

Angular WebApp Security with JWT

(Reverse engineering Jhipster Generated Code)

arnaud.nauwynck@gmail.com

this document: <https://github.com/Arnaud-Nauwynck/Presentations/Web/Security-JWT-Angular-ReverseEngineering-Jhipster>

Step 1: create Jhipster app

```
$ jhipster
INFO! Using bundled JHipster
JHIPSTER
https://www.jhipster.tech
Welcome to JHipster v7.8.1
Application files will be generated in folder: C:\arn\test-jhipster
Documentation for creating an application is at https://www.jhipster.tech/creating-an-app/
If you find JHipster useful, consider sponsoring the project at https://opencollective.com/generator-jhipster

WARNING! Your Node version is not LTS (Long Term Support), use it at your own risk! JHipster does not support non-LTS releases,
?
? Which *type* of application would you like to create? (Use arrow keys)
> Monolithic application (recommended for simple projects)
  Gateway application
  Microservice application
```

Choose App Type > Security Type

```
? Which *type* of application would you like to create? Monolithic application (recommended for simple projects)
? What is the base name of your application? myapp
? Do you want to make it reactive with Spring WebFlux? No
? What is your default Java package name? com.mycompany.myapp
? Which *type* of authentication would you like to use? (Use arrow keys)
> JWT authentication (stateless, with a token)
  OAuth 2.0 / OIDC Authentication (stateful, works with Keycloak and Okta)
  HTTP Session Authentication (stateful, default Spring Security mechanism)
```

End Choices (Db, Angular, Maven, ..)

```
? Which *type* of database would you like to use? SQL (H2, PostgreSQL, MySQL, MariaDB, Oracle, MSSQL)
? Which *production* database would you like to use? PostgreSQL
? Which *development* database would you like to use? H2 with disk-based persistence
? Which cache do you want to use? (Spring cache abstraction) No cache - Warning, when using an SQL database, this will disable the Hibernate 2nd level cache!
? Would you like to use Maven or Gradle for building the backend? Maven
? Do you want to use the JHipster Registry to configure, monitor and scale your application? No
? Which other technologies would you like to use? WebSockets using Spring Websocket
? Which *Framework* would you like to use for the client? Angular
? Do you want to generate the admin UI? Yes
? Would you like to use a Bootswatch theme (https://bootswatch.com/)? Default JHipster
? Would you like to enable internationalization support? No
? Please choose the native language of the application English
? Besides JUnit and Jest, which testing frameworks would you like to use?
? Would you like to install other generators from the JHipster Marketplace? No
```

Start Generating

```
G n ration d'une paire de cl s RSA de 2048 bits et d'un certificat auto-sign  (SHA256withRSA) d'une validit  de 99999 jours
pour : CN=Java Hipster, OU=Development, O=com.mycompany.myapp, L=, ST=, C=
KeyStore 'C:\arn\test-jhipster\src\main\resources\config\tls\keystore.p12' generated successfully.
```

Generating a RSA Key pair, for exposing https instead of http
(but self-signed certificate)

Finished Generating

```
Application successfully committed to Git from C:\arn\test-jhipster.

If you find JHipster useful consider sponsoring the project https://www.jhipster.tech/sponsors/

Server application generated successfully.

Run your Spring Boot application:
./mvnw (mvnw if using Windows Command Prompt)

Client application generated successfully.

Start your Webpack development server with:
npm start

> myapp@0.0.1-SNAPSHOT clean-www
> rimraf target/classes/static/app/{src,target/}

Congratulations, JHipster execution is complete!
Sponsored with ❤ by @oktadev.

arnaud@DESKTOP-2EGCC8R /cygdrive/c/arn/test-jhipster
$
```

Created Files

```
create .prettierrc
create .prettierignore
create package.json
  force .yo-rc.json
  force .yo-rc-global.json
create .gitattributes
create .editorconfig
create .gitignore
create sonar-project.properties
create .husky\pre-commit
create .lintstagedrc.js
create mvnw
create mvnw.cmd
create .mvn\jvm.config
create .mvn\wrapper\maven-wrapper.jar
create .mvn\wrapper\maven-wrapper.properties
create npmw
create npmw.cmd
create src\main\resources\banner.txt
```

(truncated ...) total files :

```
$ find . | wc -l
105964
```

Lot of npm files (in node_modules/**) ... but ok .gitignore

```
$ git status  
On branch master  
nothing to commit, working tree clean
```

```
$ git log  
commit 86ba746ae24834926b6b79e85cecef20481d44ab (HEAD -> master)  
Author: Arnaud Nauwynck <arnaud.nauwynck@gmail.com>  
Date:   Thu May 19 14:13:09 2022 +0200  
  
Initial version of myapp generated by generator-jhipster@7.8.1
```

```
$ cat .gitignore  
#####  
# Project Specific  
#####  
/src/main/webapp/content/css/main.css  
/target/classes/static/**  
/src/test/javascript/coverage/  
  
#####  
# Node  
#####  
/node/  
node_tmp/  
node_modules/  
npm-debug.log.*  
.awcache/*  
.cache-loader/*
```

```
$ ls -1  
README.md  
angular.json  
checkstyle.xml  
jest.conf.js  
mvnw  
mvnw.cmd  
ngsw-config.json  
node_modules  
npmw  
npmw.cmd  
package-lock.json  
package.json  
pom.xml  
sonar-project.properties  
src  
target  
tsconfig.app.json  
tsconfig.json  
tsconfig.spec.json  
webpack
```

```
$ find . -maxdepth 1 -type d  
.br/>./.devcontainer  
./.git  
./husky  
./.mvn  
./node_modules  
./src  
./target  
./webpack
```

Src Files

\$ find src/main wc -l	464	\$ find src/main/java wc -l	92
		\$ find src/main/webapp wc -l	322
		\$ find src/main/resources wc -l	29
		\$ find src/main/docker wc -l	20

Created Files - Categories

```
$ find . -maxdepth 1 -type d
.
./.devcontainer
./.git
./.husky
./.mvn
./node_modules
./src
./target
./webpack
```

src/
(in git)

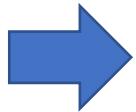
src/main
(in artifact delivery)

src/test
(during test only)

```
$ find ./src -maxdepth 2
./src
./src/main
./src/main/docker
./src/main/java
./src/main/resources
./src/main/webapp
./src/test
./src/test/java
./src/test/resources
```

src/main/webapp

Technical layers organization:
account, admin, config, core, entities, home, layouts, login, shared



```
$ find ./src/main/webapp/app -maxdepth 1
./src/main/webapp/app
./src/main/webapp/app/account
./src/main/webapp/app/admin
./src/main/webapp/app/app-routing.module.ts
./src/main/webapp/app/app.constants.ts
./src/main/webapp/app/app.module.ts
./src/main/webapp/app/config
./src/main/webapp/app/core
./src/main/webapp/app/entities
./src/main/webapp/app/home
./src/main/webapp/app/layouts
./src/main/webapp/app/login
./src/main/webapp/app/shared
```

src/mainjava

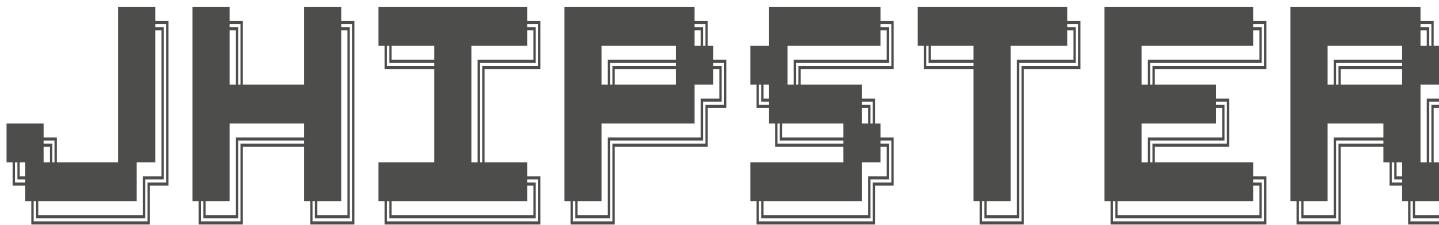
Technical layers organization:
aop, config, domain, managment, security, service, web

```
$ find ./src/main/java -maxdepth 4
./src/main/java
./src/main/java/com
./src/main/java/com/mycompany
./src/main/java/com/mycompany/myapp
./src/main/java/com/mycompany/myapp/aop
./src/main/java/com/mycompany/myapp/ApplicationWebXml.java
./src/main/java/com/mycompany/myapp/config
./src/main/java/com/mycompany/myapp/domain
./src/main/java/com/mycompany/myapp/GeneratedByJHipster.java
./src/main/java/com/mycompany/myapp/management
./src/main/java/com/mycompany/myapp/MyappApp.java
./src/main/java/com/mycompany/myapp/repository
./src/main/java/com/mycompany/myapp/security
./src/main/java/com/mycompany/myapp/service
./src/main/java/com/mycompany/myapp/web
```

Launching Backend: mvn -Pdev spring-boot:run

```
$ mvn -Pdev spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.mycompany.myapp:demo >-----
[INFO] Building Demo 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> spring-boot:3.3.5:run (default-cli) > test-compile @ demo >>>

[INFO] --- spring-boot:3.3.5:run (default-cli) @ demo ---
[INFO] Attaching agents: []
```



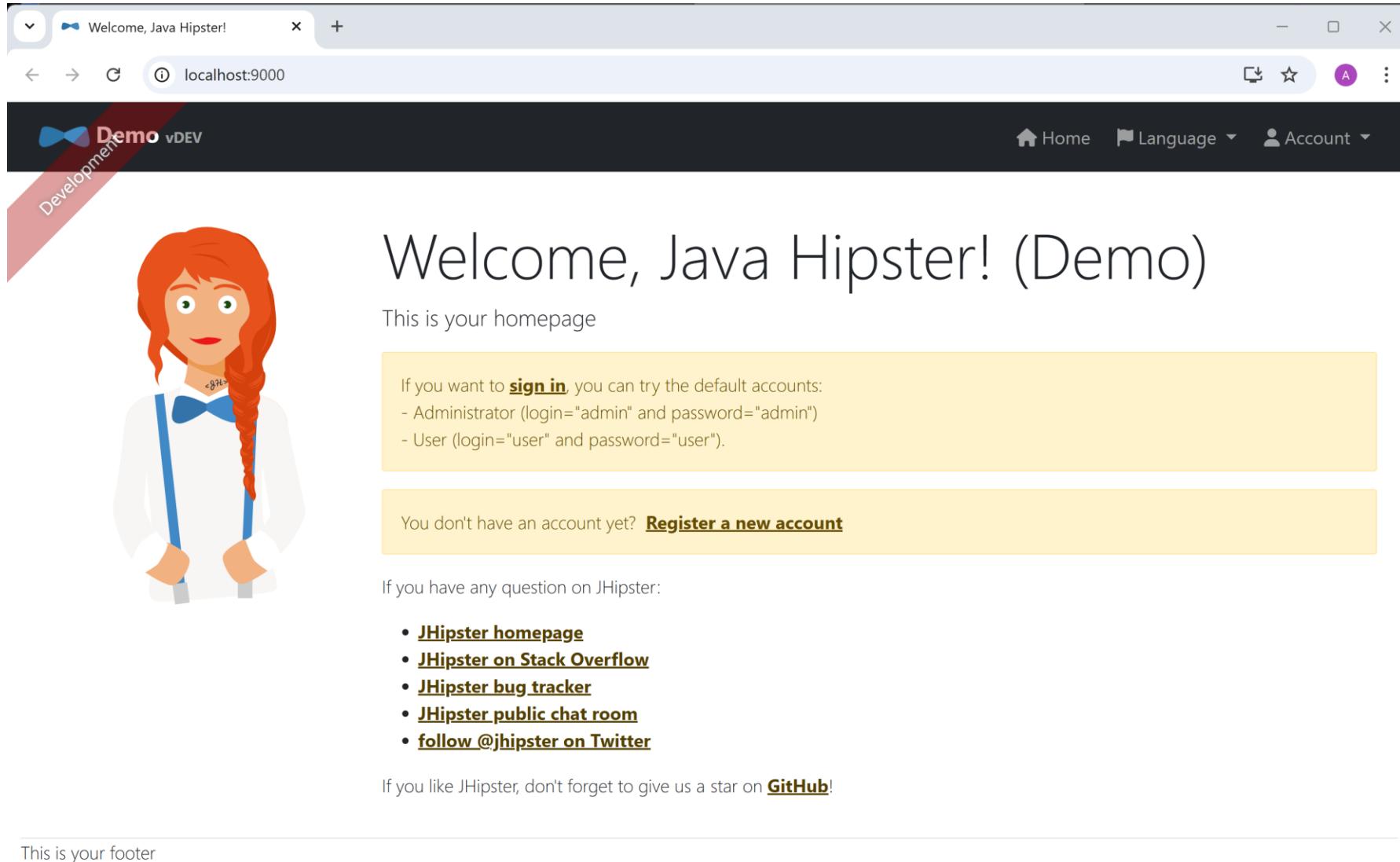
```
2024-11-19T17:05:44.632+01:00  INFO 7000 --- [ restartedMain] com.mycompany.myapp.DemoApp          : Started DemoApp in 11.616 seconds (process runni
ng for 12.128)
2024-11-19T17:05:44.644+01:00  INFO 7000 --- [ restartedMain] com.mycompany.myapp.DemoApp          :
-----
Application 'demo' is running! Access URLs:
Local:          http://localhost:8080/
External:       http://192.168.0.50:8080/
Profile(s):    [dev, api-docs]
```

