# Authentication tutorial (Java)

In this tutorial, you'll continue working on an application that uses meetup locations as the data model. You'll add the ability to log in to the client application and send the authentication token with any request.

## Step One: Import project into IntelliJ and explore starting code

Before you begin, import both the client and server starter code into IntelliJ. Review both projects.

### Client

If you run the application, you'll notice the new menu option, option 6, for logging in:

```
----Meetup Locations Menu----
1: List Locations
2: Show Location Details
3: Add a Location
4: Update a Location
5: Delete a Location
6: Login
0: Exit
```

Take a moment to examine the code that makes this login feature work. It starts in `App.java` with `handleLogin()`:

```
private void handleLogin() {
    String username = consoleService.promptForString("Username: ");
    String password = consoleService.promptForString("Password: ");
    String token = authenticationService.login(username, password);
    if (token != null) {
        locationService.setAuthToken(token);
    } else {
        consoleService.printErrorMessage();
    }
}
```

If you successfully log in, you receive an authentication token. You take that token and pass it to the location service which uses it when accessing any API endpoint that requires authentication.

### Server

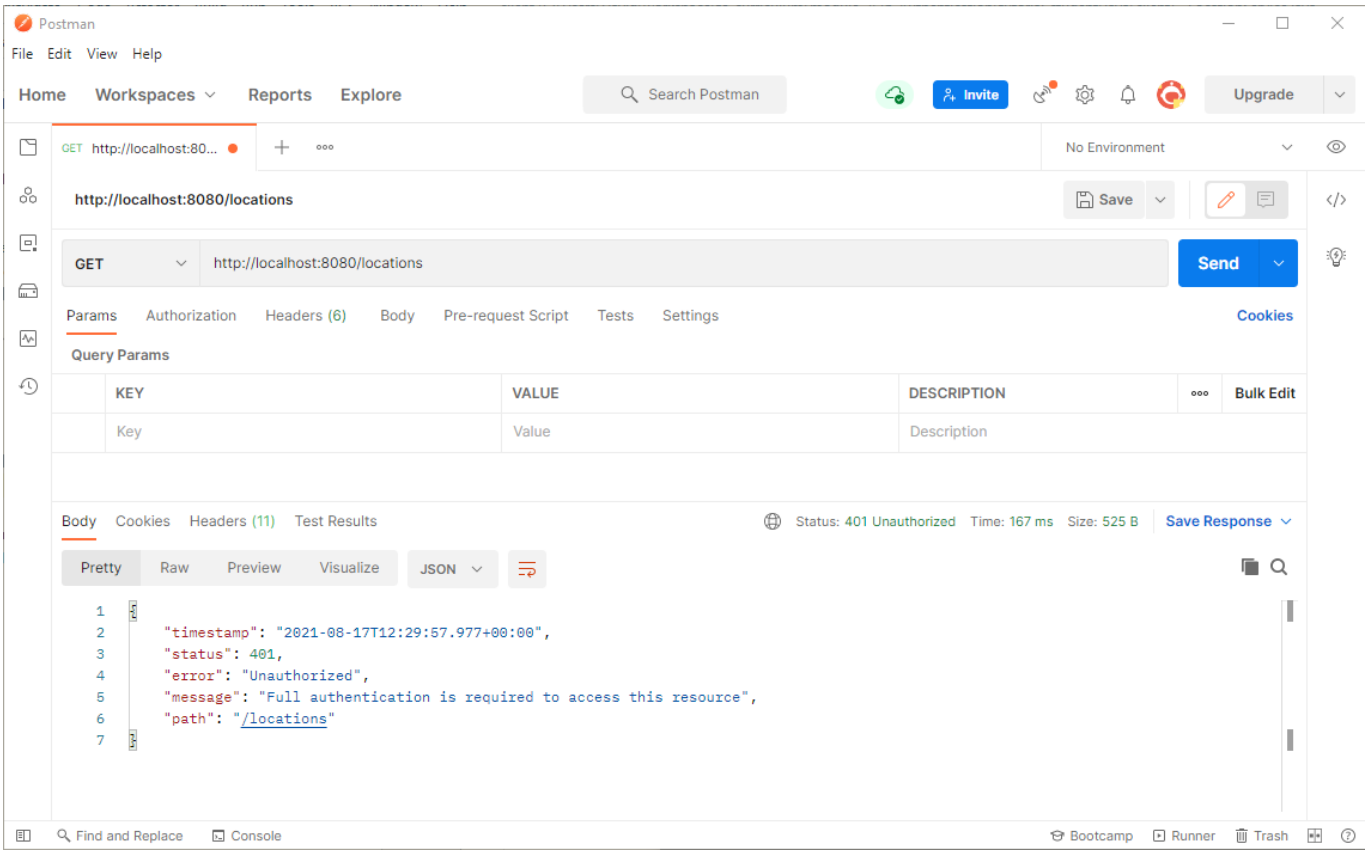
The server application should look familiar to you as it picks up where you left off in the previous tutorial. There's a new package called `security` that contains the code needed to implement authentication.

