# / JAVA DB SCHOOL ADVANCE SPRING 4

## / Recap

## / Security

## Web Security

Website security is any action or application taken to ensure website data is not exposed to cybercriminals or to prevent exploitation of websites in any way.

Website security protects your visitors from:

- Stolen data
- Phishing schemes
- Session hijacking
- Malicious redirects

## HTTP

- HTTP stands for Hypertext Transfer Protocol. At it’s most basic, it allows for the communication between different systems.
- It’s most commonly used to transfer data from a web server to a browser in order to allow users to view web pages.
- It’s the protocol that was used for basically all early websites.

## HTTPS

- HTTPS stands for Hypertext Transfer Protocol Secure.
- The problem with the regular HTTP protocol is that the information that flows from server to browser is not encrypted, which means it can be easily stolen.
- HTTPS protocols remedy this by using an SSL (secure sockets layer) certificate, which helps create a secure encrypted connection between the server and the browser, thereby protecting potentially sensitive information from being stolen as it is transferred between the server and the browser.

## Authentication

- Authentication is the process of identifying users and validating who they claim to be.
- One of the most common and obvious factors to authenticate identity is a password.
- If the user name matches the password credential, it means the identity is valid, and the system grants access to the user.

## Authorization

- Authorization happens after a user’s identity has been successfully authenticated. It is about offering full or partial access rights to resources like database, funds, and other critical information to get the job done.

## / Basic Auth

## Protocol

- In the context of an HTTP transaction, basic access authentication is a method for an HTTP user agent (e.g. a web browser) to provide a user name and password when making a request.
- In basic HTTP authentication, a request contains a header field in the form of
    - Authorization: Basic <credentials>,
    - where credentials is the Base64 encoding of ID and password joined by a single colon :.

## Base64 in Java

```java
byte[] decodedBytes = Base64.getDecoder().decode(encodedString);
String decodedString = new String(decodedBytes);
```

```java
String originalInput = "test input";
String encodedString = Base64.getEncoder().encodeToString(originalInput.getBytes());
```

## Base64

- In programming, Base64 is a group of binary-to-text encoding schemes that represent binary data (more specifically, a sequence of 8-bit bytes) in an ASCII string format by translating the data into a radix-64 representation.
- Common to all binary-to-text encoding schemes, Base64 is designed to carry data stored in binary formats across channels that only reliably support text content. Base64 is particularly prevalent on the World Wide Web where its uses include the ability to embed image files or other binary assets inside textual assets such as HTML and CSS files

## Concerns

- The BA mechanism does not provide confidentiality protection for the transmitted credentials. They are merely encoded with Base64 in transit and not encrypted or hashed in any way.
- Therefore, basic authentication is typically used in conjunction with HTTPS to provide confidentiality.
- Because the BA field has to be sent in the header of each HTTP request, the web browser needs to cache credentials for a reasonable period of time to avoid constantly prompting the user for their username and password. Caching policy differs between browsers.

## Logout

- The browser should clean the cache.

-
  ```
  <script>document.execCommand('ClearAuthenticationCache');</script>
  ```

