Securing APIs

Module 2: 15

Today’s Objectives

1. Authentication
2. JWT
3. Authorization
4. Using JWT and Authorization in Spring Boot
5. Using JWT in the Client

Authentication

Authentication is the act of validating that users are whom they claim to be. This is the first step in any security process.

Common forms of authentication:

- Username/Password
- One-time pins
- Authentication apps.
- Biometrics

Authentication Process

1. Credentials only transported by POST using TLS (HTTPS)
2. Error messages should be generic and not identify the source of the failure
   a. Bad Error Messages
      - Invalid Password
      - Login failed, invalid ID
      - Account disabled
      - Unable to login, the user is not active
   b. Good Error Message
      - Login failed; invalid User Id or Password

If HTTP is a stateless protocol, how does login work?

Shouldn’t the user need to login again with each request?

JWT (JSON Web Tokens)

An Internet Standard for creating self contained data with an optional signature and/or encryption to secure a number of claims. The tokens are signed either using a private or public key.

Allows for a client to identify itself to the server as having already been authenticated allowing for multiple requests after login.

This allows for multiple requests to occur after a single login regardless of the stateless nature of HTTP and REST.

JWT Workflow

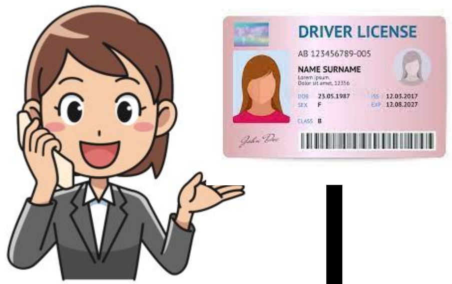
On login the server generates a *Secure Token* with a JSON payload and a signature.

The client then sends this token with each further request, the server can verify it was created by the server using the signature.

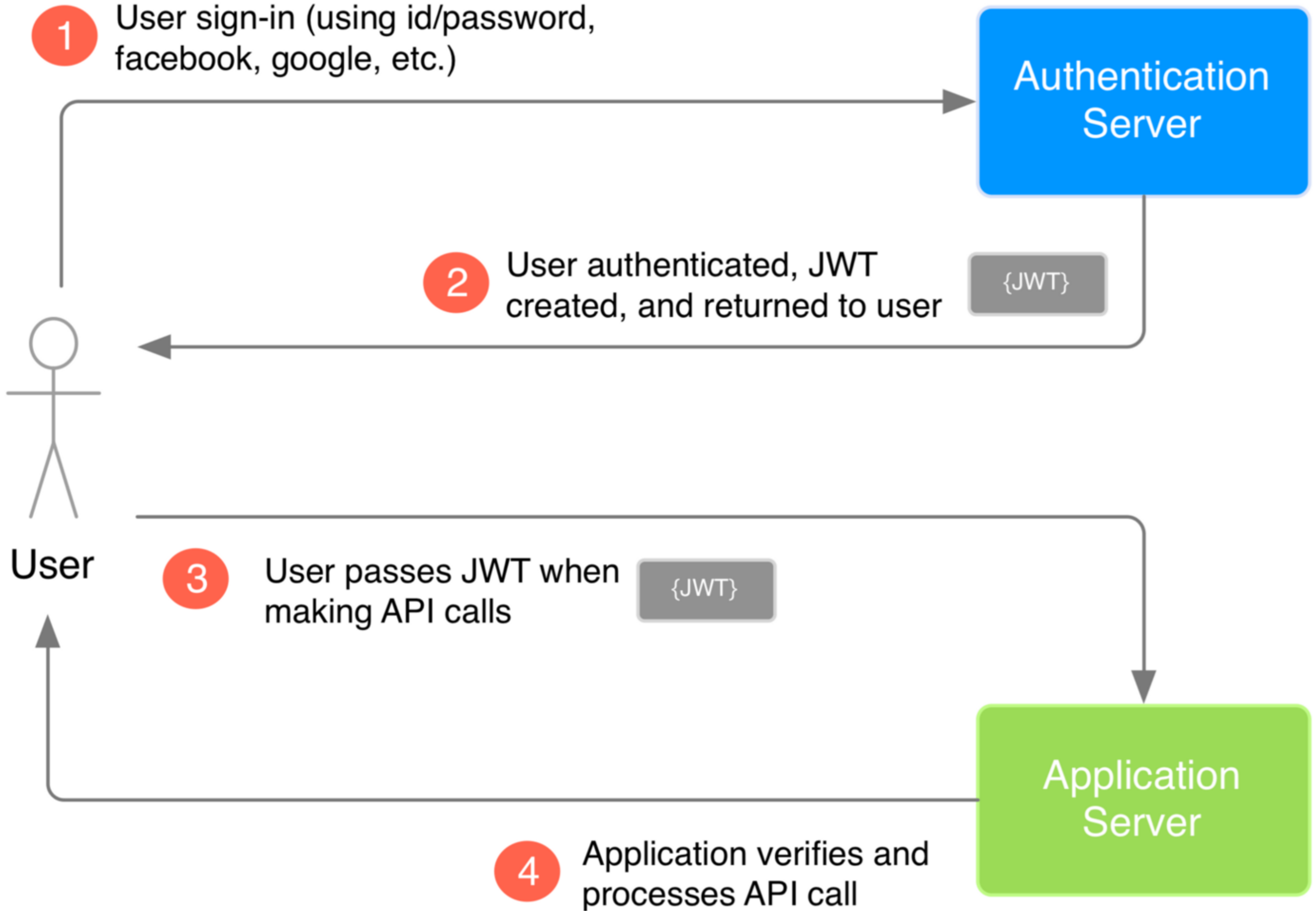
The payload contains information about the user, when it expires, and what the user is allowed to do (Authorizations).