## Step Two: Run the applications

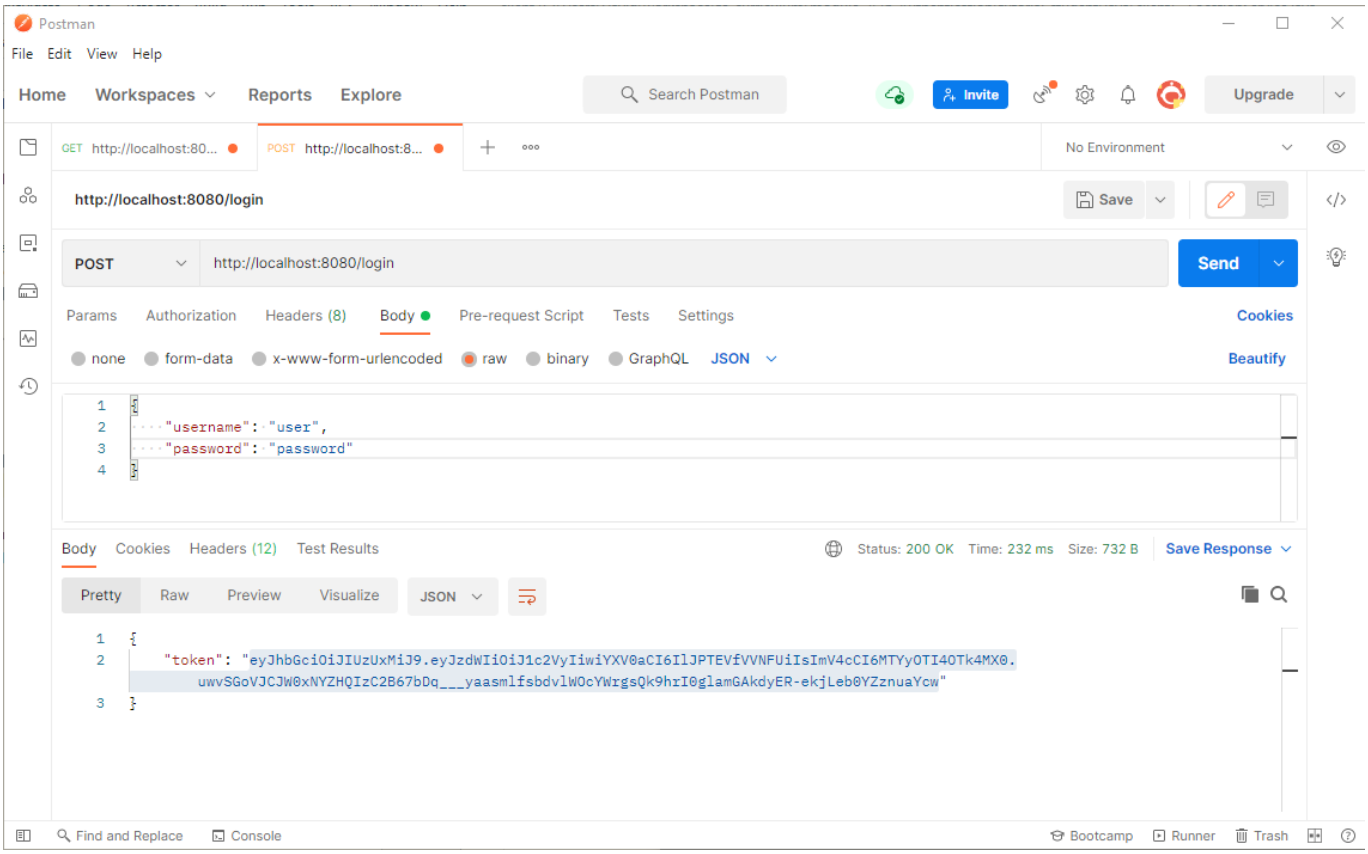
Now that you've set up your project in IntelliJ and reviewed the starting code, run both of them to verify that they start up as expected. It's best to make sure the applications run before adding anything new.

