

Web-Security

Authentifikation und Autorisierung

HTTP Tokens

Token-basierte Authentifikation (Authorization)



Moderne Web-Schnittstellen verwenden gerne eine besondere Art der Autorisierung, die zwingend mit HTTPS verwendet werden sollte

- Basis ist ein Token, eine Art Geheimnis, welches man mit der Anfrage mit präsentieren muss
- Dieses Geheimnis erhält man nach erfolgreicher Authentifikation, ggf. auch bei einem externen Anbieter
- Token hat eine Struktur und eine Signatur
- Token wird als HTTP-Authorization-Header vom Typ Bearer übertragen

JSON Web Token (eine Möglichkeit für Tokens)

- aaaaaaaaaa.bbbbbbbbbbbb.cccccccccccc (jeweils base64-codiert)
- Header.payload.signature
- Header gibt in JSON-Format an: JWT und Hash-Algorithmus
- Payload liefert wichtige Information (Claims): Subject, Issuer, Audience, Expiration, Not-valid-before, Ausstellzeitpunkt
- Signatur über Header, Payload und zusätzlichem Secret

HTTP Tokens

OAuth2



- Verteiltes Autorisierungsprotokoll mit standardisierter Programmierschnittstelle
- Offengelegtes Protokoll
- Token-basierter Schutz auf Ressourcen
- 4 Rollen: Resource Owner, Resource Server, Client und Authorization Server
- Gut mit REST verknüpfbar und mittels JSON Web Token (JWT)
- Liefert jedoch eigentlich keine Aussagen über die vorher notwendige Authentifikation!

HTTP Tokens

OpenID Connect



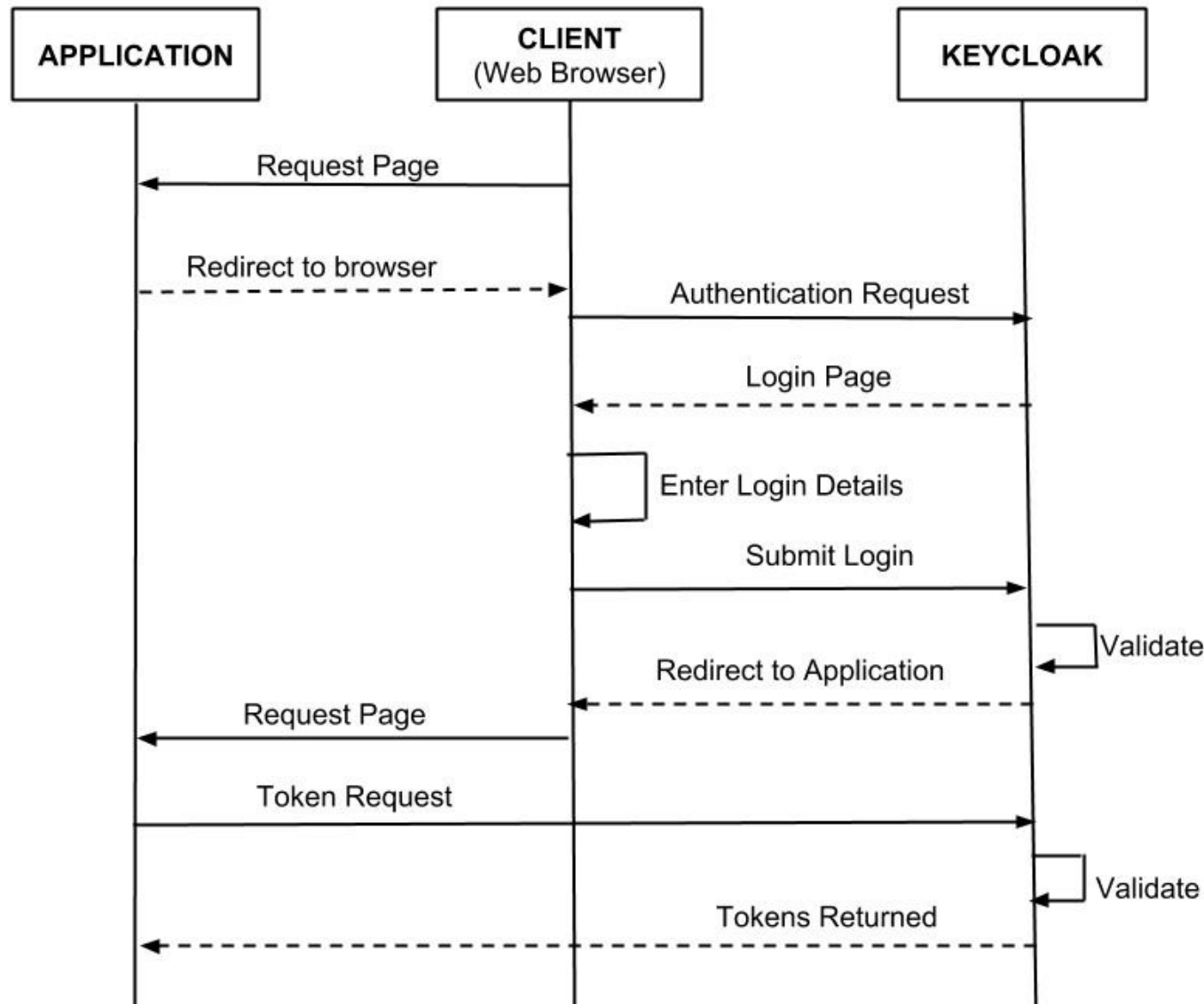
- Verteiltes Authentifizierungssystem in der Web-Landschaft
- Offengelegtes Protokoll, das auf dem OAuth 2.0-Protokoll basiert (Autorisierung einen IdP zu fragen)
- Ausschließlich unter OpenSource bereitgestellte Software
- Dient der föderierten Authentifikation
- Relativ einfach für die Nutzer und daher gerne genutzt im Web-Kontext

