MEET NJOROGE

I'm Njoroge. I run the private bookstore called XYZ.

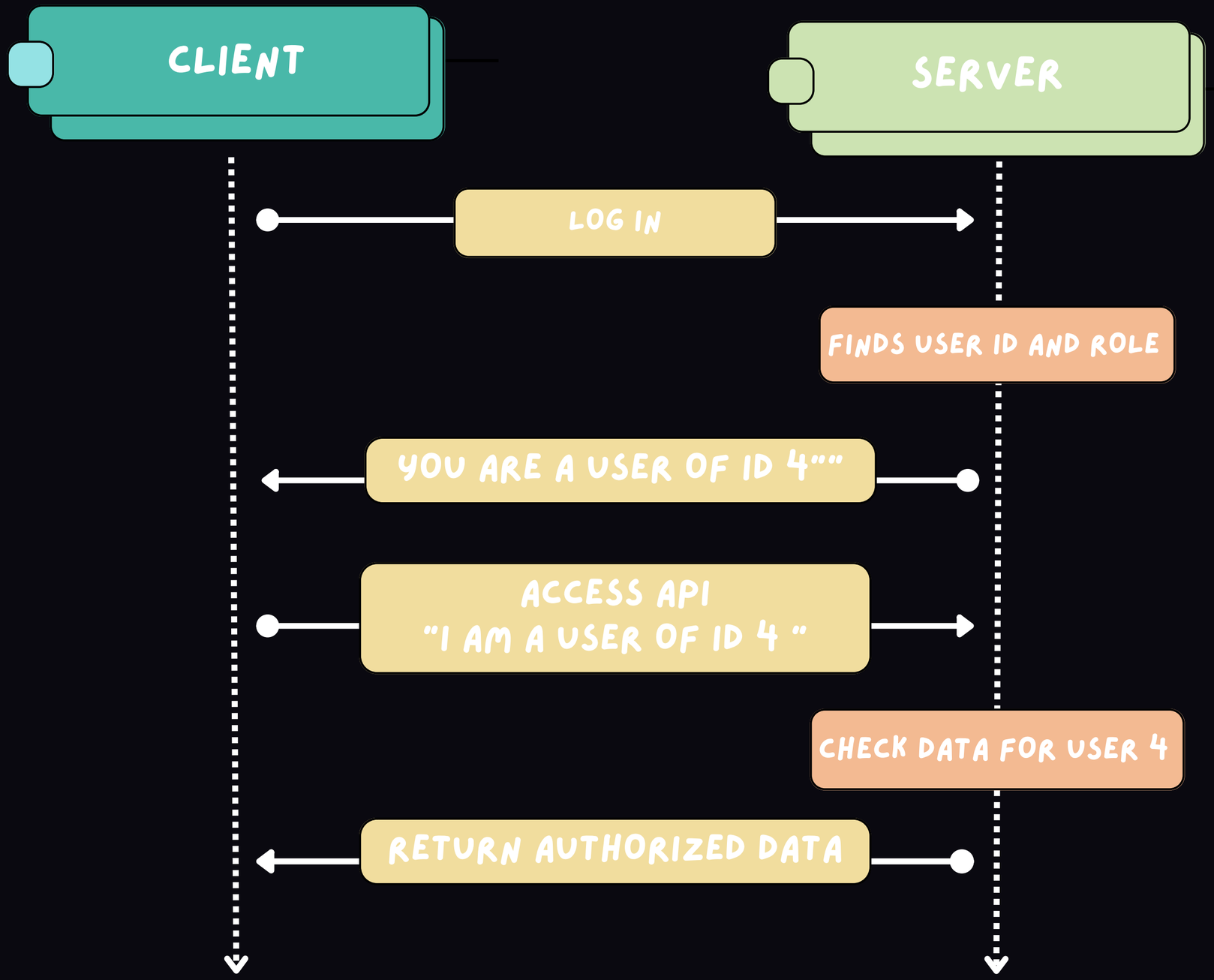
About XYZ

Registered users are able to

Only registered users are able to access the inventory to see what's on offer

Administrators get more priviledge

Admins can add books and do a whole load of stuff


CLIENT
SERVER
LOG IN
FINDS USER ID AND ROLE
YOU ARE A USER OF ID 4""
ACCESS API
"I AM A USER OF ID 4 "
CHECK DATA FOR USER 4
RETURN AUTHORIZED DATA