Launching Frontend: npm run start (= ng serve)

```
$ npm run start  
> demo@0.0.1-SNAPSHOT start  
> ng serve --hmr  
  
Node.js version v23.2.0 detected.
```

```
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **  
  
✓ Compiled successfully.  
Notifications are disabled  
Reason: DisabledByGroupPolicy Please make sure that the app id is set correctly.  
Command Line: C:\web\demo-jhipster\demo\node_modules\node-notifier\vendor\snoreToast\snoretoast-x64.exe -pipeName \\.\pi  
cbca1e849 -p C:\web\demo-jhipster\demo\webpack\logo-jhipster.png -m "Build successful" -t Demo  
[Browsersync] Proxying: http://localhost:4200  
[Browsersync] Access URLs:  
-----  
  Local: http://localhost:9000  
  External: http://192.168.0.50:9000  
-----  
  UI: http://localhost:3001  
UI External: http://192.168.0.50:3001  
-----
```

Home page - not logged-in



The screenshot shows a web browser window with the title "Welcome, Java Hipster!" and the URL "localhost:9000". The page has a dark header with a logo and navigation links for "Home", "Language", and "Account". A red ribbon banner on the left says "Development Demo vDEV". The main content features a cartoon character with orange hair and a bow tie, and the text "Welcome, Java Hipster! (Demo)". It includes instructions for logging in with default accounts ("Administrator" or "User") and a link to register a new account. Below this, there's a section for questions with links to the JHipster homepage, Stack Overflow, bug tracker, public chat room, and Twitter. At the bottom, there's a GitHub link and a footer note.

Welcome, Java Hipster!

localhost:9000

Development Demo vDEV

Home Language Account

Welcome, Java Hipster! (Demo)

This is your homepage

If you want to [sign in](#), you can try the default accounts:

- Administrator (login="admin" and password="admin")
- User (login="user" and password="user").

You don't have an account yet? [Register a new account](#)

If you have any question on JHipster:

- [JHipster homepage](#)
- [JHipster on Stack Overflow](#)
- [JHipster bug tracker](#)
- [JHipster public chat room](#)
- [follow @jhipster on Twitter](#)

If you like JHipster, don't forget to give us a star on [GitHub](#)!

This is your footer

Click sign-in => /login page

The screenshot shows a web browser window with the URL `localhost:9000/login`. The page title is "Sign in".

The page includes the following elements:

- Header:** A dark header bar with a blue ribbon icon and the text "Demo vDEV". It also features "Development" and "vDEV" text.
- Top Right:** Navigation icons for refresh, search, and other browser functions.
- Header Buttons:** "Home", "Language", and "Account" buttons.
- Form Fields:** "Username" field containing "admin" and "Password" field containing ".....".
- Checkboxes:** "Remember me" checkbox.
- Buttons:** A blue "Sign in" button.
- Callout Boxes:** Two yellow rectangular boxes at the bottom.
 - The top box contains the text "[Did you forget your password?](#)".
 - The bottom box contains the text "You don't have an account yet? [Register a new account](#)".

Home Page after logged-in

A screenshot of a web browser window displaying the JHipster homepage at localhost:9000. The browser's address bar shows the URL. The page has a dark header with a red diagonal banner on the left containing the text "Development Demo vDEV". On the right side of the header are links for "Home", "Entities", "Administration", "Language", and "Account". Below the header, there is a large illustration of a person with orange hair in a white shirt and blue bow tie. The main content area features a large heading "Welcome, Java Hipster! (Demo)". Below it, a message says "This is your homepage". A green callout box contains the text "You are logged in as user 'admin'.". Further down, there is a section for questions with links to the JHipster homepage, Stack Overflow, bug tracker, public chat room, and Twitter account. At the bottom, there is a link to GitHub.

Welcome, Java Hipster! (Demo)

This is your homepage

You are logged in as user "admin".

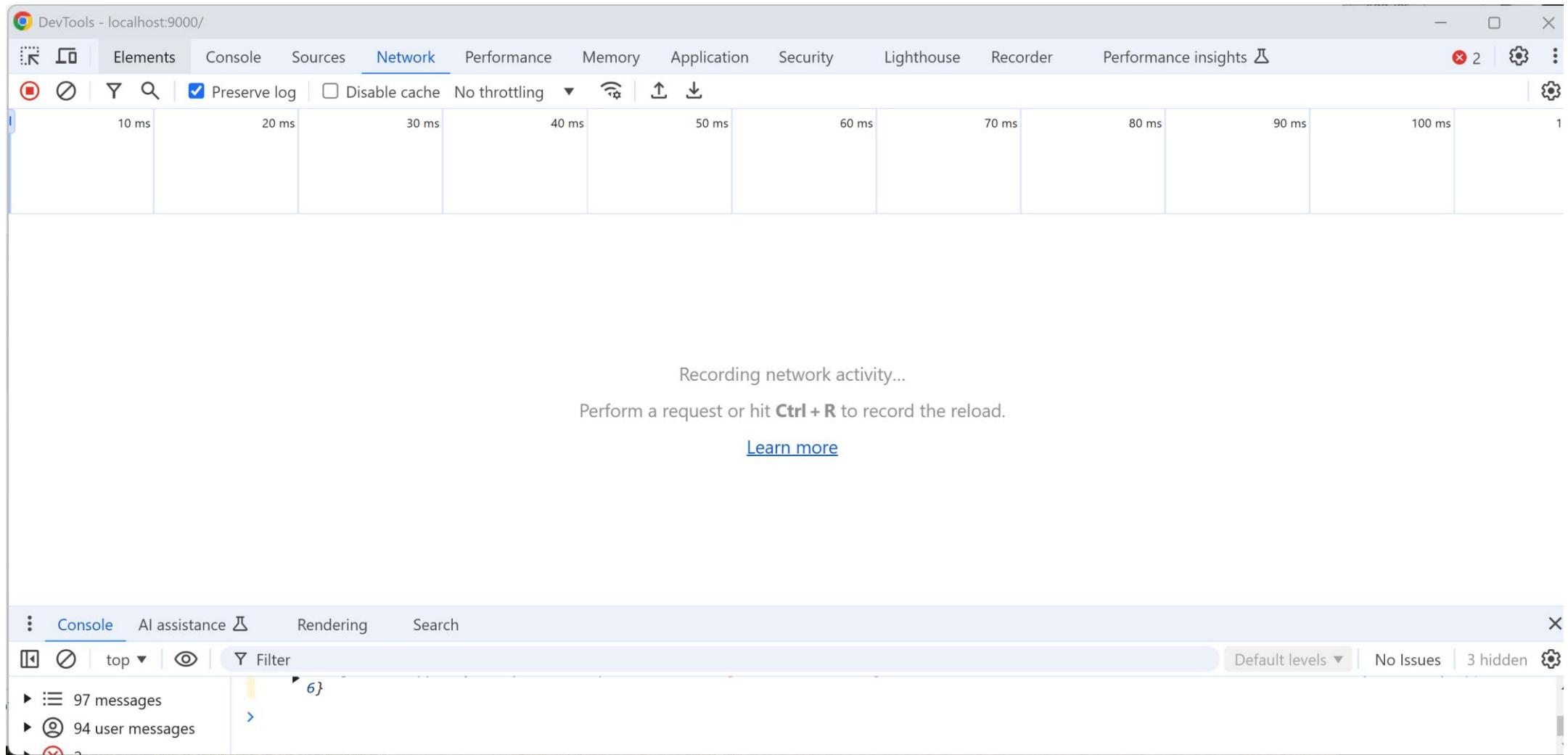
If you have any question on JHipster:

- [JHipster homepage](#)
- [JHipster on Stack Overflow](#)
- [JHipster bug tracker](#)
- [JHipster public chat room](#)
- [follow @jhipster on Twitter](#)

If you like JHipster, don't forget to give us a star on [GitHub](#)!

Debuging Security from "Http" viewpoint

Chrome Dev Tools > Network



Sign Out => NO Http call

The screenshot shows a web application interface. At the top, there is a dark navigation bar with several items: Home, Entities, Administration, Language, and Account. The 'Account' item has a red box drawn around it. A dropdown menu is open under the 'Account' item, containing three options: Settings, Password, and Sign out. The 'Sign out' option is also highlighted with a red box. Below the navigation bar, the main content area displays a welcome message: "Welcome, Java Hipster! (Demo)". Underneath this, a green banner states: "This is your homepage". At the bottom of the banner, it says: "You are logged in as user 'admin'.".

Home Entities Administration Language Account

Settings

Password

Sign out

Welcome, Java Hipster! (Demo)

This is your homepage

You are logged in as user "admin".

Sign-in => 3 Http Requests

The screenshot shows the Network tab in Google Chrome DevTools. It displays three requests made during a sign-in process:

Name	Status	Type	Initiator	Size	Time
account	401	xhr	login.component.ts:32	927 B	53 ...
authenticate	200	xhr	login.component.ts:44	1.6 kB	284...
account	200	xhr	login.component.ts:44	1.3 kB	46 ...

At the bottom of the Network tab, it shows "3 requests | 3.8 kB transferred | 602 B resources".

The Console tab is active in the bottom panel, showing the following log entries:

- NavigationEnd {id: 6, url: '/', urlAfterRedirects: '/', type: 1} (router.mjs:6854)
- 148 messages
- 142 user messages

Request [1/3]: Http GET /api/account no auth Header => 401 Unauthorized

Name	Headers	Preview	Response	Initiator	Timing	Cookies
✖ account						
⌚ authenticate						
⌚ account						
	General					
	Request URL:	http://localhost:9000/api/account				
	Request Method:	GET				
	Status Code:	401 Unauthorized				
	Remote Address:	[::1]:9000				
	Referrer Policy:	strict-origin-when-cross-origin				
	Response Headers	Raw				
	Access-Control-Allow-Origin:	*				
	Cache-Control:	no-cache, no-store, max-age=0, must-revalidate				
	Connection:	close				
	Content-Length:	0				
	Content-Security-Policy:	default-src 'self'; frame-src 'self' data; script-src 'self' 'unsafe-inline' 'unsafe-eval' https://storage.googleapis.com; style-src 'self' 'unsafe-inline'; img-src 'self' data; font-src 'self' data;				
	Date:	Tue, 19 Nov 2024 16:14:54 GMT				
	Expires:	0				
	Permissions-Policy:	camera=(), fullscreen=(self), geolocation=(), gyroscope=(), magnetometer=(), microphone=(), midi=(), payment=(), sync-xhr=()				
	Pragma:	no-cache				

3 requests | 3.8 kB transferred | 602 B resources

MISSING Header "Authorization" => FAILING with 401

Notice : why first Failing http GET /api/account ?

as soon as you click on the "SIGN IN" button

=> the authentication is immediatly cleared

=> JHipster immediatly refresh the current user account (? why? it could be just cleared)

Then there is a pending call to "/api/authenticate"

(This could take for example 2 seconds... In the mid-time, still no calls are possible)

