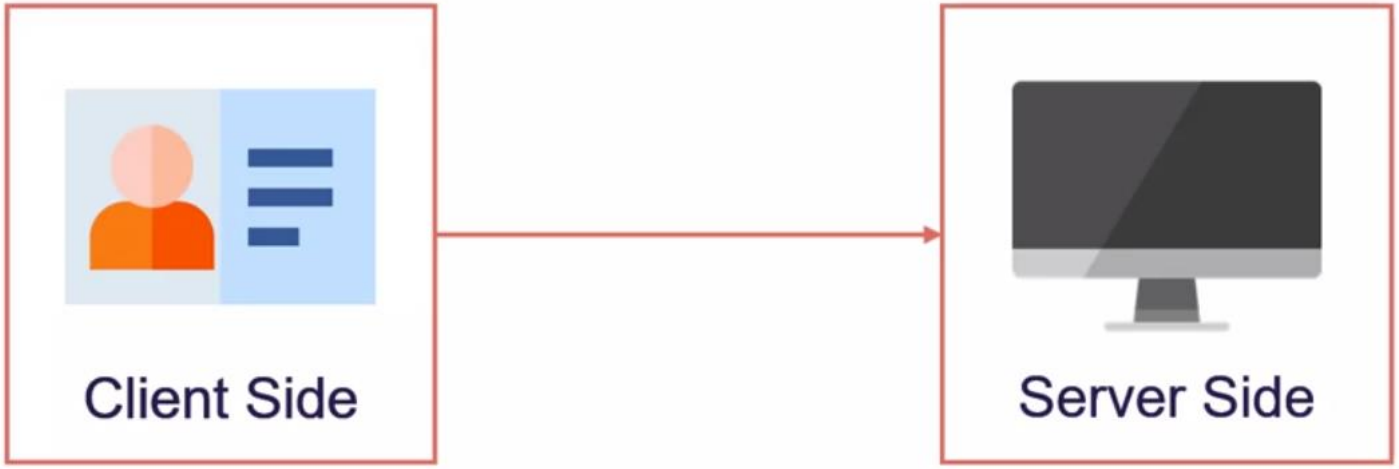
Nodejs- Authentication and Authorization

Laiba Imran

Authentication

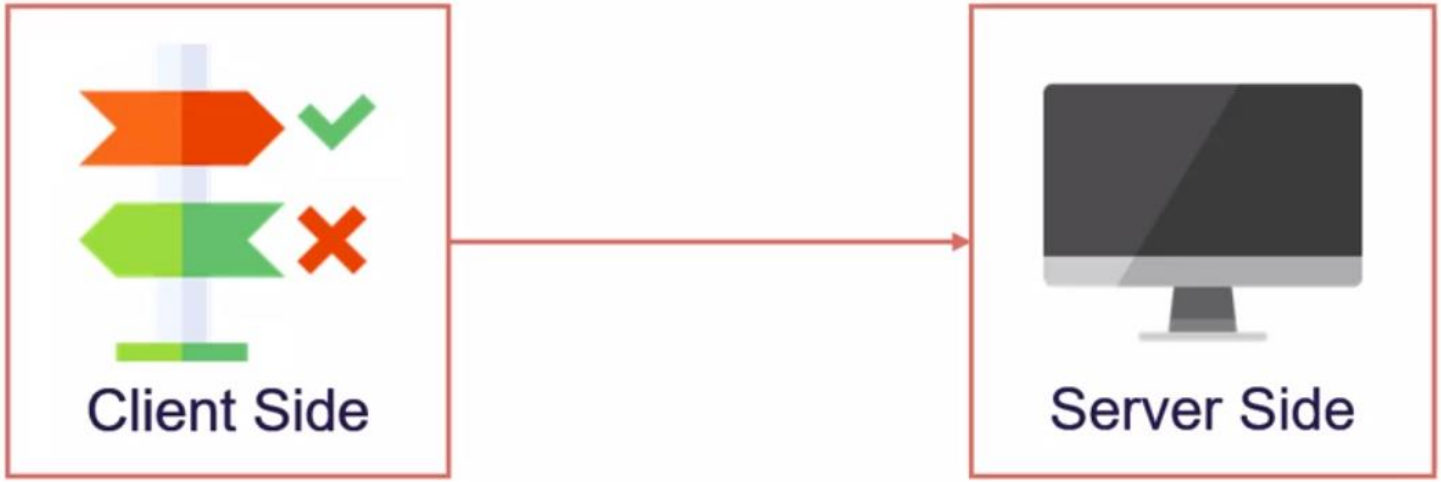
Who are you?



Authentication is a process that scrutinizes the user at the login level.

Authorization

What are you allowed to do?



Authorization is the process of specifying access rights/privileges to resources.

Server-Based Authentication

Authentication → Server-Based

HTTP is stateless; every request requires a new login.