This payload is called a **claim**.

The BMV issues an ID that identifies details about the holder (the claim)



The ID is presented to a bartender to identify the holder and prove drinking age. The bartender can verify the ID is valid using security features built into the ID (the signature)


1
User sign-in (using id/password, facebook, google, etc.)
Authentication Server
2
User authenticated, JWT created, and returned to user
{JWT}
User
3
User passes JWT when making API calls
{JWT}
Application Server
4
Application verifies and processes API call

The JWT Token

Header

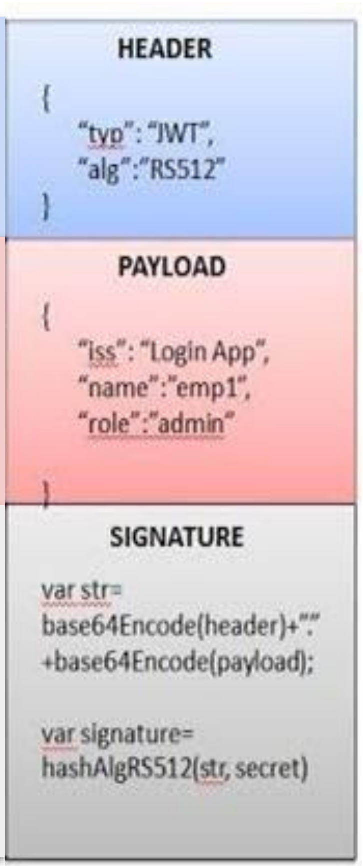
Identifies the type of Token and the algorithm used to generate it.

Payload

Contains the “claims” or who the user is and what they are authorized to do.

Signature

Contains a hash of the header, the payload, and a secret key known only to the server. The server can then recreate the hash to verify the token is valid.



HEADER

```
{
    "typ":"JWT",
    "alg":"RS512"
}
```

PAYLOAD

```
{
    "iss":"Login App",
    "name":"emp1",
    "role":"admin"
}
```

SIGNATURE

```
var str=
base64Encode(header)+"."
+base64Encode(payload);

var signature=
hashAlgRS512(str, secret)
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NT
Y3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5M
DIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

JWT Token Decoding Tool

1 Header

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

2 Payload

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

3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

The JWT Payload

sub (Subject) - whom the token is for (often the username)
auth (Authorities/Roles) - What the user is authorized to do in the application
exp (Expires) - Timestamp of when the token expires

The **Issued at Time (iat)** is a Unix timestamp, which is the number of seconds, minus leap seconds, that have elapsed since the Unix Epoch: 1970-01-01 00:00:00 UTC

Authorization

Authorization is the process of giving users to access specific resources or functionality in an application.   Authorities or Access Controls determine what privileges a user has within an application.

Role Based Authorization

- Accessed decision based on the individual’s responsibilities within an organization (**Role**).
- Easy to understand and administer
- *Examples:* Manager vs Employee, Doctor vs Lab Tech vs Patient

Permission Based Authorization

- Accessed decision based on who the identity of the individual.
- Applies when permissions need to be user-specific
- *Examples:* A user can see only their 401K or paycheck, Only Aniyah can DROP the Customer Table.

Authentication vs Authorization

Authentication is the “key” to the application. It lets you in, but does not say what you can do once you get in. (e.g. Login, New user Registration)

A door person checks IDs and allows entry into an event based on criteria like age or holding a ticket.

Authorization (Access Control) says what a user can do one they have been permitted entry. (e.g. you can see only your paycheck, only a manager can assign work)

NOTICE

Employees only

Rules inside the event determine what the person can do once inside, like staff-only areas.

Setting Authorization Roles On the Server

Spring has a security framework for JWT called *Spring Security*, which can be used with Spring Boot to add JWT Authentication and Authorization to an API.

The @PreAuthorize annotation can be added to controller methods to enforce authentication or authorization before the method can be called. Once applied if the user does not have permission to access the resource, then a **401 “Not Authorized”** status will be returned

The isAuthenticated() method can be used with @PreAuthorize to verify that the user has a valid *JWT token* prior to the method being called.

