Softwareentwicklung (SW)
APIs mit JSON Web Token (JWT)

Prof. Dr. Alixandre Santana
alixandre.santana@oth-regensburg.de

Wintersemester
2024/2025

Lernziele

- To understand token authentication
- To Generate a JWT
- To authenticate a user using a token with Postman
- To programmatically authenticate a user and then make a request (using a RestController)

Agenda

1. Motivation
2. JWT Flow Architecture
3. Implementation of the JWTFilter and JWT Service
4. Testing the API with Postman
5. Implementation of a Client to consume the API with a Token

1- REST and Authentication

- Is your API public or private?
- How sensitive is the data?

Solutions:

- ❑ HTTPS Communication
- ❑ Authentication/Authorization (Spring Security):
- ❑ HttpBasic, HttpDigest, Token Based Authorization, OAuth

1- REST and Authentication

It is up to each analyst or developer to use the insights discussed here to identify situations where the systems they work on can benefit from using this technology.

1- The need of Stateless Authentication

- **Definition**: Stateless authentication is an authentication method where each request from the client to the server must contain all the information needed to understand and process the request. The server does not store any session information about the client.
- **Key Concept**: Unlike traditional session-based authentication, stateless authentication does not rely on storing session data on the server.
- **Examples**: JSON Web Tokens (JWT), OAuth 2.0.

1- Advantages of Stateless Authentication

- **Scalability**: Easier to scale because the server does not need to maintain session state information.
- **Performance**: Reduces server load and latency as it eliminates the need to read/write session data.
- **Security**: Minimizes the risk of session hijacking since there are no sessions to hijack.
- **Simplicity**: Simplifies architecture, especially for microservices and distributed systems.

Agenda

1. Motivation
2. JWT Flow Architecture
3. Implementation of the JWTFilter and JWT Service
4. Testing the API with Postman
5. Implementation of a Client to consume the API with a Token

2- Tokenbasierte Autorisierung - Steps

1 –The client application authenticates itself to the server by providing its credentials

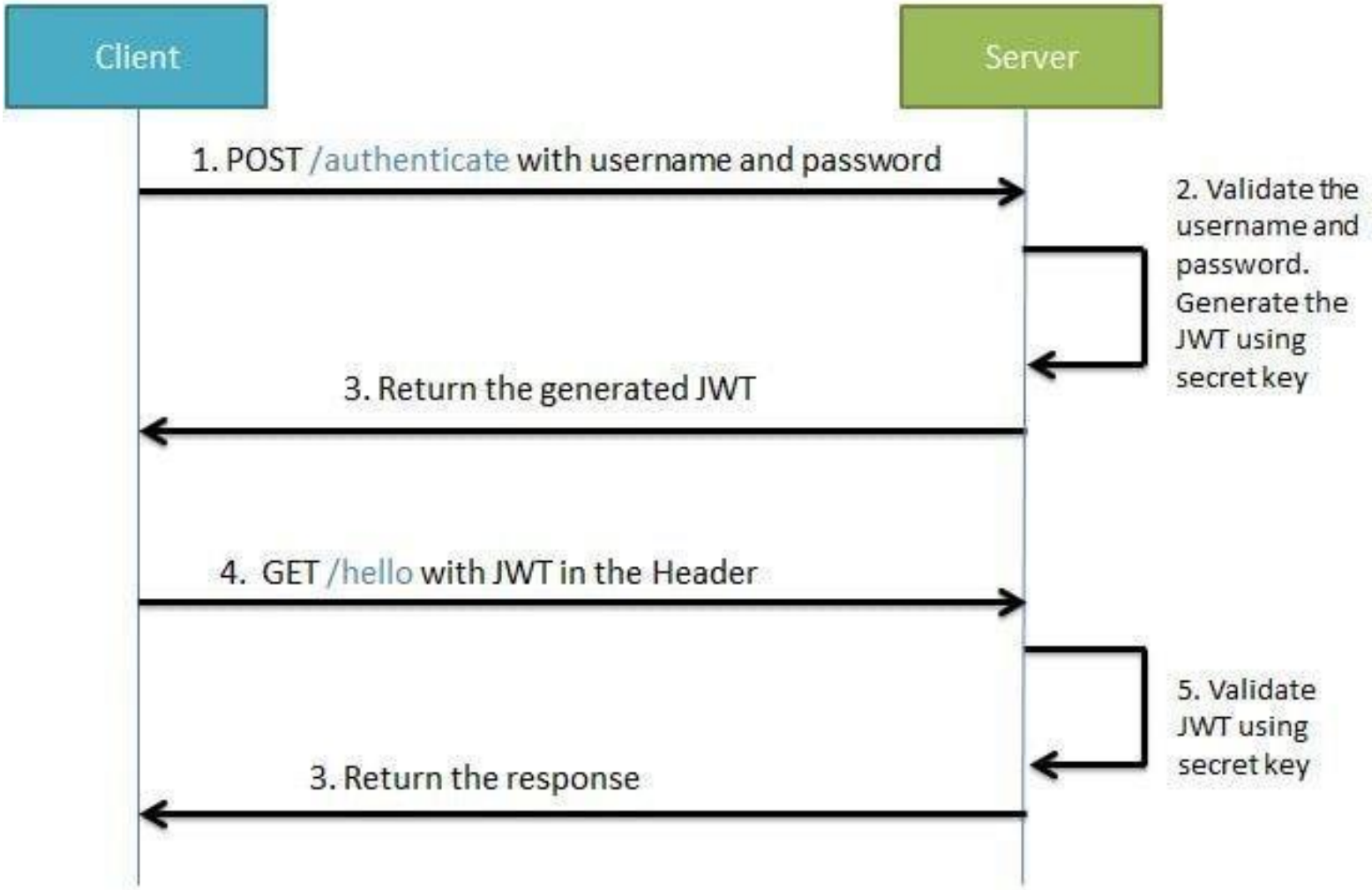
2 –The server application validates the credentials and, if successful, stores an authorization token that is associated with the client's source IP/login. The token can be valid for one operation or for a period of time (e.g. 24 hours).

