# JWT: Its commonly used for managing the Authorization, it uses between two parties to communicate securely its specification called RFC7519 which outline how JWT should be structured

# For Authorization we have multiple mechanism like Session Token, JSON web token ETC in all one thing is common they follow HTTP (STATELESS) protocol

# STATELESS means we have to send all data in a request for required to access that page like authentication details, so in order to overcome that problem now a day we are using some Authorization strategies like Session Token, JSON web token

# Session Token Example

# Now Suppose Airtel Customer have some query So he will call Customer support, Representative listen customer query and make a ticket number and stored in a server and give that same ticket number to that customer so next time call he no need to explain entire scenario again he just gives that ticket number to that available Representative and he will come to know

# So technically first user authenticate using Username and password server will authenticate and make a Session Id then stored in a server and provide that same Session ID in response and from next request Client send Session ID as a part of every request in a header. Server will verify that Session ID with Stored Session ID in a server and provide response

# Problem with Session Token

# It will work fine if we have only server for our App, So that server storing User details and providing us a Session ID so next time no need to authorize

# BUT what if we have multiple servers for single App and we are using LOAD BALANCER in between to communicate with that server

# So, for Authorization Load Balancer will call first Server and that server will Authenticate user and Stored Session ID in server and provide that same Session Id in response for next request we will send that Session ID in header but this time Load Balancer will give request to second server and that server will not have any info about our Session ID that info is stored in server one

# We can use Shared Session Cache (REDIS) like all server storing info in a shared cache memory, but there is problem when this REDIS server goes down all server session will be LOST

# Some follow pattern like they configured Load Balancer So they will send request to that server where it was gone first but this can cause a problem with Micro Service

# JSON Web Token

# So Instead of Storing Authorization details in server, server will send that all info in response along with JSON Signature in the form of Token

# JSON Web Token

POM.xml

<dependency>

<groupId>com.thetransactioncompany</groupId>

<artifactId>cors-filter</artifactId>

<version>1.3.2</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.8</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-core</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.0</version>

</dependency>

Web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE xml>

<web-app xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee*

*http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*

version=*"3.1"*>

<display-name>Archetype Created Web Application</display-name>

<servlet>

<servlet-name>One</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>One</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<filter>

<filter-name>CORS</filter-name>

<filter-class>com.thetransactioncompany.cors.CORSFilter</filter-class>

<init-param>

<param-name>cors.allowOrigin</param-name>

<param-value>\*</param-value>

</init-param>

<init-param>

<param-name>cors.supportsCredentials</param-name>

<param-value>false</param-value>

</init-param>

<init-param>

<param-name>cors.supportedHeaders</param-name>

<param-value>Accept, Origin, X-Requested-With,

Content-Type,Last-Modified,Authorization,responseType</param-value>

</init-param>

<init-param>

<param-name>cors.supportedMethods</param-name>

<param-value>GET, POST, HEAD, OPTIONS</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CORS</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

</web-app>

One-servlet.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc.xsd"*>

<mvc:annotation-driven></mvc:annotation-driven>

<context:annotation-config></context:annotation-config>

<context:component-scan base-package=*"p1"*></context:component-scan>

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/Output/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

</beans>

Employee.java

**public** **class** Employee {

**int** eid;

String eName;

**int** roll;

**public** Employee(**int** eid, String eName, **int** roll) {

**this**.eid = eid;

**this**.eName = eName;

**this**.roll = roll; // **GETTERs,SETTERs,toString() and Empty Constructor**

EmployeeController.java

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.authentication.BadCredentialsException;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.userdetails.UserDetails;

@RestController

**public** **class** EmployeeController{

@Autowired

AuthenticationManager authenticationManager;

@Autowired

MyUserDetailService myUserDetailService;

@Autowired

JwtUtil jwtUtil;

@RequestMapping(value = "hello")

**public** String hello() {

**return** "Hello World...";

}

@RequestMapping(value = "/authenticate",method = RequestMethod.***POST***)

**public** ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest jwtRequest) **throws** Exception {

**try** {

System.***out***.println(jwtRequest);

**this**.authenticationManager.authenticate(

**new** UsernamePasswordAuthenticationToken(jwtRequest.getUsername(), jwtRequest.getPassword()));

}**catch**(BadCredentialsException e) {

**throw** **new** Exception("Incorrect Username and Password",e);

}

UserDetails userDetails = **this**.myUserDetailService.loadUserByUsername(jwtRequest.getUsername());

String jwt = **this**.jwtUtil.generateToken(userDetails);