When receiving the "/api/authenticate" response, then immediatly after,

Jhipster redo another refresh for the current user account (this one will succeed)

Request [2/3]: Http POST /api/authenticate with request Body = { username, password }

Name	X Headers	Payload	Preview	Response	Initiator	Timing	Cookies
account							
authenticate							
account							

▼ General

Request URL: http://localhost:9000/api/authenticate

Request Method: POST

Status Code: 200 OK

Remote Address: [:1]:9000

Referrer Policy: strict-origin-when-cross-origin

► Response Headers (20)

► Request Headers (17)

X Headers	Payload	Preview	Response	Initiator	Timing	Cookies

▼ Request Payload view source

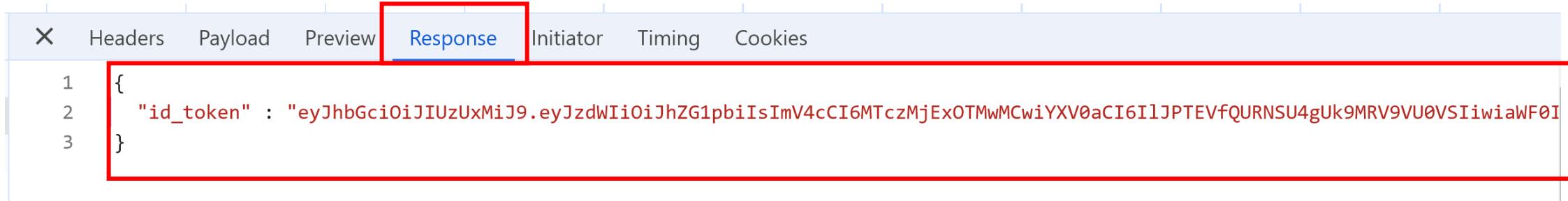
```
{username: "admin", password: "admin", rememberMe: false}
```

password: "admin"

rememberMe: false

username: "admin"

Request [2/3] Response Body: { jwt token }



The screenshot shows the Network tab of a browser developer tools interface. The 'Response' tab is selected, highlighted by a red box. The response body is displayed as a JSON object:

```
1 {  
2   "id_token" : "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbkiIsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IiJPTEVfQURNSU4gUk9MRV9VU0VSIiwiaWF0I  
3 }
```

the response is a **JSON** object

It contains only 1 field, named "**id_token**"

We will see next, that the string value "eyJH..." is called a **JWT Token**

Request [3/3]: Http GET /api/account => status code: 200 OK

The screenshot shows the Network tab in Google DevTools for a request to `http://localhost:9000/api/account`. The request method is GET and the status code is 200 OK. The response body contains a large JSON object.

Request URL: http://localhost:9000/api/account

Request Method: GET

Status Code: 200 OK

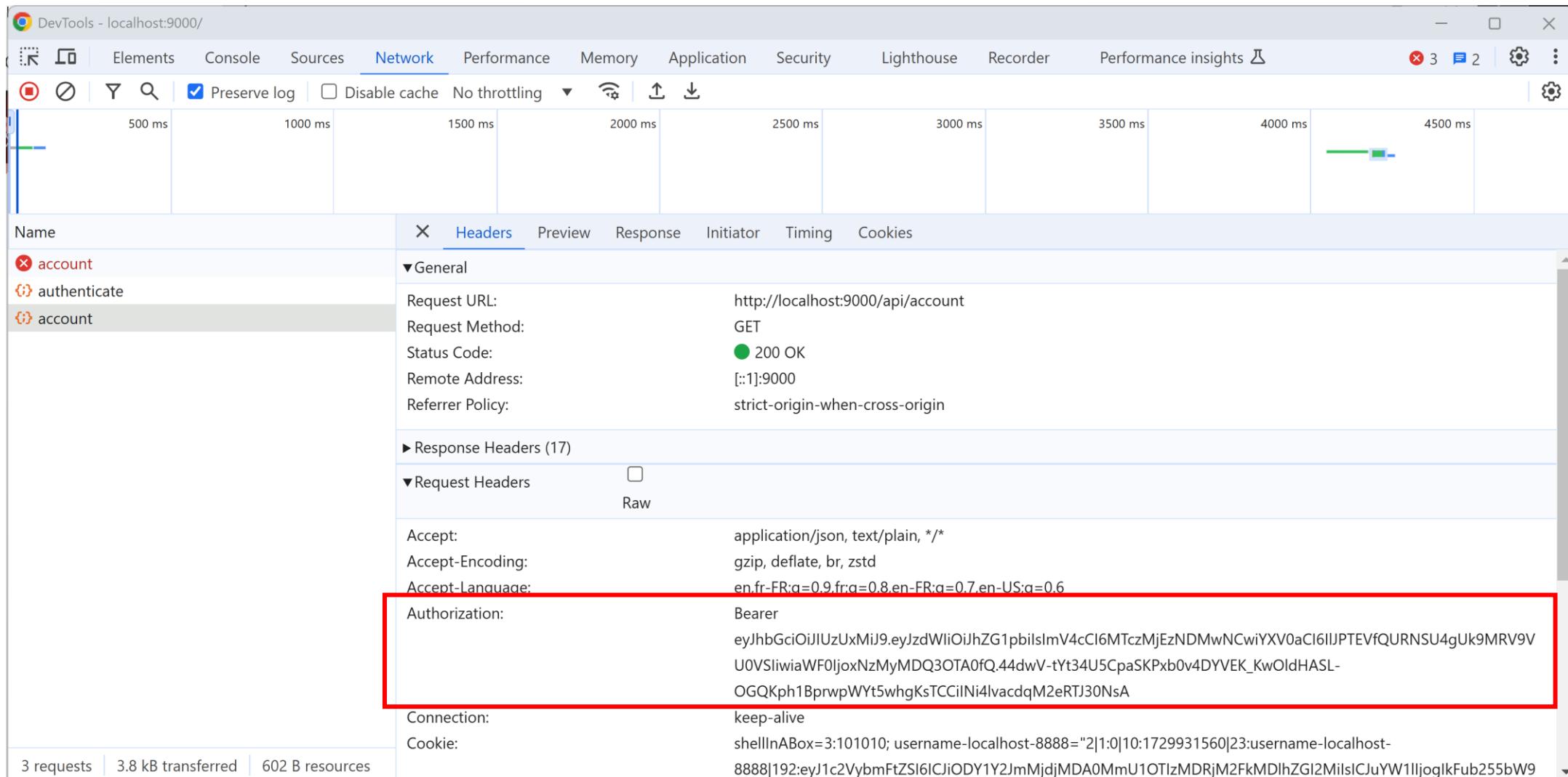
Response Headers (17):

- Accept: application/json, text/plain, */*
- Accept-Encoding: gzip, deflate, br, zstd
- Accept-Language: en,fr-FR;q=0.9,fr;q=0.8,en-FR;q=0.7,en-US;q=0.6
- Authorization: Bearer eyJhbGciOiJIUzUxMiJ9eyJzdWliOiJhZG1pbilslmV4cCl6MTczMjEzNDMwNCwiYXV0aCl6IiJPTEVfQURNSU4gUk9MRV9VU0VSlwiaWF0ljoxNzMyMDQ3OTA0fQ.44dwV-tYt34U5CpaSKPxbov4DYVEK_KwOldHASL-OGQKph1BprwpWYt5whgKsTCCiINi4lvacdqM2eRTJ30NsA
- Connection: keep-alive
- Cookie: shellInABox=3:101010; username=localhost-8888="2|1:0|10:1729931560|23:username=localhost-8888|192:eyJ1c2VybmFtZSI6ICJiODY1Y2JmMjdjMDA0MmU1OTIzM2FkMDlhZGI2MilsICJuYW1lIjogIkFub255bW9

Network Performance:

- 500 ms
- 1000 ms
- 1500 ms
- 2000 ms
- 2500 ms
- 3000 ms
- 3500 ms
- 4000 ms
- 4500 ms

Difference between Failing Request [1/3] and Success Request [3/3] (Authenticated)



The screenshot shows the Network tab in Google DevTools for a request to `http://localhost:9000/api/account`. The request was made via GET and returned a 200 OK status. The response includes various headers such as Accept, Accept-Encoding, Accept-Language, and Authorization, which is highlighted with a red box. The Authorization header contains a complex token string.

Name	Value
Request URL	<code>http://localhost:9000/api/account</code>
Request Method	GET
Status Code	200 OK
Remote Address	<code>[:1]:9000</code>
Referrer Policy	strict-origin-when-cross-origin
Accept	<code>application/json, text/plain, */*</code>
Accept-Encoding	<code>gzip, deflate, br, zstd</code>
Accept-Language	<code>en,fr-FR;q=0.9,fr;q=0.8,en-ES;q=0.7,en-US;q=0.6</code>
Authorization	<code>Bearer eyJhbGciOiJIUzUxMiJ9eyJzdWliOiJhZG1pbilsmV4cCl6MTczMjEzNDMwNCwiYXV0aCI6IiJPTEVfQURNSU4gUk9MRV9VU0VSlwiaWF0ljoxNzMyMDQ3OTA0fQ.44dwV-tYt34U5CpaSKPxbov4DYVEK_KwOldHASL-OGQKph1BprwpWYt5whgKsTCCiINi4lvacdqM2eRTJ30NsA</code>
Connection	<code>keep-alive</code>
Cookie	<code>shellInABox=3:101010; username=localhost-8888="2 1:0 10:1729931560 23:username=localhost-8888 192:eyJ1c2VybmFtZSI6ICJiODY1Y2JmMjdjMDA0MmU1OTIzMDRjM2FkMDlhZGI2MilsICJuYW1lIjogIkFub255bW9</code>

Request [3/3]: /api/account Response Body: Detailed Information on Account

The screenshot shows the Network tab in Google DevTools for a request to `localhost:9000/api/account`. The response body is displayed in a code editor-like interface with line numbers, showing a JSON object representing an account.

```
1  {
2    "id" : 1,
3    "login" : "admin",
4    "firstName" : "Administrator",
5    "lastName" : "Administrator",
6    "email" : "admin@localhost",
7    "imageUrl" : "",
8    "activated" : true,
9    "langKey" : "en",
10   "createdBy" : "system",
11   "createdDate" : null,
12   "lastModifiedBy" : "system",
13   "lastModifiedDate" : null,
14   "authorities" : [ "ROLE_USER", "ROLE_ADMIN" ]
15 }
```

Below the table, the network summary shows 3 requests, 3.8 kB transferred, and 602 B resources.

Http Request status 200.. ONLY If authenticated + authorized

Http client **always need to pass Http Header "Authorization" = "Bearer \${jwtToken}"**

=> this is for **Authentication** of the user (similar to checking its password)
The Token is a proof of identity

Then, only after, the server can check for Authorization for that calling user on a given action
=> this is "**Authorization ROLES checking**"

Example, some actions may be reserved only for Admins, some other only for normal users.
Most actions on resources depends on both the resource and the corresponding user (without ROLE).

