

# Authentication and Authorization

This document describes user authentication procedure and the circumstances under which the authentication is necessary. The document ends with the description of Authorization mechanism.

## JWT authentication:

The authentication mechanism implemented in the solution is JWT authentication.

The user willing to communicate with the application will initially have to send their *username* and *password* within the HTTP request to the */login* endpoint. If the credentials are known to the application, the user will receive the JWT token in the HTTP response.

Once the token is received, the idea is to place it in the header of subsequent requests sent to the application. Once the subsequent requests are received by the application, the value stored in Authorization header **might** be validated by the application. This will depend on the authentication rules under which the application is configured - some application functionalities will not require authentication while others will.

Based on these rules and the validation, users will either be allowed the access to functionality or not.

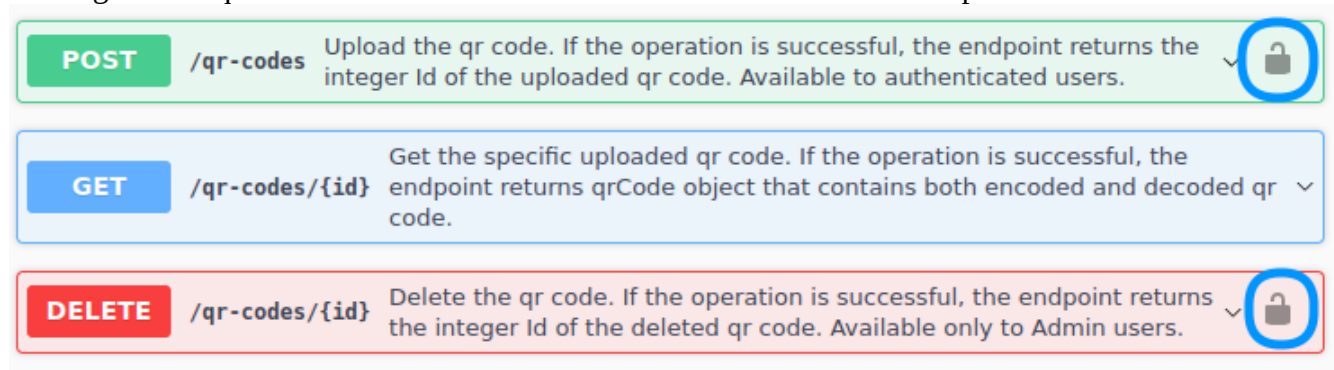
## In Swagger:

The following is the detailed description on how to authenticate an user and communicate with the application via Swagger interface that can be accessed on localhost:8080/swagger-ui/index.html .

Some endpoints shown in the Swagger documentation will be marked with the lock on the side. These locks can be shown in two variants, as either locked or unlocked. These variants and their meaning are explained bellow.

### 1. Initial state:

The framed icons on this image suggest there is no Authorization header present in the HTTP requests that will be sent by the user. The endpoints that require authentication will be unavailable for the user sending these requests. POST and DELETE will result with 401 HTTP response.



So let us create the new user, log in and set the Authorization header to their requests.

## 2. Creating a new user:

The username of a new user must be unique in the system. There are no other rules implemented for validating *username* and *password*. Roles allowed in the request are **ROLE\_STUDENT**, **ROLE\_TEACHER** and **ROLE\_ADMIN**.

The following image shows the HTTP request example which is sent to create a new user.

The screenshot shows an API client interface for the **auth-controller**. The selected method is **POST** for the endpoint **/auth/signup**. The description states: "Create the account. If the operation is successful, the endpoint returns the username of the created user in string format." There are **Cancel** and **Reset** buttons. The **Parameters** section shows "No parameters". The **Request body** is marked as **required** and the content type is set to **application/json**. The request body is a JSON object: 

```
{  "username": "primus",  "password": "primus",  "role": "ROLE_STUDENT"}

```

 Blue annotations highlight the JSON body and include instructions: "1. Fill out the form and remember the username and password", "2. choose one of the three valid role values: ROLE\_STUDENT, ROLE\_TEACHER, ROLE\_ADMIN", and "3. Submit the request". A large blue **Execute** button is at the bottom.