**return** ResponseEntity.*ok*(**new** JwtResponse(jwt));

}

}

SecurityConfigurer.java

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

**import** org.springframework.security.config.http.SessionCreationPolicy;

**import** org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

**import** org.springframework.security.crypto.password.PasswordEncoder;

**import** org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfigurer **extends** WebSecurityConfigurerAdapter{

@Autowired

MyUserDetailService myUserDetailService;

@Autowired

JwtRequestFilter jwtRequestFilter;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(myUserDetailService);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.csrf().disable().authorizeRequests().antMatchers("/authenticate").permitAll()

.anyRequest().authenticated()

.and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.***STATELESS***);

http.addFilterBefore(**this**.jwtRequestFilter, UsernamePasswordAuthenticationFilter.**class**);

}

@Override

@Bean

**public** AuthenticationManager authenticationManagerBean() **throws** Exception {

**return** **super**.authenticationManagerBean();

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

}

MyUserDetailService.java

@Service

**public** **class** MyUserDetailService **implements** UserDetailsService{

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

**return** **new** User(username,"$2a$10$sCrpbptlhsO9GLmLF4.c2Ont0vhl89uwXUtBIIhU.A9c4o1wUqcFK",**new** ArrayList<>());

}

}

JwtUtil.java

**import** java.util.function.Function;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.stereotype.Service;

**import** io.jsonwebtoken.Claims;

**import** io.jsonwebtoken.Jwts;

**import** io.jsonwebtoken.SignatureAlgorithm;

@Service

**public** **class** JwtUtil {

**private** String SECRET\_KEY = "secret";

**public** String extractUsername(String token) {

**return** extractClaim(token, Claims::getSubject);

}

//retrieve expiration date from jwt token

**public** Date extractExpiration(String token) {

**return** extractClaim(token, Claims::getExpiration);

}

**public** <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

**final** Claims claims = extractAllClaim(token);

**return** claimsResolver.apply(claims);

}

**private** Claims extractAllClaim(String token) {

**return** Jwts.*parser*().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

**private** Boolean isTokenExpired(String token) {

**final** Date expiration = extractExpiration(token);

**return** expiration.before(**new** Date());

}

**public** String generateToken(UserDetails userDetails) {

Map<String, Object> claims = **new** HashMap<>();

**return** createToken(claims, userDetails.getUsername());

}

**private** String createToken(Map<String, Object> claims, String subject) {

**return** Jwts.*builder*().setClaims(claims).setSubject(subject).setIssuedAt(**new** Date(System.*currentTimeMillis*()))

.setExpiration(**new** Date(System.*currentTimeMillis*() + 1000 \* 60 \* 60 \* 10))

.signWith(SignatureAlgorithm.***HS256***, SECRET\_KEY).compact();

}

**public** Boolean validateToken(String token, UserDetails userDetails) {

**final** String username = extractUsername(token);

**return** (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

JwtRequest.java

**public** **class** JwtRequest {

**private** String username;

**private** String password;

**public** JwtRequest() {

}

**public** JwtRequest(String username, String password) {

**this**.username = username;

**this**.password = password;

} **// Getters and Setters**

@Override

**public** String toString() {

**return** "JwtRequest [username=" + username + ", password=" + password + "]";

JwtResponse.java

**package** p1.SpringSecurityDemo;

**public** **class** JwtResponse {

**private** String jwt;

**public** JwtResponse() {

}

**public** JwtResponse(String jwt) {

**this**.jwt = jwt;

} **// Getters and Setters**

@Override

**public** String toString() {

**return** "JwtResponse [jwt=" + jwt + "]";

JwtRequestFilter.java

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

**import** org.springframework.stereotype.Component;

**import** org.springframework.web.filter.OncePerRequestFilter;

@Component

**public** **class** JwtRequestFilter **extends** OncePerRequestFilter{

@Autowired

MyUserDetailService myUserDetailService;

@Autowired

JwtUtil jwtUtil;

@Override

**protected** **void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

**throws** ServletException, IOException {

String authorizationHeader = request.getHeader("Authorization");

String userName = **null**;

String jwt = **null**;

**if**(authorizationHeader != **null** && authorizationHeader.startsWith("Bearer ")) {

jwt = authorizationHeader.substring(7);

userName = **this**.jwtUtil.extractUsername(jwt);

}

**if**(userName != **null** && SecurityContextHolder.*getContext*().getAuthentication() == **null**) {

UserDetails userDetails = **this**.myUserDetailService.loadUserByUsername(userName);

**if**(**this**.jwtUtil.validateToken(jwt, userDetails)) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken =

**new** UsernamePasswordAuthenticationToken(userDetails,**null**,userDetails.getAuthorities());

usernamePasswordAuthenticationToken.setDetails(**new** WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.*getContext*().setAuthentication(usernamePasswordAuthenticationToken);

