

Security & Privacy

TD2 (Password Management)

Esilv 2025

arnaud.nauwynck@gmail.com

Outline (= Same as Course)

Authentication

Authorization - Permission

Confidentiality/Encryption

Backend-end: storing password in Database

Cryptographic "Hash" function, Salt

Who is "John The Ripper" ?

Summary

Authentication	 Transmit Password to Http
Authorization - Permission	password=>authentication=>role
Confidentiality/Encryption	password not in clear over internet
Backend-end: storing password in Database	storing password ...
Cryptographic "Hash" function, Salt	... storing hashed+salted password
Who is "John The Ripper" ?	finding password from hashed

Outline



Authentication

Authorization - Permission

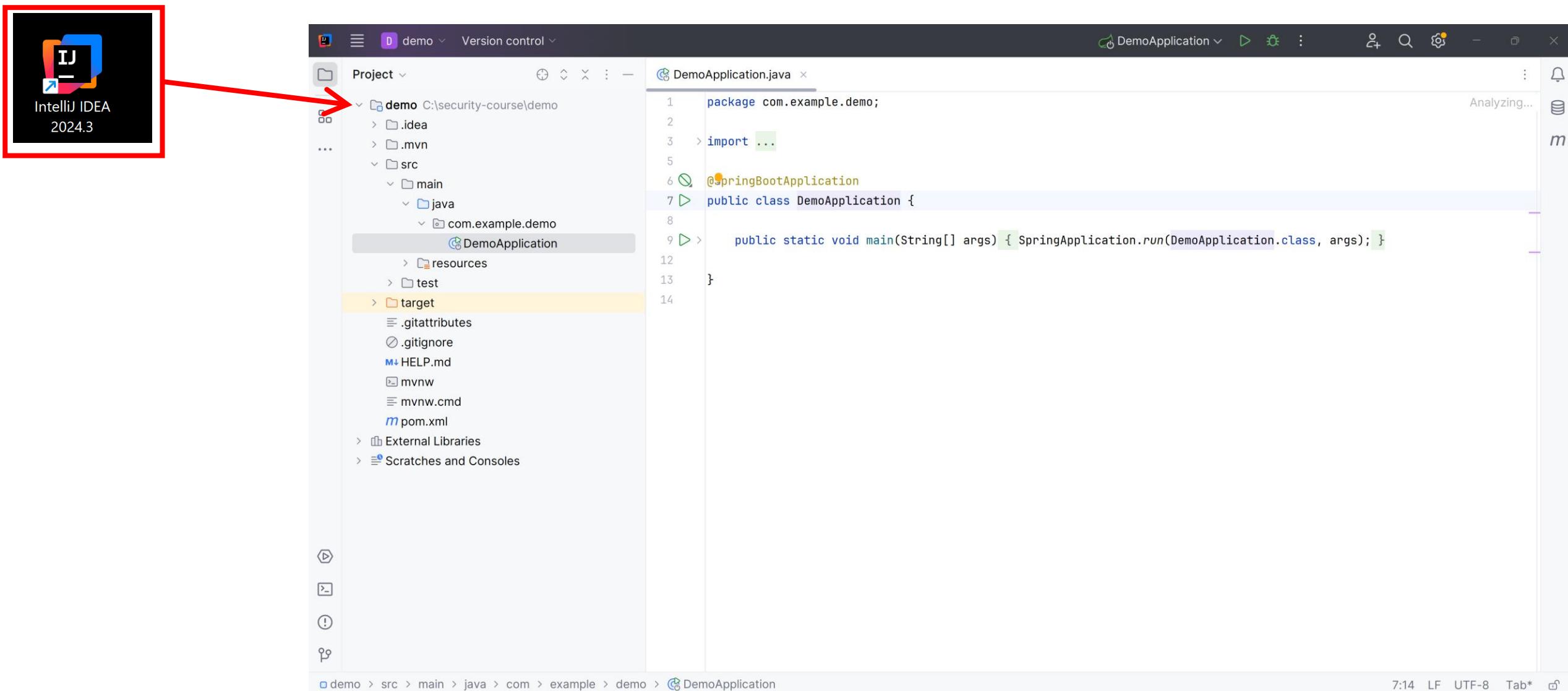
Confidentiality/Encryption

Backend-end: storing password in Database

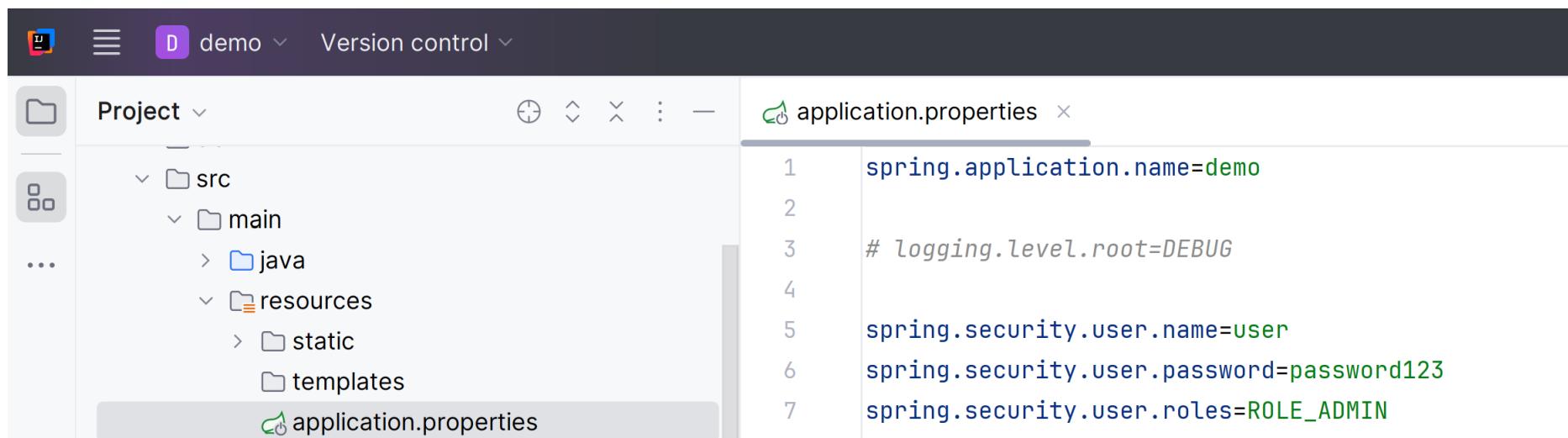
Cryptographic "Hash" function, Salt

Who is "John The Ripper" ?

Pre-Requisite Step1: relaunch your IntelliJ IDE reopen project c:\security-course\demo



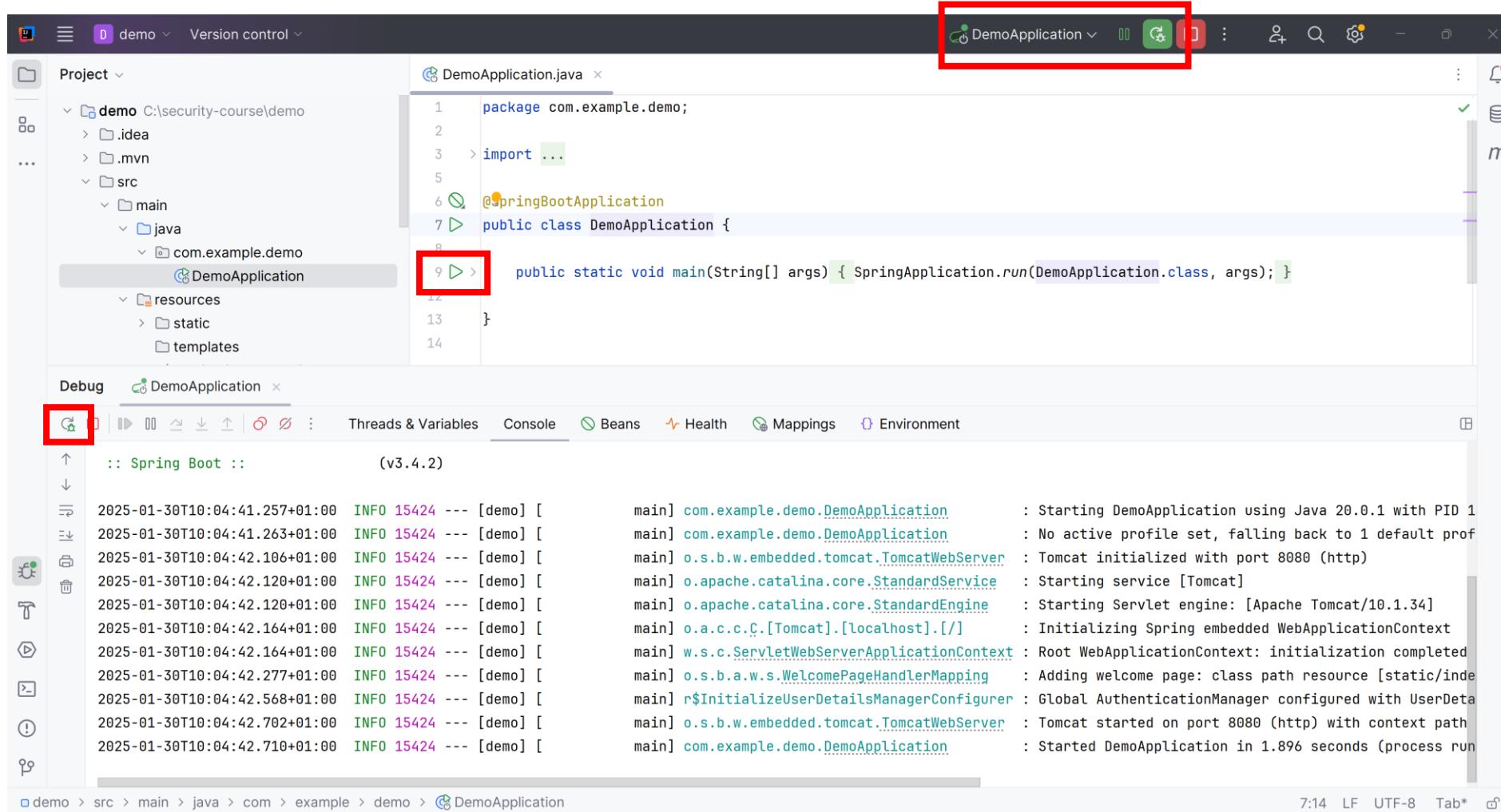
Pre-Requisite Step 2: check file
src/main/resources/application.properties
for "user" / "password123"
roles: ROLE_ADMIN



