

Lab 8

Part 1:

Write a simple application with 3 REST endpoints:

/shop is accessible by everyone

/orders is accessible by all employees

/payments is accessible only by employees of the finance department

Make 2 in memory users:

Bob (who is an employee of the sales department)

Mary (who is an employee of the finance department)

Test your application using postman so that Bob can only access /shop and /orders, and that Mary can access all 3 endpoints.

Part2:

Modify the given code of WebSecurityProject2 so that we have 2 more endpoints:

/manager is accessible only by managers

/topmanager is accessible only by top managers

Put 2 more users in the database one with role manager and one with role topmanager.

Test your application using postman

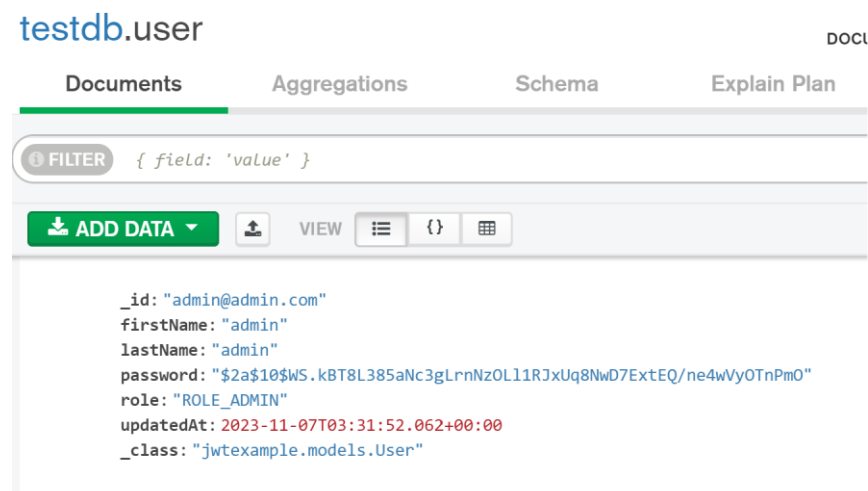
Part3:

Run the given application **springjwtdemo**

Make a screenshot of every step given below.

Put these screenshots in a document that you submit for this exercise.

First check if user admin is saved in the database.



Then in postman check that you can only access the **/api/all** endpoint

The screenshot shows the Postman interface for a GET request to `localhost:8080/api/all`. The request is configured with the method `GET` and the URL `localhost:8080/api/all`. The `Auth` tab is selected, showing `No Auth`. The response status is `200 OK` with a response time of `102 ms` and a body size of `355 B`. The response body is displayed in the `Body` tab, showing the text `everyone can see this`.

GET localhost:8080/api/all

Send

Params Auth Headers (8) Body Pre-req. Tests Settings

Type

No Auth

This request does not use any authorization. Learn more about [authorization](#)

Body 200 OK 102 ms 355 B Save as example

Pretty Raw Preview Visualize Text

1 everyone can see this

The screenshot shows the Postman interface for a GET request to `localhost:8080/api/users`. The request is configured with the method `GET` and the URL `localhost:8080/api/users`. The `Auth` tab is selected, showing `No Auth`. The response status is `403 Forbidden` with a response time of `66 ms` and a body size of `439 B`. The response body is displayed in the `Body` tab, showing a JSON object with the following structure:

```
{
  "timestamp": "2023-11-07T03:35:22.332+00:00",
  "status": 403,
  "error": "Forbidden",
  "path": "/api/users"
}
```

GET localhost:8080/api/users

Send

Params Auth Headers (8) Body Pre-req. Tests Settings

Type

No Auth

This request does not use any authorization. Learn more about [authorization](#)

Body 403 Forbidden 66 ms 439 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2023-11-07T03:35:22.332+00:00",
3   "status": 403,
4   "error": "Forbidden",
5   "path": "/api/users"
6 }
```

Then call the **/signup** to signup a new user:

POST localhost:8080/auth/signup

Params Auth Headers (10) Body ● Pre-req. Tests Settings

raw JSON Beautify

```
1 {
2   "firstName": "bob",
3   "password": "user",
4   "email": "john@gmail.com",
5   "lastName": "johnson"
6 }
```

Body 200 OK 367 ms 489 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huQGdtYWlsLmNvbSIsIm1hdCI6MTY5OTMyMzY1MCwiZXhwIjoxNjk5MzI3MjUwZjQ.G-iz_VDiJkwkcyxvsUH4PHBz-5YqY6nCD4HdvoZMWfA"
3 }
```

