Spring security - I

Teaching Faculty: Dr. Muhyieddin Al-Tarawneh

Prepared by Muhyieddin **AL-TARAWNEH**, Umur **INAN**

Authentication

- Verifies you are who you say you are.
- Methods:
  - Login Form
  - HTTP Authentication
  - HTTP Digest
  - Custom Authentication Method

Authorization

- Decides if you have permission to access a resource.
- Methods:
  - Access Control for URLs
  - Secure Objects and Methods
  - Access Control Lists

Single Factor Authentication:

- It requires a password to grant user access to a particular system.
- This is the simplest form.

Two Factor Authentication:

- It requires two-step verification process which not only requires a password but also a piece of information only the user know.
- There are three main types of two-factor authentication:
    - Additional login credentials only the account holder should know. This includes things like security question answers and PIN numbers.
    - Devices the account holder owns that receive additional login credentials. This most commonly takes the form of a security token, mobile phone app, or tablet device app.
    - Biometric login credentials unique to the account owner. This includes retina scans and fingerprints.

Multi-Factor Authentication

- This is the most advanced method which requires two or more levels of security from independent categories to grant user access to the system.

ENCODING

- Encoding is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage.
    - **Base64** Binary-to-text encoding schemas that represent binary data in an ASCII string format by translating it into a radix-64 representation

Encryption

- Process of encoding information.
- Converts the original representation of the information into an alternative form known as ciphertext.
- Ideally, only authorized parties can decipher a ciphertext back to plaintext and access the original information.

Hash Functions

- It is a mathematical algorithm that maps data to a bit array of fixed size. (md5,sha1,sha2, ...)
- This is a one-way function.
- It cannot be inverted.
- The only way to find a message that produces a given hash is to attempt a brute-force search of possible inputs.

JWT

- JSON Web Tokens are an open, industry standard method for representing claims securely between two parties.

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

- JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

JWT use cases -- authorization

- This is the most common scenario for using JWT.
- Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.
- Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

JWT use cases -- Information Exchange

- JSON Web Tokens are a good way of securely transmitting information between parties.
- Because JWTs can be signed—for example, using public/private key pairs—
- You can be sure the senders are who they say they are.
- Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

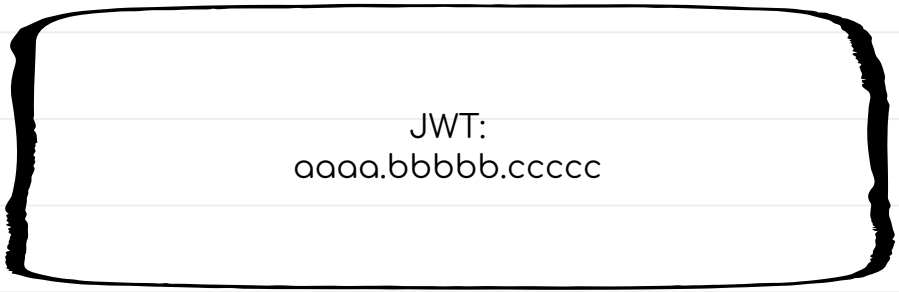
JWT Breakdown

- 3 important parts of Structure.

HEADER

PAYLOAD

VERIFY SIGNATURE



JWT - Header

- The header *typically* consists of two parts.
  - The type of the token, which is JWT.
  - The signing algorithm being used, such as HMAC SHA256 or RSA.

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

Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

JWT - PAYLOAD

- It contains the claims.
- Claims are statements about an entity (typically, the user) and additional data.

```
{

  "sub": "1234567890",
  "name": "John Doe",
  "admin": true

}
```

The payload is then **Base64Url** encoded to form the second part of the JSON Web Token.

