Blok 1 – Client Server Interaction

Maaike van Putten
Software developer & trainer
www.brightboost.nl

Overview

HTTP & HTML FORMS

State management

AJAX & refresh components

PRG, validation & error handling

Last time's problem

HTTP has no memory

- Every request is brand new to the server

The stateless problem

Hi, my name is Maaike

Hello Maaike!

5 seconds later...

The stateless problem

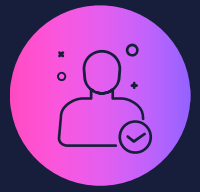
What's my name?

Who are you?

This is HTTP. Zero memory between requests.

What do these have in common?

What needs to be remembered across requests?



User is logged in as ”John”

Shopping cart has 3 items

User prefers Dutch language

User is in step 2 of checkout

Where do we store state?

Client-side

Browser stores the data

Server-side

Server stores the data

Cookies

Client-side storage

What are cookies?

Think of a membership card

- Server gives you a card
- You carry it with you
- You show it on every visit
- Server reads it to identify you

How cookies work

1. Browser → Server: GET /login

2. Server → Browser: Set-Cookie: Username=john

<html>Welcome!</html>

3. Browser saves cookie locally

4. Browser → server: GET /profile

Cookie: username=john

5. Server reads cookie

“This is john!”

Cookie ingredients

```
Set-Cookie: theme=dark; Max-Age=86400; Path=/; HttpOnly; Secure
```

Cookie ingredients

The actual cookie

```
Set-Cookie: theme=dark; Max-Age=86400; Path=/; HttpOnly; Secure
```

Cookie ingredients

How long to keep
(24 hours)

```
Set-Cookie: theme=dark; Max-Age=86400; Path=/; HttpOnly; Secure
```

Cookie ingredients

Which URLs get
this cookie

```
Set-Cookie: theme=dark; Max-Age=86400; Path=/; HttpOnly; Secure
```

Cookie ingredients

JS can’t access it.

```
Set-Cookie: theme=dark; Max-Age=86400; Path=/; HttpOnly; Secure
```

Cookie ingredients

Only sent over HTTPS

```
Set-Cookie: theme=dark; Max-Age=86400; Path=/; HttpOnly; Secure
```

Cookie attributes

Attribute	Purpose	Example
Max-Age	Lifespan in seconds	3600 = 1 hour
Path	Which URLs	\ = all URLs
HttpOnly	Block JavaScript access	Security flag
Secure	HTTPS only	Security flag
SameSite	CSRF protection	Lax or Strict

What to store in a cookie

Good uses

- User preferences (theme, language)
- Sessions IDs
- Analytics tracking
- ”Remember me” tokens

Bad uses

- Passwords
- Credit card numbers
- Large data (4KB limit)
- Sensitive information

Demo

- Write cookies
- ~~Eat~~ Read cookies

Exercise 1 (see GH)

Build a theme switcher:

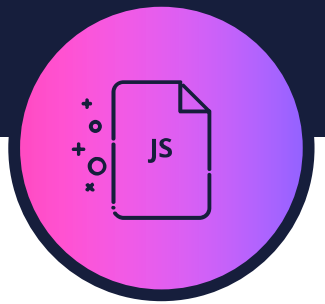
1. Create a page with theme selector form
2. Save theme preference in cookie
3. Read cookie to apply theme
4. Test in DevTools > Application > Cookies

Success criteria:

- Theme persists across refreshes
- Cookie visible in DevTools
- Can manually edit cookie

Cookie security problems

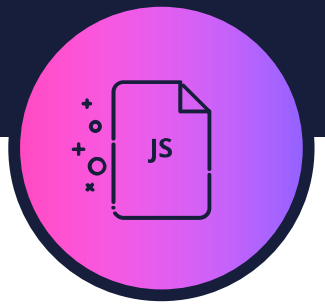
```
// (javascript)
// In browser console:
document.cookie;
// "theme=dark; username=john; sessionid=abc123"
```



Problem 1: users can read cookies

Solution: Use `HttpOnly` flag > JavaScript can’t access

```
// (javascript)
// User changes cookie:
document.cookie = "username=admin";
// Next request pretends to be admin!
```



Problem 2: users can modify cookies

Never trust cookies. Validate on server

Anyone on the network can read this!

```
GET /profile HTTP/1.1
Cookie: username=john; password=secret123
```

Problem 3: Sent in plain text (HTTP)

Solution: Always use HTTPS + `Secure` flag

Sessions

Server-side storage

What are sessions?