3 –The server sends a success message to the client and sends the token.

4 –The client calls services on the server, always sending the authorization token.

5 –The server validates each request by checking if a valid token was sent. If a token was not sent, has already expired, or is invalid, the server sends a message back to the client requesting authorization.

2- Token Based Authorization



https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/

3- JSON Web Token (JWT)

3- JWT - components

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

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),
  your-256-bit-secret
) ☐ secret base64 encoded
```

https://jwt.io/

Agenda

1. Motivation
2. JWT Flow Architecture
3. Implementation of the JWTFilter and JWT Service
4. Testing the API with Postman
5. Implementation of a Client to consume the API with a Token

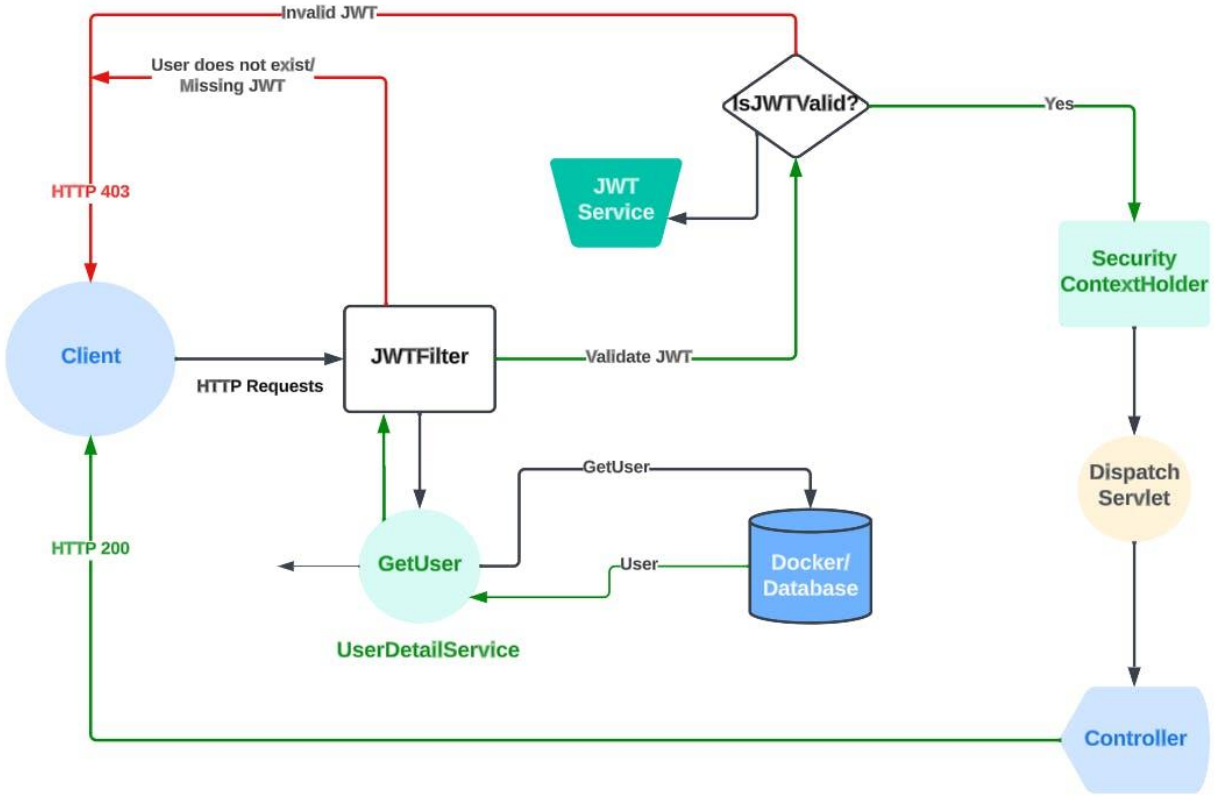
JWT (RFC 7519) mit Spring Boot