The screenshot shows a code editor interface with a dark theme. The top bar includes icons for file, edit, and version control, with the project name 'demo' selected. The left sidebar displays the project structure under 'Project': 'src' is expanded, showing 'main' (with 'java' and 'resources' subfolders), 'static', 'templates', and 'application.properties'. The 'resources' folder is also expanded. The right pane shows the contents of 'application.properties'. The code is as follows:

```
spring.application.name=demo
# logging.level.root=DEBUG
spring.security.user.name=user
spring.security.user.password=password123
spring.security.user.roles=ROLE_ADMIN
```

Pre-Requisite Step 3: relaunch webapp server



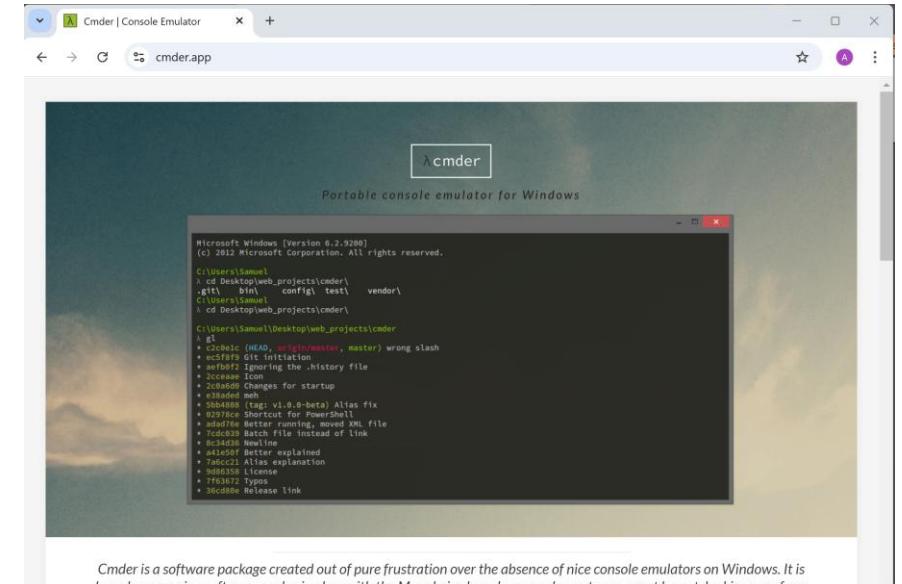
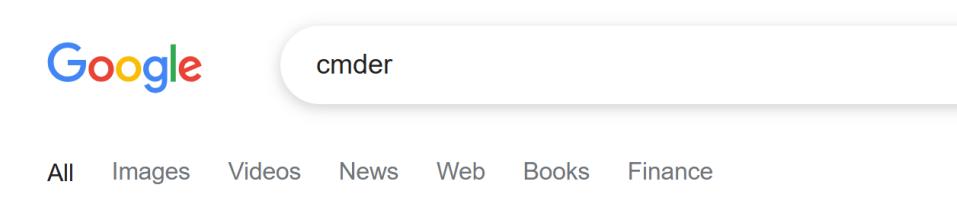
Pre-Requisite : Install (unix) shell tools "curl" and "base64"

on linux => built-in tools on system

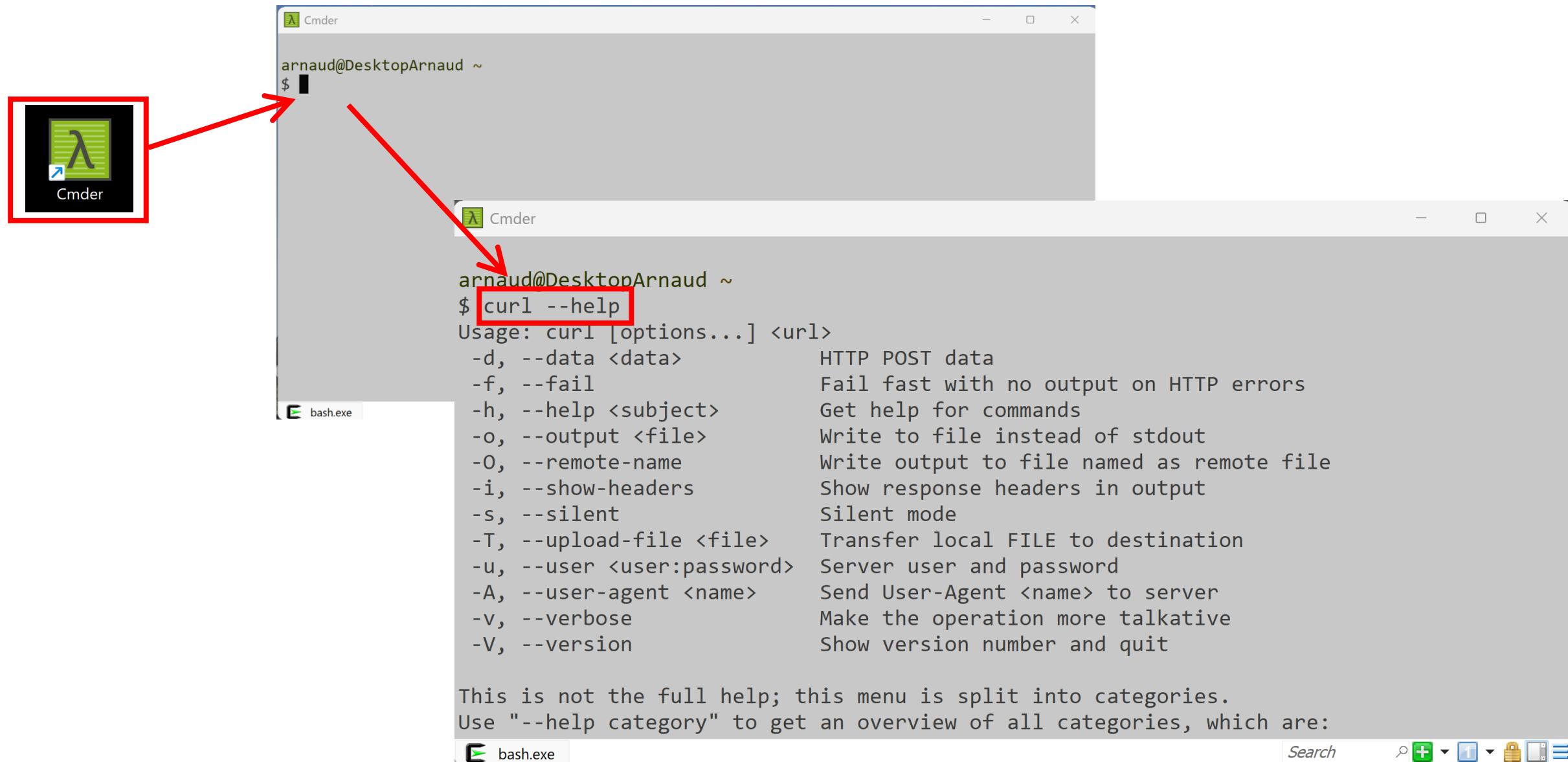
on mac => same (or use "brew install" ?)

on windows => recommended solution: install "cmder"

<https://cmder.app/>



Launch cmder console, test "curl --help"



curl main arguments

curl -v	<= for verbose
-k	<= accept all https certificate
-u "user:password"	<= basic user authentication
-x POST, PUT, DELETE	<= http verb
-H "header:value"	<= http header
url	<= http url
-d	<= http request body data

Test different curl requests..