```
@PreAuthorize("isAuthenticated()")
@RequestMapping(path = "/hotels", method = RequestMethod.GET)
public List<Hotel> list() {
    return hotelDAO.list();
}
```

@PreAuthorize() can be applied at the class level of the controller to be applied to every controller method in it.

Using JWT from the Client

1. Login with a POST request
2. Retrieve the returned JWT security token
3. Set the token in an Authorization header as “Bearer token” in further requests

```
Authorization : "Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2VyIiwiYX…"
```

In Postman the Authorization tab on the request can be used to create the header. Select “Bearer Token” as the authorization type and paste the retrieve token into the token field.

Authorization by Class & Anonymous Access

- Authorization rules can be specified for the entire class by adding the **`@PreAuthorize`** annotation on the class itself.
- Can override rule at method level if needed
- Anonymous (non-authenticated) access can be granted with **`@PreAuthorize("permitAll")`**

Other Server Side Authorization restrictions

1. Anonymous (Guest) Access (allow for anyone)

```
@PreAuthorize("permitAll")
```

1. Roles can be checked before an action is taken

```
@PreAuthorize("hasRole('ADMIN')")
```

Method level @PreAuthorize() settings override the class level.

So if the class is set to isAuthenticated() then a particular method can be given more or less restrictive access by adding a @PreAuthorize annotation to that method.

Related HTTP Status Codes

401 - Not Authorized

Authentication Error

Returned when a user that has not yet been authenticated tries an action that requires authentication.

403 - Forbidden

Authorization Error

Returned when a user who has been authenticated tries an action that they are not authorized to do.

Getting the Current User on the Server

Methods with @RequestMapping will be called by Spring Boot. Pre-defined arguments can be added to the method signature to have Spring Boot pass our controller method objects that we need.

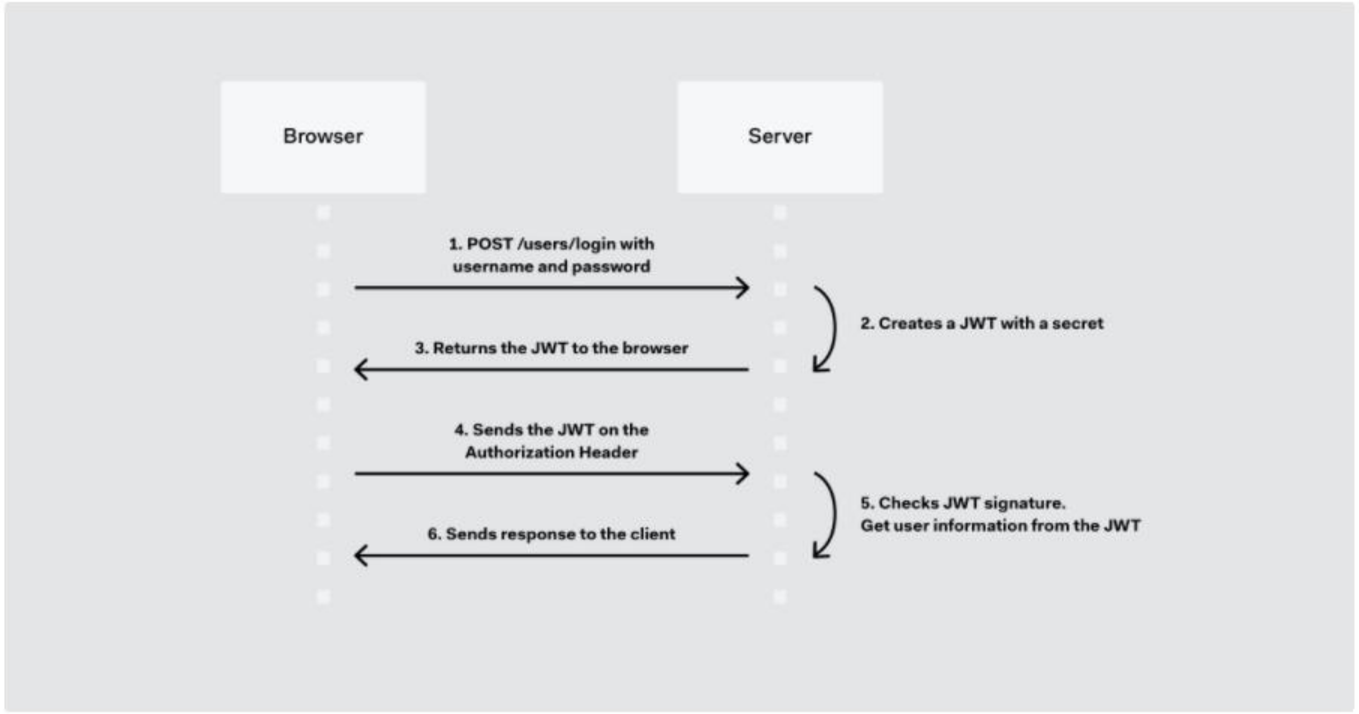
Add an argument for **Principal principal** to the controller method. Principal is an object that represents the current user. With this argument in the method signature, Spring Security will pass the current Principal object and getName() can be called to get the username.