## Step Three: Test the REST API in Postman

After starting the server application, you'll need to test the REST API in Postman before writing any client code. Open Postman and try to get a list of locations by visiting http://localhost:8080/locations. This and every request mapping in the `LocationController` returns a `401 Unauthorized` response.



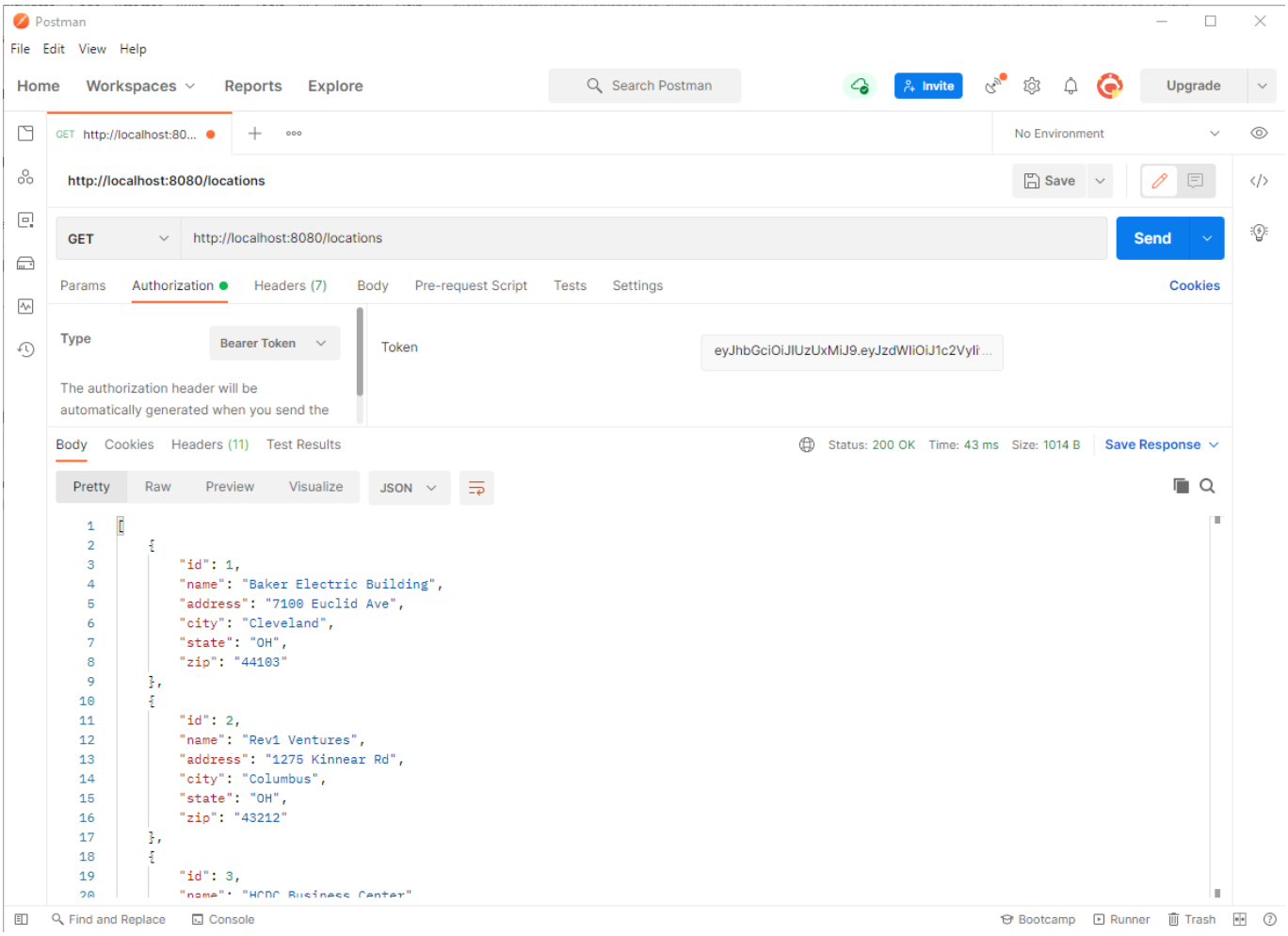

Before sending any requests to the API, you need to log in and then use the authentication token that's sent back to you in any subsequent requests. Start by sending a `POST` request to `/login` with a username and password. There's a user in the system with a username of `user` and a password of `password`.




> Note: If you send a bad username and password, what happens?

The response is a JSON key-value pair where the key is `token`. You want to copy everything inside the double quotes because this is your authentication token. Now you can go back to the `GET` locations request that failed.

Under "Authorization", select the type `Bearer Token` and paste your token into the token field. This time, you're authenticated and receive a list of locations.




# Step Four: List all locations

Now that you've tested the API using Postman, you should know what you need to do in the client. You'll need to log in using the client application, store the authentication token, and then pass that token in an authorization header each time you make a call to the API.

Start by running the client application and selecting option 1 to list all of the locations. As expected, it doesn't work. If you check the log, you'll see the client received a `401 Unauthorized` response, like you did in Postman prior to including the token.