Multiple Authentication → Inconvenient

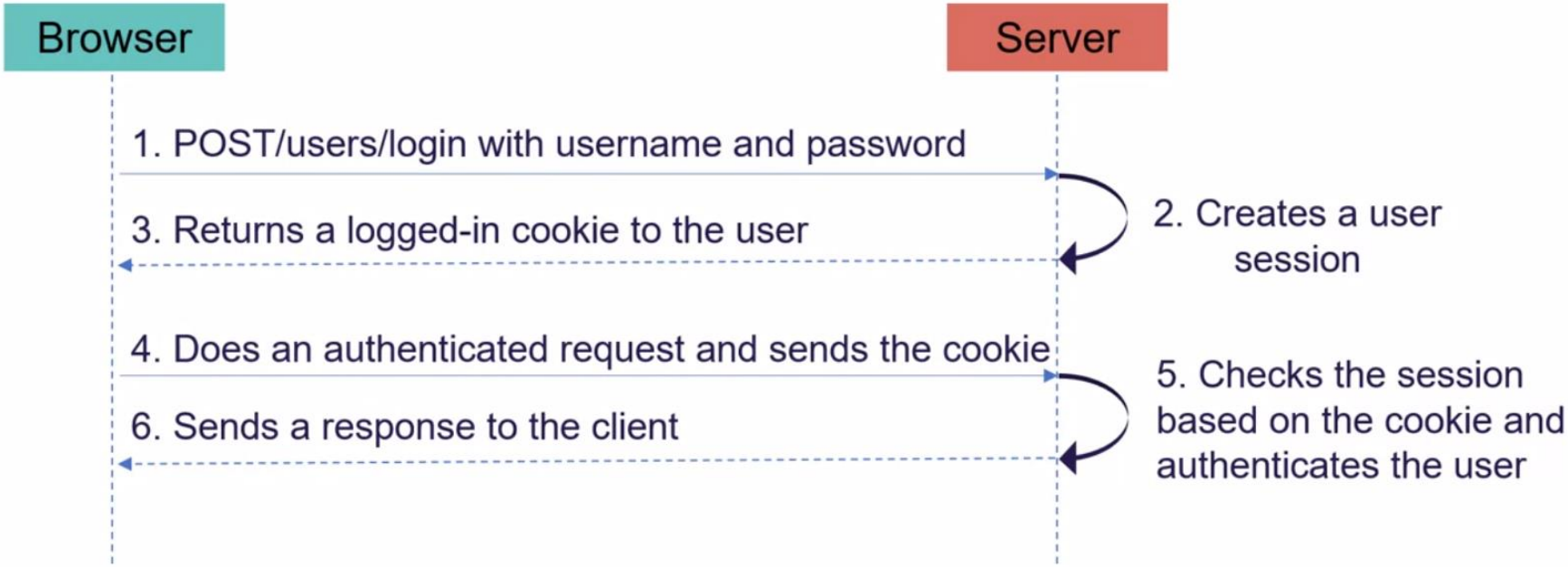
Users need to remain logged in for the entire session.

Server-Based Authentication

A **session** in web development refers to a way to store user-specific information across multiple requests while the user interacts with a web application.

- When a user logs in, the server creates a session and stores the session data (like user information) on the server-side (usually in memory or a database).
- The server generates a Session ID, which is a unique identifier for that session, and sends this ID to the client (browser) in a cookie.
- For each subsequent request, the client sends the Session ID in the cookie, and the server uses this ID to retrieve the associated session data from its storage.