}

}

filterChain.doFilter(request, response);

}

}

**Calling URL using Postman**

1. Hit with Postman by Passing URL

Type :: **POST** :: <http://localhost:8080/SpringRestPro/authenticate>

Body :: {

"username" : **"Sam"**,

"password" : **"123"**

}

1. Will get token

{"jwt": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJTYW0iLCJleHAiOjE1NzM1ODA2MjcsImlhdCI6MTU3MzU0NDYyN30.uZdJ6-CRoafSi3UByc86bWgLY0\_H0VIDYWnzCgyNnt8"}

1. Hit with Postman by Passing URL which required Auth

Type :: **POST** :: <http://localhost:8080/SpringRestPro/hello>

Body :: {

"username" : **"Sam"**,

"password" : **"123"**

}

Postman Headers :: Authorization :: Bearer ***\_SPACE\_***eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJTYW0iLCJleHAiOjE1NzM1NzE4MDgsImlhdCI6MTU3MzUzNTgwOH0.oYeQ8yfMvaKgfHLlH2ks1KR\_UN9SOi7JKDJKyKIuzf4

**Structure of JWT**

**eyJhbGciOiJIUzI1NiJ9**.**eyJzdWIiOiJTYW0iLCJleHAiOjE1NzM1NzE4MDgsImlhdCI6MTU3MzUzNTgwOH0**.**oYeQ8yfMvaKgfHLlH2ks1KR\_UN9SOi7JKDJKyKIuzf4** Separated by **Dot**

These JWT is divided into **three-part** **Header**, **Payload**, **Signature**

**Header:** Contain info like type of Algorithm for creating token and token type

{

"alg": **"HS256"**,

"typ": **"JWT"**

}

**Payload:** Contains Info which we do need to exchange from server like authorization details

{

"sub": **"1234567890"**, **//** Unique Id Created by JWT

"name": **"John** **Doe"**, **//** User Name

"iat": **1516239022 //** Issued At

}

**Signature:** Its use to validate Header and Payload which we pass in request is correct or not, with the help of secret key which we have configured in **JwtUtils class** above

**HMACSHA256**(

base64UrlEncode(**header**) + **"."** +

base64UrlEncode(**payload**),

**secretKey**

}

So, if someone temper with name in Payload the whole token needs to be change

**JWT is NOT USE FOR AUTHENTICATION**

We need to provide Authentication details like username and password and after verify it will give those details in TOKEN format, later we can use **Local Storage** or **Cookie** to send that token in subsequent request

**What if someone steals my JWT and uses it themselves?**

**Technically it will work**, remember JWT is not use for Authentication, to solve this we can use **HTTPs** connection and some other Authorization technique with **JWT**, some common which getting used now a day is **OAuth** to solve that issue

**Can we disable JWT?**

Like someone stolen my session ID and accessing my data so I can simply logout so then session Id will be expired, But JWT does not maintain any state in that token we have all info which we can use to access some URL (We can set expiration date also in JWT), but without setting expiration date someone can access that APP URL after Logoff

# Calling URL using *Angular*

# User.ts

export class User{

    username: string;

    password: string;

}

# MyService.ts

  authenticateUser(user : User){

        return this.httpClient.post(MyService.ROOT\_URL+"authenticate",user);

    }

   hello(user : User){

        console.log(">>> " + 'Bearer ' + localStorage.getItem('token'));

        return this.httpClient.post(MyService.ROOT\_URL+"hello",user,{

            headers : new HttpHeaders({

                'responseType' : 'text',

                'Authorization' : 'Bearer ' + localStorage.getItem('token')

            })

        });   }

One.component.html

<button (click) = "authenticateUser()">Authenticate-User</button><br><br>

<button (click) = "hello()">Hello</button>

One.component.ts

authenticateUser(){

        console.log('authenticateUser');

        let user = new User();

        user.username = 'Sam';

        user.password = '123';

        this.myServ.authenticateUser(user).subscribe((response) => {

            let token = JSON.parse(JSON.stringify(response));

            console.log(token.jwt);

            localStorage.setItem('token',token.jwt);

        });

    }

hello(){

        console.log('hello');

        let user = new User();

        user.username = 'Sam';

        user.password = '123';

        this.myServ.hello(user).subscribe((response) => {

            console.log(response);

            console.log(JSON.parse(JSON.stringify(response)));

        });