## / API Key Auth

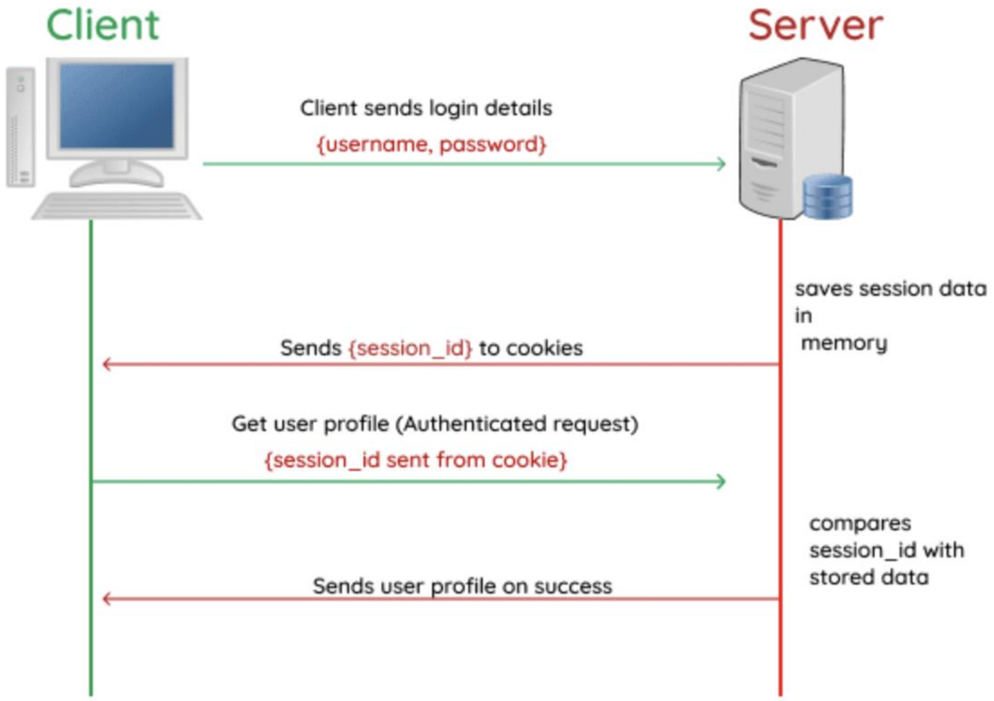
## TL;DR

“ Passwords are for humans, API keys for automated tasks or applications. Using API keys, you don’t have to worry about expiring passwords or multi-factor authentication in your automation. But make sure to keep those API keys secret.”

## Introduction

- Similar with Basic Auth a secret is sent using a header
- This secret represents the API Key which is a random String which is stored by the server and checked each time
- Usually this kind of authentication is used between services
- The Api key should not be exposed to the clients

## / Session Auth


Client
Server
Client sends login details
{username, password}
saves session data in memory
Sends {session_id} to cookies
Get user profile (Authenticated request)
{session_id sent from cookie}
compares session_id with stored data
Sends user profile on success