Server-Based Authentication

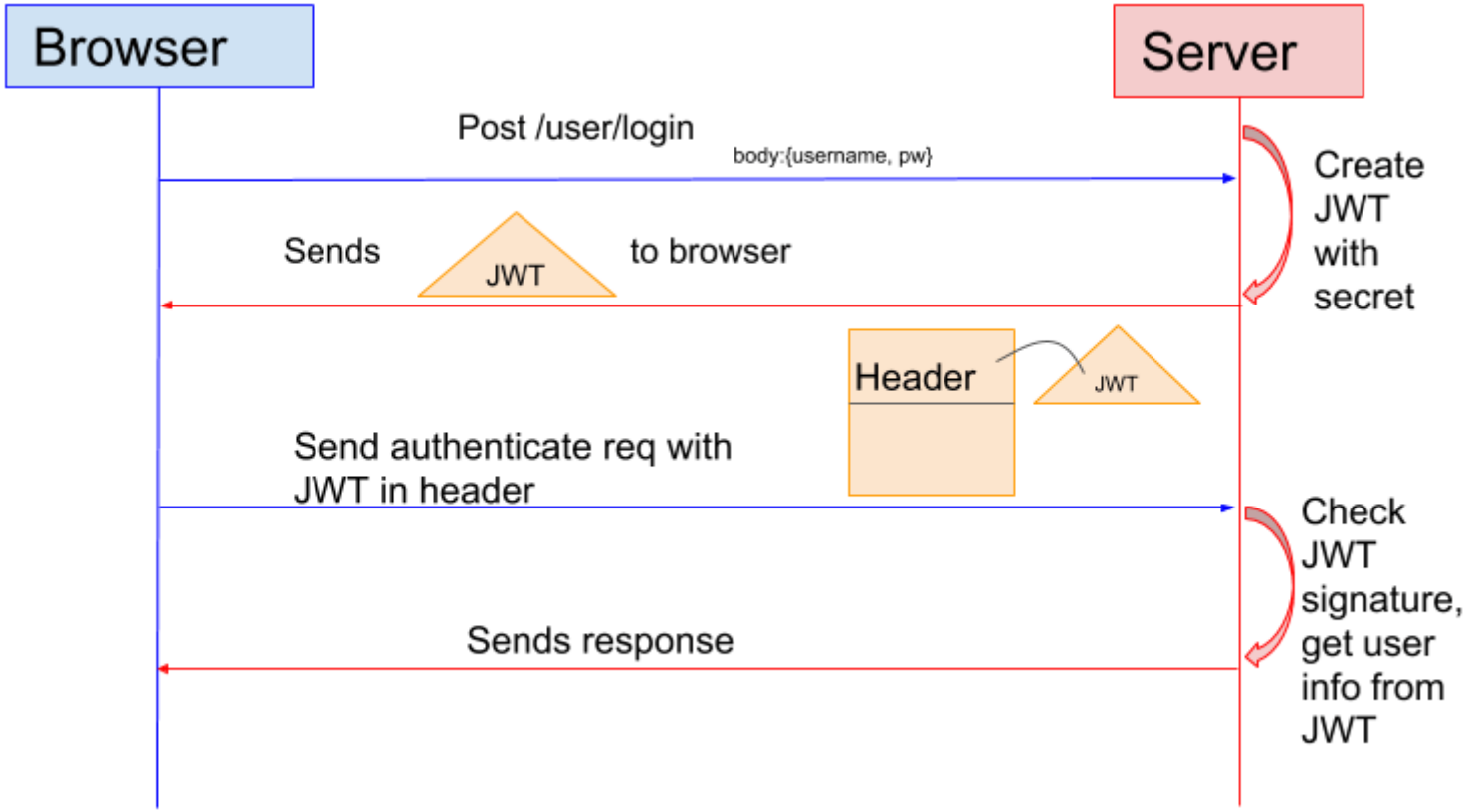


Token-Based Authentication

A **JWT** in web development refers to a JSON Web Token that is used across multiple requests while the user interacts with a web application.

- When a user logs in, the server creates a JWT (JSON Web Token) containing user information (like user ID, roles, etc.), signs it with a secret key, and sends it to the client.
- The JWT is stored in local Storage on the client-side.
- For each subsequent request, the client sends the JWT in the Authorization header.
- The server verifies the JWT's signature using the secret key and extracts the user information from the token.

Token-Based Authentication



What is JSON Web Token (JWT)

Why JWT?

Provides ease of client-side processing

Can be transmitted faster due to its small size

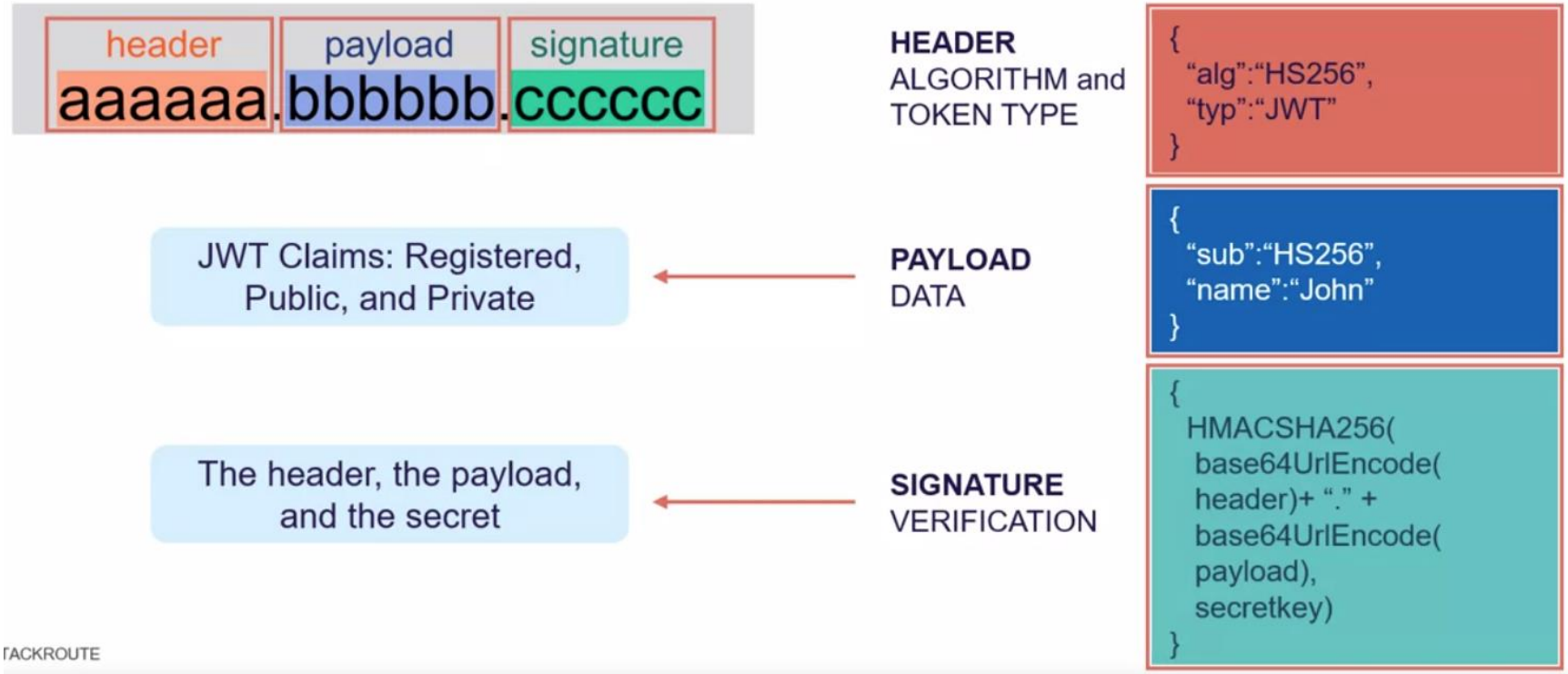
Can be transmitted securely using public/private keys