`http://localhost:8080/index.html`

```
curl -v http://localhost:8080/index.html
```

```
curl -v -u user:password123 http://localhost:8080/index.html
```

```
curl -v -u user:password123 -H "accept: text/html" http://localhost:8080/index.html
```

```
curl -v -H "Cookie: JSESSIONID=..." \
      -H "accept: text/html" http://localhost:8080/index.html
```

Explain different Http response status

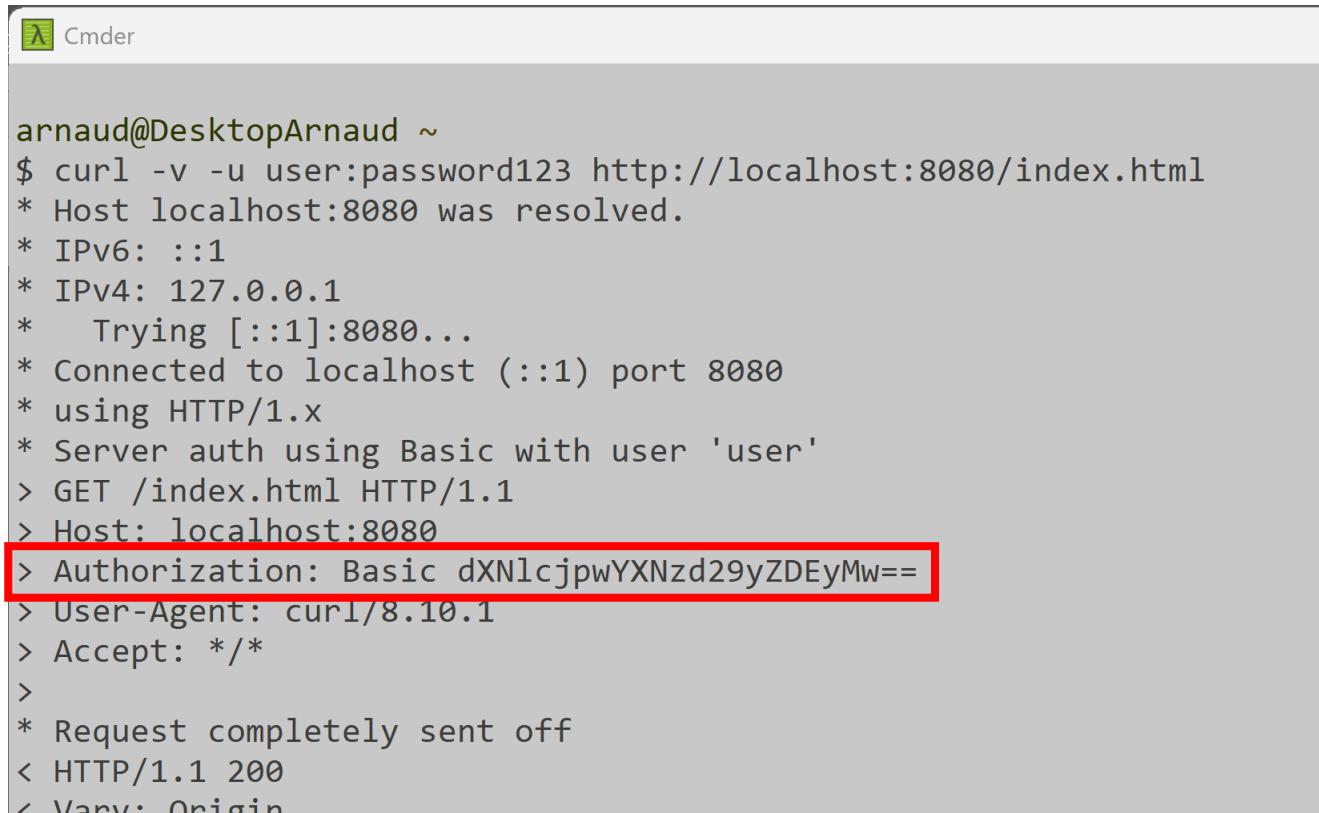
200, 301, 400, 403, 404, ...

example with http 200

```
Cmder
arnaud@DesktopArnaud ~
$ curl -v -u user:password123 http://localhost:8080/index.html
* Host localhost:8080 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
* Trying [::1]:8080...
* Connected to localhost (::1) port 8080
* using HTTP/1.x
* Server auth using Basic with user 'user'
> GET /index.html HTTP/1.1
> Host: localhost:8080
> Authorization: Basic dXNlcjpwYXNzd29yZDEyMw==
> User-Agent: curl/8.10.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200
< vary: origin
< Vary: Access-Control-Request-Method
< Vary: Access-Control-Request-Headers
< Last-Modified: Wed, 29 Jan 2025 14:18:28 GMT
< Accept-Ranges: bytes
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 0
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Pragma: no-cache
< Expires: 0
< X-Frame-Options: DENY
< Content-Type: text/html
< Content-Length: 16
< Date: Thu, 30 Jan 2025 09:36:44 GMT
<
```



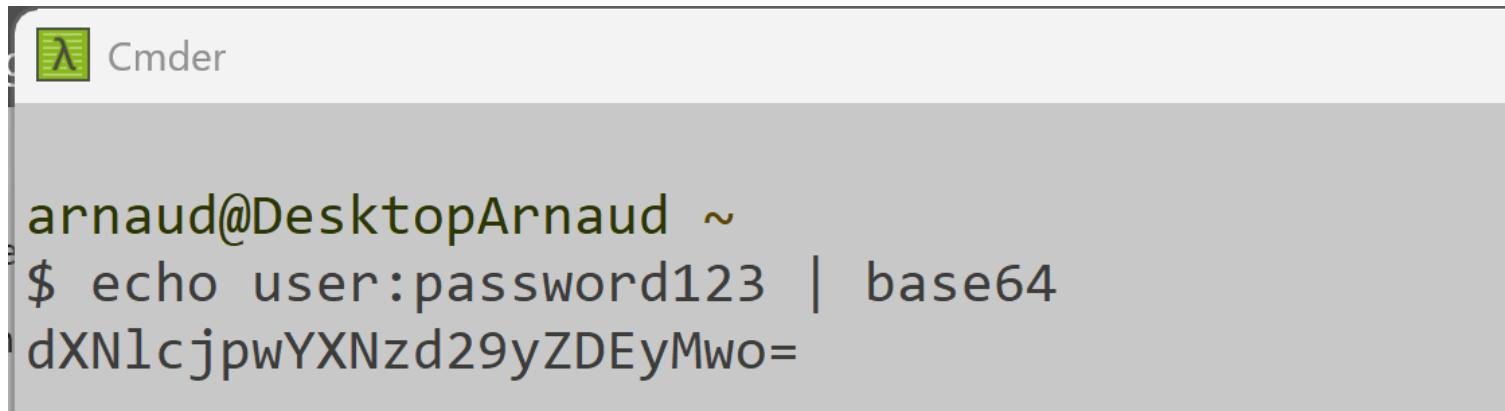
```
curl -v -u "user:password123"
=> see Http Header "Authorization: Basic ..."
```



```
arnaud@DesktopArnaud ~
$ curl -v -u user:password123 http://localhost:8080/index.html
* Host localhost:8080 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
* Trying [::1]:8080...
* Connected to localhost (::1) port 8080
* using HTTP/1.x
* Server auth using Basic with user 'user'
> GET /index.html HTTP/1.1
> Host: localhost:8080
> Authorization: Basic dXNlcjpwYXNzd29yZDEyMw==
> User-Agent: curl/8.10.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200
< Vary: Origin
```

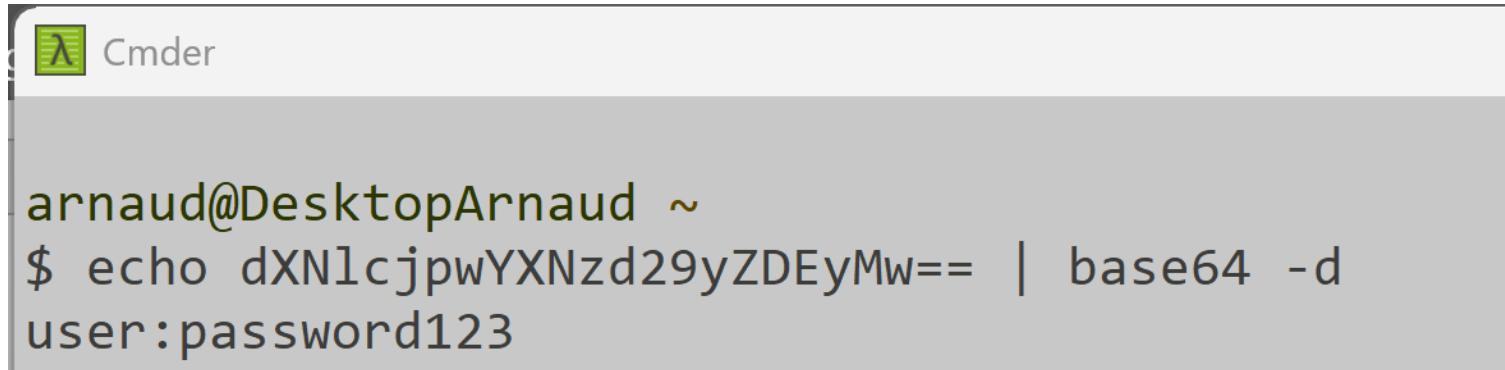
Encode / Decode base64 password, using "base64 -d"

```
echo user:password123 | base64
```



```
arnaud@DesktopArnaud ~
$ echo user:password123 | base64
dXNlcjpwYXNzd29yZDEyMwo=
```

```
echo dXNlcjpwYXNzd29yZDEyMw== | base64 -d
```



```
arnaud@DesktopArnaud ~
$ echo dXNlcjpwYXNzd29yZDEyMw== | base64 -d
user:password123
```

Test curl Basic Authorization Header (base64)

```
curl -v -H "Authorization: Basic dXNlcjpwYXNzd29yZDEyMw==" \  
      -H "accept: text/html" http://localhost:8080/index.html
```



Remember ...
When taking Http Request screenshots, or sharing screens to have IT support
... Do not reveal your "base64 encoded" password !!!