**auth-controller**

**POST** /auth/signup Create the account. If the operation is successful, the endpoint returns the username of the created user in string format.

**Parameters** **Cancel** **Reset**

No parameters

**Request body** **required** **application/json**

```
{  "username": "primus",  "password": "primus",  "role": "ROLE_STUDENT"}
```

1. Fill out the form and remember the username and password

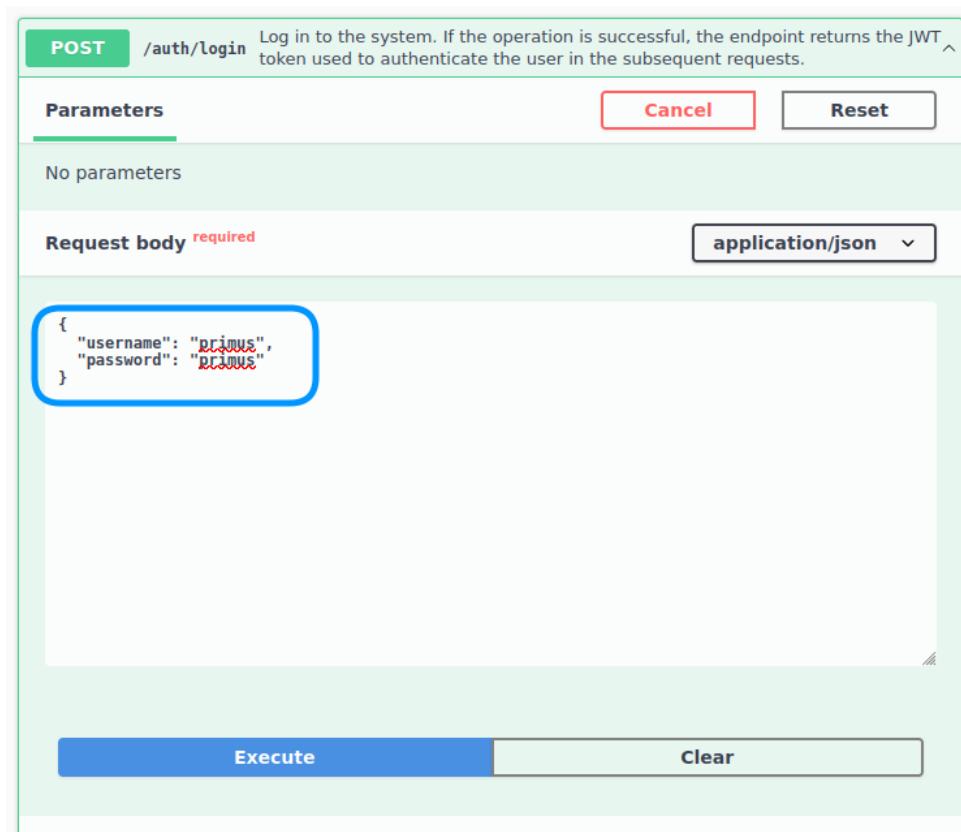
2. choose one of the three valid role values:  
ROLE\_STUDENT, ROLE\_TEACHER, ROLE\_ADMIN

3. Submit the request

**Execute**

### 3. Login with the new user:

The following request needs to be sent in order to login the user to the application.



POST /auth/login Log in to the system. If the operation is successful, the endpoint returns the JWT token used to authenticate the user in the subsequent requests.