Check if this user is added to the database:

testdb.user DO

Documents Aggregations Schema Explain Plan

FILTER { field: 'value' }

ADD DATA VIEW

```
_id: "admin@admin.com"
firstName: "admin"
lastName: "admin"
password: "$2a$10$WS.kBT8L385aNc3gLrnNzOL11RJxUq8NwD7ExtEQ/ne4wVyOTnPM0"
role: "ROLE_ADMIN"
updatedAt: 2023-11-07T03:31:52.062+00:00
_class: "jwtexample.models.User"
```

```
_id: "john@gmail.com"
firstName: "bob"
lastName: "johnson"
password: "$2a$10$r7pBlUSnta10xak4tJzG0gD7jpr7GtaiCqrjZkQ6CbOkLlzY1ly"
role: "ROLE_USER"
updatedAt: 2023-11-07T03:39:50.336+00:00
_class: "jwtexample.models.User"
```

Then let this user signin to the application

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/auth/signin
- Body:** JSON (raw view)

```
{
  "email": "john@gmail.com",
  "password": "user"
}
```
- Response:** 200 OK, 310 ms, 489 B. The response body is shown in JSON (Pretty view):

```
{
  "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huQGdtYWlsLmNvbSIsIm1hdCI6MTY5OTMyNDM1OCwiZXhwIjoxNjk5MzI3OTU4fQ.1P3mhrYqfQ_wFuppqcM1N2piLEZl9SeCB4Zyik9-FK0"
}
```

Copy the provided token in a text file.

We can now check the content of the token at <https://jwt.io/>

First copy the secret key from **application.properties**

The screenshot shows the `application.properties` file with the following configuration:

```
1 spring.data.mongodb.host=localhost
2 spring.data.mongodb.port=27017
3 spring.data.mongodb.database=testdb
4
5 # JWT secret key
6 token.secret.key=48a868a4042f634ac04a117f00a87202131dd7c46c4b32c4acb3edc5e15f4511
7
8 # JWT expiration is 1 hour
9 token.expirationms=3600000
```

Paste this secret in jwt.io

VERIFY SIGNATURE


```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  48a868a4042f634ac04a1`  
)
```


☒ secret base64 encoded

Also select the base 64 encode checkbox.

Then copy and paste the token in the **Encoded** field

← → ↺ jwt.io

 Debugger Libraries Introduction Ask

Crafted by  Auth0 by Okta

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqb2huQGdtYWlsLmNvbSIsIm1hdCI6MTY5OTMyODg3NywiZXhwIjoxNjk5MzMyNDc3fQ.j_1Lfx-sBVj1inD2MFwv0knEkxFA-RcH0u7hiKUePzI
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256"  
}
```

PAYLOAD: DATA