JWT - SIGNATURE

- To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

- The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

```
HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```

Advantages of JWTs

- No session to manage (stateless).
- Portable: A single token can be used with multiple backends.
- No cookies required, so it is very mobile friendly.
- Good performance.
- Decoupled / Decentralized.

Spring Security

- Spring Security is a framework that focuses on providing both **authentication** and **authorization** (or "access-control") to Java applications.

- It is built on top of Spring Framework.

Spring Security Fundamentals

- Authentication
  - Confirming truth of credentials
  - Who are you?
- Authorization
  - Define access policy for principal
  - What can you do?
- Principal
  - User that performs the action
  - Currently logged in User

- Roles
  - Coarse-grained permission.
- GrantedAuthority
  - Reflects the application-wide permissions granted to a principal.

Spring Security Fundamentals

- SecurityContextHolder
  - Helper class to provide access to the SecurityContext.

- SecurityContext
  - Holds the Authentication Object [current authenticated user].

Spring Security Fundamentals

- Authentication Object
  - Object is created upon authentication, which holds the login credentials.

- UserDetails
  - Provides the necessary information to build an Authentication object from your application's “custom” DAOs or other source of security data.

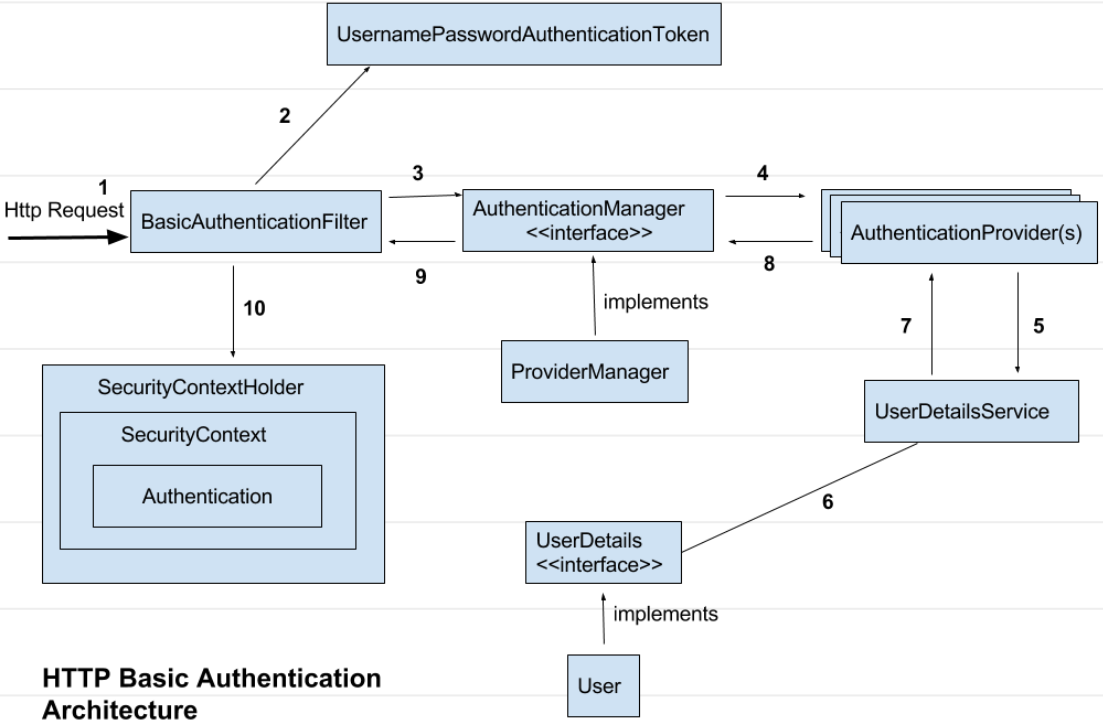
- UserDetailsService
  - loads & creates a “custom” UserDetails when passed in a String-based username.

Spring Security Fundamentals

- AuthenticationManager
    - is like a coordinator where you can register multiple providers, and based on the request type, it will deliver an authentication request to the correct provider.

- AuthenticationProvider
    - Interface that maps to a data store which stores your user data.

Spring Security architecture



HTTP Basic Authentication Architecture

Chathuranga Tennakoon
www.springbootdev.com

Authentication Provider
InMemory authentication

```
@EnableWebSecurity
public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {


    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
                .withUser("user")
                .password("user")
                .roles("USER")
                .and()
                .withUser("admin")
                .password("admin")
                .roles("ADMIN");
    }
}
```

Authentication Provider
JDBC authentication

```
public class SpringSecurityConfiguration extends
WebSecurityConfigurerAdapter {

    @Autowired
    DataSource dataSource;

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
throws Exception {

        auth.jdbcAuthentication()
                .dataSource(dataSource)
                .usersByUsernameQuery("select username,
password, enabled from users where username = ?")
                .authoritiesByUsernameQuery("select username,
authority from authorities where username = ?");

    }
}
```

Authentication Provider
JPA authentication

```
@EnableWebSecurity
public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {


    @Qualifier("JPAUserDetailService")
    @Autowired
    UserDetailsService userDetailService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailService);
    }
}
```

```
@Service
public class JPAUserDetailService implements UserDetailsService {

    @Autowired
    UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Optional<User> user = userRepository.findByUsername(username);
        user.orElseThrow(() -> new UsernameNotFoundException("Not FOUND..."));
        return new JPAUserDetails(user.get());
    }
}
```

Authentication Provider
JPA authentication

```
public class JPAUserDetails implements UserDetails {

    private String username;
    private String password;
    private boolean isActive;
    private Set<Role> roles;

    public JPAUserDetails(User user) {
        username = user.getUsername();
        password = user.getPassword();
        isActive = user.getActive() == 1 ? true : false;
        roles = user.getRoles();
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return roles.stream().map(role -> new SimpleGrantedAuthority(role.getRole()))
                .collect(Collectors.toList());

    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isEnabled() {
        return isActive;
    }
...
}
```

PasswordEncoder

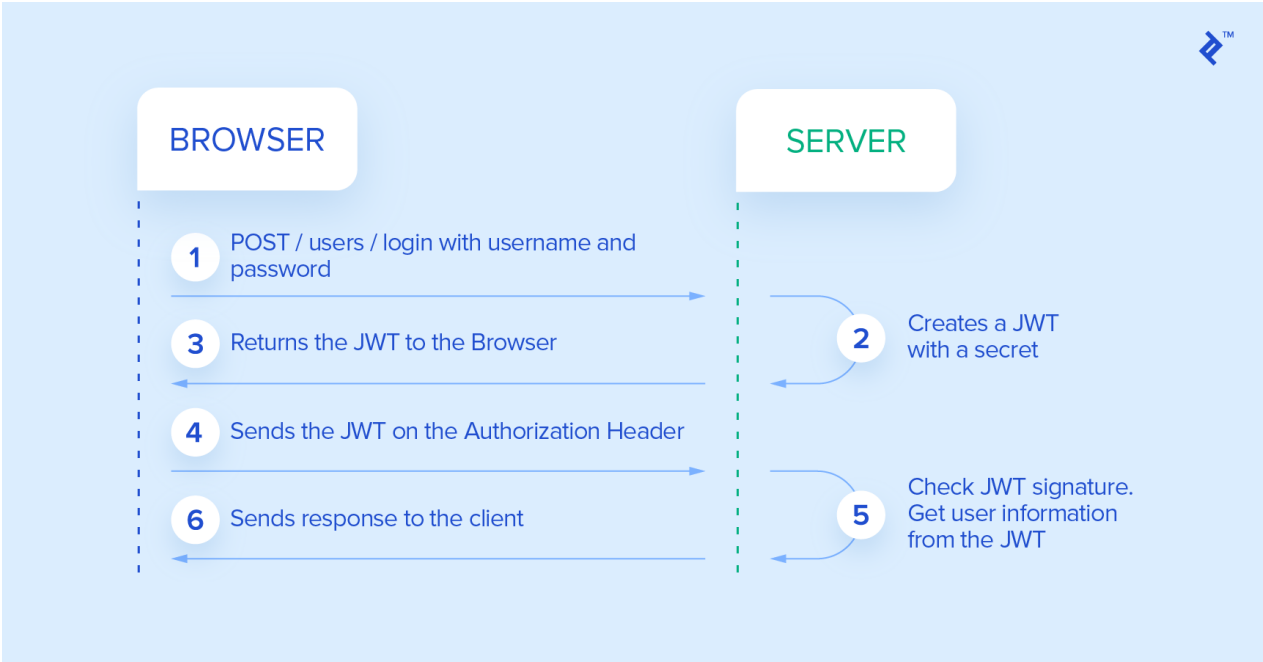
```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Main points