OpenID Connect verwendet OAuth 2.0 für die Authentifizierung und Berechtigung und erstellt dann Identitäten, die Benutzer eindeutig identifizieren indem man auf Identitäten zurück greift, die die Benutzer woanders schon haben

HTTP Tokens

OpenID Connect

<https://www.comakeit.com/blog/quick-guide-using-keycloak-identity-access-management>



FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

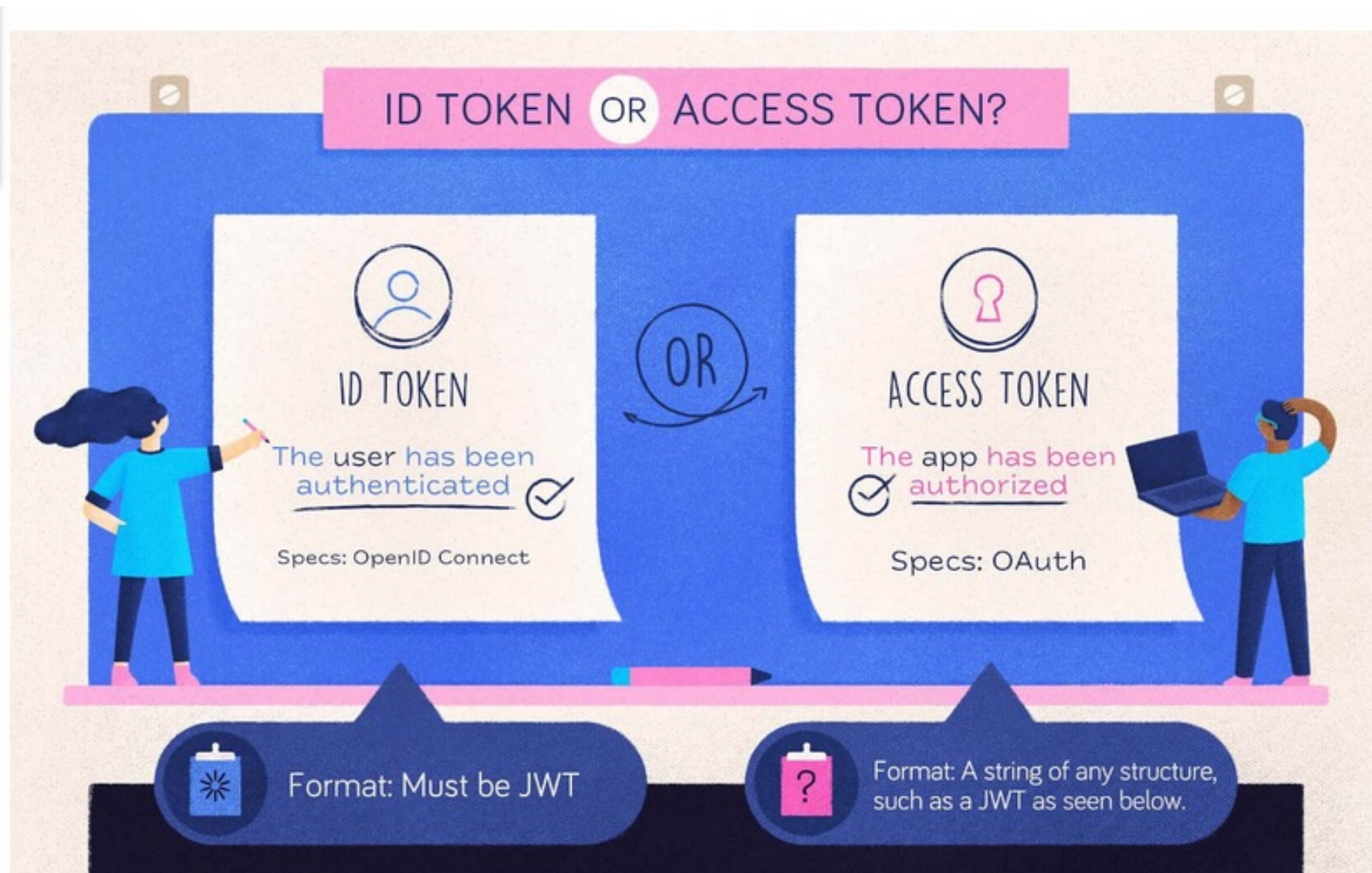
pretty ▼

<https://stackabuse.com/authentication-and-authorization-with-jwts-in-express-is/>

Host: localhost:3000

HTTP Tokens


<https://auth0.com/blog/id-token-access-token-what-is-the-difference/>



JWT

Encoded ☐

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwOi8vbXktZG9tYWwuLmFkdGgwLmNvSlsnNTYlI6ImF1dGgwZDEyMzQ1NiIsImF1ZCI6IjEyMzRhYmNkZWYiLCJleHAiOjEzMTEyODE5NDAsImhdCi6MTMxMDkzMCwibmFtZSI6IkhpbmUgRG90Iiwia2ZlZW50bmFtZSI6IkhpbmUiLCJmYW1pbHlfbmFtZSI6IkvZSJ9.bql-jxlG9B_bielkqOnjTY9Di9FIIfB6IMQINXoYsw

Decoded 

HEADER:

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "iss": "http://my-domain.auth0.com",
  "sub": "auth0|123456",
  "aud": "1234abcdef",
  "exp": 1311281970,
  "iat": 1311280970,
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe"
}
```

SIGNATURE: _____

```
RSA SHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    secret
)
```

Holds identification information

JWT

Encoded ☐

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ikk1EYzNR
MFJDTXpOR1FqQkdPRFF3UWpWRE1EVkVPRVV3TkrBe
U0wSXhPVEkzTkVWQ1F6TTVOdyJ9.eyJYIjWxwo4TVTG6
GB5FLrnQjd.....QSVWODRMvVh3biilFEookDrdqm4tPf
8hqg3FoejXegcioOCXMqY6fdMQZPqVbyostYHv92Qq
C83PNr2lDxafjZnkHkWHLOLO70cnSzQ

Decoded ☐

HEADER:

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "MDc3Q...TI3NEVCQzM5Nw"
}
```