Example: a Customer of a Bank can only view his personal "Bank Account", not others accounts !!
Even if he has role "ROLE_CUSTOMER"

NOTICE Request [2/3] ... special explicit configuration on server to allow "un-authenticated"

on the backend (java) side, "http POST /api/authenticate" is an exceptional Rest API that accepts being called with un-authenticated mode (= ".permitAll()")

Other similar endpoints are for

- getting public content files,
- health-check, monitoring, ..
- asking an email to "reset your forgotten password"
- asking an email to "register" a new account

Backend Security Config

.permitAll() for special methods

```
④ SecurityConfiguration.java x

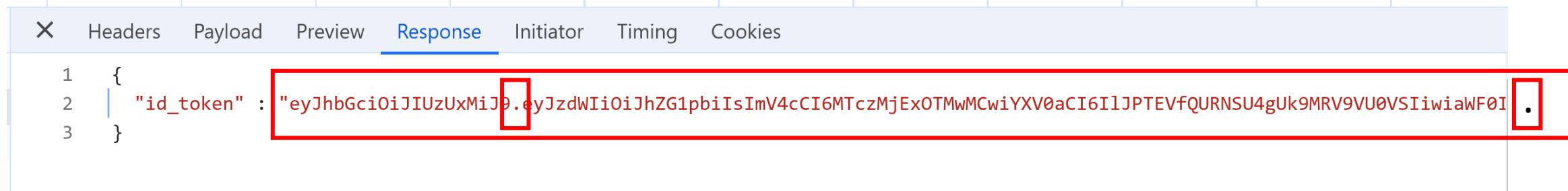
31     public class SecurityConfiguration {
48         public SecurityFilterChain filterChain(HttpSecurity http, MvcRequestMatcher.Builder mvc) throws Exception {
63             }
64             .authorizeHttpRequests( AuthorizationManagerRequestMat... authz ->
65                 // prettier-ignore
66                 authz
67                     .requestMatchers(mvc.pattern(✓ "/index.html"), mvc.pattern(✓ "/*.js"), mvc.pattern(✓ "/*.txt"))
68                     .requestMatchers(mvc.pattern(✓ "/*.ico"), mvc.pattern(✓ "/*.png"), mvc.pattern(✓ "/*.svg"), mv
69                     .requestMatchers(mvc.pattern(✓ "/app/**")).permitAll()
70                     .requestMatchers(mvc.pattern(✓ "/i18n/**")).permitAll()
71                     .requestMatchers(mvc.pattern(✓ "/content/**")).permitAll()
72                     .requestMatchers(mvc.pattern(✓ "/swagger-ui/**")).permitAll()
73                     .requestMatchers(mvc.pattern(HttpServletRequest.POST, ✓ "/api/authenticate")).permitAll()
74                     .requestMatchers(mvc.pattern(HttpServletRequest.GET, ✓ "/api/authenticate")).permitAll()
75                     .requestMatchers(mvc.pattern(✓ "/api/register")).permitAll()
76                     .requestMatchers(mvc.pattern(✓ "/api/activate")).permitAll()
77                     .requestMatchers(mvc.pattern(✓ "/api/account/reset-password/init")).permitAll()
78                     .requestMatchers(mvc.pattern(✓ "/api/account/reset-password/finish")).permitAll()
79                     .requestMatchers(mvc.pattern(✓ "/api/admin/**")).hasAuthority(AuthoritiesConstants.ADMIN)
80                     .requestMatchers(mvc.pattern(✓ "/api/**")).authenticated()
81                     .requestMatchers(mvc.pattern(✓ "/v3/api-docs/**")).hasAuthority(AuthoritiesConstants.ADMIN)
82                     .requestMatchers(mvc.pattern(✓ "/management/health")).permitAll()
83                     .requestMatchers(mvc.pattern(✓ "/management/health/**")).permitAll()
84                     .requestMatchers(mvc.pattern(✓ "/management/info")).permitAll()
85                     .requestMatchers(mvc.pattern(✓ "/management/prometheus")).permitAll()
86                     .requestMatchers(mvc.pattern(✓ "/management/**")).hasAuthority(AuthoritiesConstants.ADMIN)
87             )

```

Backend Code for Http POST /api/authenticate ... check password then create a JWT Token

```
34     @RestController
35     @RequestMapping("/api")
36     public class AuthenticateController {
37
38         @PostMapping("/authenticate")
39         public ResponseEntity<JWTToken> authorize(@Valid @RequestBody LoginVM loginVM) {
40             UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(
41                 loginVM.getUsername(),
42                 loginVM.getPassword()
43             );
44
45             Authentication authentication = authenticationManagerBuilder.getObject().authenticate(authenticationToken);
46             SecurityContextHolder.getContext().setAuthentication(authentication);
47             String jwt = this.createToken(authentication, loginVM.isRememberMe());
48             HttpHeaders httpHeaders = new HttpHeaders();
49             httpHeaders.setBearerAuth(jwt);
50             return new ResponseEntity<>(new JWTToken(jwt), httpHeaders, HttpStatus.OK);
51         }
52     }
```

Analysis of a JWT Token



The screenshot shows a browser's developer tools Network tab with the Response tab selected. The response body contains the following JSON:

```
1 {  
2   "id_token" : "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbilIsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IiJPTEVfQURNSU4gUk9MRV9VU0VSIiwiaWF0I  
3 }
```

The entire JSON object is highlighted with a red box.

JWT Token = 3 parts separated by (2) ":"

part 1,2 are only made of base64 characters (no ".")

part3 is made of ascii

eyJhbGciOiJIUzUxMiJ9

.

eyJzdWIiOiJhZG1pbilIsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IiJPTEVfQURNSU4gUk9MRV9VU0VSIiwiaWF0IjoxNzMyM

.

G9uNF_VPgAjUsO86TInRo-ihqvB62QI_6vSfr6hr_f2BQPhofB7D6XQLqe6O001lhQCHCvmuiJOjPh8ZbELuQ

JWT Token Part 1: token sign algorithm

decoding part 1 as base 64

```
$ echo eyJhbGciOiJIUzUxMiJ9 | base64 -d  
{"alg":"HS512"}
```

JWT Token Part 2: token data

```
$ echo
```

```
eyJzdWliOiJhZG1pbilsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IJPTEVfQURNSU4gUk9MRV9  
VU0VSliwiaWF0IjoxNzMyMDMyOTAwfQ== | base64 -d
```

```
{"sub":"admin","exp":1732119300,"auth":"ROLE_ADMIN ROLE_USER","iat":1732032900}
```



subject
(=username)
bearer of this token



expiry time
in seconds since epoch

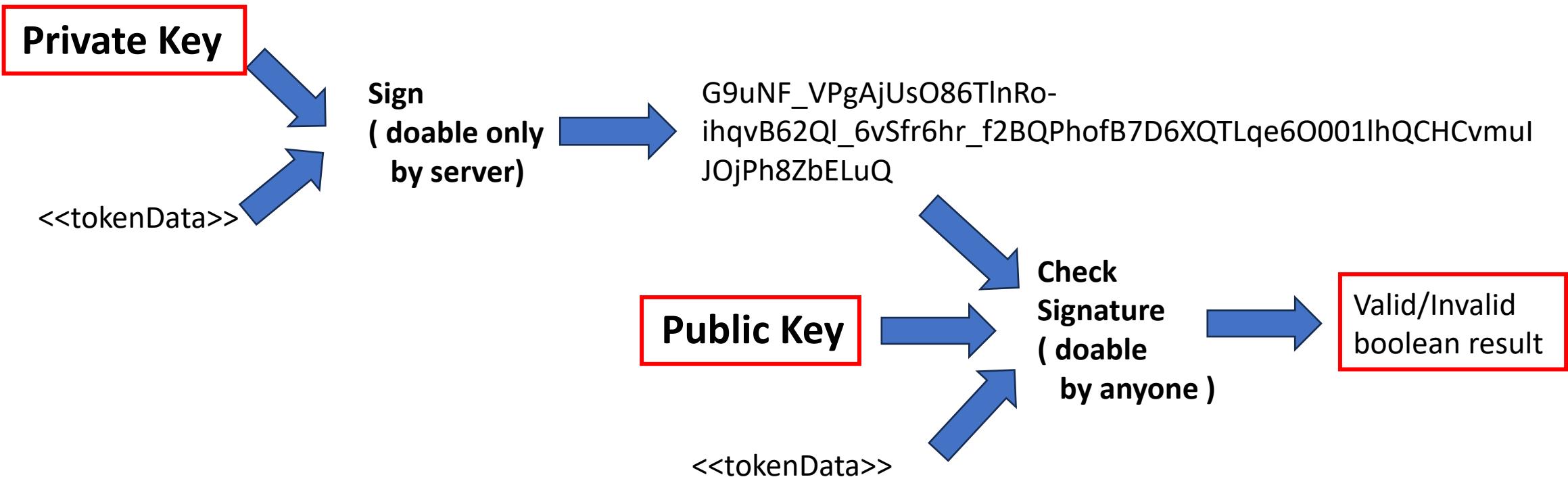


authorized Roles



time
in seconds
since epoch

JWT Token Part 3: Cryptographic Signature



Webapp keep JWT Token + Info

on return of Http POST /api/authenticate, WebApp keep informations:

- **username** (=subject) currently authenticated
- **JWT Token**



- To display "already signed-in" / "anonymous" feedback
- To authenticate all following Http Rest API Requests to backend server

then after Http GET /api/account, WebApp keep other informations:

- **list of roles**
- **detailed Account information**



- To show/hide features that could not be called by client (server would reject anyway)
- To translate website in preferred user langage, to display firstName, lastName, email ...

Login Page Component Html

a classical Form,
with 3 Fields

click to submit
=> call 'login()'

```
<> login.component.html <  
1   <div>  
2     <div class="d-flex justify-content-center">  
3       <div class="col-lg-6 col-md-8 col-sm-10">  
9         </>  
10        <form class="form" (ngSubmit)="login()" [formGroup]="loginForm">  
11          <div class="mb-3">  
12            <label class="username-label" for="username" jhiTranslate="global.form.username.label">Username</label>  
13            <input ...>  
14          </div>  
15  
24  
25          <div class="mb-3">  
26            <label for="password" jhiTranslate="login.form.password">Password</label>  
27            <input ...>  
28          </div>  
29  
37  
38          <div class="form-check">  
39            <label class="form-check-label" for="rememberMe" ...>  
40          </div>  
41  
44  
45          <button type="submit" class="btn btn-primary" data-cy="submit" jhiTranslate="login.form.button">Sign i  
46        </form>
```

LoginComponent TypeScript

The code get the 3 values
{username,password,rememberMe} from the Form, and call the service

At the end, the Login page is closed, navigate back to Home page

```
TS login.component.ts ×  
15  export default class LoginComponent implements OnInit, AfterViewInit {  
42  
43    login(): void { Show usages  
44      this.loginService.login(this.loginForm.getRawValue()).subscribe({  
45        ↑  
46          next: () :void => {  
47            this.authenticationError.set(false);  
48            if (!this.router.getCurrentNavigation()) {  
49              // There were no routing during login (eg from navigationToStoredUrl)  
50              this.router.navigate(['']);  
51            }  
52            ↑  
53          },  
54        error: () :void => this.authenticationError.set(true),  
55      });  
56    }  
57  }  
58}
```

login() / logout() High Level Facade Code

```
ts login.service.ts x
1  > import ...
9
10 @Injectable({ providedIn: 'root' }) Show usages
11 export class LoginService {
12   private readonly accountService = inject(AccountService);
13   private readonly authServiceProvider = inject(AuthServiceProvider);
14
15   login(credentials: Login): Observable<Account | null> { Show usages
16     return this.authServiceProvider.login(credentials).pipe(mergeMap(() => this.accountService.identity(true)));
17   }
18
19   logout(): void { Show usages
20     this.authServiceProvider.logout().subscribe({ complete: () => this.accountService.authenticate(null) });
21   }
22 }
```

High-Level
method
called from
component

Low Level Http Code + Response

Low-Level
Http POST
Request

```
ts auth-jwt.service.ts x

15  export class AuthServerProvider {
16    ...
17    private readonly stateStorageService : StateStorageService = inject(StateStorageService);
18    private readonly applicationConfigService : ApplicationConfigService = inject(ApplicationConfigService);
19
20    getToken(): string { Show usages
21      return this.stateStorageService.getAuthenticationToken() ?? '';
22    }
23
24    login(credentials: Login): Observable<void> { Show usages
25      return this.http
26        .post<JwtToken>(this.applicationConfigService.getEndpointFor('api/authenticate'), credentials)
27        .pipe(map(response : JwtToken => this.authenticateSuccess(response, credentials.rememberMe)));
28    }
29
30    logout(): Observable<void> { Show usages
31      return new Observable(observer : Subscriber<void> => {
32        this.stateStorageService.clearAuthenticationToken();
33        observer.complete();
34      });
35    }
36
37    private authenticateSuccess(response: JwtToken, rememberMe: boolean): void { Show usages
38      this.stateStorageService.storeAuthenticationToken(response.id_token, rememberMe);
39    }

```

Callback to Save Response

save Json "id_token" to Storage

Storing JWT Token

if selected
"rememberMe",

then save to
LocalStorage

else to temporary
SessionStorage

```
TS state-storage.service.ts ×  
4   export class StateStorageService {  
21  
22     storeAuthenticationToken(authenticationToken: string, rememberMe: boolean): void {  
23       authenticationToken = JSON.stringify(authenticationToken);  
24       this.clearAuthenticationToken();  
25       if (rememberMe) {  
26         localStorage.setItem(this.authenticationKey, authenticationToken);  
27       } else {  
28         sessionStorage.setItem(this.authenticationKey, authenticationToken);  
29       }  
30     }  
31   }
```

Stored JWT Available for later Http Calls

```
ts state-storage.service.ts × :  
4  export class StateStorageService {  
31  
32      getAuthenticationToken(): string | null { Show usages  
33          const authenticationToken : string | null = localStorage.getItem(this.authenticationKey) ?? sessionStorage.getItem(this.authenticationKey);  
34          return authenticationToken ? (JSON.parse(authenticationToken) as string | null) : authenticationToken;  
35      }  
    ↴ ↵
```

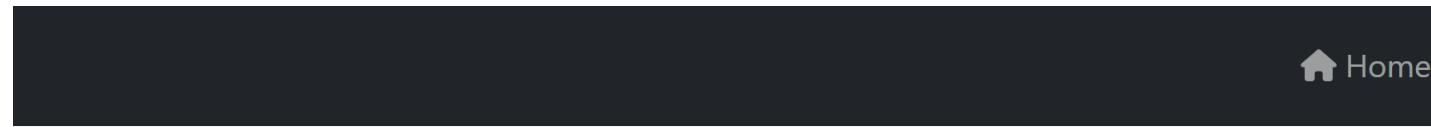
The code retries both local and session Storage,
then reparse from JSON to get JWT Bearer Token value

Viewing SessionStorage with Chrome Devtools

The screenshot shows the Chrome DevTools interface with the Application tab selected (indicated by a red box). The left sidebar lists storage types: Manifest, Service workers, Storage, Local storage (under http://localhost:9000), Session storage (under http://localhost:9000, also highlighted with a red box), IndexedDB, Cookies, Private state tokens, Interest groups, Shared storage, Cache storage, and Storage buckets. The right panel displays the contents of the Session storage for the origin http://localhost:9000. It shows a table with one entry: a key "jhi-authenticationToken" and its value, which is a long JWT token.