## Location service

Open `services/LocationService.java` and locate the `getAll()` method. To pass the authentication token, you need to create a new `HttpHeaders` instance. That class has a method for setting the `Bearer Token` called `setBearerAuth()`:

```
public Location[] getAll() {
    Location[] locations = null;
    try {

        HttpHeaders headers = new HttpHeaders();
        headers.setBearerAuth(authToken);

        locations = restTemplate.getForObject(API_BASE_URL, Location[].class);
```

```
    } catch (RestClientResponseException | ResourceAccessException e) {
        BasicLogger.log(e.getMessage());
    }
    return locations;
}
```

To get a list of locations, you previously used `RestTemplate`'s `getForObject()` method. This method is a convenience method and is used when you need to make a `GET` request. In your case, you need to send additional information with the `GET` request, so you'll use another method called `exchange()`. This method takes an `HttpEntity` as an argument. The `HttpEntity` contains the headers you need to send in the request. The `exchange()` method returns a `ResponseEntity` object that includes a `getBody()` method for accessing the body of the response:

```
public Location[] getAll() {
    Location[] locations = null;
    try {

        HttpHeaders headers = new HttpHeaders();
        headers.setBearerAuth(authToken);

        HttpEntity<Void> entity = new HttpEntity<>(headers);
        ResponseEntity<Location[]> response =
           restTemplate.exchange(API_BASE_URL, HttpMethod.GET, entity,
Location[].class);
        locations = response.getBody();

    } catch (RestClientResponseException | ResourceAccessException e) {
        BasicLogger.log(e.getMessage());
    }
    return locations;
}
```

Run the client application, log in, and then try to list all of the locations.