CLIENT

SERVER

LOG IN

FINDS USER ID AND ROLE

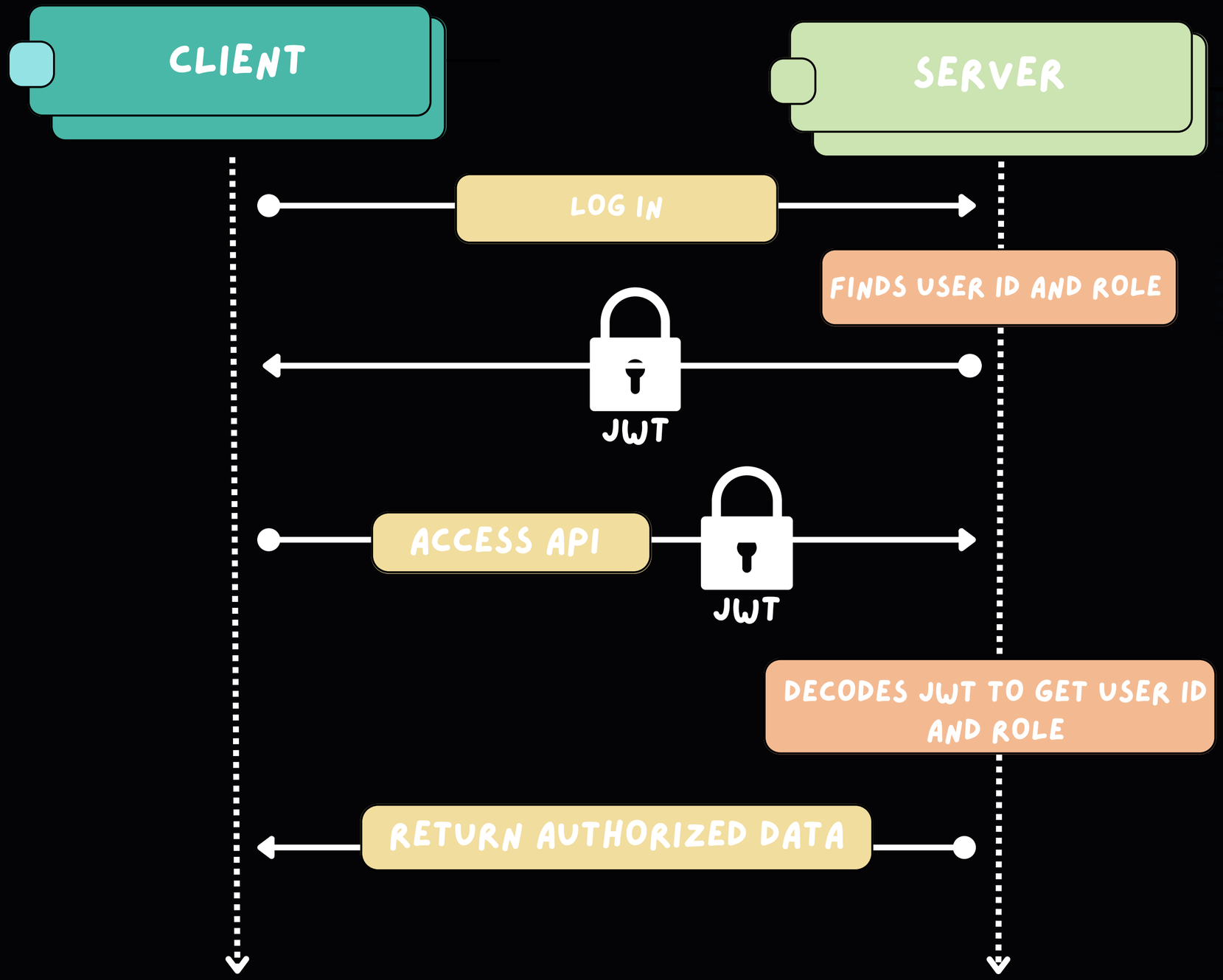
YOU ARE A USER OF ID 4""