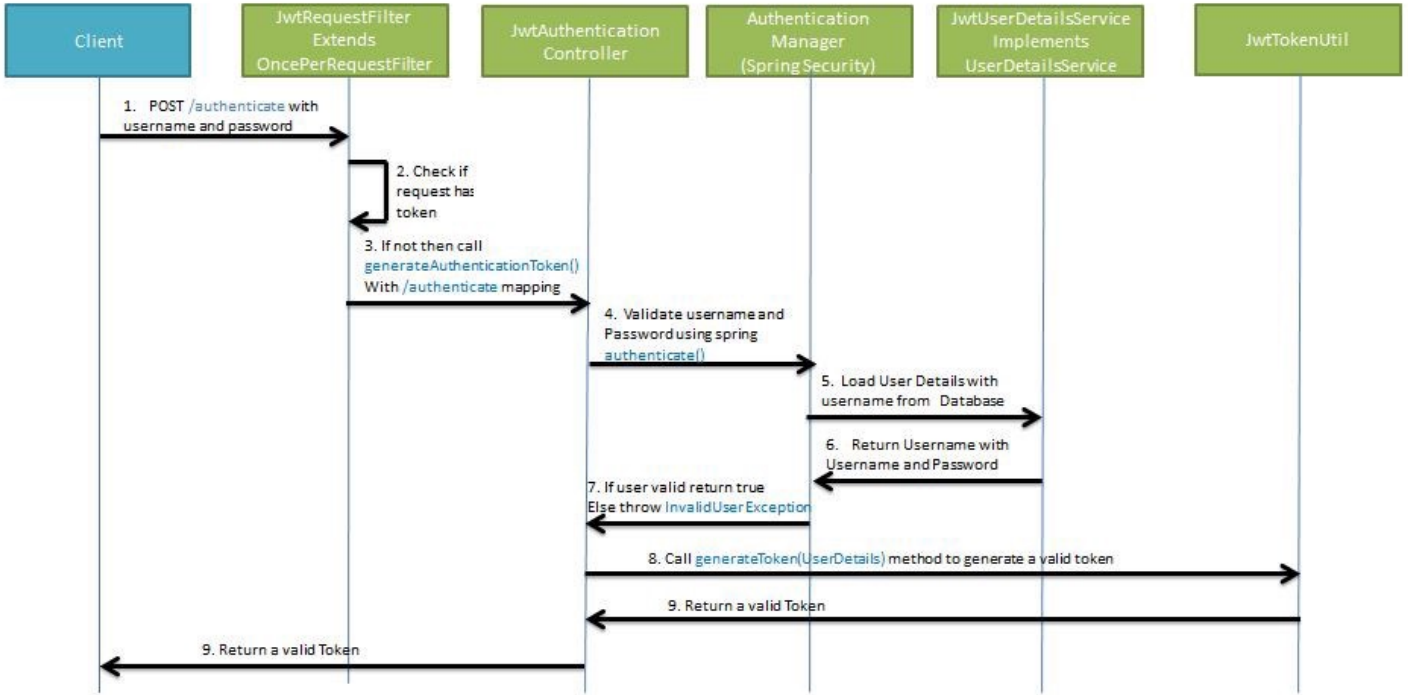
- Spring Security allows for authentication and authorization of system users. It gives access to resources “appropriately”. It works as a stabilizing factor in the enterprise infrastructure

- Securing life at its basis - at the underlying field of Creative Intelligence – guarantees stability and success at all levels.