Provides complete information to reduce database querying

Working of JWT

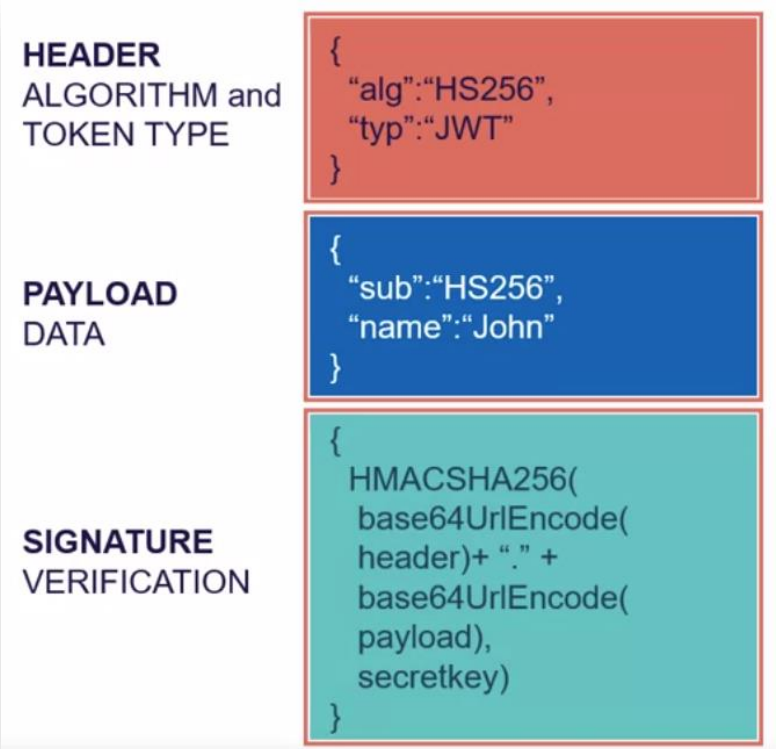
1. User first signs into the authentication server by using the server's login system.
2. The authentication server creates the JWT and sends it to the user.
3. The user passes the JWT along with the API call.
4. The application server is configured to verify that the incoming JWT is created by the authentication server.
5. The application uses the JWT to verify if the API call is coming from an authenticated user.

Structure of JWT



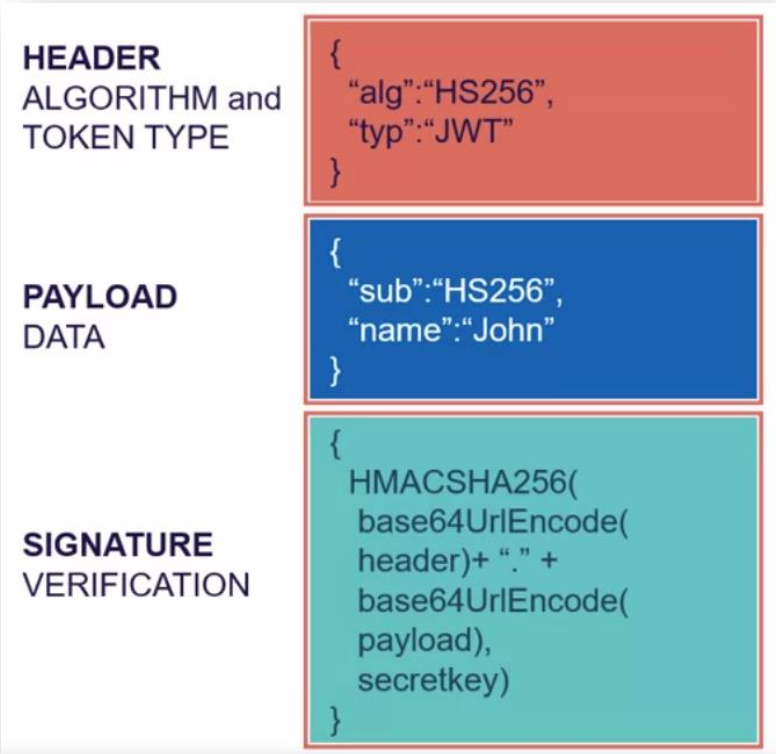
Header

- The header *typically* consists of two parts:
  - Type of the token, which is JWT
  - Signing algorithm being used, such as HMAC SHA256 or RSA.
- JSON is Base64Url encoded to form the first part of the JWT.