Parameters Cancel Reset

No parameters

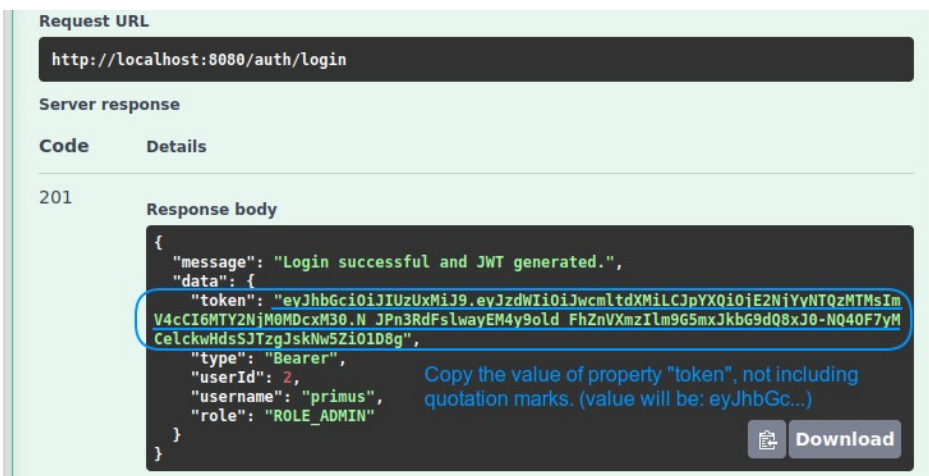
Request body required application/json

```
{  "username": "primus",  "password": "primus"}
```

Execute Clear

### 4. Acquiring JWT token:

An example of a response with the instructions is shown bellow



Request URL

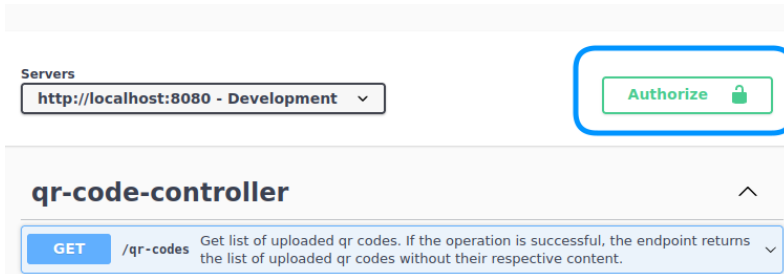
http://localhost:8080/auth/login

Server response

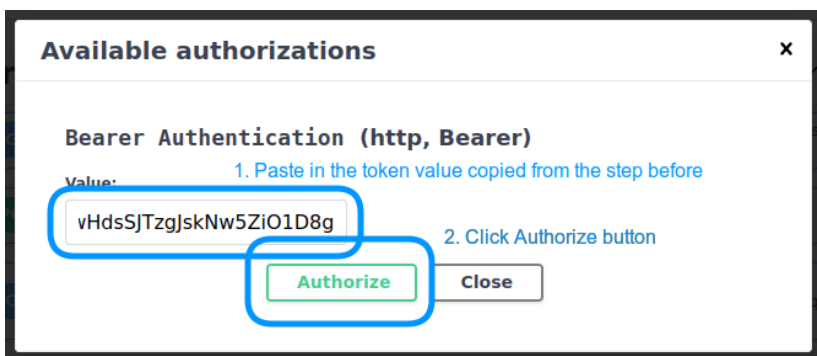
Code	Details
201	<p>Response body</p> <pre>{  "message": "Login successful and JWT generated.",  "data": {    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwcmVudXNlcjYXQ0iE2NjYyNTQzMThsImV4cCI6MTY2NjM0MDcxM30.N_JPn3RdFslwayEM4y9oId_FhZnVXmzIlm9G5mxJkbG9dQ8xJ0-NQ40F7yMCeIckwHdsSJtZgJskNw5Zi01D8g",    "type": "Bearer",    "userId": 2,    "username": "primus",    "role": "ROLE_ADMIN"  }}</pre> <p>Copy the value of property "token", not including quotation marks. (value will be: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwcmVudXNlcjYXQ0iE2NjYyNTQzMThsImV4cCI6MTY2NjM0MDcxM30.N_JPn3RdFslwayEM4y9oId_FhZnVXmzIlm9G5mxJkbG9dQ8xJ0-NQ40F7yMCeIckwHdsSJtZgJskNw5Zi01D8g)</p> <p>Download</p>

## 5. Setting the Authorization header using the acquired token:

After the value has been acquired, we are ready to authenticate this user.  
Find the *Authorization* button in the upper part of Swagger webpage and click on it.