Outline

Authentication



Authorization - Permission

Confidentiality/Encryption

Backend-end: storing password in Database

Cryptographic "Hash" function, Salt

Who is "John The Ripper" ?

Target: pages*.html with different Roles

The screenshot shows a Java IDE interface with the following details:

- Project Bar:** Shows the project name "demo" and a "Version control" dropdown.
- Project Explorer:** Displays the project structure under "src".
 - main:** Contains "java" and "resources".
 - resources:** Contains "static".
 - "index.html" is selected and highlighted.
 - Other files in "static": "page-admin.html", "page-anonymous.html", "page-authenticated.html", "page-manager.html", "page-user.html".
 - templates**
 - application.properties**- target:** Contains "h2-console".
- Structure:** Shows a collapsed node for "H1 Home page".
- Code Editor:** The "index.html" file is open with the following content:

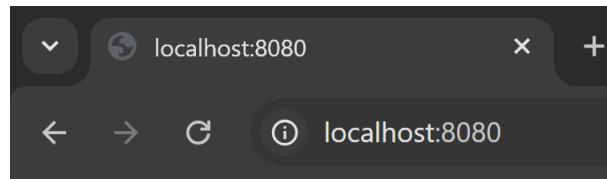
```
<H1>Home page</H1>
<BR/>
<BR/>

<A href="/page-user.html">page-user</A>
<BR/>
<A href="/page-manager.html">page-manager</A>
<BR/>
<A href="/page-authenticated.html">page-authenticated</A>
<BR/>
<A href="/page-anonymous.html">page-anonymous</A>
<BR/>
<A href="/page-admin.html">page-admin</A>
<BR/>

<BR/>

<A href="/h2-console">h2-console</A> (ADMIN)
```

"index.html" menu with anonymous access



Home page

[page-user](#)
[page-manager](#)
[page-authenticated](#)
[page-anonymous](#)
[page-admin](#)

[h2-console](#) (ADMIN)
[logout](#)

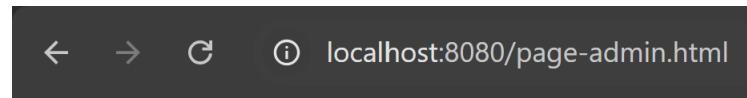
```
<H1>Home page</H1>
<BR/>
<BR/>

<A href="/page-user.html">page-user</A>
<BR/>
<A href="/page-manager.html">page-manager</A>
<BR/>
<A href="/page-authenticated.html">page-authenticated</A>
<BR/>
<A href="/page-anonymous.html">page-anonymous</A>
<BR/>
<A href="/page-admin.html">page-admin</A>
<BR/>
<BR/>

<A href="/h2-console">h2-console</A> (ADMIN)
<BR/>

<A href="/logout">logout</A>
```

Page-XYZ ... accessible only if user has role XYZ



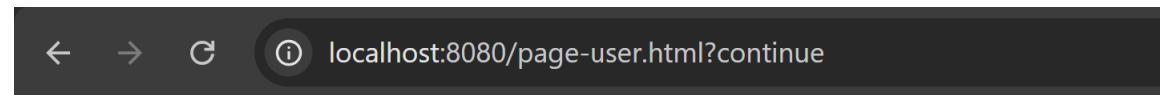
[Home](#)

... should see only if you have role "ADMIN"

<H1>Page Admin</H1>

Home

... should see only if you have role "ADMIN"



This application has no explicit mapping for /error, so you are seeing this as a fallback.
Fri Jan 31 19:12:37 CET 2025
There was an unexpected error (type=Forbidden, status=403).

class SecurityConf [1/3]

to configure ... (and disable cors & csrf)

```
package com.example.demo;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;
import java.util.List;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
            .formLogin(withDefaults())
            .logout(withDefaults())
            .authorizeHttpRequests((requests) -> configureHttpAuth(requests))
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .csrf(csrf -> csrf
                .ignoringRequestMatchers("/h2-console/**")
                .disable()) // Disable CSRF for API requests
            .headers(headers -> headers.frameOptions(frame -> frame.sameOrigin()))
            .build();
    }
}
```

// ... continue next page

class SecurityConf [2/3]

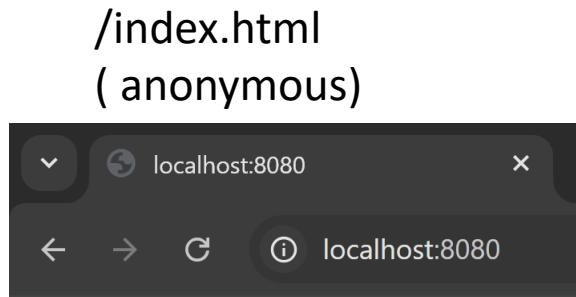
```
private static void configureHttpAuth(  
    AuthorizeHttpRequestsConfigurer<HttpSecurity>.AuthorizationManagerRequestMatcherRegistry requests) {  
    requests  
        .requestMatchers(HttpMethod.GET, "*.css", "*.js", "*.png").permitAll()  
        .requestMatchers("/", "/index.html").permitAll()  
        .requestMatchers("/page-admin.html").hasRole("ADMIN")  
        .requestMatchers("/page-user.html").hasRole("USER")  
        .requestMatchers("/page-manager.html").hasRole("MANAGER")  
        .requestMatchers("/page-anonymous.html").permitAll()  
        .requestMatchers("/page-authenticated.html").authenticated()  
        .requestMatchers("/h2-console/**")  
            .authenticated() // TODO .. should be .hasRole("ADMIN")  
            // .hasRole("ADMIN")  
        .requestMatchers("/api/**").authenticated()  
        .anyRequest().permitAll();  
}
```

class SecurityConf [3/3]

```
@Bean
public CorsFilter corsFilter() {
    CorsConfiguration config = new CorsConfiguration();
    config.setAllowCredentials(true);
    config.setAllowedOrigins(List.of("*"));
    config.setAllowedHeaders(List.of("*"));
    config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "HEAD", "OPTIONS"));
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", config);
    return new CorsFilter(source);
}

private UrlBasedCorsConfigurationSource corsConfigurationSource() {
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    CorsConfiguration config = new CorsConfiguration();
    config.setAllowCredentials(true);
    config.setAllowedOrigins(List.of("*")); // Allow all origins
    config.setAllowedHeaders(List.of("*")); // Allow all headers
    config.setAllowedMethods(List.of("*")); // Allow all HTTP methods
    source.registerCorsConfiguration("/**", config);
    return source;
}
```

Relaunch & Test navigation



Home page

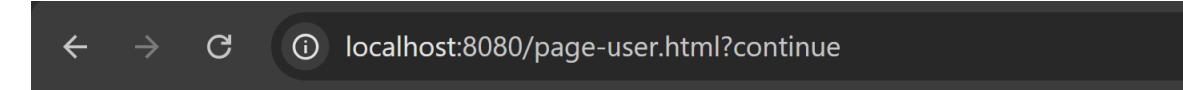
[page-user](#)
[page-manager](#)
[page-authenticated](#)
[page-anonymous](#)
[page-admin](#)

[h2-console](#) (ADMIN)
[logout](#)



Page Admin

[Home](#)
... should see only if you have role "ADMIN"

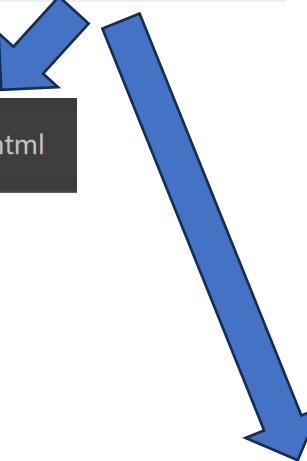
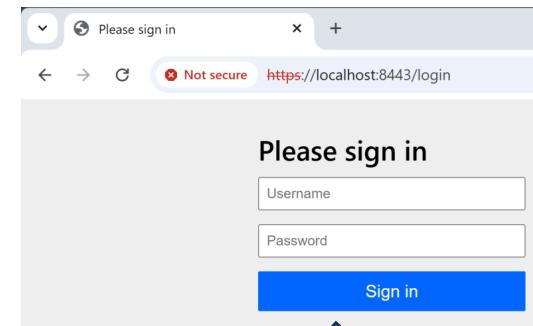