Think of a coat check

- You give your coat to the restaurant
- They give you a ticket (number)
- Your coat stays with them
- You only carry the small ticket
- Show ticket and get coat back

Sessions store data on the server

The user only carries a session ID (stored in a cookie)

How sessions work

1. User visits site

Browser → Server: GET /login

2. Server creates session

- Session ID: “abc123” (random)
- Storage: { username: null, cart: []}

How sessions work

1. User visits site

Browser → Server: GET /login

2. Server creates session

- Session ID: “abc123” (random)
- Storage: { username: null, cart: []}

3. Server sends ID via c

Server → browser: Set-Cookie: J

How sessions work

er creates session

n ID: “abc123” (random)
e: { username: null, cart: []}

3. Server sends ID via cookie

Server → browser: Set-Cookie: JSESSIONID=abc123

4. User logs in

Browser → Server: PO
Co

How sessions work

ia cookie

ie: JSESSIONID=abc123

4. User logs in

Browser → Server: POST / login
Cookie: JSESSIONID=abc123

5. Server updates

“abc123” → {username:

How sessions work

ST / login
okie: JSESSIONID=abc123

5. Server updates session storage

“abc123” → {username: “john”, cart: []}

6. User visits anoth

Browser → Server: Cookie

How sessions work

s session storage

e: “john”, cart: []}

6. User visits another page

Browser → Server: Cookie: JSESSIONID=abc123

7. Server looks up s

“Welcome john!”

How sessions work

s another page

Cookie: JSESSIONID=abc123

7. Server looks up session

“Welcome john!”

What's stored where?

Browser

```
JSESSIONID: abc123
```

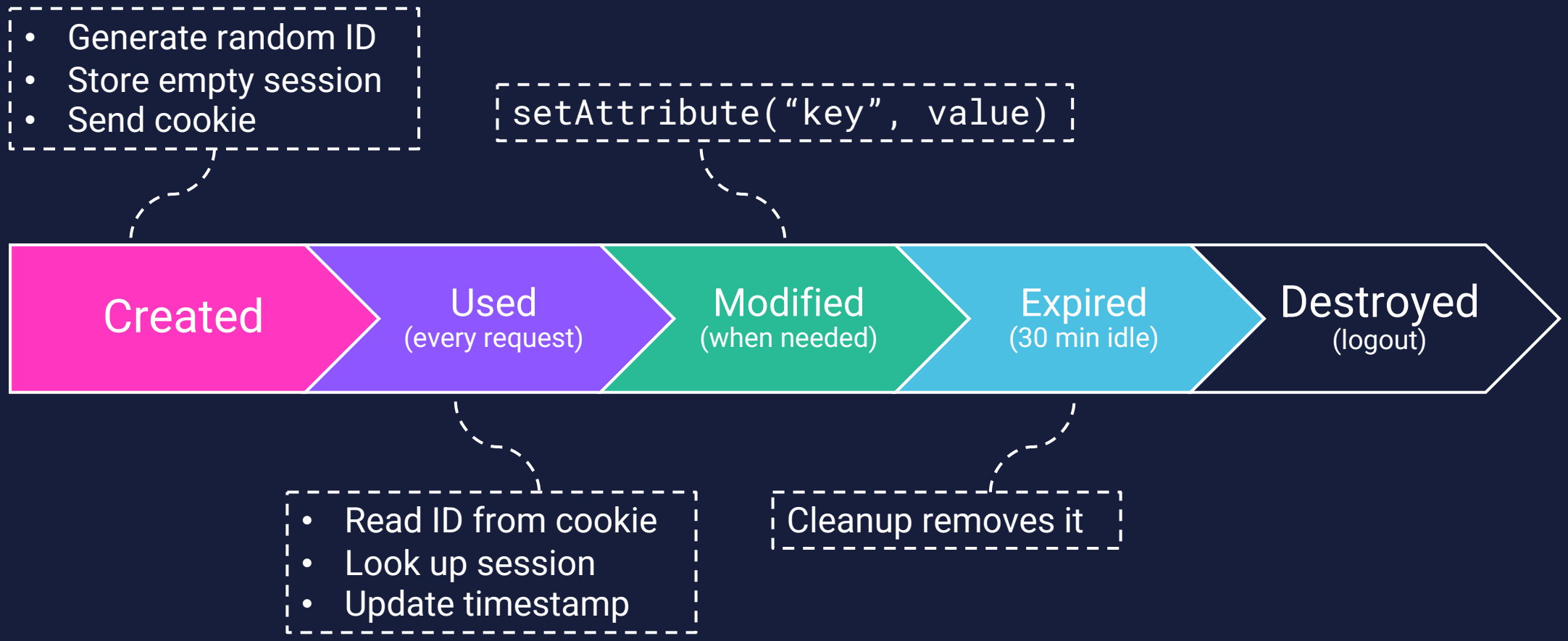
(Just the ID!)