PAYLOAD:

```
{
  "iss": "https://tenant.auth0.com/",
  "sub": "MCHiB...OsMFrzyb@clients",
  "aud": "https://glossary.com",
  "iat": 1627566690,
  "exp": 1627653090,
  "azp": "MCHiB1T...qQ30OsMFrzyb",
  "scope": "create:term update:term",
  "gtz": "authorization_code",
}
```

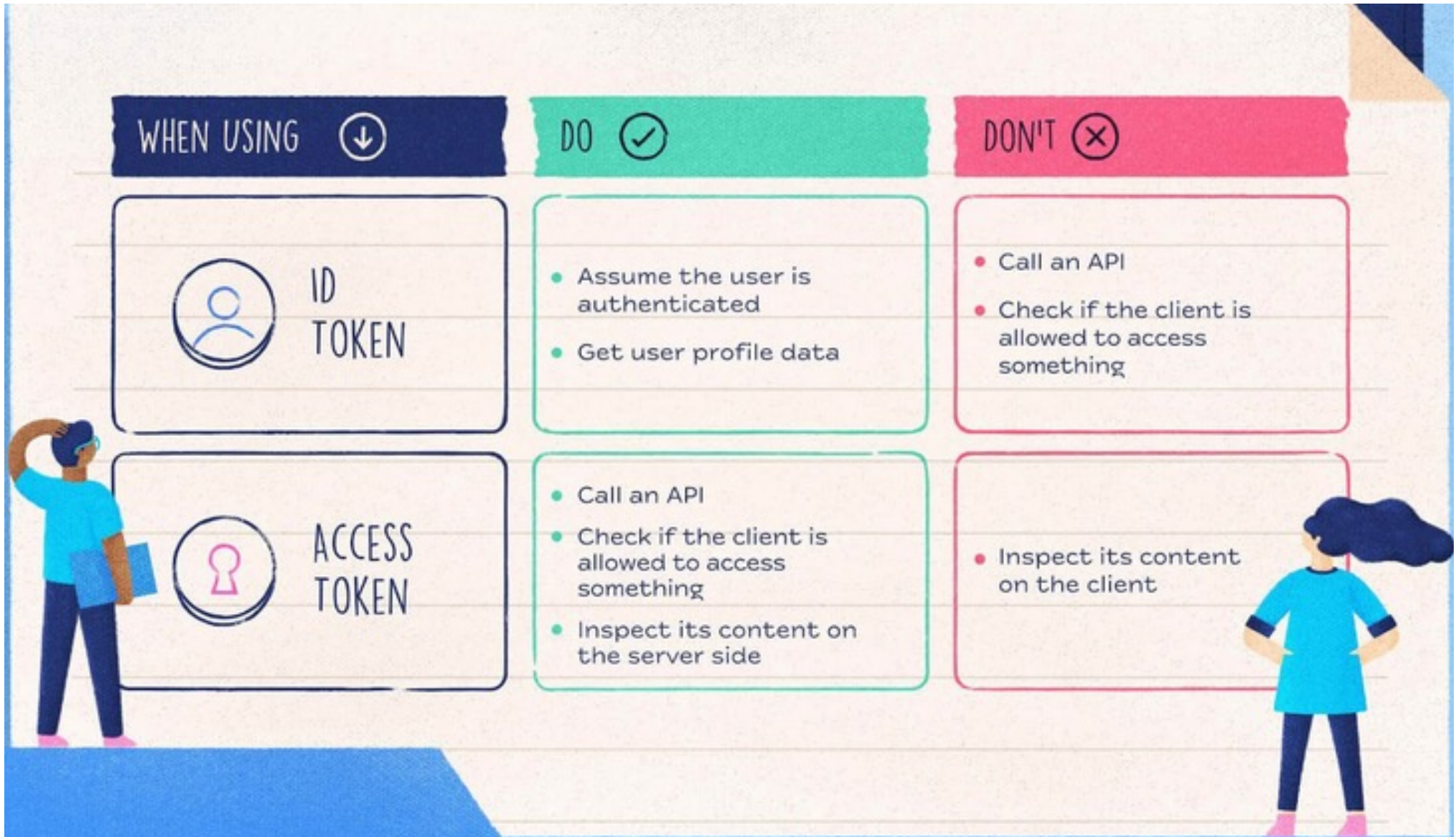
SIGNATURE: _____

```
RSA SHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    secret  
)
```