Whitelabel Error Page

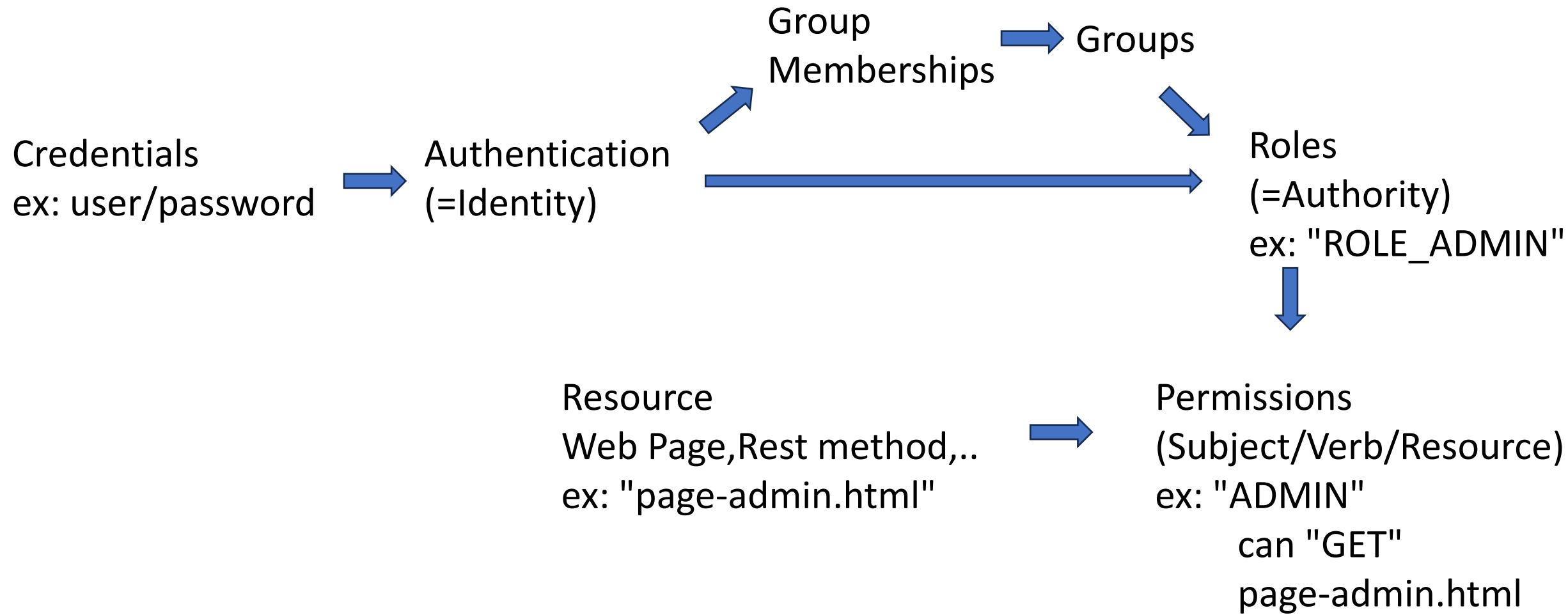
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Jan 31 19:12:37 CET 2025

There was an unexpected error (type=Forbidden, status=403).



Reminder..



Outline

Authentication

Authorization - Permission



Confidentiality/Encryption

Backend-end: storing password in Database

Cryptographic "Hash" function, Salt

Who is "John The Ripper" ?

Generate your Https self-signed certificate

```
keytool -genkeypair -alias mycert -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore mycert.p12  
-validity 3650
```

Troubleshooting ?

"keytool" is a tool in your jdk/bin directory

check "where keytool" => it should be found in your "PATH" environment variable

check "echo \$PATH"

... to fix, use windows "env" to edit your environement variables, or export PATH="\$PATH:\$JAVA_HOME/bin"



```
arnaud@DesktopArnaud /cygdrive/c/security-course/demo  
$ where keytool  
C:\apps\jdk\jdk-20.0.1+9\bin\keytool.exe
```

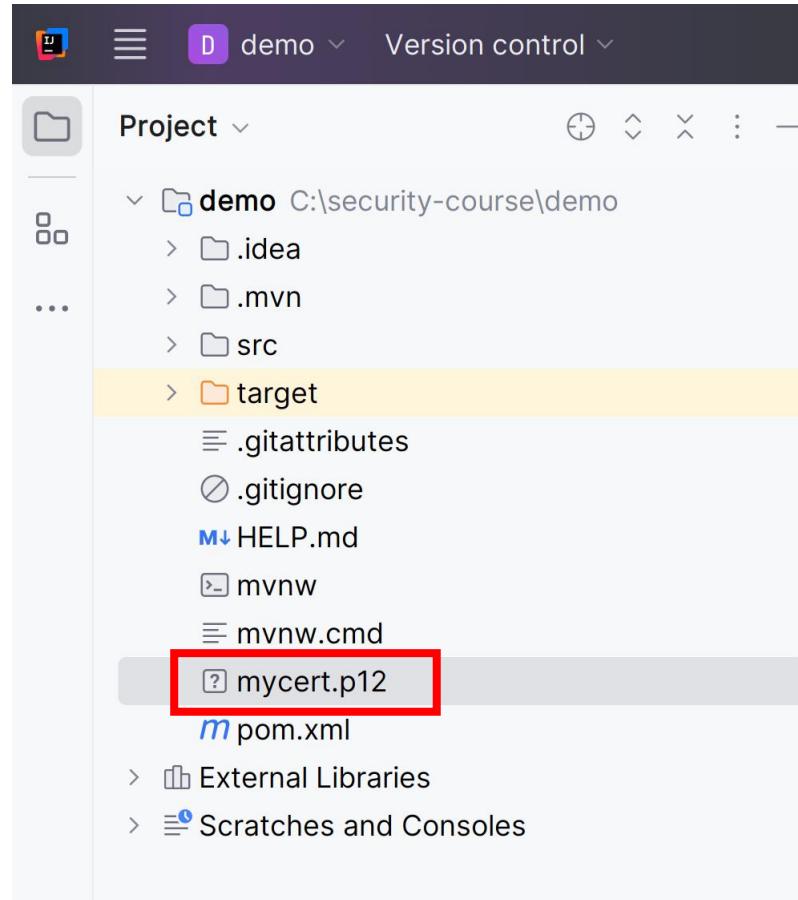
interactive answer to prompt (any is ok) lastly "Is it CN=.... ? " [no]: type yes !!

```
  Cmder
Quel est le nom de votre entreprise ?
[Unknown]:
Quel est le nom de votre ville de résidence ?
[Unknown]: Quel est le nom de votre état ou province ?
[Unknown]:
arnaud@DesktopArnaud /cygdrive/c/security-course/demo
$ keytool -genkeypair -alias mycert -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore mycert.p12 -validity 3650
Entrez le mot de passe du fichier de clés :

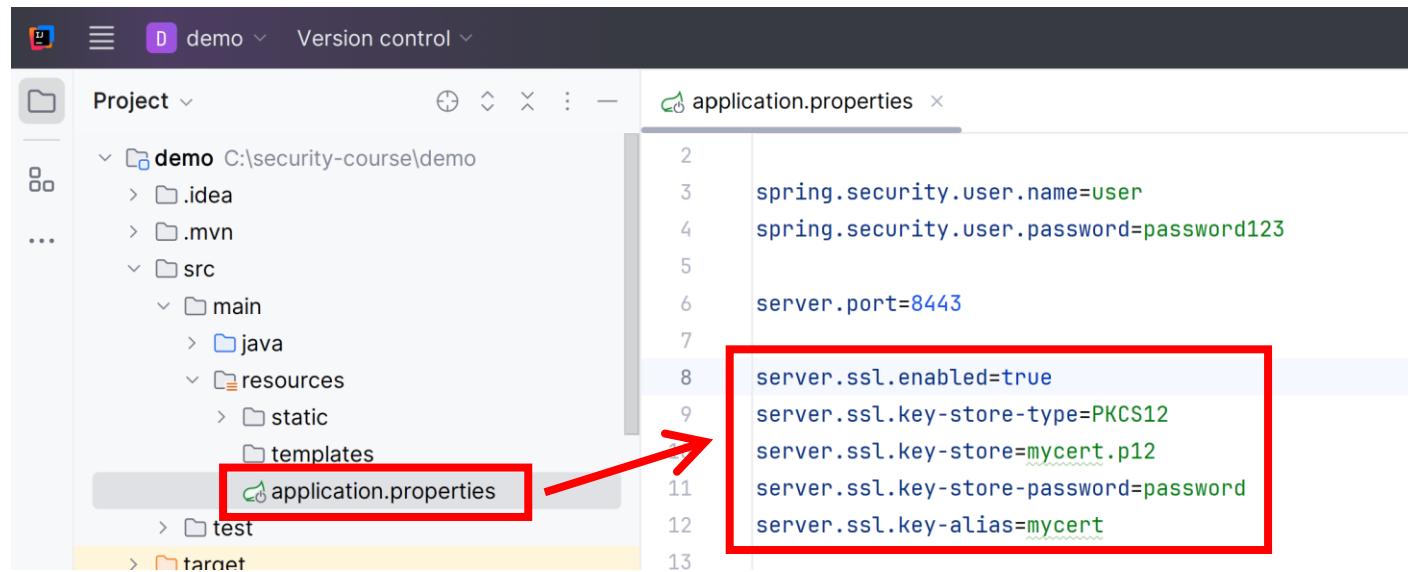
Ressaisissez le nouveau mot de passe :

Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press
ENTER to use the default value in braces.
Quels sont vos nom et prénom ?
[Unknown]: Nauwynck Arnaud
Quel est le nom de votre unité organisationnelle ?
[Unknown]: esilv
Quel est le nom de votre entreprise ?
[Unknown]: sg
Quel est le nom de votre ville de résidence ?
[Unknown]: nanterre
Quel est le nom de votre état ou province ?
[Unknown]: fr
Quel est le code pays à deux lettres pour cette unité ?
[Unknown]: fr
Est-ce CN=Nauwynck Arnaud, OU=esilv, O=sg, L=nanterre, ST=fr, C=fr ?
[non]: oui
Génération d'une paire de clés RSA de 2048 bits et d'un certificat auto-signé (SHA384withRSA)
d'une validité de 3650 jours
pour : CN=Nauwynck Arnaud, OU=esilv, O=sg, L=nanterre, ST=fr, C=fr
```