Session based authentication

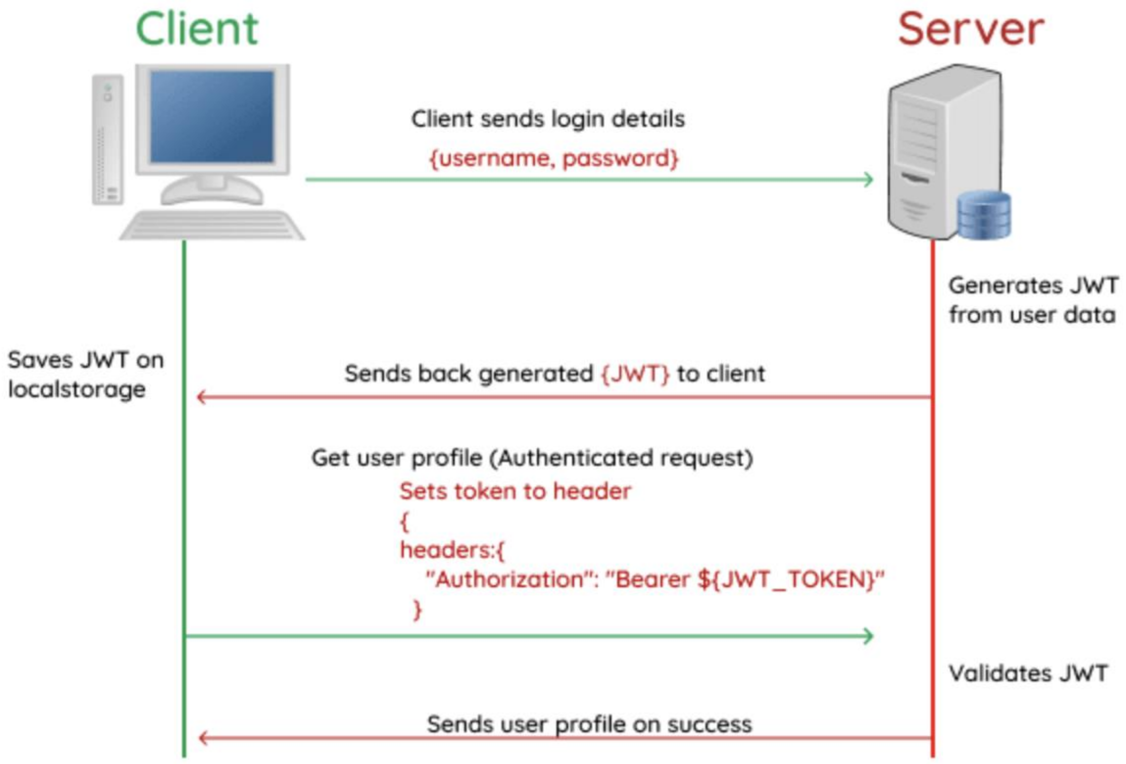
## Session Based Auth

- Session based authentication is one in which the user state is stored on the server’s memory.
- When using a session based auth system, the server creates and stores the session data in the server memory when the user logs in and then stores the session id in a cookie on the user browser.
- The session id is then sent on subsequent requests to the server and the server compares it with the stored session data and proceeds to process the requested action.

## Session Based Auth

- To obtain the session that is stored on the server side but also in browser / client cache the user should present a proof of identity
  - Username + password
  - Email + sms code
- Logout is done by deleting the session id from server side and client cache

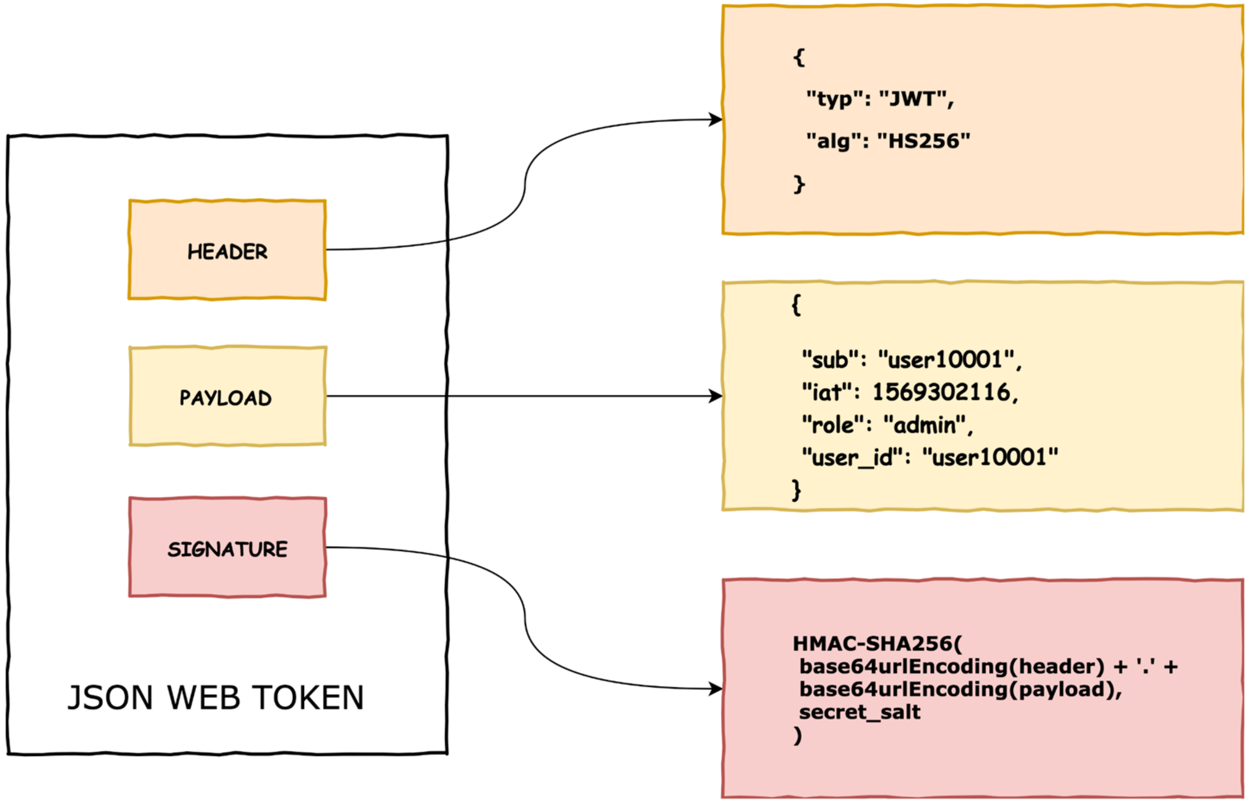
## / JWT


Client
Server
Client sends login details
{username, password}
Generates JWT from user data
Saves JWT on localstorage
Sends back generated {JWT} to client
Get user profile (Authenticated request)
Sets token to header
{
headers:{
"Authorization": "Bearer ${JWT_TOKEN}"
}
Validates JWT
Sends user profile on success