```
  @PreAuthorize("hasRole('ADMIN')")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    @RequestMapping(path = "/reservations/{id}", method = RequestMethod.DELETE)
    public void delete(@PathVariable int id, Principal principal) throws
ReservationNotFoundException {
        auditLog("delete", id, , principal.getName());
        reservationDAO.delete(id);
    }
```

Using JWTs in a client app

1. POST the credentials to Login
2. Retrieve the token from the response body
3. Store the token for use in future requests
4. Sending the JWT

How JWT works


Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the browser
4. Sends the JWT on the Authorization Header
5. Checks JWT signature. Get user information from the JWT
6. Sends response to the client

```
public class AuthenticationService {
    private static final String API_BASE_URL = "http://localhost:8080/";
    private final RestTemplate restTemplate = new RestTemplate();

    public String login(String username, String password) {
        CredentialsDto credentialsDto = new CredentialsDto();
        credentialsDto.setUsername(username);
        credentialsDto.setPassword(password);
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        HttpEntity<CredentialsDto> entity = new HttpEntity<>(credentialsDto, headers);
        String token = null;
        try {
            //**** Add code here to send the request to the API and get the token from the response.
            ResponseEntity<TokenDto> response = restTemplate.exchange(API_BASE_URL + "login", HttpMethod.POST, entity,
TokenDto.class);
            TokenDto body = response.getBody();
            if (body != null) {
                token = body.getToken();
            }
        } catch (RestClientResponseException | ResourceAccessException e) {
            BasicLogger.log(e.getMessage());
        }
        return token;
    }

}
```

Sending the JWT

Since a Authorization header needs to be added to get requests, the exchange() method must be used for authorized requests instead of getForObject().

1. Create an HTTP Entity with the **Authorization Header** and JWT

```
HttpHeaders headers = new HttpHeaders();
headers.setBearerAuth(AUTH_TOKEN);
HttpEntity entity = new HttpEntity<>(headers);
```

1. Include the header in the GET request using the RestTemplate.exchange() method. With Exchange, the HTTP Method must be specified and getBody() must be called to deserialize the JSON into an Object.

e.g.

```
hotels = restTemplate.exchange(BASE_URL + "hotels", HttpMethod.GET,
                               makeAuthEntity(), Hotel[].class).getBody();
```

The exchange Method

```
private HttpEntity makeAuthEntity() {
    HttpHeaders headers = new HttpHeaders();
    headers.setBearerAuth(AUTH_TOKEN);
    HttpEntity entity = new HttpEntity<>(headers);
    return entity;
}

hotels = restTemplate.exchange(BASE_URL + "hotels",
    HttpMethod.GET, makeAuthEntity(), Hotel[].class).getBody();
```

Using **`exchange`** here allows us to attach an **`HttpEntity`** with **`AUTH_TOKEN`** in **`Bearer Auth`** header even though **`GET`** has no body to attach headers to in the **`HttpEntity`**

DTO – Data Transfer Object

- A data transfer object (DTO) is an object that carries data between processes. DTOs have flat structures without any business logic. A DTO only contains storage, and accessors…

Let’s Code