Server

```
Session "abc123":
- username: "john"
- cart: [item1, item2]
- loginTime: ...
- role: "admin"
```

(All the actual data!)

Session lifecycle



Where to store sessions?

Type	Pros	Cons	Use case
In-memory	Fast, simple	Lost on restart	Development
Database	Survives restarts	Slower	Small apps
Redis	Fast + persistent	Requires setup	Production

We’ll build in-memory (simplest)

Session hijacking

The #1 security risk with sessions

How session hijacking works

Victim logs in at coffee shop

Server → Victim:
JSESSIONID=abc1213

Attacker on same WiFi sniffs traffic

Attacker sees:
JSESSIONID=abc1213

Attacker sets same cookie

Attacker → Server:
JSESSIONID=abc123

The server thinks the attacker IS the victim!

Session ID = identity

Defending against hijacking

- Always use HTTPS (+ `Secure` flag)
- Set `HttpOnly` flag (no JavaScript access)
- Set `SameSite` flag (CSRF protection)
- Regenerate ID after login
- Short timeout (30 minutes)

- Optional: IP validation (problematic with mobile)

Mini exercise 2 (see GH)

Build login system:

1. Create LoginPage with username/password form
2. Store username in session on successful login
3. Create ProfilePage that shows username from session
4. Test: Login and Profile shows name
5. Test: Close tab, reopen and still logged in

Success criteria:

- JSESSIONID cookie appears in DevTools
- Session persists across requests
- Different tabs share same session

Mini exercise 3 (see GH)

Session hijacking demo:

1. Login and view secret page
2. Copy JSESSIONID from DevTools
3. Open incognito window
4. Manually add cookie with same JSESSIONID
5. Access secret page and **it works!**

Discussion questions:

- Why did this work?
- How does HTTPS Prevent this?
- What else could we do?

⚠ This demonstrates why HTTPS is mandatory!

Cookies vs Sessions

Aspect	Cookies	Sessions
Stored	Browser	Server
Contains	Actual data	Session ID only
Size limit	~4KB	No limit
Security	User can read/edit	Hidden from user
Speed	Fast	Slightly slower
Survives restart	Yes	No (in-memory)
Best for	Preferences	Authentication

Cookies vs Sessions

Cookies

Browser:
theme: “dark”
language: “nl”
tracking: ”xyz”

Server:
(nothing)

All data travels with each request

Sessions

Browser:
JSESSIONID: “abc123”

