JSON Web Tokens (JWT)

Token-based Authentication

Learning Objectives

- Explain the purpose of token-based authentication.
- Describe the various parts of a JSON web token.
- Implement JWT authentication in a full stack application using the **jsonwebtoken** library.

Authentication

Authentication

- Ideally we want users to log in only **once** to our application and to continue to be authenticated while accessing the app
- When a user logs in: Are you who you say you are?
- When a user takes further action: Who is this request coming from again?
  - HTTP is stateless
- Analogy: when you check into a hotel they give you a unique key card that you can use to access amenities and your room, but not other people's rooms.

Authentication Flow

1. User inputs their credentials (username and password) and clicks “submit”.
2. Server receives credentials, verifies user, and sends back a token.
3. Future requests from the client include the token.
4. Future requests then received by the server verify and decode the token so the server knows who made the request.

JSON Web Tokens (JWT)

Token use

- Server side: used to verify that an incoming request is valid and that the user has permissions to do their requested action
  - Some actions may be admin-only
- Client side: send with each request to verify identity and avoid having to constantly send usernames and passwords

Anatomy of a Token

- A token consists of three parts separated by a “ . ” :
  - Header: identifies which algorithm is used to generate the signature
  - Payload: holds claims (or key-value pairs)
  - Signature: validates the token, makes sure it wasn’t altered
- It looks something like: header.payload.signature
- The tokens are serialized using base64, so don’t panic when it looks jumbled up!
  - Base64 is a method of encoding binary values as text

Anatomy of a Token: Example

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.h4rrxefpgMJi-rOvmJFTKNKhOpziPFKoNvKm6XB3SAM
```

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),
  very-secret-123
) secret base64 encoded
```

npm install jsonwebtoken

jsonwebtoken

- A node library that implements jwt
- We will use two main functions from it:
  - sign: Takes in data and a secret, returns a token
  - verify: Takes in a token and a secret, returns the data if valid or throws an error if not

Example of using jsonwebtoken library

```
1   const jwt = require('jsonwebtoken');
2
3   // secret key to sign your tokens, private to server
4   const SECRET_KEY = 'bananapotatobanana';
5
6   // create a token
7   // ex: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJ1c2VybmFtZSI6InN1cmZlckdhbDQ4IiwiaWF0IjoxNjEwMDQ4MDk2fQ.
    7zZR2jMb5lFSa8lsks0OgJosxKs3ZhiU1SEficAWhPM
8   const token = jwt.sign({ username: 'surferGal48' }, SECRET_KEY);
9
10  // verify it's a valid token and return the data
11  // ex: { username: 'surferGal48', iat: 1610048204 }
12  const verifyGood = jwt.verify(token, SECRET_KEY);
13
14  // Throws JsonWebTokenError: invalid token
15  const verifyBad = jwt.verify('jioejiwjf.jjiojefi.randomrandom', SECRET_KEY);
```

Frequently Asked Questions

So what happens if someone gets access to my token?

- This would be bad! Anyone with your token can essentially try to impersonate you.
- To avoid problems associated with this, it's good practice to give your token a pretty short expiration time.
- Example:

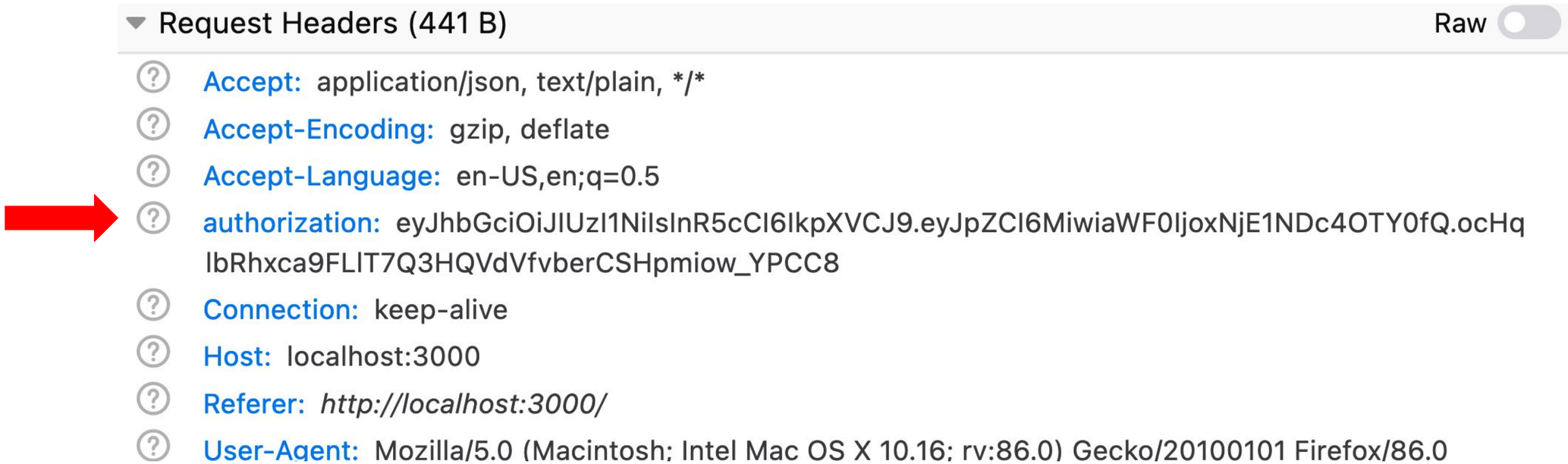
```
jwt.sign({
  data: 'foobar'
}, 'secret', { expiresIn: '1h' });
```

Why would I choose JWT over sessions?

- Less things to store and manage on the server!
- When you use sessions, you have to store all your session ids on the server and manage which ones are active vs inactive
- With JWT the server doesn't store anything. It's completely on the client to store the token. The jwt verify and decode methods only require the secret string to verify the token.

How does the client store and send JWTs?

- Storage: Use localStorage available from the browser
- Sending: Set the JWT as an “Authorization” header on HTTP requests



How do we keep the secret safe?

- The secret value used to generate a token's signature should be kept safe
- What if we want to push our code to Github?
- Environment variables can be set on your local machine and called from inside your code:
  - From the terminal: `JWT=yourSecretKeyHere`
  - From your code: `const SECRET_KEY = process.env.JWT;`