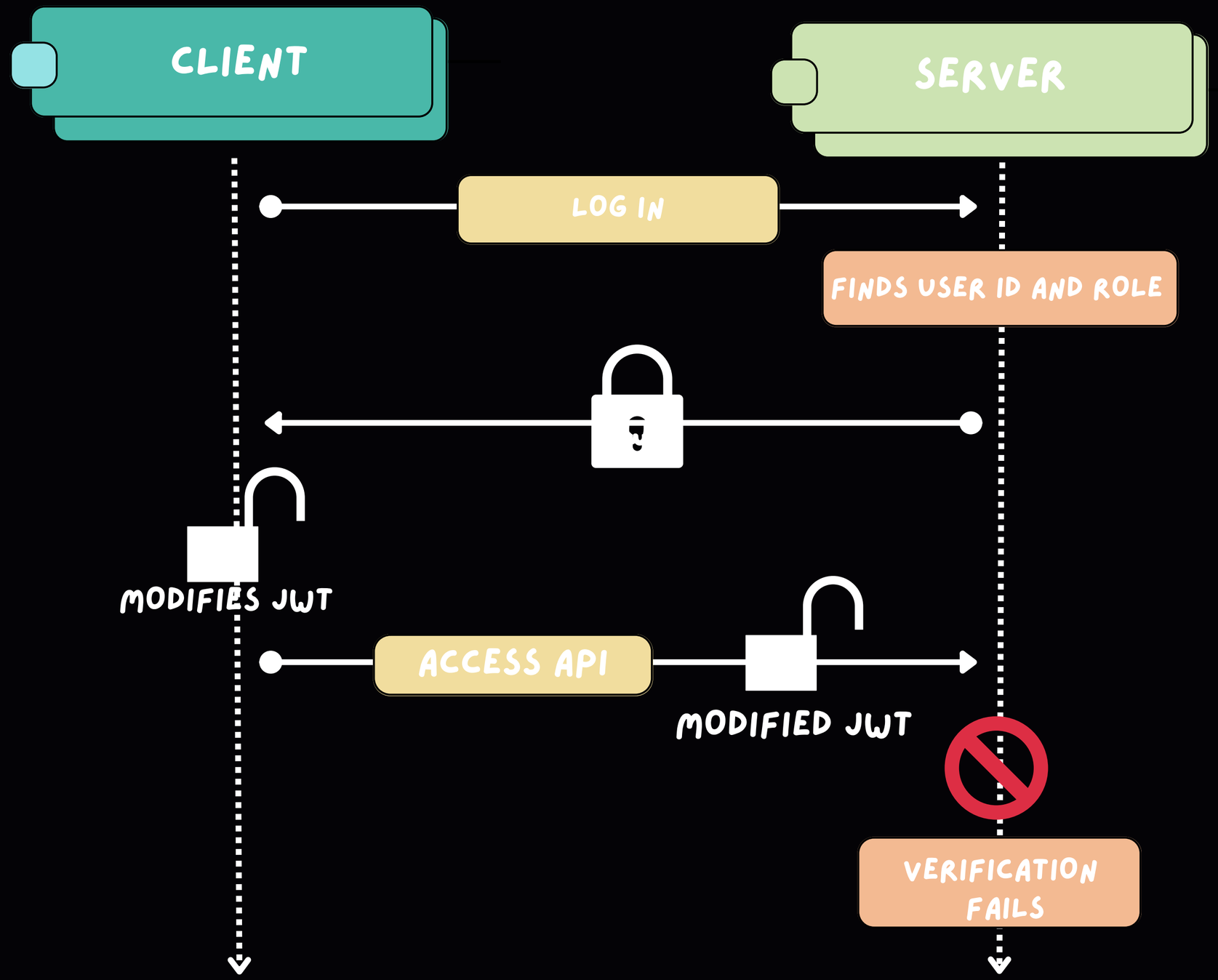
MALICIOUS USER CHANGES ID & ROLE

ACCESS API
"I AM AN ADMIN OF ID 5 "

CHECK DATA FOR ADMIN 5

RETURN AUTHORIZED DATA


CLIENT
SERVER
LOG IN
FINDS USER ID AND ROLE
JWT
ACCESS API
JWT
DECODES JWT TO GET USER ID AND ROLE
RETURN AUTHORIZED DATA