Payload

- The second part of the token is the payload, which contains the claims.
- Claims are statements about an entity (typically, the user) and additional data.
- There are three types of claims:
  - Registered
  - Public (defined in IANA)
  - Private



Payload

- Registered:
  - predefined claims that are not mandatory but recommended
  - **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience)
- Public (defined in IANA):
  - custom claims that should be registered to avoid name collisions; they can be used across different applications.
  - email, role, or organization
  - If someone defines a claim called role, it is advisable to register it to ensure that no other party uses the same name for a different purpose.
- Private:
  - custom claims that are not registered or standardized, which means they are unique to the specific application or context.
  - user_id or favorite_color

Structure of JWT

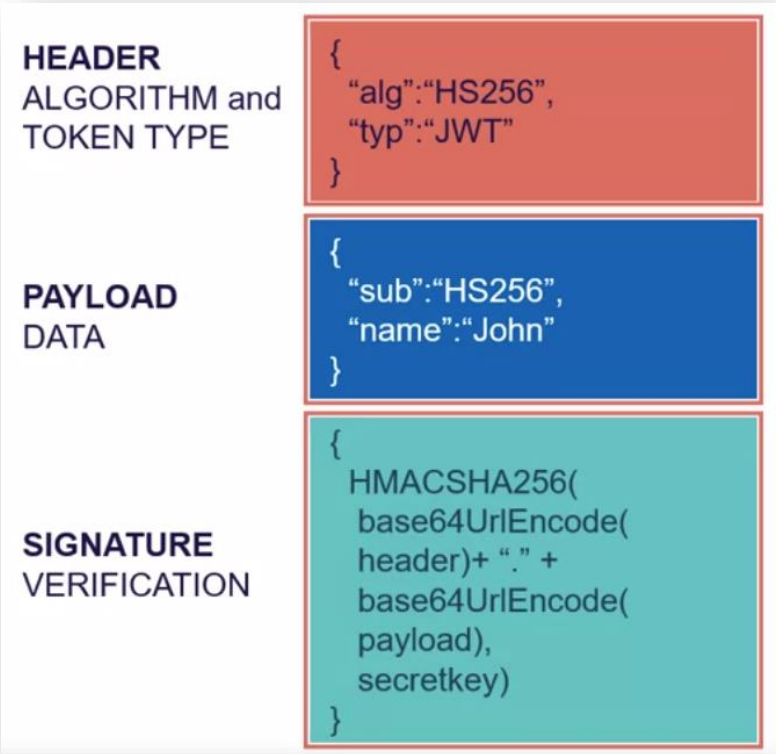
PAYLOAD
DATA

```
{
    "sub": "98765",
    "name" : "John"
}
```

iss	The issuer of the token
sub	The subject of the token
aud	The audience of the token
exp	The expiration in NumericDate value
nbf	The time before which the JWT must not be accepted for processing
iat	The time when the JWT was issued
jti	The unique identifier for the JWT

Signature

- To create the signature part you have to take the
  - encoded header
  - the encoded payload
  - a secret
  - the algorithm specified in the header
  - and sign that