```
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt-api</artifactId>
   <version>0.12.5</version>
</dependency>
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt-impl</artifactId>
   <version>0.12.5</version>
</dependency>
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt-jackson</artifactId>
   <version>0.12.5</version>
</dependency>
```

https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/

3. JwtFilter and JwtService

They intercept incoming requests, validates JWT tokens, and authenticates users if a valid token is present:



https://timife.hashnode.dev/exploring-the-inner-workings-of-jwt-in-spring-security-an-in-depth-analysis

3. JWT Filter´s tasks

1. Check the Request Header
2. Check the ToKen content
3. Extract the user information
4. Provide or Reject Access to the resource

3- Spring Boot RestTemplate – getForObject

```
RestTemplate restTemplate = new RestTemplate();
String fooResourceUrl
  = "http://localhost:8080/spring-rest/foos";
Foo foo = restTemplate
  .getForObject(fooResourceUrl + "/1", Foo.class);
```

3- Spring Boot RestTemplate – postForObject()

```
RestTemplate restTemplate = new RestTemplate();

HttpEntity<Foo> request = new HttpEntity<>(new Foo("bar"));

Foo foo = restTemplate.postForObject(fooResourceUrl, request,
Foo.class);
```

3. Spring Boot RestTemplate – main methods

HTTP Method	REST Template Method	Description
POST	postForObject	Post object and expect its instance as a response.
	postForEntity	Post object and expect a response of ResponseEntity type.
	postForLocation	Post object and read location of it.
GET	getForObject	Read the object based on a given URL.
	getForEntity	Read the ResponseEntity instance based on a given URI.
PUT	put	Perform the PUT of a given object against a given URI.
DELETE	delete	Delete objects based on a given URI.
HEAD	headForHeaders	Read headers based on a given URL and return the HttpHeaders object.
OPTIONS	optionsForAllow	Return the value of the Allow header for a given URI.
Any	Exchange	Perform any HTTP method against a server and exchange HttpEntity and ResponseEntity instances.

4- Hands On - Our API

- **[POST] /api/register** → Register a new user
- **[POST] /api/authenticate** → Authenticate a user
- **[GET] /api/demo** → Retrieve the current authenticated user

https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/

Agenda

1. Motivation
2. JWT Flow Architecture
3. Implementation of the JWTFilter and JWT Service
4. Testing the API with Postman
5. Implementation of a Client to consume the API with a Token

Agenda

1. Motivation
2. JWT Flow Architecture
3. Implementation of the JWTFilter and JWT Service
4. Testing the API with Postman
5. Implementation of a Client to consume the API with a Token

References

- https://www.baeldung.com/rest-template
- https://www.youtube.com/watch?v=_XbXkVdoG_0
- https://www.freecodecamp.org/news/how-to-setup-jwt-authorization-and-authentication-in-spring/
- https://www.javainuse.com/webseries/spring-security-jwt/chap6
- https://www.javainuse.com/webseries/spring-security-jwt/chap6#google_vignette
- https://www.unlogged.io/post/spring-security-6---oauth-2-0-social-login-with-springboot-3