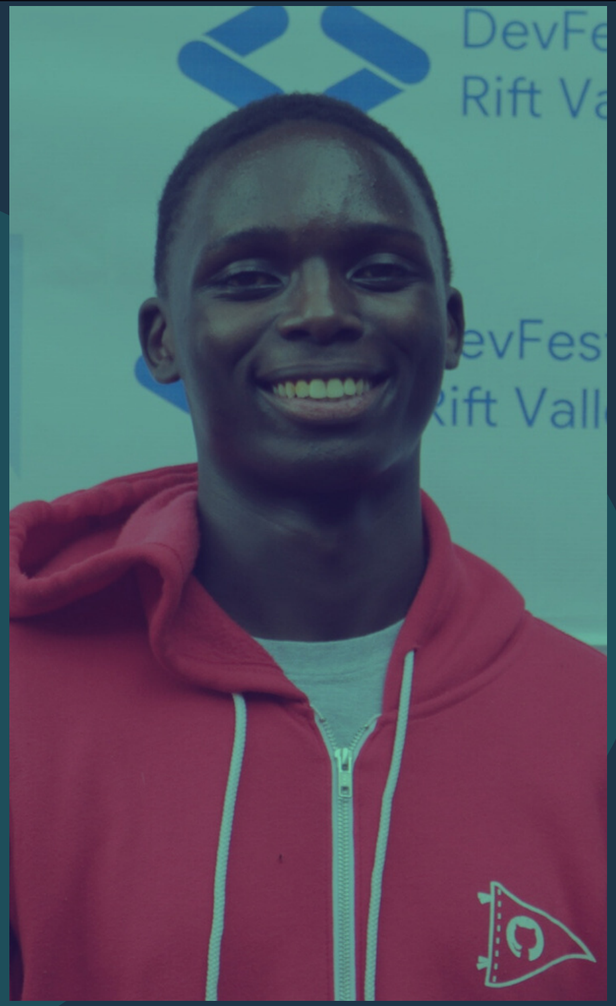

CLIENT
SERVER
LOG IN
FINDS USER ID AND ROLE
MODIFIES JWT
ACCESS API
MODIFIED JWT
VERIFICATION FAILS

SECURING

RESTFUL APIS

WITH JWTS

WHO AM I ?



CLIFFORD OUMA

OSS Community Manager, Open Terms Archive
Frontend Developer, Applantus

Our mission

We are to help Njoroge implement JSON Web Tokens to his bookstore to be able to:

- Authenticate users to see books
- Authorize admins only to be able to add books

WHAT WE WILL COVER

01

UNDERSTANDING JWTS

How JWTs work under the hood

02

IMPLEMENTING JWT IN A RESTFUL API

What to takenote when implementing JWTs to a REST API

03

SIMPLE DEMO

We'll do a simple demo of the implementation

SCAN TO GET SLIDES

01

UNDERSTANDING JWTS

JWT under the hood

Session-based Authentication

User logs in

Server validates and creates a session in DB

Session ID provided

A session ID is sent to client and saved as a cookie

Session ID used to authenticate

Session ID used in subsequent requests and authenticates user

Token-based Authentication

User logs in

A JWT is generated upon successful login

JWT provided

JWT is provided in to the client and usually stored in local storage

JWT used to authenticate

JWT sent in auth header for subsequent requests, is verified and authenticates user

WHAT IS A JSON WEB TOKEN (JWT)

01 | Is an open standard

02 | it is compact and self contained

03 | Helps in transmission of info as JSON object

03 | Is digitally signed hence trusted

JWT COMPONENTS

XXX.YYY.ZZZ

HEADER (XXX)

- Contains the token type and the hashing algorithm
- Is Base64Url encoded to form

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

PAYLOAD (YYY)

- Contains the information being transmitted (claims)
- Can be reserved, public or private claims
- Is Base64url encoded

```
{
  'sub': '12345',
  'name': 'John Doe',
  'admin': true
}
```

SIGNATURE (ZZZ)

- Created encoded header and payload
- Signs both with a secret and uses algo in the header

```
HMACSHA256(
  base64UrlEncode(header)+'.'+
  base64UrlEncode(payload), secret )
```