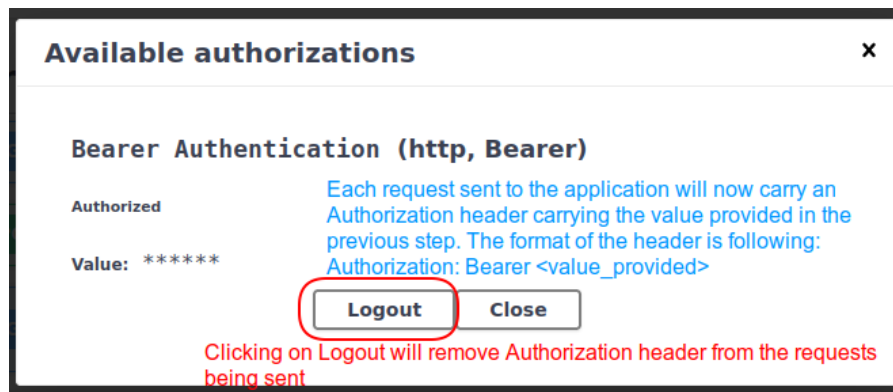


The following form will appear:



The value inserted will be the acquired JWT token from the previous step.

## 6. Authorization header is now set



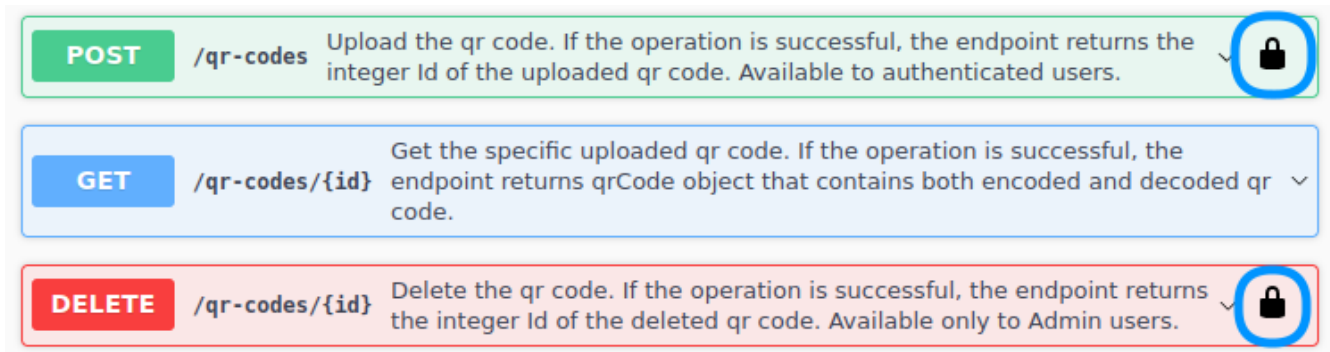
The authorization header is now set to be sent with HTTP requests.  
At any time *Logout* button can be clicked – this will remove the Authorization header from the HTTP requests being sent to the application.

Click on close to keep the set Authorization header in HTTP requests to be sent.

## 7. Authorization header is detected as present by Swagger

After the Authentication header has been set, next to those endpoints which require user authentication, the lock marks will change to a locked state. This way Swagger shows us that he is aware of the Authentication header that has been set.

Authorization header will be checked by the application for the requests sent to these two endpoints. If the header is valid, users will be considered successfully authenticated and their requests accepted by the authenticator. If the header is invalid, the user will receive 401 HTTP response with the message that they were not authenticated successfully.



## 8. After the successful authentication, authorization is performed

After the user is authenticated, authorization rules are consulted and are applied to the user. RBAC has been implemented in order to restrict the access to certain application functionalities (to authorize the users for functionalities).

Restrictions/constraints are applied based on the role a user is registered under within the application. In this application, there are 3 possible roles:

- **ROLE\_STUDENT**
- **ROLE\_TEACHER**
- **ROLE\_ADMIN**

The only RBAC permission constraint implemented is the following:

- DELETE /qr-codes/{id} endpoint can only be accessed by the user with *role* = **ROLE\_ADMIN**

This means that if an authenticated user with *role* = **ROLE\_STUDENT** or **ROLE\_TEACHER** attempts to access DELETE /qr-codes/{id} endpoint, they will receive 403 HTTP response because they are not authorized to perform the operation.

This also means that if an authenticated user with *role* = **ROLE\_ADMIN** attempts to access DELETE /qr-codes/{id} endpoint they will be authorized to access the endpoint.