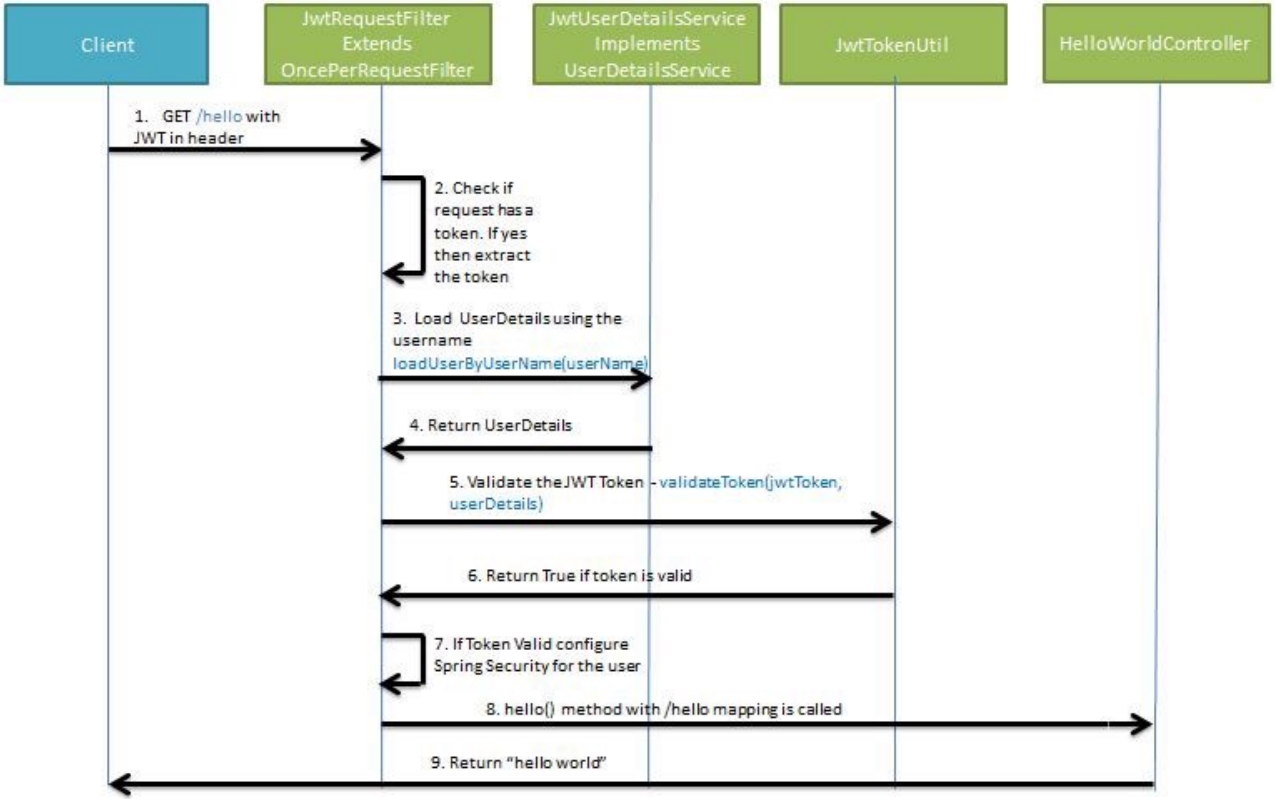
Implementing JWT


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
Check JWT signature. Get user information from the JWT
6
Sends response to the client

Sequence flow of generating a JWT



Validating JWT



JwtTokenUtil

- The JwtTokenUtil is responsible for performing JWT operations like creation and validation.
- It makes use of the io.jsonwebtoken.Jwts for achieving this.

- See uitl.JwtUtil class

Make an authenticate API endpoint

- Accepts user ID and password
- Returns JWT as a response

- Permit all calls to the endpoint /authenticate

- See controller.AuthController

Intercept all incoming requests

- Extract JWT from header
- Validate and set execution context

- This will request will intercept every request just once then examine the header

- See filter.JwtFilter