02

IMPLEMENTING JWT

Let's see how we can implement JWTs in a REST API

BEFORE IMPLEMENTATION

ACCESS TOKEN

Is a JWT to give access to certain data
Is short-lived

REFRESH TOKEN

Is a JWT that helps create a new access token upon expiry
Is long-lived

Basic implementation cases

The basic scenarios that need to be catered for

Generating a JWT

We need to create a JWT upon login

Verify JWT

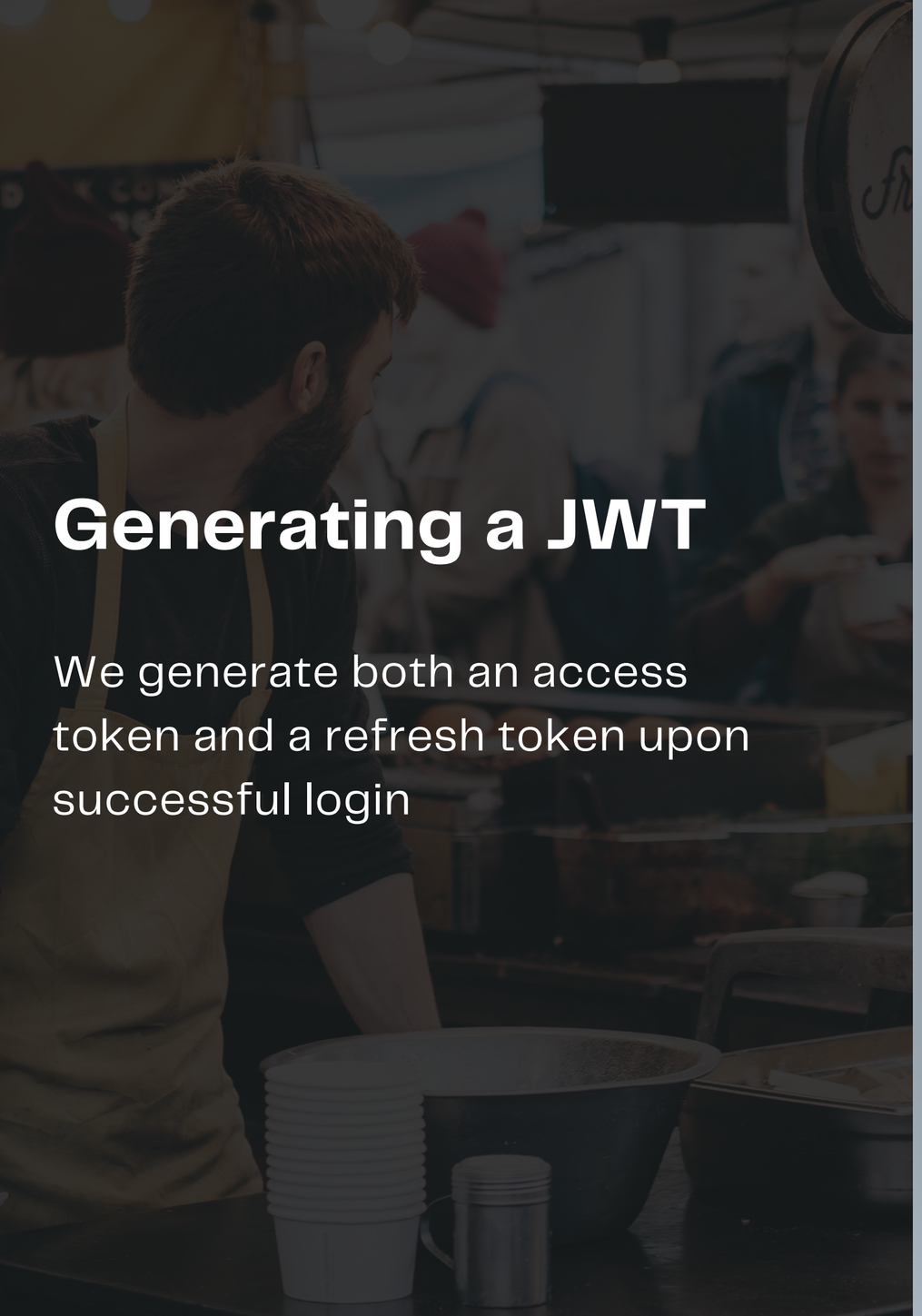
We need to verify and decode JWT upon user request