Server:
Session abc123:
- username: ”john”
- cart: [...]
- preferences: {...}

Only ID travels, data stays on server

JWT tokens

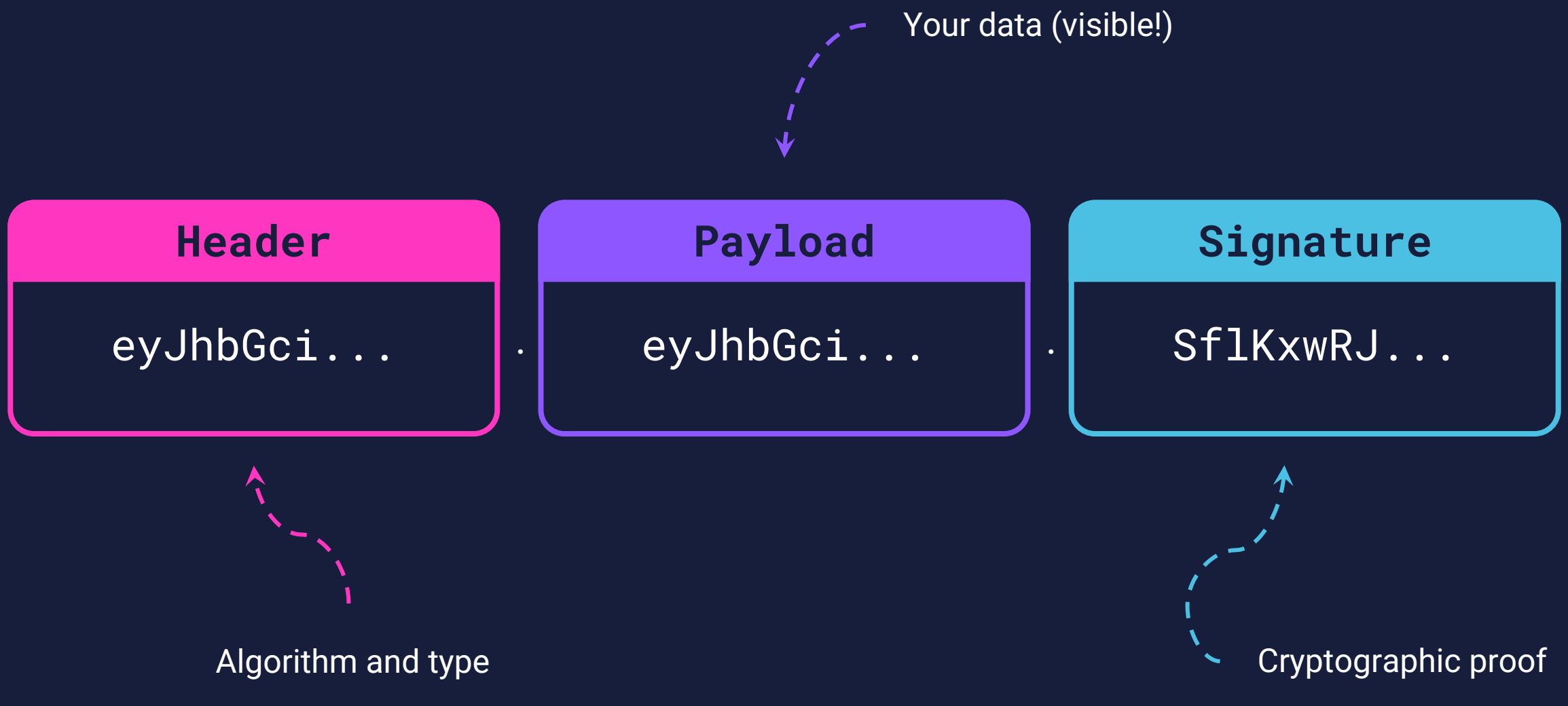
A third approach

What are tokens?

Think of a passport

- Contains your information
- Digitally signed (can’t be forged)
- You carry it
- Anyone can verify it’s real
- No database lookup needed

JWT structure



Anyone can read it, only server can verify it.

JWT decoded

Header

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

.

Payload

```
{
  "userId": 123,
  "username": "john",
  "role": "admin",
  "iat": 1516239022,
  "exp": 1516242622
}
```

.

Signature

```
HMACSHA256(
  header + payload,
  secretKey
)
```

JWT authentication flow

1. User logs in

Browser → Server: username + password

2. Server validates and

Token includes: {userId, use
Signs with secret key

JWT authentication flow

in

r: username + password

2. Server validates and creates JWT

Token includes: `{userId, username, role, exp}`
Signs with secret key

3. Server sends JWT

Server → Browser: {“tok

JWT authentication flow

nd creates JWT

username, role, exp}

3. Server sends JWT

Server → Browser: {“token”: “eyJhbGci...”}

4. Browser stores JWT

Server → Browser: {“token

JWT authentication flow

T

ken": "eyJhbGci..."}

4. Browser stores JWT

Server → Browser: {"token": "eyJhbGci..."}

5. Browser includes J

Browser → Server:
GET /api/profile
Authorization: Bearer eyJhbG

JWT authentication flow

s JWT

token": "eyJhbGci..."}

5. Browser includes JWT in requests

Browser → Server:
GET /api/profile
Authorization: Bearer eyJhbGci...