Key	Value
jhi-authenticationToken	"eyJhbGciOiJIUzUxMiJ9eyJzdWIiOiJhZG1pbilIsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IlJPTEVfQURNSU4gUk9MRV9VU0VSIiwiaWF0IjoxNzMyMDMyOTAwfQ.G"

Sign-in with "rememberMe" option



Sign in

Username

Password



Remember me

[Did you forget your password?](#)

You don't have an account yet? [Register a new account](#)

rememberMe => JWT in LocalStorage

The screenshot shows the Google Chrome DevTools Application tab interface. The left sidebar lists storage types: Manifest, Service workers, Storage, Session storage, IndexedDB, Cookies, and Private state tokens. The Storage section is expanded, and the Local storage item under http://localhost:9000 is selected and highlighted with a red box. The main panel displays the contents of the Local storage for the origin http://localhost:9000. It shows two items: 'jhi-authenticationToken' with a value of a long JWT string, and 'loglevel:webpack-dev-server' with a value of 'INFO'. A red box highlights the 'jhi-authenticationToken' key in the list.

DevTools - localhost:9000/

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights x 2 2 ⋮

Application

- Manifest
- Service workers
- Storage

Storage

- Local storage
- Session storage
- IndexedDB
- Cookies
- Private state tokens

http://localhost:9000

Origin http://localhost:9000

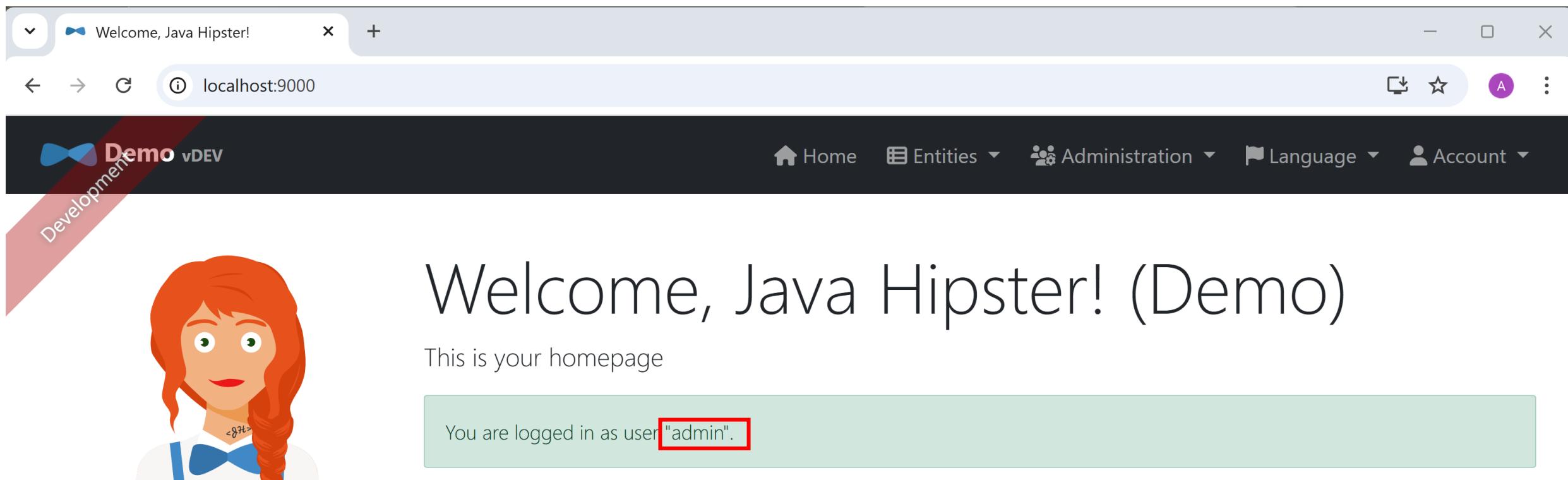
Key	Value
jhi-authenticationToken	"eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbilsImV4cCI6MTczNDYzMzc1NCwiYXV0...
loglevel:webpack-dev-server	INFO

LocalStorage :

resist to a Close Browser Tab + Re-Open

(!= SessionStorage, purged)

Closed + re-Opened Tab => still "logged-in" (even resist to Quit Browser application, relaunch it, reopen Tab)



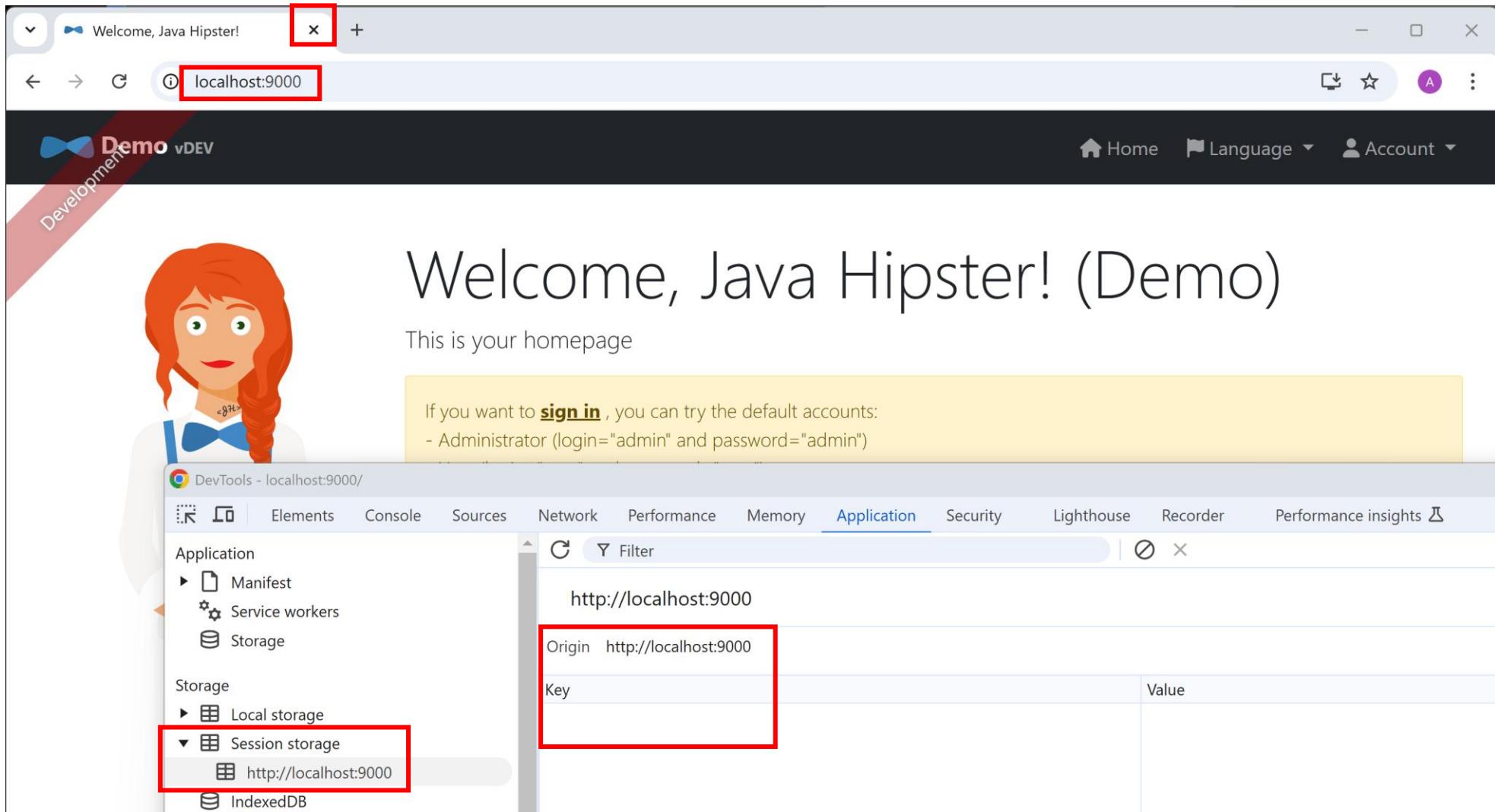
The screenshot shows a web browser window with the following details:

- Tab Bar:** Shows a single tab titled "Welcome, Java Hipster!".
- Address Bar:** Displays the URL "localhost:9000".
- Header:** A dark header bar with a red diagonal banner on the left containing the text "Development". It includes navigation icons, a search bar, and user account dropdowns for "Home", "Entities", "Administration", "Language", and "Account".
- Content Area:** The main content displays a cartoon character with orange hair and a blue bow tie. The text "Welcome, Java Hipster! (Demo)" is prominently displayed. Below it, the message "This is your homepage" is shown. A green callout box at the bottom contains the text "You are logged in as user **"admin"**".