Confirm authorization

We need to allow or deny access based on authorization

Generate new token upon expiry

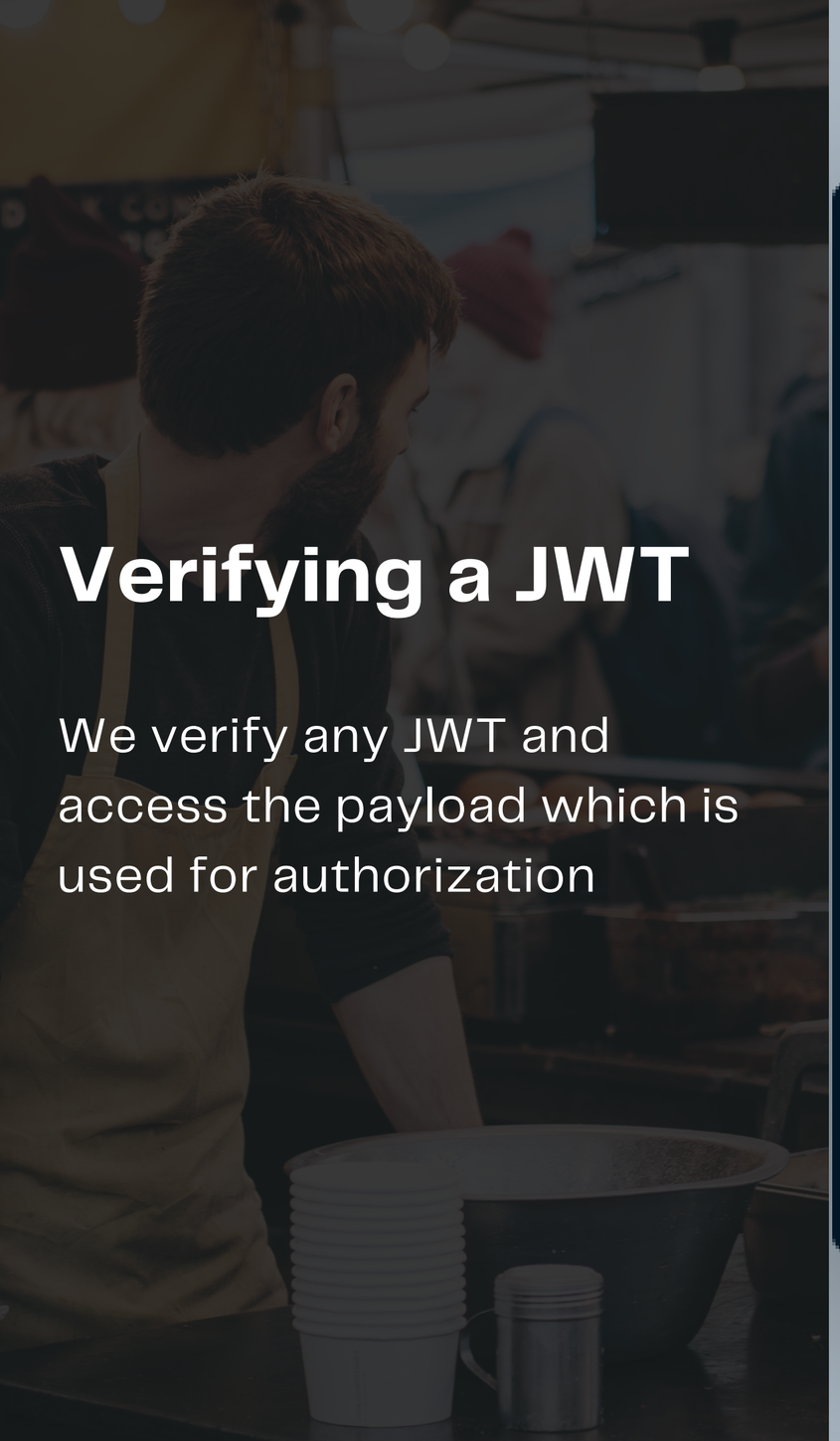
We need to generate



Generating a JWT

We generate both an access token and a refresh token upon successful login

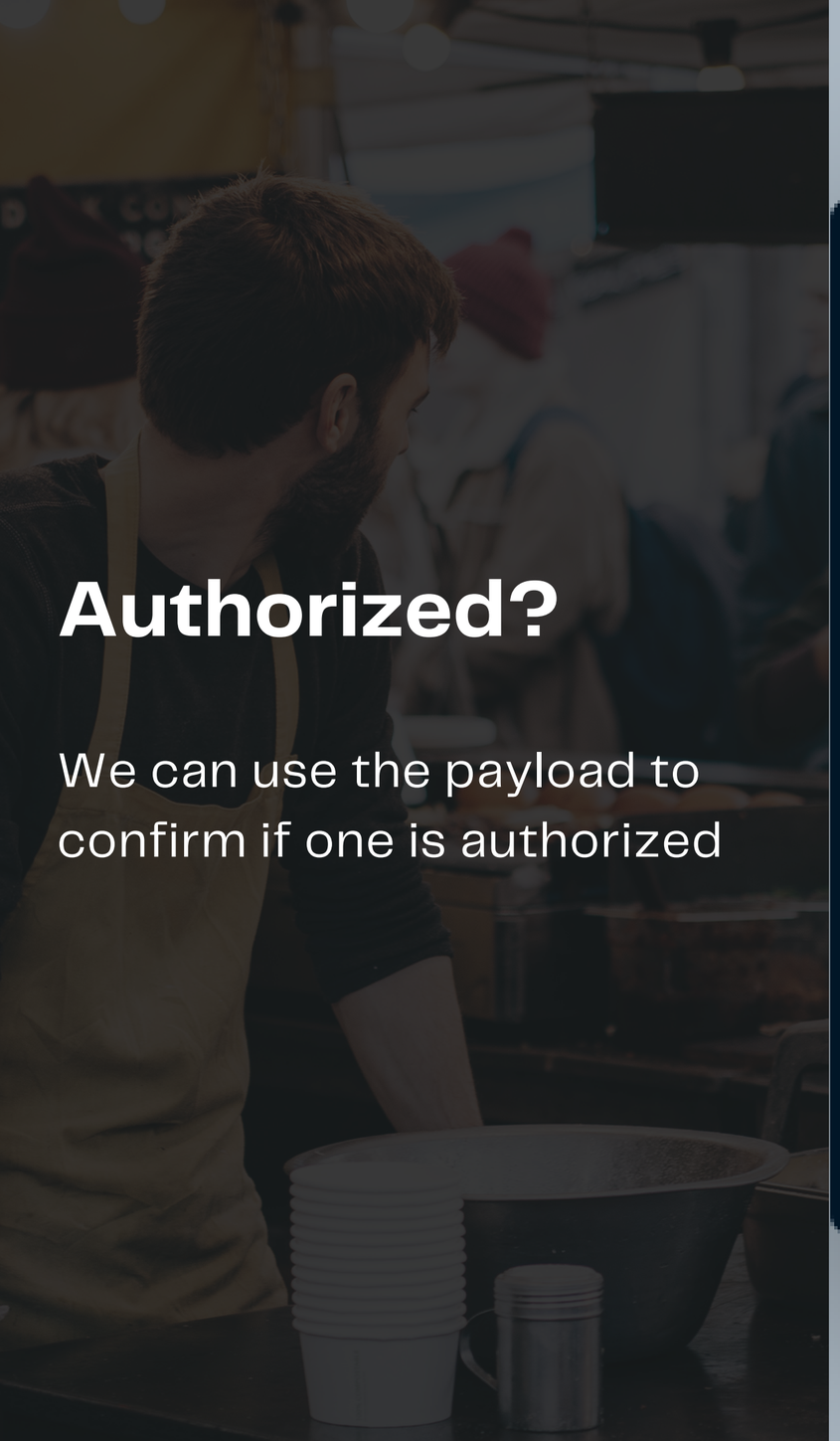
```
//import sign from jwt
const {sign} = require('jsonwebtoken');

//Create access token function
const createAccessToken = (payload) => {
    return sign({payload}, ACCESS_TOKEN_SECRET, {
        expiresIn: '15m'
    });
}

//Create refresh token function
const createRefreshToken = (payload) => {
    return sign({payload}, REFRESH_TOKEN_SECRET, {
        expiresIn: '7d'
    });
}
```