6. Server verifies JWT

- Checks signature
- Check expiration
- Extract user info

JWT authentication flow

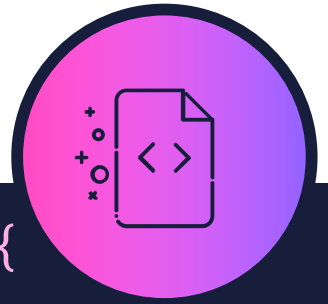
ser includes JWT in requests

→ Server:
profile
ion: Bearer eyJhbGci...

6. Server verifies JWT

- Checks signature
- Check expiration
- Extract user info

Generating JWT (Java)



```java
public static String generateToken(String username, String role) {
    Date now = new Date();
    Date expiry = new Date(now.getTime() + 86400000); // 24h

    return Jwts.builder()
        .setSubject(username)
        .claim("role", role)
        .setIssuedAt(now)
        .setExpiration(expiry)
        .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
        .compact();
}

// Returns: "eyJhbGciOiJIUzI1NiIsInR5c..."
```

Verifying JWT (Java)

```java
public static Claims validateToken(String token) {
    try {
        Claims claims = Jwts.parser()
            .setSigningKey(SECRET_KEY)
            .parseClaimsJws(token)
            .getBody();

        String username = claims.getSubject();
        String role = claims.get("role", String.class);

        return claims; // Valid!

    } catch (ExpiredJwtException e) {
        throw new UnauthorizedException("Token expired");
    } catch (SignatureException e) {
        throw new UnauthorizedException("Invalid token");
    }
}
```

```
// In browser console:
let token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...";
let payload = JSON.parse(atob(token.split(".")[1]));
console.log(payload);

// Shows:
// {
//   userId: 123,
//   username: "john",
//   role: "admin"
// }
```

Anyone can read JWT!

JWT is not encrypted, just signed!

Mini exercise 6 (see GH)

JWT tokens

JWT vs Sessions

Aspect	JWT	Sessions
Server storage	None (stateless)	Yes (stateful)
Scalability	Easy	Harder
Revocation	Hard	Easy
Size	Larger (~500 bytes)	Smaller (ID only)
Security	Contents visible	Contents hidden
Best for	APIs, microservices	Web applications

JWT vs Sessions

Sessions

Browser: JSESSIONID=abc123

↓

Server: Look up in storage

↓

Find user data

Stateful (server must store)

JWT

Browser: JWT with all data

↓

Server: Verify signature

↓

Read data from token

Stateless (no server storage)

When to use what?

Use JWT for:

- REST APIs
- Microservices
- Mobile apps
- Cross-domain auth
- Horizontal scaling

Use sessions for:

- Traditional web apps
- Wicket applications
- When you need immediate logout
- When storing large data
- When controlling both ends

What we've learned

- Why HTTP needs state management
- Cookies (client-side storage)
- Sessions (server-side storage)
- JWT tokens (stateless alternative)
- Security risks and defenses

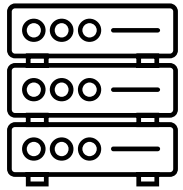
Remember these differences

Cookies

Client stores data

Small, visible

For preferences

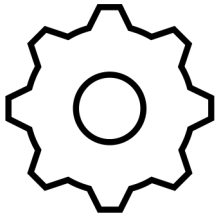


Sessions

Server stores data

Large, hidden

For authentication



JWT

Client stores + server validates

Stateless, visible

For APIs

Security first

- Always use HTTPS in production
- Never store passwords in cookies/JWT
- Always set `HttpOnly` on session cookies
- Use short session timeouts
- Regenerate session ID after login

Debugging tips

When you get stuck:

- Use DevTools > Application > Cookies
- Use DevTools > Application > Storage
- Log session ID on every request
- Check cookie attributes in Network tab
- Print session contents to console

Next up:

Kaasus time!

Next up:

Session 3: Ajax & refresh components

Questions or suggestions?

Maaike.vanputten@brightboost.nl

See you next time! 💖