## Step Five: Get a single location

If you try to get the details for a single location, you'll receive the same `401 Unauthorized` response. In the `LocationService` class, find the `getOne()` method. The changes you need to make here are similar to the ones you made for the `getAll()` method. Try to update this method on your own before looking at the answer below:

```
public Location getOne(int id) {
    Location location = null;
    try {
        HttpHeaders headers = new HttpHeaders();
        headers.setBearerAuth(authToken);
        HttpEntity<Void> entity = new HttpEntity<>(headers);
        ResponseEntity<Location> response = restTemplate.exchange(API_BASE_URL +
```

```
id,
                                                HttpMethod.GET, entity,
Location.class);
        location = response.getBody();
    } catch (RestClientResponseException | ResourceAccessException e) {
        BasicLogger.log(e.getMessage());
    }
    return location;
}
```

> Note: If you're having trouble with this, here's a tip that may help: run the program in debug mode and place breakpoints in the `LocationService.java` methods `getAll()` or `getOne()`. Step through the execution of the program line by line to see what's happening.

## Step Six: Refactor entity and headers

Whenever you duplicate code, ask yourself, "Is there an opportunity to refactor this?"

The answer is usually, "yes."

You used the same three lines of code each of the previous methods:

```
HttpHeaders headers = new HttpHeaders();
headers.setBearerAuth(authToken);
HttpEntity<Void> entity = new HttpEntity<>(headers);
```

You can extract that code into a new `private` method called `makeAuthEntity()` because it's only used within `LocationService.java`:

```
private HttpEntity<Void> makeAuthEntity() {
    HttpHeaders headers = new HttpHeaders();
    headers.setBearerAuth(authToken);
    return new HttpEntity<>(headers);
}
```

Now, in your `GET` methods, you can call the `makeAuthEntity()` as the third argument to `restTemplate.exchange()`:

```
public Location[] getAll() {
    Location[] locations = null;
    try {
        ResponseEntity<Location[]> response = restTemplate.exchange(API_BASE_URL,
                                    HttpMethod.GET, makeAuthEntity(),
Location[].class);
        locations = response.getBody();
    } catch (RestClientResponseException | ResourceAccessException e) {
        BasicLogger.log(e.getMessage());
```

```
        }
        return locations;
    }

    public Location getOne(int id) {
        Location location = null;
        try {
            ResponseEntity<Location> response = restTemplate.exchange(API_BASE_URL +
    id,
                                        HttpMethod.GET, makeAuthEntity(),
    Location.class);
            location = response.getBody();
        } catch (RestClientResponseException | ResourceAccessException e) {
            BasicLogger.log(e.getMessage());
        }
        return location;
    }
```

> Note: You'll also use this for the `DELETE` request because it's another operation that doesn't send a request body.

## Step Seven: Create and update location

The `add()` and `update()` methods also avoid duplicating logic by calling the method `makeLocationEntity()`. This method is different from the method you previously created because it also needs to send the location as content in the request body.

However, if you try to log in right now and create or update a location, it won't work. This is because the `makeLocationEntity()` method doesn't include the authentication token in the request header.

Try to modify this method on your own before looking at the answer below:

```
private HttpEntity<Location> makeLocationEntity(Location location) {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(authToken);
    return new HttpEntity<>(location, headers);
}
```

If you run the application, you can now create and update a location.

## Step Eight: Delete a location

The last step is to modify the delete method. This is similar to the `GET` methods because you're not sending anything in the request body. You need to replace the convenience method `restTemplate.delete()` with the lower level `exchange()` method. Try to do this on your own before looking at the answer below:

```
public boolean delete(int id) {
    boolean success = false;
    try {
        restTemplate.exchange(API_BASE_URL + id, HttpMethod.DELETE,
                              makeAuthEntity(), Void.class);
        success = true;
    } catch (RestClientResponseException | ResourceAccessException e) {
        BasicLogger.log(e.getMessage());
    }
    return success;
}
```

Now, you can run the client and the server. If you encounter problems, review the steps again to ensure that you completed each step correctly. If the requests work in Postman, but not in the client, you might need to review the `LocationService` again.

## Summary

In this tutorial, you learned:

- How to test a secure API using Postman
- How to send an authentication token using `RestTemplate`
- How to use the `RestTemplate exchange()` method for `GET`, `POST`, `PUT`, and `DELETE`.