SessionStorage => auto purge on Close Tab

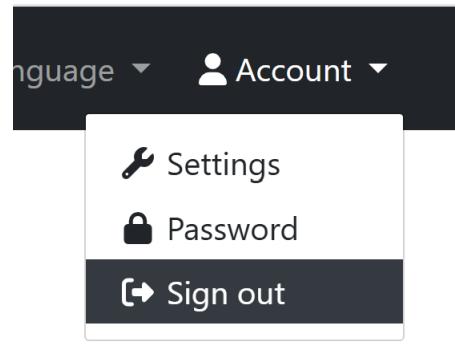
Close Tab => even Chrome Dev Tools panel is Closed

Re-Open new tab with same URL => then re-open Chrome Dev Tools => SessionStorage was cleared



On Sign-Out

=> All Security information must be cleared!
(Local + Session)

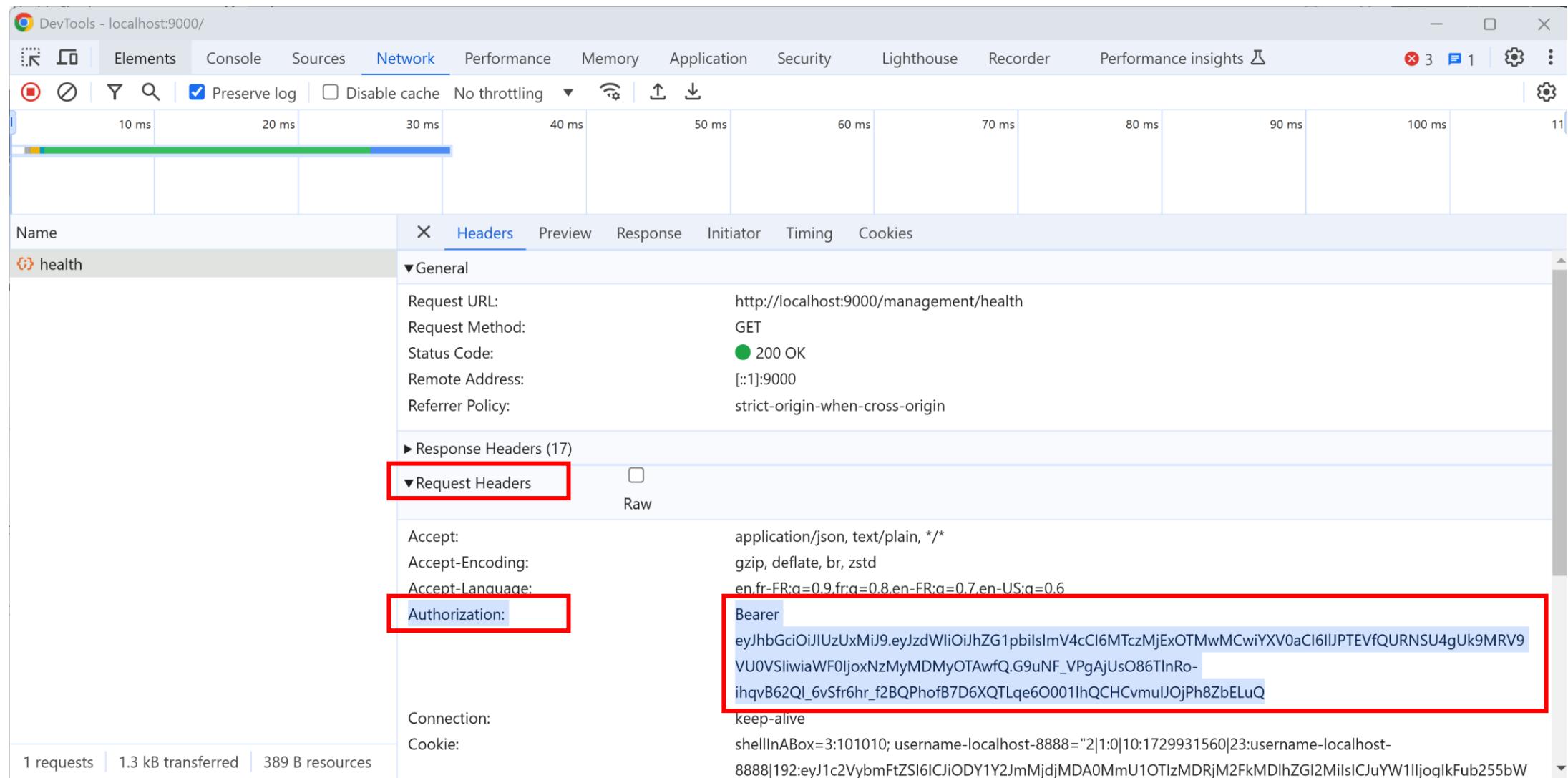


```
37     clearAuthenticationToken(): void { Show usages
38       sessionStorage.removeItem(this.authenticationKey);
39       localStorage.removeItem(this.authenticationKey);
40   }
```

A screenshot of the Chrome DevTools Application tab. The tab bar at the top includes Elements, Console, Sources, Network, Performance, Memory, Application (which is selected and highlighted in blue), Security, Lighthouse, and Recorder. On the left, a sidebar lists sections like Application, Storage, and Cookies. Under Storage, 'Local storage' and 'Session storage' are expanded, showing entries for 'http://localhost:9000'. The 'Session storage' section has an entry for 'http://localhost:9000' highlighted with a light gray background. The main panel shows a table with 'Key' and 'Value' columns. At the bottom of the table, there is a large red button labeled 'Clear storage'.

Passing JWT back in Http Rest API Calls

Debugging Authenticated Rest API Calls



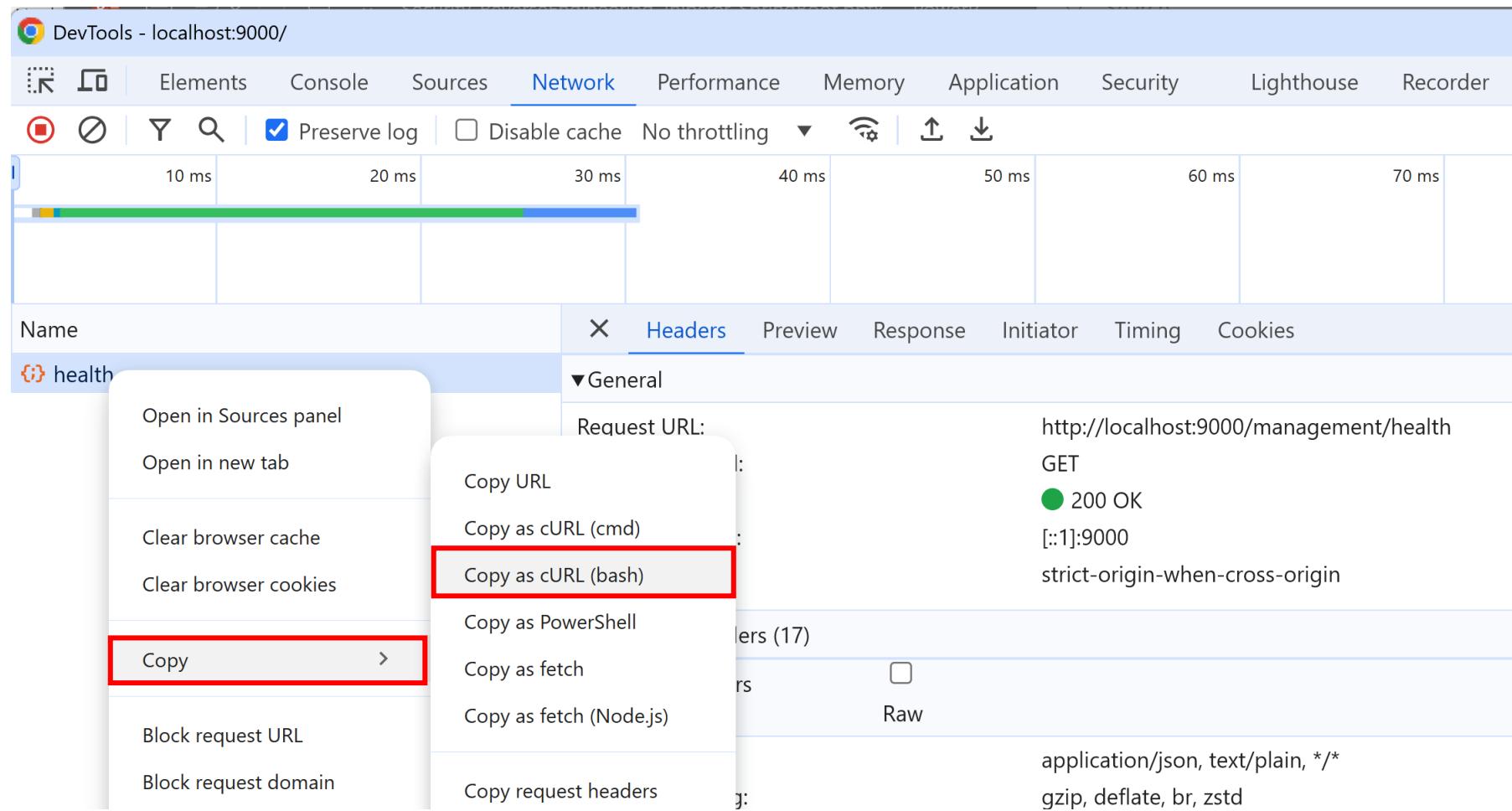
The screenshot shows the Network tab in Google DevTools for a request to `http://localhost:9000/management/health`. The request was made via GET and received a 200 OK status. The Request Headers section is expanded, showing the `Authorization` header highlighted with a red box. The value of the `Authorization` header is `Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbilsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IjUPTEVfQURNSU4gUk9MRV9VU0VSlisiaWF0ljoxNzMyMDMyOTAwfQ.G9uNF_VPgAjUsO86TInRo-ihqvB62Ql_6vSfr6hr_f2BQPhofB7D6XQLqe6O001lhQCHCvmulJOjPh8ZbELuQ`, which is a standard JWT token.

Name	Value
Request URL	<code>http://localhost:9000/management/health</code>
Request Method	GET
Status Code	200 OK
Remote Address	<code>[:1]:9000</code>
Referrer Policy	strict-origin-when-cross-origin
Accept	<code>application/json, text/plain, */*</code>
Accept-Encoding	<code>gzip, deflate, br, zstd</code>
Accept-Language	<code>en,fr-FR;q=0.9,fr;q=0.8,en-FR;q=0.7,en-US;q=0.6</code>
Authorization	<code>Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWliOiJhZG1pbilsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IjUPTEVfQURNSU4gUk9MRV9VU0VSlisiaWF0ljoxNzMyMDMyOTAwfQ.G9uNF_VPgAjUsO86TInRo-ihqvB62Ql_6vSfr6hr_f2BQPhofB7D6XQLqe6O001lhQCHCvmulJOjPh8ZbELuQ</code>
Connection	<code>keep-alive</code>
Cookie	<code>shellInABox=3:101010; username=localhost-8888="2 1:0 10:1729931560 23:username=localhost-8888 192:eyJ1c2VybmFtZSI6ICJiODY1Y2JmMjdjMDA0MmU1OTIzMjM2FkMDlhZG12MilsICJuYW1ljoqlkFub255bW</code>

Header "Authorization: Bearer \${token}" ...

All Rest API calls are Authenticated using Http Header "Authorization",
containing a Bearer token

Minimalist Check using Curl



Full Curl Command ... can simplify

Check Minimalist Curl Command

```
token=..."  
curl 'http://localhost:9000/management/health' -H "Authorization: Bearer ${token}"
```

```
arnaud@DesktopArnaud /cygdrive/c/arn/devPerso  
$ token="eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbjIsImV4cCI6MTczMjExOTMwMCwiYXV0aCI6IlJPTEVfQURNSLTlnRo-ihqvB62Ql_6vSfr6hr_f2BQPhofB7D6XQLqe60001lhQCHCvmuIJ0jPh8ZbELuQ"  
  
arnaud@DesktopArnaud /cygdrive/c/arn/devPerso  
$ curl 'http://localhost:9000/management/health' -H "Authorization: Bearer ${token}"  
{"status": "UP", "components": {"db": {"status": "UP", "details": {"database": "H2", "validationQuery": "SELECT 1", "lastSync": "2022-01-19T19:52:08.102Z", "threshold": 10485760, "path": "C:\\web\\demo-jhipster\\demo\\."}, "status": "UP"}, "readinessState": {"status": "UP"}}, "groups": ["liveness", "readiness"]}
```

HttpClient Code in Angular : NO extra Code for Header !

```
TS health.service.ts ×  
1  > import ...  
7  
8  @Injectable({ providedIn: 'root' }) Show usages  
9  export class HealthService {  
10    private readonly http :HttpClient = inject(HttpClient);  
11    private readonly applicationConfigService :ApplicationConfigService = inject(ApplicationConfigService);  
12  
13    checkHealth(): Observable<Health> { Show usages  
14      return this.http.get<Health>(this.applicationConfigService.getEndpointFor('management/health'));  
15    }  
}
```

HttpClient ... using Http Interceptors

```
ts app.config.ts ×
45
46     export const appConfig: ApplicationConfig = { Show usages
47     @↑   providers: [
48         provideRouter(routes, ...routerFeatures),
49         importProvidersFrom(BrowserModule),
50         // Set this to true to enable service worker (PWA)
51         @↑   importProvidersFrom(ServiceWorkerModule.register('ngsw-worker.js', { enabled: false })),
52         importProvidersFrom(TranslationModule),
53         @↓   provideHttpClient(withInterceptorsFromDi()), // This line is highlighted with a red box
54         Title,
55         @↑   { provide: LOCALE_ID, useValue: 'en' },
56         @↑   { provide: NgbDateAdapter, useClass: NgbDateDayjsAdapter },
57         httpInterceptorProviders, // This line is highlighted with a red box
58         @↑   { provide: TitleStrategy, useClass: AppPageTitleStrategy },
59         // jhipster-needle-angular-add-module JHipster will add new module here
60     ],
61 };
```