Verifying a JWT

We verify any JWT and access the payload which is used for authorization

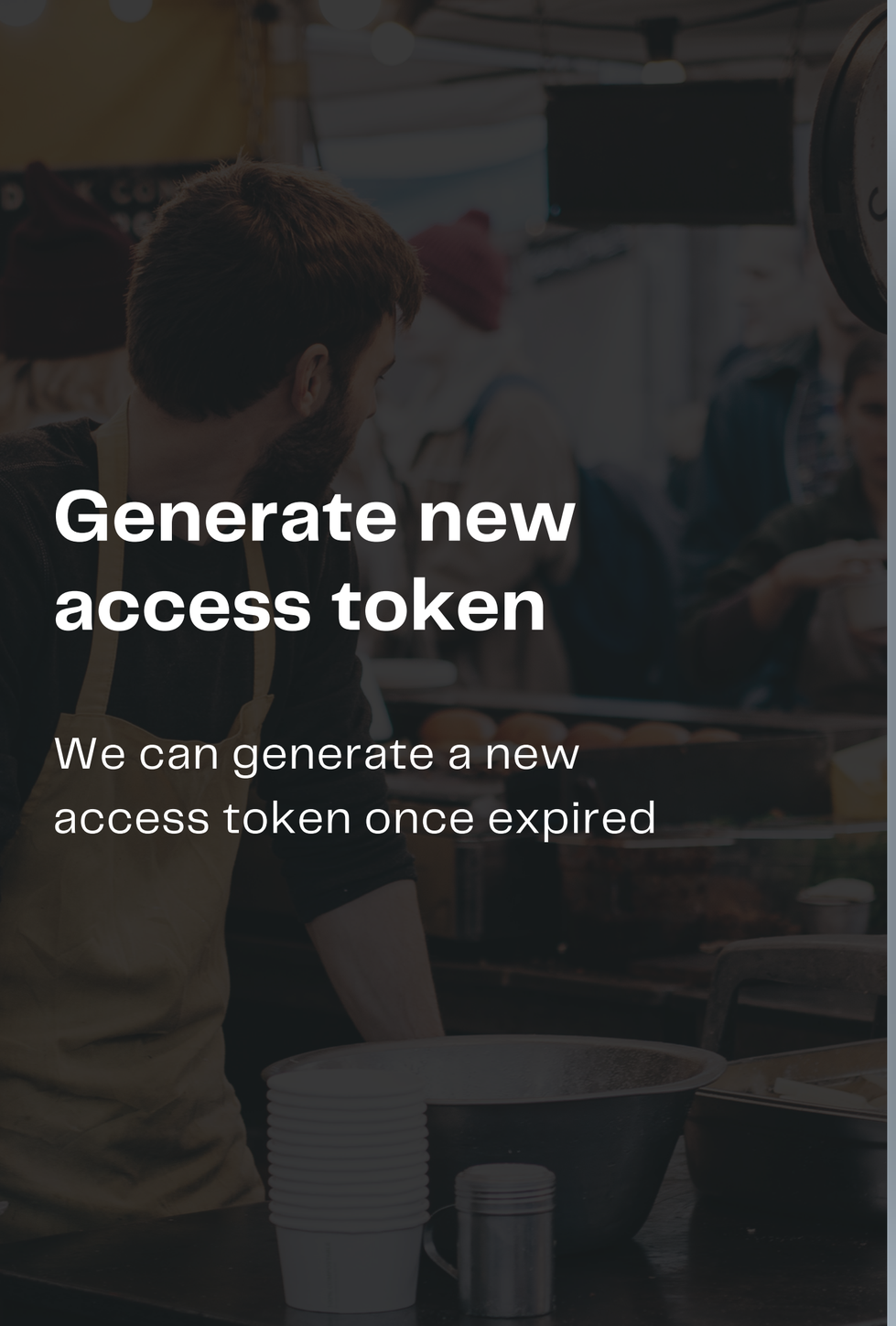
```
//import verify from jwt
const {verify} = require('jsonwebtoken');

//Check if authenticated
const isAuth = req => {
    const authorization = req.headers['authorization'];
    if(!authorization) throw new Error('You need to
login');
    //Bearer <token>
    const token = authorization.split(' ')[1];
    const {payload} = verify(token, ACCESS_TOKEN_SECRET);
    return {payload};
}
```



Authorized?

We can use the payload to confirm if one is authorized

```
app.post('/books', async (req, res) => {
    //pick book details from req.body
    try {
        //1. Check if user is logged in and get role
        const {payload} = isAuth(req);
        if(payload !== 'value') throw new Error('Not
authorized');Grant access
    } catch (error) {
        res.json({error: error.message});
    }
})
```



Generate new access token

We can generate a new access token once expired

```
app.post('/refresh_token', (req, res) => {
    //Get refresh token from cookies
    const token = req.cookies.refreshToken;
    //If we don't have a token in our request
    if(!token) return res.send({accessToken: ''});
    //If we have a token, verify it
    let payload = null;
    try {
        payload = verify(token, REFRESH_TOKEN_SECRET);
    } catch (error) {
        return res.send({accessToken: ''});
    }
    //Token exists, create new refresh and access token
    const accessToken = createAccessToken(_payload);
    const refreshToken = createRefreshToken(_payload);

    return res.send({accessToken});
})
```

LET'S DO A SIMPLE DEMO

CONCLUSION

Why you should consider JWTs

VERSATILE

JWTs can have various purposes: auth, info exchange e.t.c

SECURE AND VERIFIABLE

Are encrypted and signed by the issuer

SELF CONTAINED & COMPACT

The JWT contains all the required info needed


THANK YOU!

RESOURCE PAGE

GitHub Repo link:

bit.ly/jwt-session-code

Slides:

bit.ly/jwt-session-slides

Feedback Form:

bit.ly/jwt-session-feedback

SCAN TO GET SLIDES