

# SECURING APIs

# 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)

# COMMON @PREAUTHORIZE USAGE

- `@PreAuthorize("isAuthenticated()")`: The user must be authenticated.
- `@PreAuthorize("permitAll")`: The user doesn't have to be authenticated.
- `@PreAuthorize("hasRole('ADMIN')")`: The user must be authenticated and have the role **ADMIN**.
- `@PreAuthorize("hasAnyRole('ADMIN', 'USER')")`: The user must be authenticated and have either the **ADMIN** or **USER** role.

# 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.

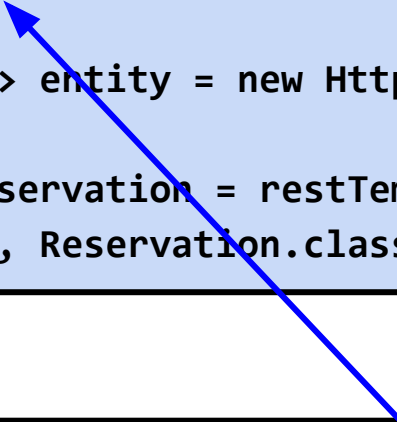
When you annotate a method with **@RequestMapping**, that method has a flexible signature, and you can choose from a range of supported controller method arguments.

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

# CLIENT AUTHORIZATION

# PASSING A BEARER TOKEN USING RESTTEMPLATE

```
HttpHeaders headers = new HttpHeaders();  
headers.setContentType(MediaType.APPLICATION_JSON);  
headers.setBearerAuth(authToken);  
  
HttpEntity<Reservation> entity = new HttpEntity<>(newReservation, headers);  
  
Reservation returnedReservation = restTemplate.postForObject(API_BASE_URL +  
"reservations", entity, Reservation.class);
```



All we have to do to add authentication with a token is set the **Authorization** property of the header to bearer authentication using the token String.

# THE EXCHANGE METHOD

Just as we looked at convenience methods like `@GetMapping` vs. the more generic `@RequestMapping`, the methods we have been using so far with `RestTemplate` have been specific versions of the more generic call `exchange`. The `exchange` method requires you to specify the `HttpMethod` you are using and returns a `ResponseEntity` object.

```
ResponseEntity<Reservation> response = restTemplate.exchange(API_BASE_URL +  
"reservations", HttpMethod.POST, entity, Reservation.class);
```

Class type of `ResponseEntity`

`HttpMethod`

`HttpEntity`

Class type to return

URL

# THE EXCHANGE METHOD

```
try {  
    ResponseEntity<Reservation> response = restTemplate.exchange(API_BASE_URL  
+ "reservations", HttpMethod.POST, entity, Reservation.class);  
    return response.getBody();  
} catch (RestClientResponseException | ResourceAccessException e) {  
    BasicLogger.log(e.getMessage());  
}
```

**Class type of ResponseEntity**

The **ResponseEntity** object will contain the specified class type as its **body** property. We can read the object using the **getBody()** getter.

# THE EXCHANGE METHOD

```
Hotels[] hotels restTemplate.getForObject(API_BASE_URL + "hotels", Hotel[].class);
```

In order to authenticate calls using an API, we need to pass headers that contain the authentication info. But `getForObject` doesn't provide us the ability to pass an object or entity since `GET` does not have a body!



# THE EXCHANGE METHOD

```
Hotels[] hotels restTemplate.getForObject(API_BASE_URL + "hotels", Hotel[].class);
```

In order to authenticate calls using an API, we need to pass headers that contain the authentication info. But `getForObject` doesn't provide us the ability to pass an object or entity since `GET` does not have a body!

```
ResponseEntity<Hotel[]> response = restTemplate.exchange(API_BASE_URL + "hotels",  
HttpMethod.GET, entity, Hotel[].class);
```

The `exchange` method allows us to pass an object or entity and ALSO allows us to set the `HttpMethod` - so we can use it to make a `GET` request with an entity attached.

# THE EXCHANGE METHOD

Create the entity using just the headers..

```
HttpHeaders headers = new HttpHeaders();  
headers.setBearerAuth(authToken);  
HttpEntity<Void> entity = new HttpEntity<>(headers);  
  
ResponseEntity<Hotel[]> response = restTemplate  
    .exchange(API_BASE_URL + "hotels", HttpMethod.GET, entity, Hotel[].class);  
  
Hotels [] hotels = response.getBody();
```

Set the bearer auth property of the header to the token.

Since we are not attaching a body, specify `Void` as the `HttpEntity` type.

Pass the entity and specify `HttpMethod GET`.

LET'S ADD AN AUTH\_TOKEN TO  
ADDARESERVATION...