List of Dependency Injected HttpInterceptors

Interceptor for
adding Http Header

other Interceptors,
for renewing,
showing errors,
showing notifications..

```
ts index.ts ×  
1 import { HTTP_INTERCEPTORS } from '@angular/common/http';  
2  
3 import { AuthInterceptor } from 'app/core/interceptor/auth.interceptor';  
4 import { AuthExpiredInterceptor } from 'app/core/interceptor/auth-expired.interceptor';  
5 import { ErrorHandlerInterceptor } from 'app/core/interceptor/error-handler.interceptor';  
6 import { NotificationInterceptor } from 'app/core/interceptor/notification.interceptor';  
7  
8 export const httpInterceptorProviders :{ provide: InjectionToken<readonly Http... } = [ Show usages  
9 {  
10   provide: HTTP_INTERCEPTORS,  
11   useClass: AuthInterceptor,  
12   multi: true,  
13 },  
14 {  
15   provide: HTTP_INTERCEPTORS,  
16   useClass: AuthExpiredInterceptor,  
17   multi: true,  
18 },  
19 {  
20   provide: HTTP_INTERCEPTORS,  
21   useClass: ErrorHandlerInterceptor,  
22   multi: true,  
23 },  
24 {  
25   provide: HTTP_INTERCEPTORS,  
26   useClass: NotificationInterceptor,  
27   multi: true,  
28 },  
29 ];
```

AuthInterceptor

```
TS auth.interceptor.ts ✘ ...  
1 > import ...  
7  
8 @Injectable() Show usages  
9 export class AuthInterceptor implements HttpInterceptor {  
10   private readonly stateStorageService : StateStorageService = inject(StateStorageService);  
11   private readonly applicationConfigService : ApplicationConfigService = inject(ApplicationConfigService);  
12  
13 ⓘ intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> { no usages  
14     const serverApiUrl : string = this.applicationConfigService.getEndpointFor('');  
15     if (!request.url || (request.url.startsWith('http') && !(serverApiUrl && request.url.startsWith(serverApiUrl))))  
16       return next.handle(request);  
17     }  
18  
19     const token: string | null = this.stateStorageService.getAuthenticationToken();  
20     if (token) {  
21       request = request.clone({  
22         setHeaders: {  
23           Authorization: `Bearer ${token}`,  
24         },  
25       });  
26     }  
27     return next.handle(request);  
28 }
```

do not add token
for http urls
other than backend
Api server

add Bearer
JWT Token
for backend calls

Displaying Angular Features
Conditionaly on Authenticated / Authorized

@if (account()) { .. } @else { .. }

```
<> home.component.html <

1   <div class="row">
6     <div class="col-md-9">
11    <div>
12      @if (account() !== null) {
13        <div class="alert alert-success">
14          @if (account(); as accountRef) {
15            <span id="home-logged-message" jhiTranslate="home.logged.message" [translateValue]="">
16              You are logged in as user "{{ accountRef.login }}".</span>
17            >
18          }
19        </div>
20      } @else {
21        <div class="alert alert-warning">
22          <span jhiTranslate="global.messages.info.authenticated.prefix">If you want to </
23          <a class="alert-link" (click)="login()" jhiTranslate="global.messages.info.authenticated.button">
24            sign in</a>
25          <span jhiTranslate="global.messages.info.authenticated.suffix">
26            >, you can try the default accounts:<br />- Administrator (login="admin" and p
27            (login="user" and password="user")).</span>
28          >
29        </div>
```

account() signal computed from accountService.getAuthenticationState()

```
ts home.component.ts ×
14 ① styleUrl: './home.component.scss',
15 ② imports: [SharedModule, RouterModule],
16 }
17 export default class HomeComponent implements OnInit, OnDestroy {
18   account: WritableSignal<Account | null> = signal<Account | null>(null);
19
20   private readonly destroy$ : Subject<void> = new Subject<void>();
21
22   private readonly accountService : AccountService = inject(AccountService);
23   private readonly router : Router = inject(Router);
24
25 ③ ngOnInit(): void { Show usages
26   this.accountService
27     .getAuthenticationState()
28     .pipe(takeUntil(this.destroy$))
29     .subscribe(account : Account | null => this.account.set(account));
30 }
```

AccountService.authenticationState ... computed from http GET /api/account response

```
TS account.service.ts x
1  > import ...
11
12  @Injectable({ providedIn: 'root' }) Show usages
13  export class AccountService {
14    private readonly userIdentity = signal<Account | null>(null);
15    private readonly authenticationState = new ReplaySubject<Account | null>(1);
16    private accountCache$?: Observable<Account> | null;
```

see next
where authenticate()
method is called

```
28  authenticate(identity: Account | null): void { Show usages
29    this.userIdentity.set(identity);
30    this.authenticationState.next(this.userIdentity());
31    if (!identity) {
32      this.accountCache$ = null;
33    }
34 }
```

only location where
authenticationState is "set"

Caching Detailed Account Info of current User (Bearer of JWT)

... this code is caching
the fetch response
of "http GET /api/account"

= get detailed firstname,
lastname, email, etc...
for the current authenticated
(JWT) user

```
51     identity(force?: boolean): Observable<Account | null> {
52       if (!this.accountCache$ || force) {
53         this.accountCache$ = this.fetch().pipe(
54           tap((account: Account) : void => {
55             this.authenticate(account);
56           ...
57           ...
58           ...
59           ...
60           ...
61           ...
62           ...
63           ...
64           ...
65           ...
66           ...
67           ...
68           ...
69           ...
70         })
71       }
72     }
73   }
74
75   private fetch(): Observable<Account> { Show usages
76     return this.http.get<Account>(this.applicationConfigService.getEndpointFor('api/account'));
77   }
78 }
```

http GET /api/account (caching called just after POST /api/authenticate)

The screenshot shows the Network tab in Google Chrome DevTools. A single request for `/api/account` is listed, which was triggered by a previous `POST /api/authenticate` request. The request details are as follows:

- Name:** account
- Request URL:** `http://localhost:9000/api/account`
- Request Method:** GET
- Status Code:** 200 OK
- Remote Address:** `[::1]:9000`
- Referrer Policy:** strict-origin-when-cross-origin

Below the request details, the response headers are listed:

- Accept:** application/json, text/plain, */*
- Accept-Encoding:** gzip, deflate, br, zstd
- Accept-Language:** en,fr-FR;q=0.9,fr;q=0.8,en-FR;q=0.7,en-US;q=0.6
- Authorization:** Bearer eyJhbGciOiJIUzUxMiJ9eyJzdWIiOiJhZG1pbilsImV4cCl6MTczMjEzM**DA1OSwiYXV0aCI6IJPTEVfQURNSU4gUk9MRV9VU0VSlwiaWF0ljoxNzMyMDQzMjU5fQ.LjE2aT9dBCNiipiY-qZIBk-_0TrtLa_KHhrvoA0JhcZ6R_vdgHbt4Mk8POmBueaj3PiwMloj4_APelO0ld7dDw**
- Connection:** keep-alive
- Cookie:** shellInABox=3:101010; username-localhost-8888="2|1:0|10:1729931560|23:username-localhost-8888|192:eyJ1c2VybmFtZSI6ICJiODY1Y2JmMjdjMDA0MmU1OTIzM**DRjM2FkMDlhZGI2MilsICJuYW1ljljoglkFub255bW**

At the bottom of the DevTools interface, the summary indicates 3 requests, 3.8 kB transferred, and 602 B resources.

Response of http GET /api/account : detailed Information

The screenshot shows the Network tab in Google Chrome DevTools. A red box highlights the 'account' entry in the list. Another red box highlights the 'Response' tab in the header bar. The response body is displayed as a JSON object:

```
1 {  
2   "id" : 1,  
3   "login" : "admin",  
4   "firstName" : "Administrator",  
5   "lastName" : "Administrator",  
6   "email" : "admin@localhost",  
7   "imageUrl" : "",  
8   "activated" : true,  
9   "langKey" : "en",  
10  "createdBy" : "system",  
11  "createdDate" : null,  
12  "lastModifiedBy" : "system",  
13  "lastModifiedDate" : null,  
14  "authorities" : [ "ROLE_USER", "ROLE_ADMIN" ]  
15 }
```

Notice "authorities" :
[ROLE_ADMIN, ROLE_USER, ..]

```
13     |   "lastmodifiedate" : null,  
14     |   "authorities" : [ "ROLE_USER", "ROLE_ADMIN" ]  
15 }
```

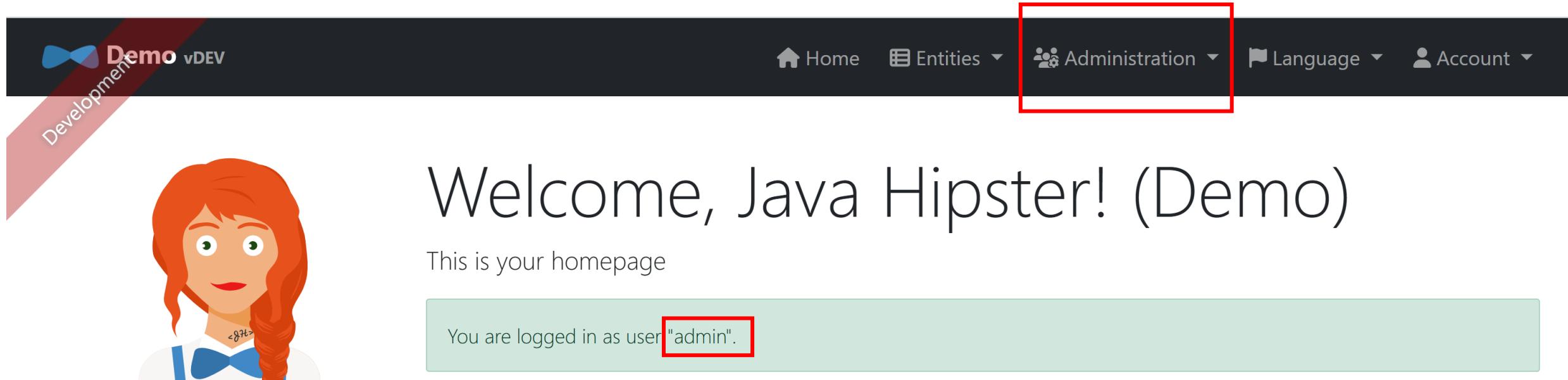
"authorities" = synonym of **"Roles"** or **"Authorizations"**
= listed of **"granted things"**

redundant with info already in JWT Token,
but the "Authorization" code works independently of the "Authentication".
Could plug authentication with OAuth2, PIN card, Face recognition, etc,
=> the code for "authorities" is separated

hasAnyAuthority : check if authorized for a Role

```
40 hasAnyAuthority(authorities: string[] | string): boolean { Show usages
41   const userIdentity :Account | null = this.userIdentity();
42   if (!userIdentity) {
43     return false;
44   }
45   if (!Array.isArray(authorities)) {
46     authorities = [authorities];
47   }
48   return userIdentity.authorities.some((authority: string) :boolean => authorities.includes(authority));
49 }
```

Showing Menu "Administration" when having ROLE_ADMIN



The screenshot shows a web application interface. At the top, there is a dark header bar with the following elements from left to right: a blue ribbon icon, the text "Demo vDEV", a "Development" badge, a "Home" link with a house icon, an "Entities" link with a grid icon, an "Administration" link with a user icon, a "Language" link with a flag icon, and an "Account" link with a person icon. The "Administration" link is highlighted with a red rectangular box. Below the header, the main content area has a light gray background. On the left side, there is a cartoon illustration of a character with orange hair and a bow tie. The central part of the page displays the text "Welcome, Java Hipster! (Demo)" in a large, bold font. Below this, a smaller text says "This is your homepage". At the bottom of the page, there is a green horizontal bar containing the text "You are logged in as user `admin`". The word "admin" is also enclosed in a red rectangular box.

Hiding Menu "Administration" when logged-in, but NOT as an admin



The screenshot shows a web application's homepage. At the top, there is a dark header bar with the following elements from left to right: a blue ribbon icon, the text "Development Demo vDEV", a "Home" link with a house icon, an "Entities" link with a grid icon, a "Language" link with a flag icon, and an "Account" link with a person icon. A red rectangular box highlights the "Language" link. Below the header, on the left, is a cartoon illustration of a character with orange hair in a white shirt and blue bow tie. To the right of the illustration, the text "Welcome, Java Hipster! (Demo)" is displayed in large font. Underneath this, the message "This is your homepage" is shown. At the bottom of the page, within a green box, is the text "You are logged in as user **"user"**". A red rectangular box highlights the word "user".