jwt.io

JWT

Debugger Libraries Introduction Ask

Crafted by auth0 by Okta

Algorithm HS256

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

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

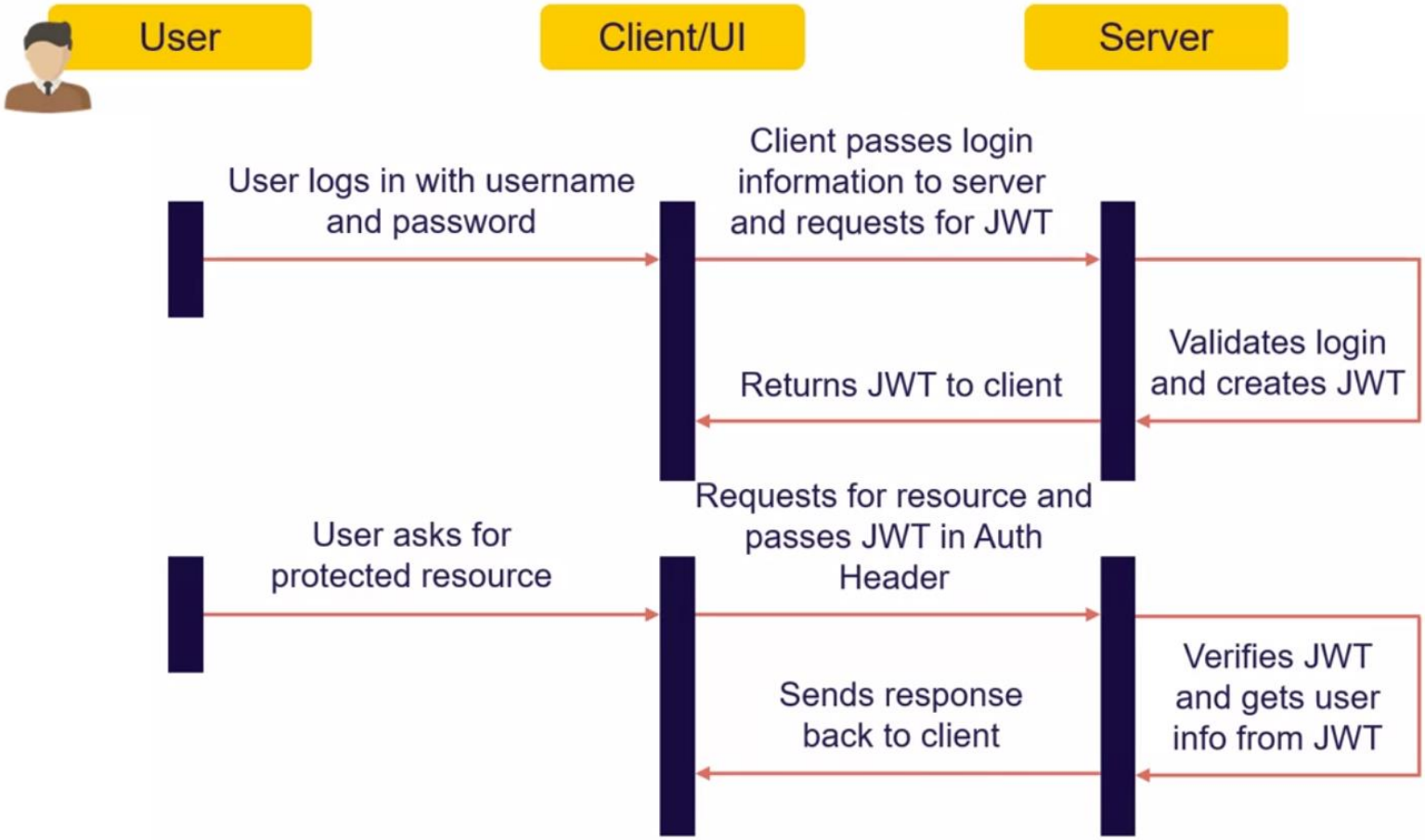
VERIFY SIGNATURE

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

Signature Verified

SHARE JWT

Dataflow using JWT



Dataflow of Nodejs App using JWT

$ npm i jsonwebtoken

```
var jwt = require('jsonwebtoken’);
var token = jwt.sign({ foo: 'bar' }, 'shhhhh');
```

- Generates token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmb28iOiJiYXIiLCJpYXQiOjE2Nzk3NDQ4NjZ9.Gq2m0f81hPqjlMgH6c_vUGeKkbhMYZQskhtVyaEPs2w

- Verify Token

```
var token =
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmb28iOiJiYXIiLCJpYXQiOjE2Nzk3NDQ4NjZ9.Gq2m0f81hPqjlMgH6c_vUGeKkbhMYZQskhtVyaEPs2w`
var decodedToken = jwt.verify(token, 'shhhhh');
```

Decoded Token: `{ foo: 'bar', iat: 1679744866 }`

https://www.npmjs.com/package/jsonwebtoken

Implementing JWT with `jsonwebtoken`

```
// Import Express
const express = require("express");

// Import jsonwebtoken for JWT
const jwt = require("jsonwebtoken");

// Import dotenv to read .env file
require("dotenv").config();
```

```
// Handling post request
app.post("/login", (req, res, next) => {
    let { email, password } = req.body;
    let existingUser = Users.find(u => u.email == email && u.password == password)
    if (!existingUser) {
        const error = Error("Wrong details please check at once");
        res.status(401).json({
            success: false,
            error: error.message
        })
    } else {
        let token;
        try {
            //Creating jwt token
            token = jwt.sign(
                { userId: existingUser.id, email: existingUser.email },
                process.env.SECRET,
                { expiresIn: "1h" }
            );
        } catch (err) {
            console.log(err);
            const error = new Error("Error! Something went wrong.");
            next(error);
        }

        res
            .status(200)
            .json({
                success: true,
                data: {
                    userId: existingUser.id,
                    email: existingUser.email,
                    token: token,
                },
            });
    }
});
```

Find User in users.json (which is imported)

Respond with 401 Status code with Erro message

When Found, generate a Token with Payload, Secret and expiry option of 1 hour

Through the error if JWT Signing failed

Send the Response with 200 OK status and JSON containing JWT

Test POST /login endpoint

POST http://localhost:3000/login Send

Query Headers 2 Auth Body 1 Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

Json Content Format

```
1 {
2   "email" : "ahmed@gmail.com",
3   "password" : "123"
4 }
```

