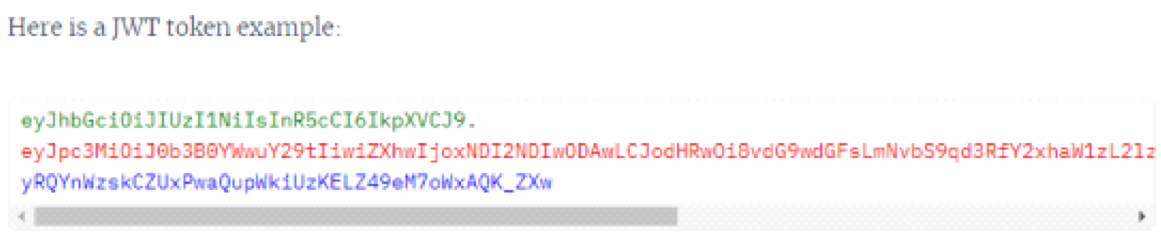


<u>Good Performance</u>: As shown above in diagram no need to call Authorization Server for validating or checking the access token, which will reduce the network call.
<u>Portable:</u> Allow to use multiple backends with single access token.
It is Very <u>Mobile Friendly,</u> because cookies are not required.
JWT contains expiration date as a claim that can be used to determine when the access token is going to expire.
It's very secure way to validate the user information, as it's digitally signed.

What is the structure of JWT?

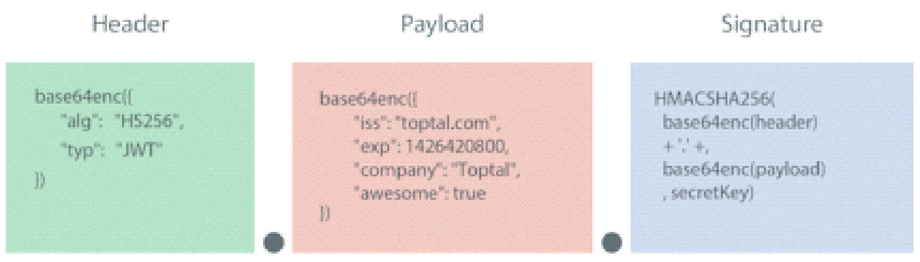
JWT consists of 3 parts - *Header.Payload.Signature*

It generate JWT token as in the form of a.b.c which represent header.payload.signature



Here is a JWT token example:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiJ0b3B0YWwuY29tIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3RfY2xhaW1zL2lz
yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK_ZXw
```



```
base64enc({
    "alg":  "HS256",
    "typ":  "JWT"
})
```

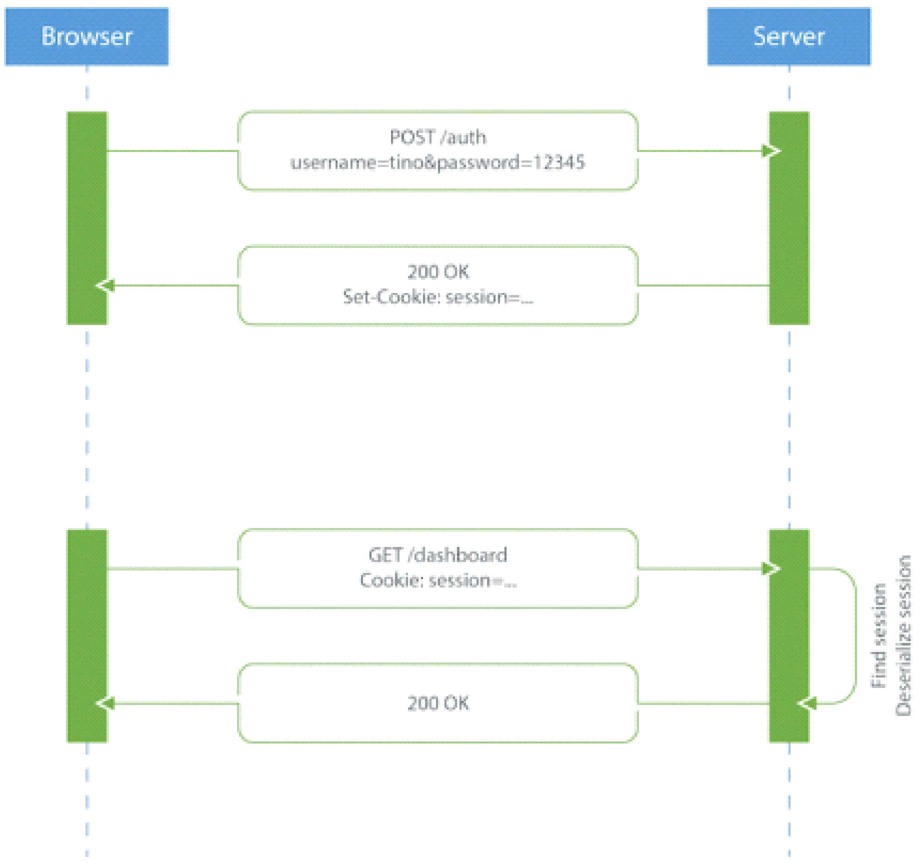
```
base64enc({
    "iss": "toptal.com",
    "exp": 1426420800,
    "company": "Toptal",
    "awesome": true
})
```

```
HMACSHA256(
 base64enc(header)
 + '.' +,
 base64enc(payload)
 , secretKey)
```

Why the Need for Web Tokens?

Before we can see all the benefits of using JWT authentication, we have to look at the way authentication has been done in the past.

- Server-Based Authentication



Because the HTTP protocol is stateless, there needs to be a mechanism for storing user information and a way to authenticate the user on every subsequent request after login. Most websites use cookies for storing user's session ID.

From <https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs>

Drawbacks of Server-Based Authentication

- **Hard to scale**: The server needs to create a session for a user and persist it somewhere on the server. This can be done in memory or in a database. If we have a distributed system, we have to make sure that we use a separate session storage that is not coupled to the application server.

- **Cross-origin request sharing (CORS)**: When using AJAX calls to fetch a resource from another domain ("cross-origin") we could run into problems with forbidden requests because, by default, HTTP requests don't include cookies on cross-origin requests.

- Token-Based Authentication

  - **Stateless, easier to scale**: The token contains all the information to identify the user, eliminating the need for the session state. If we use a load balancer, we can pass the user to any server, instead of being bound to the same server we logged in on.

  - **Reusability**: We can have many separate servers, running on multiple platforms and domains, reusing the same token for authenticating the user. It is easy to build an application that shares permissions with another application.

  - **JWT Security**: Since we are not using cookies, we don’t have to protect against cross-site request forgery (CSRF) attacks. We should still encrypt our tokens using JWE if we have to put any sensitive information in them, and transmit our tokens over HTTPS to prevent man-in-the-middle attacks.

  - **Performance**: There is no server side lookup to find and deserialize the session on each request. The only thing we have to do is calculate the HMAC SHA-256 to validate the token and parse its content.