Holds authorization information

HTTP Tokens

<https://auth0.com/blog/id-token-access-token-what-is-the-difference/>

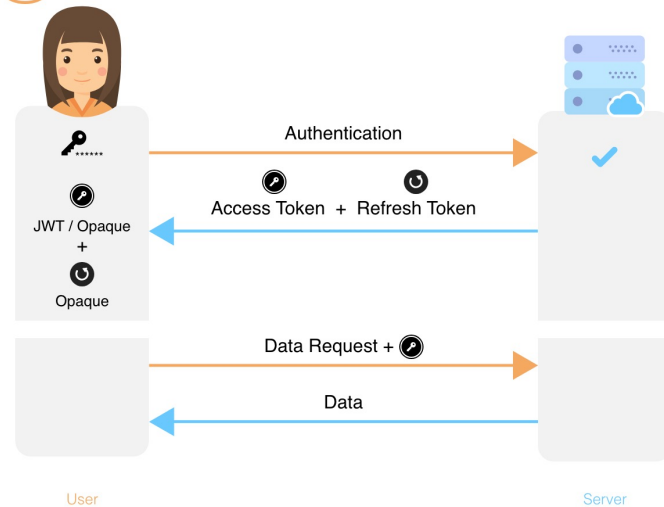


HTTP Tokens

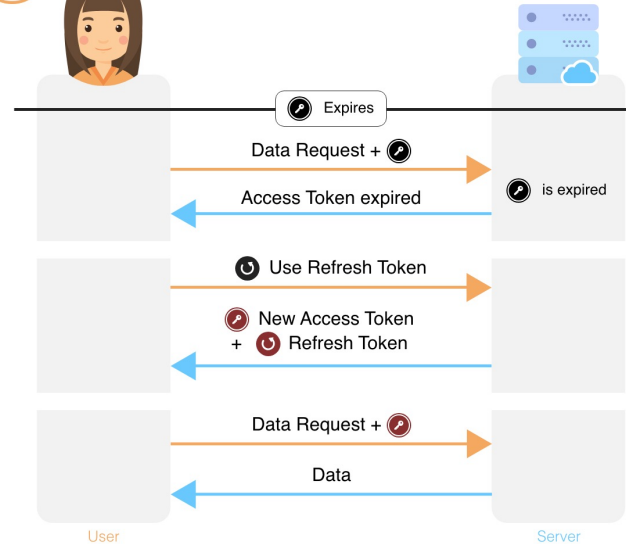
Refresh Token

<https://supertokens.com/docs/community/prologue>

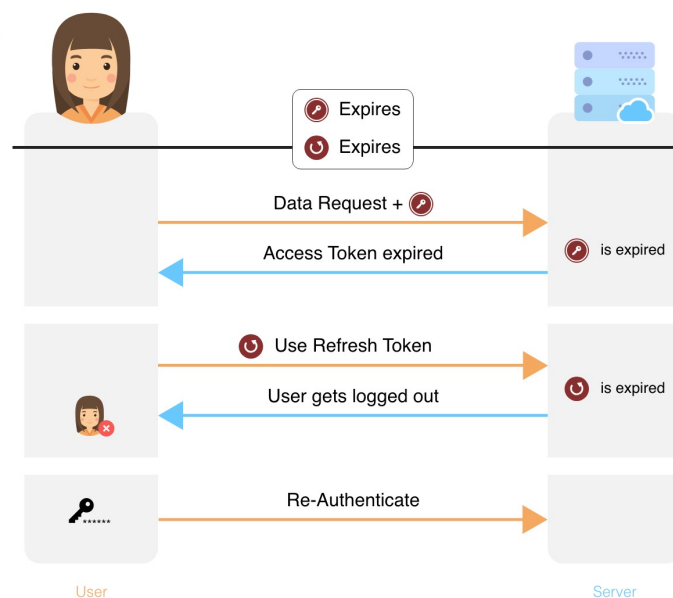
1 Authentication + API calls



2 Access Token Expires



3 Refresh + Access Tokens Expire



HTTP Tokens

JSON Web Token - einfach

```
const express = require('express');  
const app = express();  
const jwt = require('jsonwebtoken');  
  
const accessTokenSecret = 'mein access token secret';
```

```
app.use(express.json());  
  
const users = [  
  {  
    username: 'volker',  
    password: 'password123admin',  
    role: 'admin'  
  }, {  
    username: 'sander',  
    password: 'password123member',  
    role: 'member'  
  }  
];
```

HTTP Tokens

JSON Web Token - einfach

```
app.post('/login', (req, res) => {  
  // Read username and password from request body  
  const { username, password } = req.body;  
  console.log("User " + username + " with pw " + password)  
  // Filter user from the users array by username and password  
  const user = users.find( predicate: u => { return u.username === username && u.password === password });  
  
  if (user) {  
    // Generate an access token  
    const accessToken = jwt.sign( payload: { username: user.username, role: user.role }, accessTokenSecret);  
  
    res.json({  
      accessToken  
    });  
  } else {  
    res.send('Username or password incorrect');  
  }  
}
```

HTTP Tokens

JSON Web Token - einfach

```
const authenticateJWT = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (authHeader) {
    const token = authHeader.split(' ')[1];

    jwt.verify(token, accessTokenSecret, options: (err, user) => {
      if (err) {
        return res.sendStatus(403);
      }

      req.user = user;
      next();
    });
  } else {
    res.sendStatus(401);
  }
};
```


HTTP Tokens

JSON Web Token - einfach

```
const books = [  
  {  
    "author": "a",  
    "language": "aa",  