Status: 200 OK Size: 250 Bytes Time: 69 ms

Response Headers 6 Cookies Results Docs { }


```
1 {
2   "success": true,
3   "data": {
4     "userId": 1,
5     "email": "ahmed@gmail.com",
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJ1c2VySWQiOjEsImVtYWlsIjoiYWhtZWRAZ21haWwuY29tIiwia
        WF0IjoxNjgwMTY4NjU2LCJleHAiOjE2ODAxNzIyNTZ9
        ._eMMBDJInxcGfNvwkN3DaJSNh3E8jFedGyMvE3uf6Eg"
7   }
8 }
```


Success

POST http://localhost:3000/login Send

Query Headers 2 Auth Body 1 Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

Json Content Format

```
1 {
2   "email" : "ahmed@gmail.com",
3   "password" : "1233"
4 }
```

Status: 401 Unauthorized Size: 62 Bytes Time: 8 ms

Response Headers 6 Cookies Results Docs { }

```
1 {
2   "success": false,
3   "error": "Wrong details please check at once"
4 }
```

Fail

Verify JWT

```
app.get('/accessResource', (req, res) => {
    const token = req.headers.authorization.split(' ')[1];
    //Authorization: 'Bearer TOKEN'
    if (!token) {
        res.status(200).json({
            success: false,
            message: "Error! Token was not provided."});
    }
    //Decoding the token
    jwt.verify(token, process.env.SECRET, (err, decodedToken) => {
        if (decodedToken) {
            res.status(200).json({
                success: true,
                data: {
                    userId: decodedToken.userId,
                    email: decodedToken.email
                }
            });
        } else {
            res.status(401).json({
                success: false,
                error: err.message
            });
        }
    });
})
```

Get the Token in the Header

When Token is not sent in the header

Verify the token with the Secret and returns a callback function. On successful decoding, `decodedToken` would contain the Payload, otherwise, it will be undefined.

In case of error, decodedToken will be undefined and respond with an error with error message (which will be containing in err object retured by verify function

Test `GET /accessResource` endpoint

GET localhost:3000/accessResource Send

Query Headers 2 Auth 1 Body Tests Pre Run

None Basic Bearer OAuth 2 Ntlm Aws

Bearer Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsImVtYWlsIjoiYWhtZWRAZ21haWwuY29tIiwiaWF0IjoxNjgwMTU3MTUxLCJleHAiOjE2ODAxNjA3NTF9.AEaXTTMAtrxNZ9KWOI9MgCAN-eF_nI1ZYE6R6O1904g

Status: 401 Unauthorized Size: 39 Bytes Time: 30 ms

Response Headers 6 Cookies Results Docs

```
{
  "success": false,
  "error": "jwt expired"
}
```

Fail

GET localhost:3000/accessResource Send

Query Headers 2 Auth 1 Body Tests Pre Run

None Basic Bearer OAuth 2 Ntlm Aws

Bearer Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsImVtYWlsIjoiYWhtZWRAZ21haWwuY29tIiwiaWF0IjoxNjgwMTU3MTUxLCJleHAiOjE2ODAxNjA3NTF9.AEaXTTMAtrxNZ9KWOI9MgCAN-eF_nI1ZYE6R6O1904

Status: 401 Unauthorized Size: 45 Bytes Time: 9 ms

Response Headers 6 Cookies Results Docs

```
{
  "success": false,
  "error": "invalid signature"
}
```

Fail

GET localhost:3000/accessResource Send

Query Headers 2 Auth 1 Body Tests Pre Run

None Basic Bearer OAuth 2 Ntlm Aws

Bearer Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsImVtYWlsIjoiYWhtZWRAZ21haWwuY29tIiwiaWF0IjoxNjgwMTY5NDgwLCJleHAiOjE2ODAxNzMwODB9.P7wP1ki3dJViXK824Tr6zF_o2jWq7mireuUI9vmT0N8

Status: 200 OK Size: 62 Bytes Time: 9 ms

Response Headers 6 Cookies Results

```
{
  "success": true,
  "data": {
    "userId": 1,
    "email": "ahmed@gmail.com"
  }
}
```

Success

National
Of Computer a

Create an Authentication Middleware for Verifying JWT for every Protected Resource/Endpoint

```
const auth = (req, res, next) => {
    const token = req.headers.authorization.split(' ')[1];
    //Authorization: 'Bearer TOKEN'
    if (!token) {
        res.status(200).json({
            success: false,
            message: "Error! Token was not provided."
        });
    }
    //Decoding the token
    try {
        const decodedToken = jwt.verify(token, process.env.SECRET);
        req.token = decodedToken;
        next();
    } catch (err) {
        res.status(401).json({
            success: false,
            message: err.message
        });
    }
}
```

Upon successful decoding, add token property to request object and move to next middleware

In case of error, respond back from middleware

Use Authentication Middleware

Use Middleware when a request is received

```
app.get('/accessResource2', auth, (req, res) => {

    var decodedToken = req.token

    res.status(200).json({
        success: true,
        data: {
            userId: decodedToken.userId,
            email: decodedToken.email
        }
    });
})
```

Get the token from request object (added by middleware)

OAuth2

Stands for Open Authorization

Is designed to grant users access to a set of resources

Is an authorization protocol that authorizes users in an application

Uses external service providers to authorize users

Facebook | Gmail | GitHub

OAuth2

OAuth2 is an authorization protocol, NOT an authentication protocol.