# Rest Template *Start Here*

**RestTemplate:** It is use to consume REST API Spring RestTemplate.class is part of spring-web, introduced in Spring 3. RestTemplate class provides overloaded methods for different HTTP methods, such as **GET, POST, PUT, DELETE etc.**

**Example**

**Need to create two projects**

1) Spring Web Maven

2) Spring QuickStart

**1) Spring Web Maven**

**Employee.java**

Stringeid**;**

String firstName;

StringlastName**; Getters and Setters**

**EmployeeController.java**

@RestController

**public** **class** EmployeeController {

@RequestMapping(value = "getAllEmployee",method = RequestMethod.***GET***)

**public** List<Employee> getAllEmployee() {

@RequestMapping(value = "getAllEmployee/{id}",method = RequestMethod.***GET***)

**public** Employee getEmpById(@PathVariable **int** id) {

@RequestMapping(value = "getAllEmployee/addEmployee")

**public** **void** addEmployee(@RequestBody Employee emp) {

@RequestMapping(value = "getAllEmployee/deleteEmployee/{id}")

**public** **void** deleteEmployee(@PathVariable **int** id) {

**Keep the Server Running.....**

**2) Spring QuickStart Maven**

**Pom.xml**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.1.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>5.2.2.RELEASE</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.8</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.9.8</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-annotations</artifactId>

<version>2.9.8</version>

</dependency>

**AppConfig.*java***

@Configuration

@ComponentScan(basePackages = "p1")

**public** **class** AppConfig {

@Bean

**public** RestTemplate restTemplate() {

RestTemplate restTemplate = **new** RestTemplate();

MappingJackson2HttpMessageConverter converter = **new** MappingJackson2HttpMessageConverter();

converter.setObjectMapper(**new** ObjectMapper());

restTemplate.getMessageConverters().add(converter);

**return** restTemplate;

**Employee.java**

Stringeid**;**

String firstName;

*String lastName; Getters and Setters*

**EmployeeService.java**

**@Service**

**public** **class** EmployeeService {

@Autowired

RestTemplate restTemplate;

**final** String ROOT\_URI = "http://localhost:8080/SpringWebPro/getAllEmployee";

**public** List<Employee> getAllEmployee() {

ResponseEntity<Employee[]> response = **this**.restTemplate.getForEntity(ROOT\_URI, Employee[].**class**);

**return** Arrays.*asList*(response.getBody());

}

**public** Employee getById(**int** id) {

ResponseEntity<Employee> response = restTemplate.getForEntity(ROOT\_URI + "/"+id, Employee.**class**);

**return** response.getBody();

}

**public** HttpStatus addEmployee(Employee Employee) {

ResponseEntity<Employee[]> response = restTemplate.postForEntity(ROOT\_URI+ "/addEmployee", Employee, Employee[].**class**);

**return** Arrays.*asList*(response.getBody());

}

**public** **void** updateEmployee(Employee Employee) {

restTemplate.put(ROOT\_URI, Employee);

}

**public** **void** deleteEmployee(**int** id) {

restTemplate.delete(ROOT\_URI +"/deleteEmployee/"+ id);

**public** Employee **reqParam**() { ***//Send Request Param***

**int** eid = 1;

String eName = "Shahnawaz";

**int** sal = 100;

URI targetUrl= UriComponentsBuilder.*fromUriString*(***ROOT\_URL***) // Build the base link

.path("/reqParam") // Add path

.queryParam("eid", eid)

.queryParam("eName", eName)

.queryParam("sal", sal) // Add one or more query params

.build() // Build the URL

.encode() // Encode any URI items that need to be encoded

.toUri();

System.***out***.println("targetUrl " + targetUrl);

ResponseEntity<Employee> responseEntity = **this**.restTemplate.getForEntity(targetUrl, Employee.**class**);

**return** responseEntity.getBody();

**RestTemplate getForEntity() method does not support request headers. Please use exchange() method if headers are necessary.**

**public** Employee **reqHeader**() {

HttpHeaders headers = **new** HttpHeaders();

headers.set("data1", "data1");

headers.set("data2", "data2");

HttpEntity<Employee> requestEntity = **new** HttpEntity<>(**null**, headers);

ResponseEntity<Employee> responseEntity =

**this**.restTemplate.exchange(***ROOT\_URL***+"reqHeader", HttpMethod.***GET***,requestEntity,Employee.**class**);

**return** responseEntity.getBody();

**Client.java**

AnnotationConfigApplicationContext app = **new** AnnotationConfigApplicationContext(AppConfig.**class**);

EmployeeService es = app.getBean(EmployeeService.**class**);

List<Employee> li = es.getAllEmployee();

System.***out***.println(li);

Employee e = es.getById(0);