Hiding Menu "Administration" when not logged-in

The screenshot shows a web application's homepage. At the top, there is a dark header bar with the following elements from left to right: a blue ribbon icon, the text "Demo vDEV", a red diagonal banner with the word "Development" in white, a house icon labeled "Home", a flag icon labeled "Language" with a dropdown arrow, and a user icon labeled "Account" with a dropdown arrow. A red rectangular box highlights the "Language" menu item. Below the header, there is a large illustration of a woman with orange hair in a braid, wearing a white shirt and a blue bow tie. To her right, the text "Welcome, Java Hipster! (Demo)" is displayed in a large, sans-serif font. Underneath this, a smaller text says "This is your homepage". Below the illustration and text, there is a yellow callout box containing the following content:

If you want to [sign in](#), you can try the default accounts:

- Administrator (login="admin" and password="admin")
- User (login="user" and password="user").

At the bottom of the page, another yellow callout box contains the text: "You don't have an account yet? [Register a new account](#)".

Showing/Hiding Features based on Role

navbar.component.html

```
1  <nav data-cy="navbar" class="navbar navbar-dark navbar-expand-md bg-dark">
2    <div class="container-fluid">
20      <div class="navbar-collapse collapse" id="navbarResponsive" [ngbCollapse]="isNavigationOpen">
21        <ul class="navbar-nav ms-auto">
22          <li
23            </li>
24          <li
25            </li>
26          <li
27            *jhiHasAnyAuthority="'ROLE_ADMIN'">
28            ngbDropdown
29            class="nav-item dropdown pointer"
30            display="dynamic"
31            routerLinkActive="active"
32            [routerLinkActiveOptions]:"{ exact: true }"
33          >
34            <a class="nav-link dropdown-toggle" ngbDropdownToggle href="javascript:void(0)">
35              <span>
36                <fa-icon icon="users-cog"></fa-icon>
37                <span jhiTranslate="global.menu.admin.main">Administration</span>
38              </span>
39            </a>
40            <ul class="dropdown-menu" ngbDropdownMenu aria-labelledby="admin-menu">
41              <li>
42                <a
43                  class="dropdown-item"
44                  routerLink="/authority">
45                    <span jhiTranslate="global.menu.common.action">Authorities</span>
46                  </a>
47                </li>
48              </ul>
49            </li>
50          <li
51            </li>
52        </ul>
53      </div>
54    </div>
55  </nav>
```

The menu
"Administration"
is visible only if user
has ROLE_ADMIN

Code for Directive *jhiHasAnyAuthority=".."

compute if current user account has role

if has role, then display the element
else hide the element

```
ts has-any-authority.directive.ts ×

16  @Directive({ Show usages
17  standalone: true,
18  selector: '[jhiHasAnyAuthority]',
19 })
20  export default class HasAnyAuthorityDirective {
21  public authorities : InputSignal<string | string[]> = input<string | string[]>([], { alias: 'jhiHasAnyAuthority' }

23  private readonly templateRef : TemplateRef<any> = inject(TemplateRef<any>);
24  private readonly viewContainerRef : ViewContainerRef = inject(ViewContainerRef);

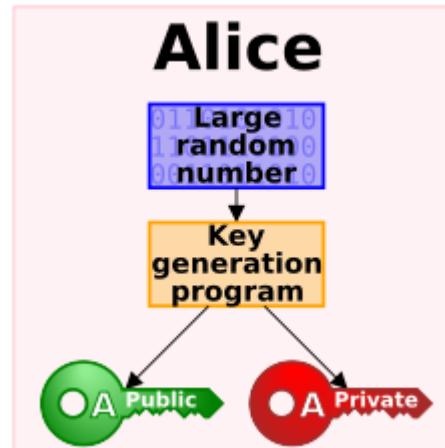
26  constructor() { no usages
27    const accountService : AccountService = inject(AccountService);
28    const currentAccount : Signal<Account | null> = accountService.trackCurrentAccount();
29    const hasPermission : Signal<boolean | undefined> = computed(
30      () : boolean | undefined => currentAccount()?.authorities &&
31                                accountService.hasAnyAuthority(this.authorities()));

33  effect(
34    () : void => {
35      if (hasPermission()) {
36        this.viewContainerRef.createEmbeddedView(this.templateRef);
37      } else {
38        this.viewContainerRef.clear();
39      }
40    }
41  )
```

Network & Infrastructure Deployments Options

Reminder: Public/Private Asymmetric Cryptography

https://en.wikipedia.org/wiki/Public-key_cryptography



- Only the server can sign with the **Private Key**
- Everybody can check the signature with the **Public Key**

Private Key should be stored carefully on servers and protected.

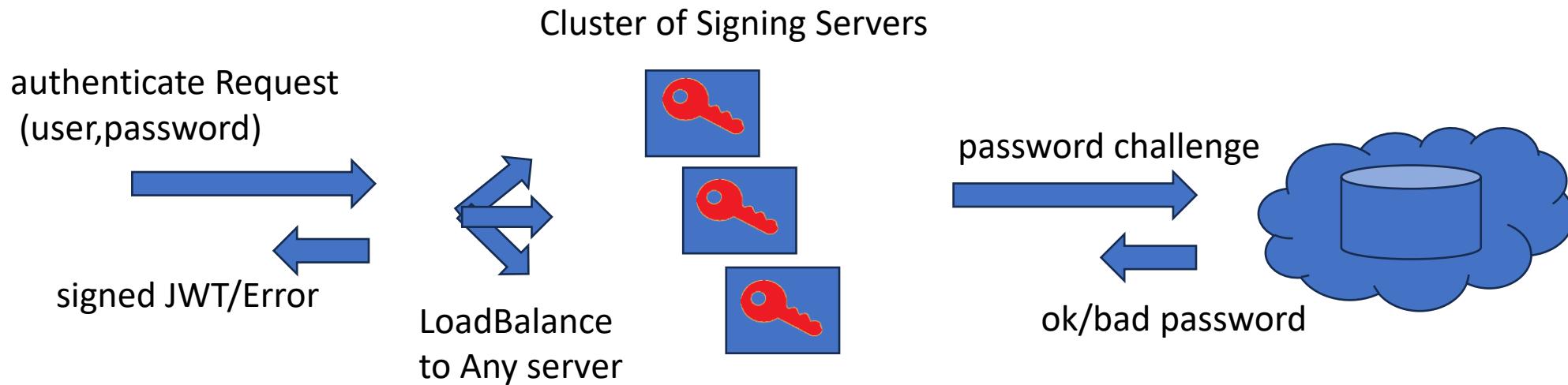
Public Key can be broadcasted.

Signing a JWT Token

= need the Private Cryptographic Key (+ cpu)

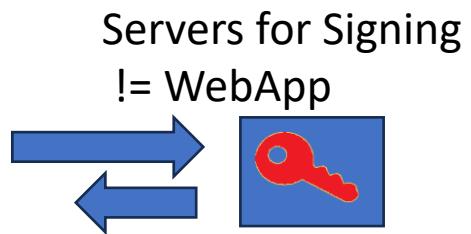
The private Key can be deployed as a secured configuration file
OR distributed in a secured cluster over Https
(chicken and egg pb: need another secret to ensure trust)

For JWT, the server is "STATELESS" : need not saving anything in Database
For checking Passwords, servers still need to connect to a LDAP realm, or Database



Checking JWT

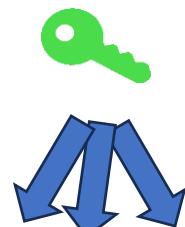
= need only Public Cryptographic Key (+ cpu)



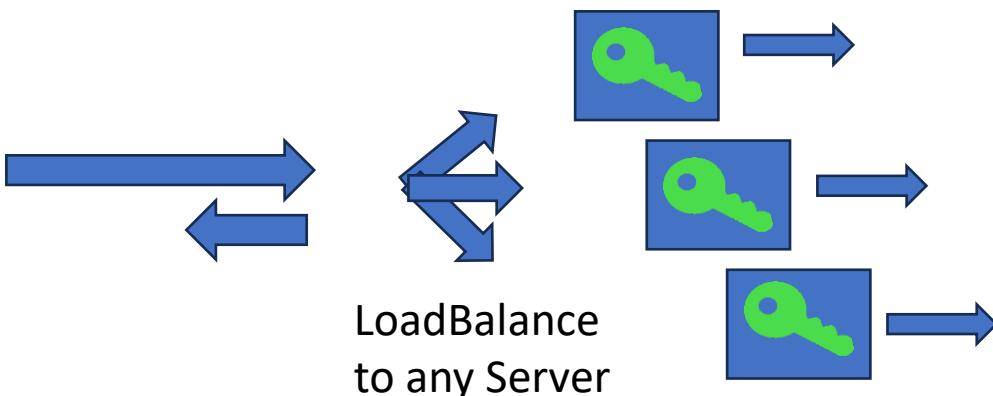
Both roles: WebApp
+ capable of Signing



The WebApp servers are "STATELESS"
The public Key can be distributed / broadcasted easily



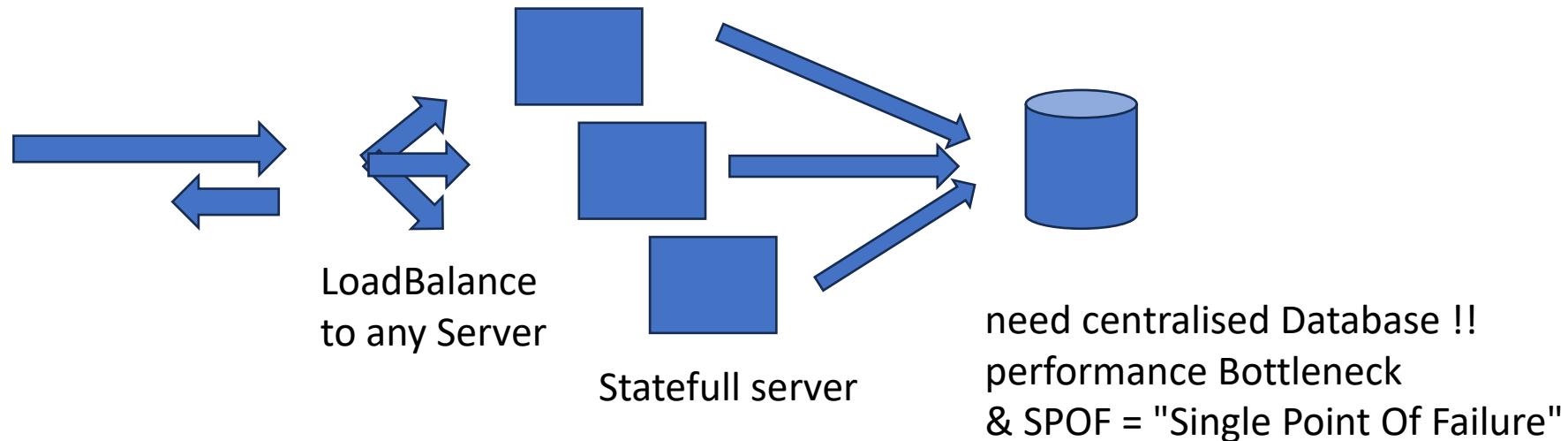
Cluster of WebApp Servers (not signing)



JWT .. easy to deploy Stateless Servers

See Next for "HTTP Session" using COOKIE => much more complex

need either "Database" to store all cookies
OR LoadBalancer supporting "Sessions Stickiness"



Conclusion

JHipster is ULTRA HIGH quality generated code

No compromise on Security

(pretty difficult to read, but interesting to learn new things done right)

There's more to it

- CRUD screen generated for your model
- can plug other frontend languages (React, VueJs)
- can plug different security mode: OAuth2 or old-style "COOKIE jsessionId"

Next Steps

- **Debugging the OAuth2 security mode**

the "login" page is not hosted in your webapp,
it is redirected to another trusted webapp,
for example "FranceConnect", "Google", "github", etc
then redirecting back to your website with a token

- **Debugging old-style COOKIE "jsessionId" security mode**

Questions

arnaud.nauwynck@gmail.com