    "title": "Erster"  
  },  
  {  
    "author": "b",  
    "language": "bb",  
    "title": "Zweiter"  
  },  
  {  
    "author": "c",  
    "language": "vc",  
    "title": "Dritter"  
  },  
];
```

```
app.get('/books', authenticateJWT, (req, res) => {  
  res.json(books);  
});
```

```
app.listen(3000, () => {  
  console.log('Authentication service started on port 3000');  
});
```

HTTP Tokens

JSON Web Token

METHOD: POST SCHEME://HOST [":" PORT] [PATH ["?" QUERY]]
http://localhost:3000/login length: 27 byte(s) Send

QUERY PARAMETERS
+ Add query parameter

HEADERS 1/2 Form
☒ Content-Type : application/json
+ Add header Add authorization

BODY 1 Text
1 { "username": "volker", "password": "password123admin" }
Text JSON XML HTML | Format body | ☒ Enable body evaluation length: 53 bytes

METHOD: GET SCHEME://HOST [":" PORT] [PATH ["?" QUERY]]
http://localhost:3000/books length: 27 byte(s) Send

QUERY PARAMETERS
+ Add query parameter

HEADERS 1/2 Form
☒ Authorization : Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6I...

BODY 1
XHR does not allow payloads for GET request.

HTTP API-Keys

Identifikation von Anwendung

Tokens sind private, personenbezogene Hilfsmittel

API-Keys sollen eine Anwendung oder ein Projekt identifizieren

- Faktisch werden API-Keys gerne dazu verwendet, die Aufrufe Anwendungen, Projekten oder doch gar Personen zuzuordnen. Diese Zuordnung ist jedoch locker und eher für statistische Zwecke geeignet, nicht um den Zugang wirklich bestimmten Benutzern zuzuordnen
- Beispiel: Google Maps benötigt ein API-Key, den man sich erzeugen muss (der erzeugt wird)

★ **New Users:** Before you can start using the Google Maps Platform APIs and SDKs, you must sign up and create a billing account. To learn more, see [Get Started with Google Maps Platform](#).

To use the Maps JavaScript API you must have an API key. The API key is a unique identifier that is used to authenticate requests associated with your project for usage and billing purposes.

<https://developers.google.com/maps/documentation/javascript/get-api-key>