It focuses on client developer simplicity.

It provides specific authorization flow for:

- Web Applications
- Desktop Applications
- Mobile Applications

Essential Components of OAuth2

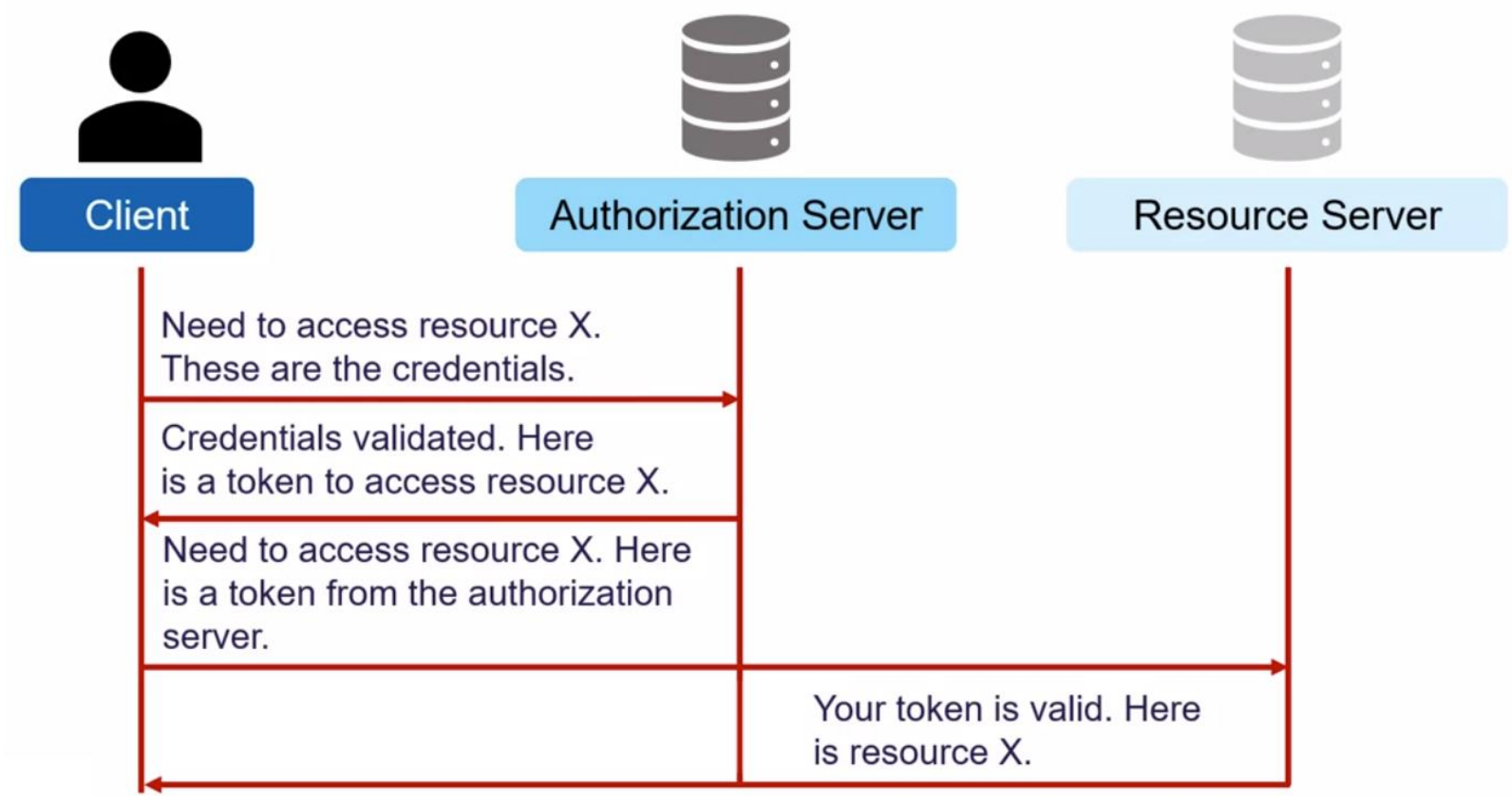
Client	The person/user trying to log in
Resource Server	The application that the client wants to log into
Authorization Server	The external application that authenticates the user identity

e.g. Gmail, GitHub user

e.g. nodejs server

e.g. Gmail

How OAuth2 Works



References

- JWT
  - https://jwt.io/
- JWT RFC
  - https://datatracker.ietf.org/doc/html/rfc7519
- IANA JWT
  - https://www.iana.org/assignments/jwt/jwt.xhtml
- jsonwebtoken package
  - https://www.npmjs.com/package/jsonwebtoken
- dotenv package
  - https://www.npmjs.com/package/dotenv
- 0Auth2
  - https://auth0.com/intro-to-iam/what-is-oauth-2