Module 2 Day 16

Web API and Authentication

Module 2 Day 16
Can you … ?

- Define and differentiate between the terms "authentication" and "authorization" as they pertain to a client-server or Web application
- Describe the general mechanics and workflow of how JSON Web Tokens (JWTs) are used to authenticate users of a client-server (including Web) application
- Use a common tool to decode an encoded JWT to inspect its contents (JWT Decoder)
- Recognize and interpret the HTTP response status codes commonly associated with authentication and authorization failures, like 401 and 403
- Write client code in Java that can authenticate with an authentication server to retrieve a JWT, and then use the JWT to authenticate subsequent requests to a Web API.
- Utilize the auth features of an application framework (Spring Boot/ASP.NET Web API) to:
    - Specify that a particular resource requires authentication to be accessed
    - Specify that a particular resource can be accessed anonymously
    - Apply simple authorization rules for resources
    - Obtain the identity of an authenticated user

Authentication vs. Authorization

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

Authorization is the process of giving the user permission to access a specific resource or function.

HTTP Status Codes: Authentication & Authorization

The HTTP Status Codes associate with authentication and authorization fall into the 4xx (client-related) group of statuses.

- **401 Unauthorized** indicates that the user has not been authenticated

- **403 Forbidden** indicates that the user is authenticated, but is not allowed to access the resource specified in the HTTP request.

Java Web Token (JWT)

- Compact size allows for quick transfer with requests.
- Often used as authorization mechanism, storing user info such as permissions or roles in payload. These are called **claims**.
- Can contain any data that can be represented in JSON.
- JWT actually contains JSON, but it's encoded.

Server Endpoint Authorization

Setting Server Authorization Rules

- **Authorization** is the process of giving a user permission to access a specific resource or function.
- Can define rule for each resource including
  - Allow anonymous access
  - Require authentication
  - Grant access based on user role

Authorization by Class & Anonymous Access

- Authorization rules can be specified for the entire class by adding the `@PreAuthorize` annotation on the class itself.

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")`

Role-Based Authorization

- It's not uncommon to have certain resources or functions only be accessible to certain people or roles.
- We can authorize access by the user's role (i.e. admin user can access but non-admin cannot)

Getting Info About the Current User

There are times where you'll need access to the current logged in user. You have secured the **`delete()`** method to users with the role **`ADMIN`**, but what if you wanted to keep an audit log of which user deleted each reservation?

Spring gives you access to information about the current user if you add an argument of type **`Principal`** to your method.

When you annotate a method with **`@RequestMapping`**, that method has a flexible signature, and you can choose from a range of supported controller method arguments.

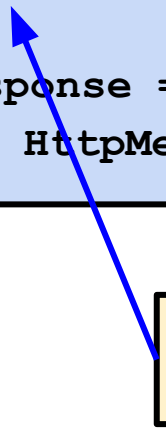
https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-ann-methods.

Client Authorization

The exchange Method

```
LoginDTO loginDTO = new LoginDTO(credentials);
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
HttpEntity<LoginDTO> entity = new HttpEntity<>(loginDTO, headers);
ResponseEntity<Map> response = null;

response = restTemplate.exchange(BASE_URL + "/login",
    HttpMethod.POST, entity, Map.class);
```



`exchange` returns a **`ResponseEntity`** of a specified type, rather than an object of the actual type.

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`**

Let's Add an AUTH_TOKEN to addAReservation...