Check created file "mycert.p12" in project
(or move it in your project)



Edit src/main/resources/application.properties to configure SSL



The screenshot shows a Java project structure in a code editor. The project is named 'demo' and contains a 'src' folder with 'main' and 'test' subfolders. Inside 'main', there is a 'resources' folder which contains 'application.properties'. A red box highlights 'application.properties' in the file list. The 'application.properties' file is open in the editor, showing configuration properties. A red box highlights the SSL-related properties: server.port=8443, server.ssl.enabled=true, server.ssl.key-store-type=PKCS12, server.ssl.key-store=mycert.p12, server.ssl.key-store-password=password, and server.ssl.key-alias=mycert. An arrow points from the highlighted 'application.properties' in the file list to the highlighted SSL properties in the editor.

```
2  
3     spring.security.user.name=user  
4     spring.security.user.password=password123  
5  
6     server.port=8443  
7  
8     server.ssl.enabled=true  
9     server.ssl.key-store-type=PKCS12  
10    server.ssl.key-store=mycert.p12  
11    server.ssl.key-store-password=password  
12    server.ssl.key-alias=mycert  
13
```

server.port=8443

server.ssl.enabled=true

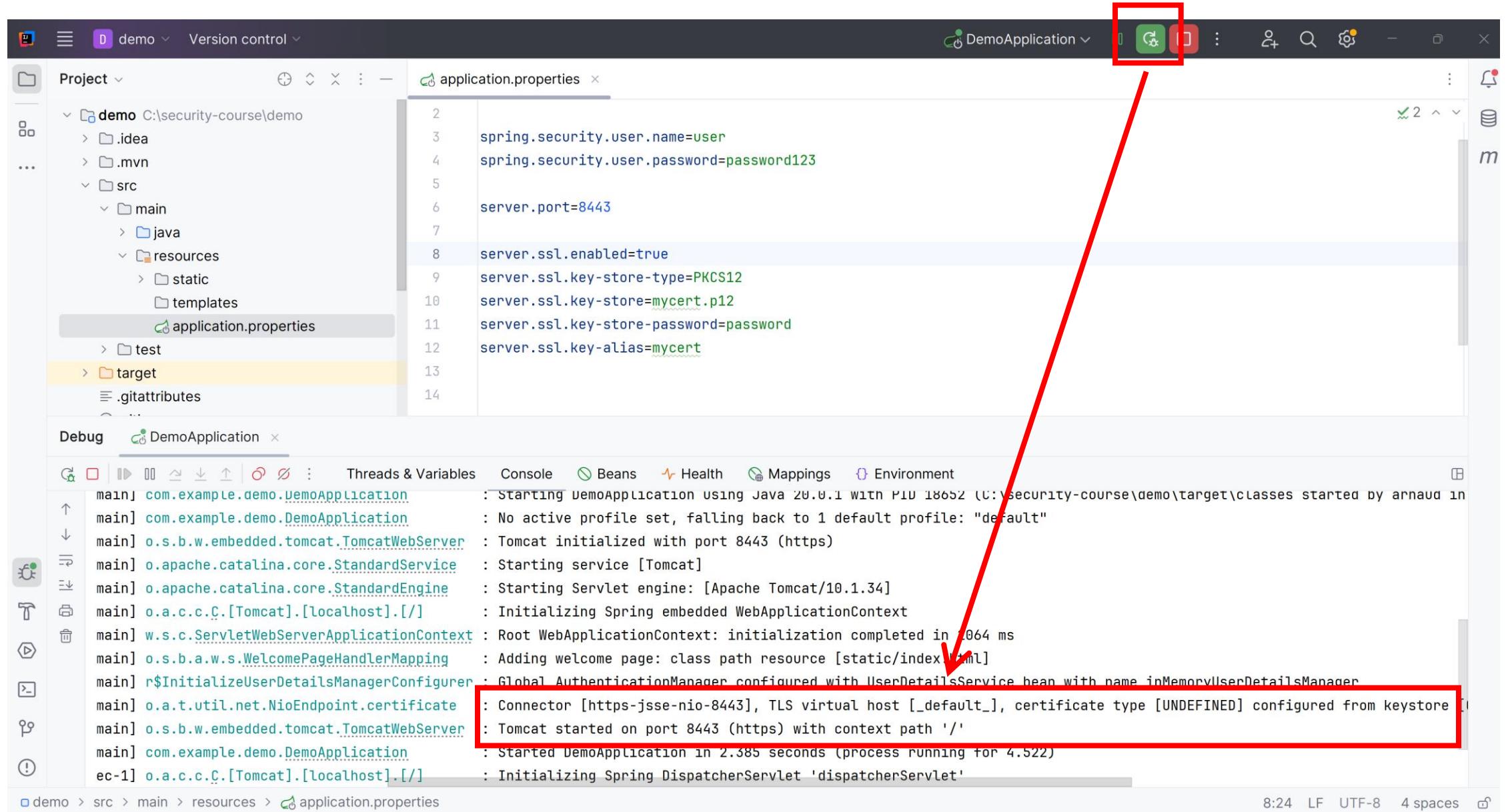
server.ssl.key-store-type=PKCS12

server.ssl.key-store=mycert.p12

server.ssl.key-store-password=password

server.ssl.key-alias=mycert

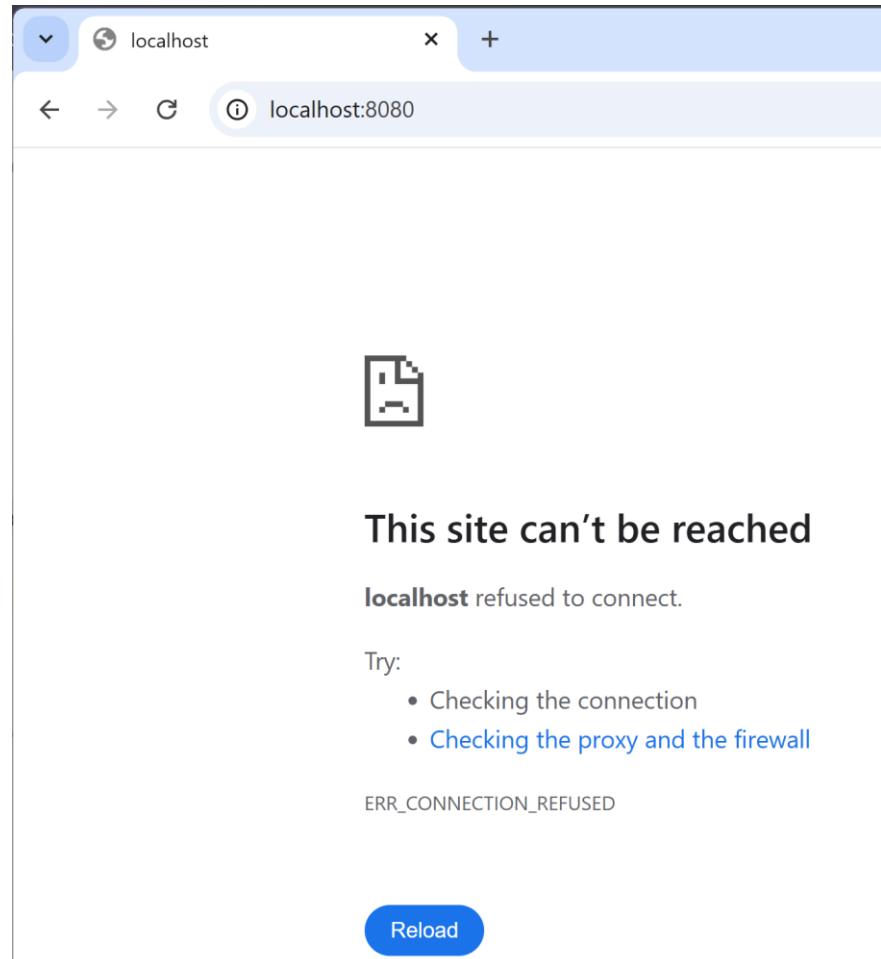
Relaunch server, check new console log



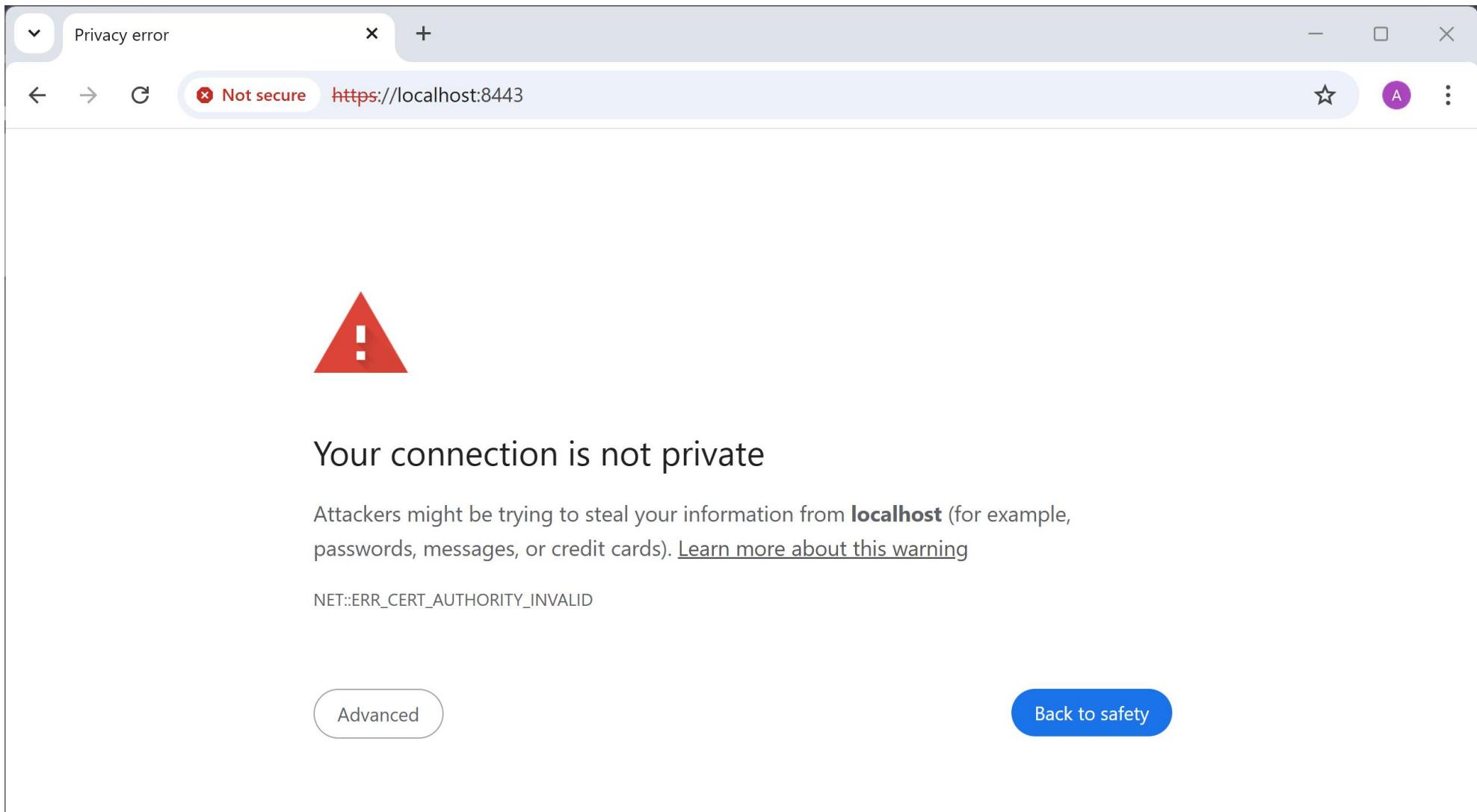
port changed 8080 (default for http)
-> 8443 (default for https)

on previous port 8080
=> "ERROR Connection refused"

OK, Normal !!



<https://localhost:8443> (self-signed certificate)
CERT_AUTHORITY_INVALID ... OK



Why ?

... because of self-signed certificate
NOT built-in recognized in your Chrome

Alternatives ?

=> pay 200 EURO/year to have a valid certificate ?

=> use GO-DADDY or other signing Authority on internet (recognized by Chrome)

=> recompile chrome to know "yourself" as valid authority

=> click on "ADVANCED" button to accept self-signed certificate

Advanced > Proceed to (unsafe)

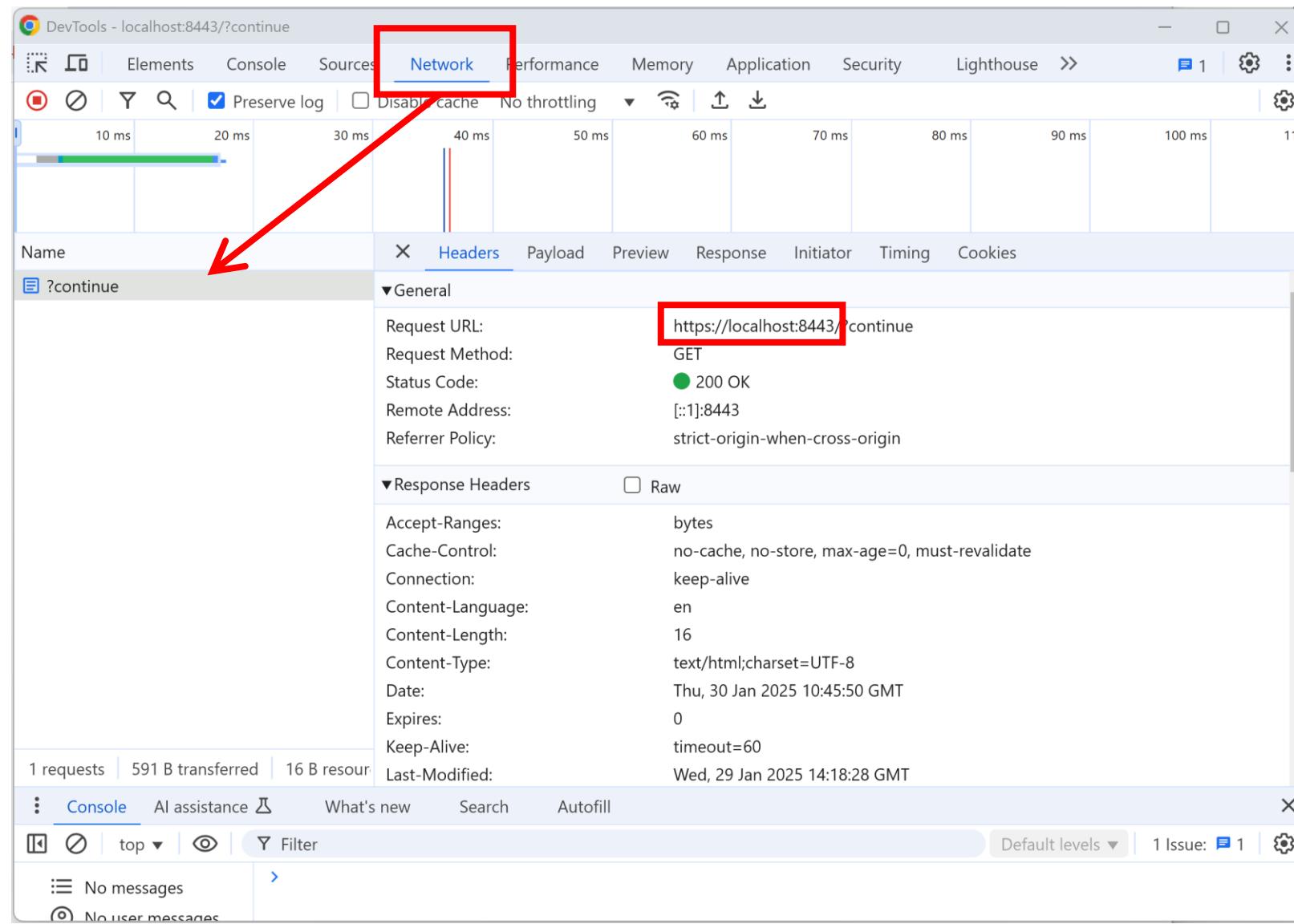
The image shows a web browser window with two tabs and a separate sign-in page.

