Unit Test Document

|  |  |
| --- | --- |
| Ticket | #33 |
| Description | Analysis of test cases for authentication |

The Authentication Strategy is based on generated JWT Tokens with specific claims (also referred to as properties of the token). One of these claims is the authorities list the token holds. These authorities (List of SimpleGrantedAuthority) are lists of permissions the specified token can grant access to.

|  |
| --- |
| Suppose a JWT Token is as follows: |
| Encoded | Decoded |
| eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2OTc3NDYzMjgsImlhdCI6MTY5NzcxMDMyOCwiYXV0aG9yaXRpZXMiOlt7ImF1dGhvcml0eSI6IlZlaGljbGVzIn0seyJhdXRob3JpdHkiOiJMb2NhdGlvbnMifV19.M2LN8ieO1TuyD6mLLBW567n41T8xH1\_fc4tcfgWI\_LA | { "exp": 1697746328, "iat": 1697710328, "authorities": [ { "authority": "Vehicles" }, { "authority": "Locations" } ]} |

The token above has the authorities Vehicles and Locations. These will be compared against the endpoint being considered in the doFilterInternal() method in the JWTAuthorisation class.

This filter is executed when a request is made to the API and can control the issued response considering authentication via the JWT Token header. The code below illustrates how the authorities are extracted from the token claims via the appropriate methods. The String[] cases represents a collection of endpoints available in the API. The default path for testing purposes is “/v1/vehicles” (a constant) and it is joined by the endpoint call (e.g. “/all”). The token must be supplied in the “Authorization” header field of the HTTPRequest.

Below are a list of assumptions for testing (which will be reported on in the testing matrix below). These assumptions provide a general outline of what the expected response should be for each of the test cases, considering if a token has been provided or if that token can access the endpoint on request.

Therefore, it should follow that:

* If the token is not present in the HTTPRequest Header, the response should fail with a 418 Status (“I’m a Teapot”).
* If the token is tampered with, when the claims are parsed via the secret, it should result in an exception that does not crash the application but results in an internal server error due to the JWT token being classified as invalid.
* If the user does not hold the necessary Authorities to access an endpoint but provides a valid token the response will be 401 Unauthorized.
* If the user provides a token and the token has the necessary authorities to access a specific endpoint, the response should be the corresponding JSON response for the request made to the endpoint

if (jwt == null) {
 response.setStatus(HttpStatus.*I\_AM\_A\_TEAPOT*.value());
 response.sendError(HttpStatus.*I\_AM\_A\_TEAPOT*.value(), "Not Authenticated");
 return;
}else {
 try {
 Claims claims = Jwts.*parser*().setSigningKey("secret").parseClaimsJws(jwt).getBody();
 List<GrantedAuthority> authorities = extractPermissions(claims);
 Authentication auth = new UsernamePasswordAuthenticationToken(null, null,authorities);
 String path = request.getServletPath();
 String[] cases = {defaultpath+"all", defaultpath+"by\_id/"};
 int i;
 for(i = 0; i < cases.length; i++)
 if(path.contains(cases[i])) break;
 switch(i) {
 case 0:
 checkAuthorities("Vehicles", response, auth);
 break;
 case 1:
 checkAuthorities("Admin", response, auth);
 break;
 default:
 auth.setAuthenticated(false);
 break;
 }
 if(!auth.isAuthenticated()){
 response.setStatus(HttpStatus.*UNAUTHORIZED*.value());
 response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, "Permissions invalid");
 return;
 }

 SecurityContextHolder.*getContext*().setAuthentication(auth);
 } catch (JwtException e) {
 throw new IllegalStateException("Invalid JWT Token - The Token is either manually modified or has expired. \n" + e.getMessage() + "\n" + Arrays.*toString*(e.getStackTrace()));
 }
}
filterChain.doFilter(request, response);

**Testing Matrix**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Token Encoded** | **Endpoint Requested** | **Permissions/Authorities in Token** | **Result Response** | **Outcome as expected?** |
| No Token Provided | /v1/vehicles/all(“Vehicles” Authority Needed) | --- | “Im a Teapot Error”Error code 418 | YES |
| eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2OTc3NDYzMjgsImlhdCI6MTY5NzcxMDMyOCwiYXV0aG9yaXRpZXMiOlt7ImF1dGhvcml0eSI6IlZlaGljbGVzIn0seyJhdXRob3JpdHkiOiJMb2NhdGlvbnMifV19.M2LN8ieO1TuyD6mLLBW567n41T8xH1\_fc4tcfgWI\_LA | /v1/vehicles/all(“Vehicles” Authority Needed) | VehiclesLocations | JSON Response received with vehicle information. | YES |
| eyJhbGciOiJIUzI1NiJ9. JleHAiOjE2OTc3NDYzMjgsImlhdCI6MTY5NzcxMDMyOCwiYXV0aG9yaXRpZXMiOlt7ImF1dGhvcml0eSI6IlZlaGljbGVzIn0seyJhdXRob3JpdHkiOiJMb2NhdGlvbnMifV19.M2LN8ieO1TuyD6mLLBW567n41T8xH1\_fc4tcfgWI\_LA(same as above, but with 2 letters missing – tampered token) | /v1/vehicles/all(“Vehicles” Authority Needed) | VehiclesLocations(CORRUPTED TOKEN) | “Internal Server Error”Error code 500Error also printed as JWTException in the server console | YES |
| eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2OTc3NTkwODUsImlhdCI6MTY5NzcyMzA4NSwiYXV0aG9yaXRpZXMiOlt7ImF1dGhvcml0eSI6IkxvY2F0aW9ucyJ9XX0.H-7OkACnhX9pPpnLb7bodXJdPL0LUxN1baz1SEHKLtw | /v1/vehicles/all(“Vehicles” Authority Needed) | Locations | “Unauthorized”Error code 401No JSON Response given due to lack of authorities | YES |