Token based authentication

## JWT Auth

- Token based authentication is one in which the user state is stored on the client. This has grown to be the preferred mode of authentication for RESTful APIs. In the token based authentication, the user data is encrypted into a JWT (JSON Web Token) with a secret and then sent back to the client.
- The JWT is then stored on the client side mostly localStorage and sent as a header for every subsequent request. The server receives and validates the JWT before proceeding to send a response to the client.

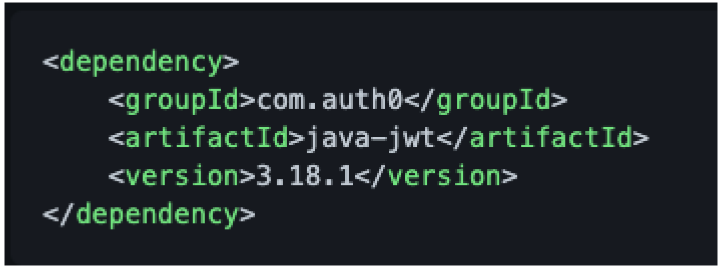


```
{
 "typ": "JWT",
 "alg": "HS256"
}
```

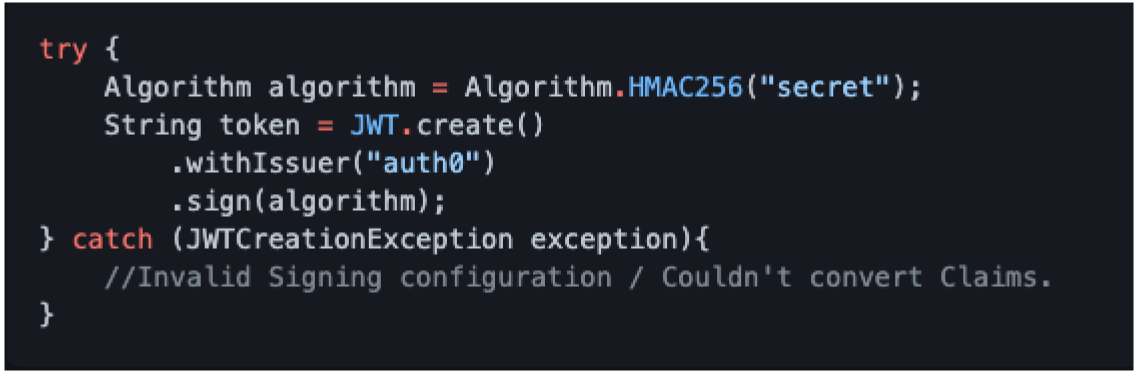
```
{

 "sub": "user10001",
 "iat": 1569302116,
 "role": "admin",
 "user_id": "user10001"
}
```

```
HMAC-SHA256(
 base64urlEncoding(header) + '.' +
 base64urlEncoding(payload),
 secret_salt
)
```

## JWT in Java



```xml
<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>3.18.1</version>
</dependency>
```



```java
try {
    Algorithm algorithm = Algorithm.HMAC256("secret");
    String token = JWT.create()
        .withIssuer("auth0")
        .sign(algorithm);
} catch (JWTCreationException exception){
    //Invalid Signing configuration / Couldn't convert Claims.
}
```

## Pro & Cons

- Pro:
  - No DB storage is required
  - Anyone which has the public key can validate the jwt
- Cons:
  - Hard to logout
  - Refresh policy should be implemented

## / Practice, practice, practice

## Implement a login with jwt without spring security

- Update the customer table to contains also a password
- Update the create customer endpoint to add password
- Add a login endpoint (username and password) which returns a jwt if the user-password pair is valid
- Without spring security update the order endpoint to obtain the client identity from the jwt
    - A header can be used to send the jwt
    - The jwt should be validated

/ Q&A

MOBILE ACADEMY