```
{  
  "sub": "john@gmail.com",  
  "iat": 1699328877,  
  "exp": 1699332477  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  48a868a4042f634ac04a1`  
)
```

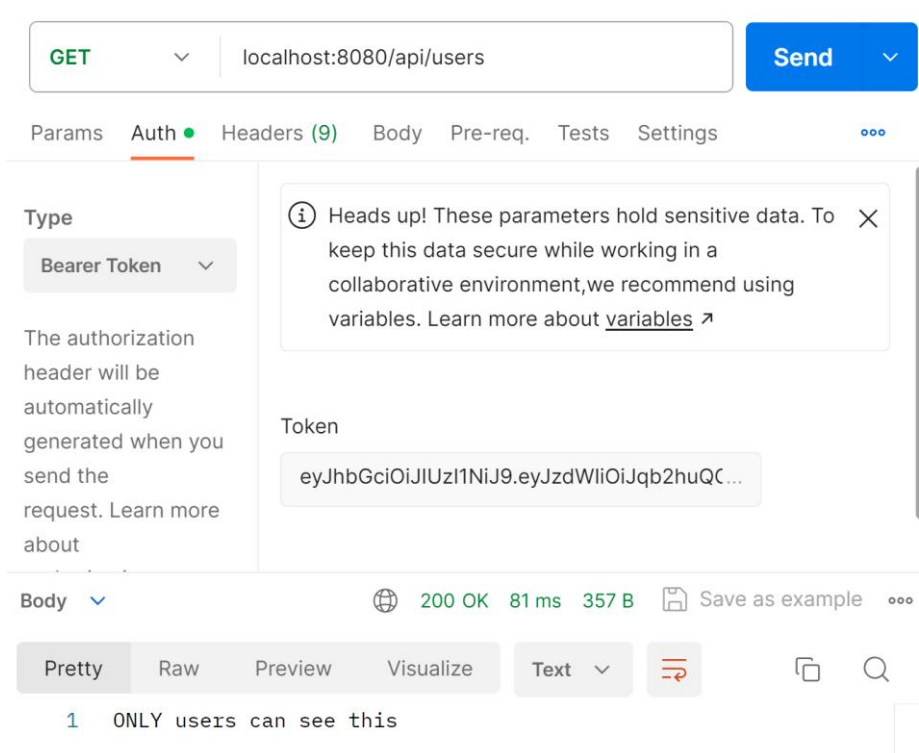
☒ secret base64 encoded

✔ Signature Verified

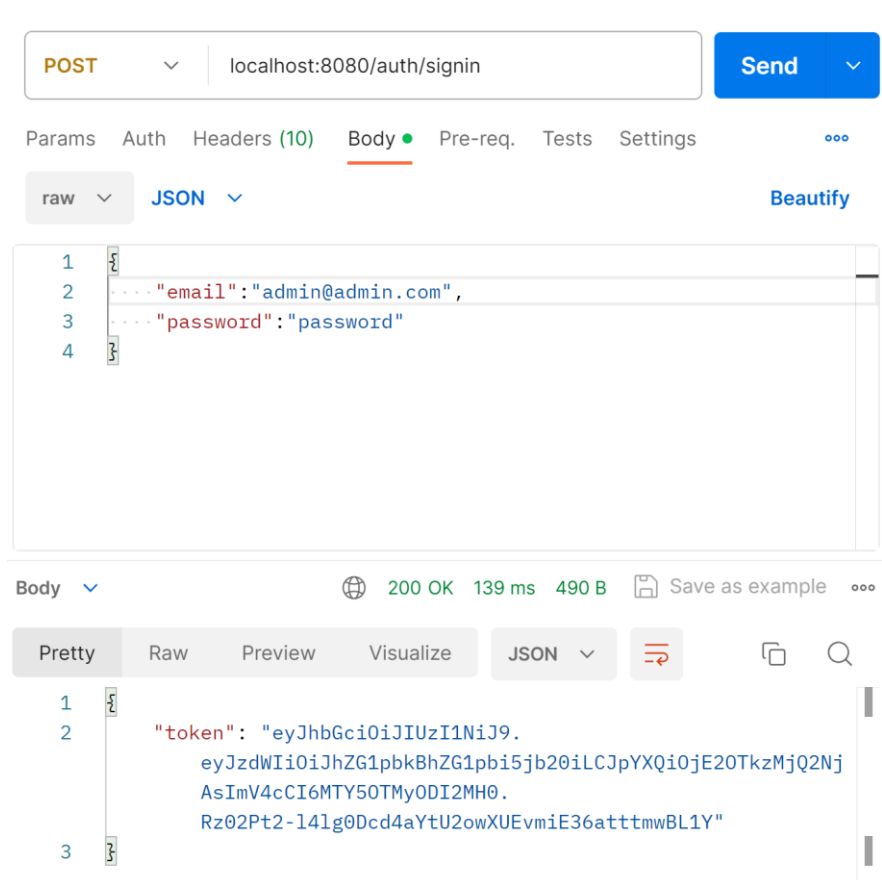
SHARE JWT

Notice that the payload contains the correct user email.

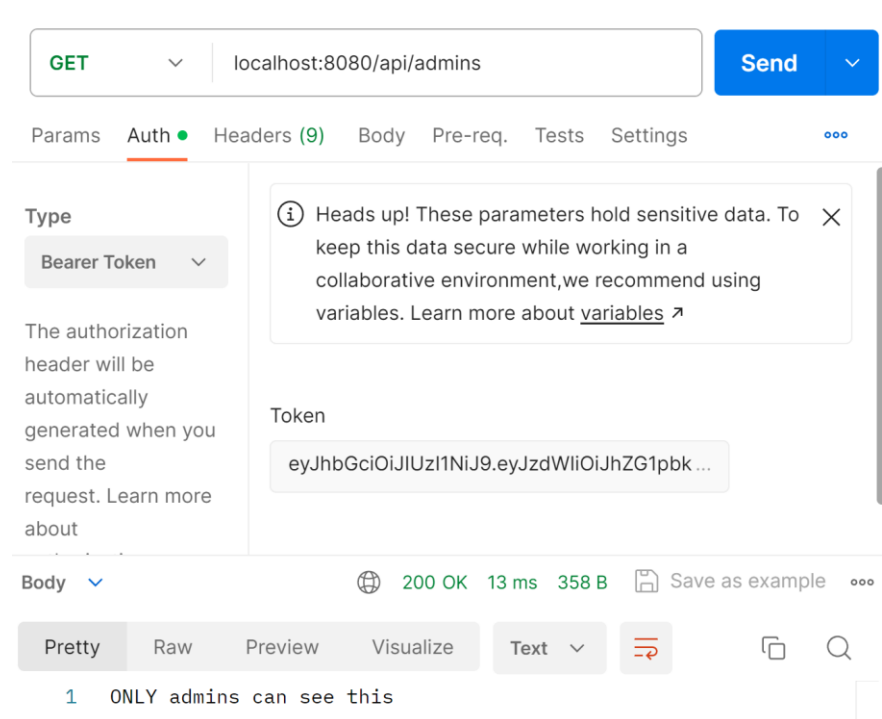
Now call the **/users** endpoint in postman using the token



Then signin as admin and get the token for admin



Copy the token and check that you can see admin data using the admin token



What to hand in?

1. A zip file of part 1
2. A zip file of part 2
3. A PDF of the screenshots from part 3