Left Tab: The title bar says "Privacy error". The address bar shows "Not secure https://localhost:8443". The content area displays a warning message: "Your connection is not private" with a red exclamation mark icon. It states: "Attackers might be trying to steal your information from **localhost** (for example, passwords, messages, or credit cards). [Learn more about this warning](#)". Below this, the error code "NET::ERR_CERT_AUTHORITY_INVALID" is shown. A red box highlights the "Advanced" button, and a red arrow points from it to the "Proceed to localhost (unsafe)" link, which is also highlighted with a red box.

Right Tab: The title bar says "Please sign in". The address bar shows "Not secure https://localhost:8443/login". The content area displays a sign-in form with fields for "Username" and "Password", and a "Sign in" button. A red box highlights the "Not secure" status in the address bar.

Chrome DevTools Network requests

... same on "https://"



The screenshot shows the Chrome DevTools Network tab interface. A red arrow points from the title text to the 'Headers' section of the request details.

Request URL: <https://localhost:8443/?continue>

Request Method: GET

Status Code: 200 OK

Remote Address: [::1]:8443

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

Response Headers

Header	Value
Accept-Ranges	bytes
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Connection	keep-alive
Content-Language	en
Content-Length	16
Content-Type	text/html; charset=UTF-8
Date	Thu, 30 Jan 2025 10:45:50 GMT
Expires	0
Keep-Alive	timeout=60
Last-Modified	Wed, 29 Jan 2025 14:18:28 GMT

1 requests | 591 B transferred | 16 B resource

Console AI assistance What's new Search Autofill

Default levels ▾ 1 Issue: 1

Chrome DevTools "Security" tab

The screenshot shows the Chrome DevTools interface with the "Security" tab selected, indicated by a red box. The main content area displays a "Security overview" with three icons: a lock, an information circle, and a warning triangle. A prominent red message states: "This page is not secure (broken HTTPS)". Below this, two warning items are listed:

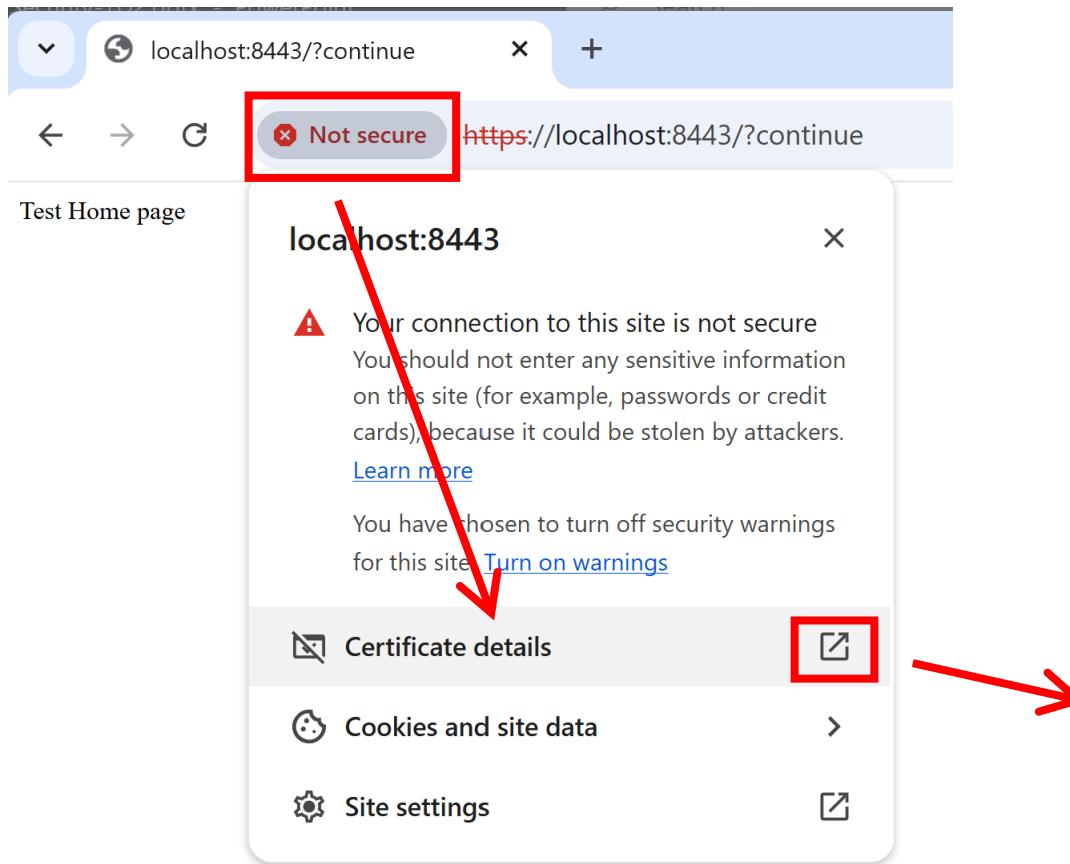
- ⚠ Certificate - Subject Alternative Name missing**: The certificate for this site does not contain a Subject Alternative Name extension containing a domain name or IP address. A "View certificate" button is provided.
- ⚠ Certificate - missing**: This site is missing a valid, trusted certificate (net::ERR_CERT_AUTHORITY_INVALID). A "View certificate" button is provided.

Two green success items are also listed:

- 🔒 Connection - secure connection settings**: The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.
- 🔒 Resources - all served securely**: All resources on this page are served securely.

The bottom of the DevTools window shows the standard toolbar with "Console" selected, along with "AI assistance", "What's new", "Search", and "Autofill". The bottom right corner shows the message "Default levels ▾ 1 Issue: 1" and a gear icon.

See Also Chrome -> Certificate Details



Suggestion ...
open view on a real web-site, "https://www.google.fr"
to see a valid signed certificate

Outline

Authentication

Authorization - Permission

Confidentiality/Encryption



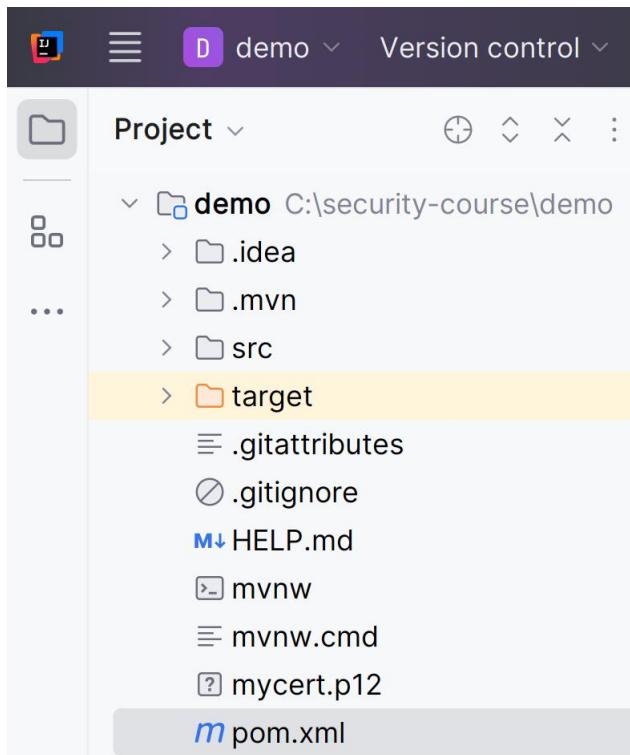
Backend-end: storing password in Database

Cryptographic "Hash" function, Salt

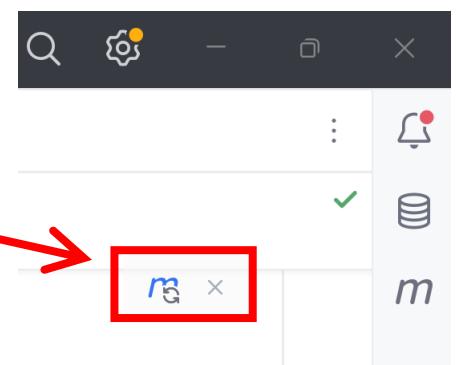
Who is "John The Ripper" ?

Edit pom.xml

Add maven dependency for Database
inside <dependencies> ..



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```



then click on refresh maven

Edit application.properties add H2 config

The screenshot shows a Java IDE interface with a dark theme. In the top bar, there are icons for file, edit, and version control, followed by the project name "demo" and "Version control". The left sidebar shows the project structure under "Project": "demo" (C:\security-course\demo) contains ".idea", ".mvn", "db", "src" (which includes "main" and "resources" folders), and "test". The "resources" folder contains "static" and "templates", and the "application.properties" file is selected. The main editor window displays the contents of the application.properties file:

```
12 server.ssl.key-store=MyCert
13
14
15 # Database configuration
16 spring.datasource.url=jdbc:h2:file:./db
17 spring.datasource.driverClassName=org.h2.Driver
18 spring.datasource.username=sa
19 spring.datasource.password=
20
21 # web console, https://localhost:8443/h2-console
22 spring.h2.console.enabled=true
23 spring.h2.console.path=/h2-console
24
```

```
# Database configuration
spring.datasource.url=jdbc:h2:file:./db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# web console, see https://localhost:8443/h2-console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Create java class SecurityDbConfiguration

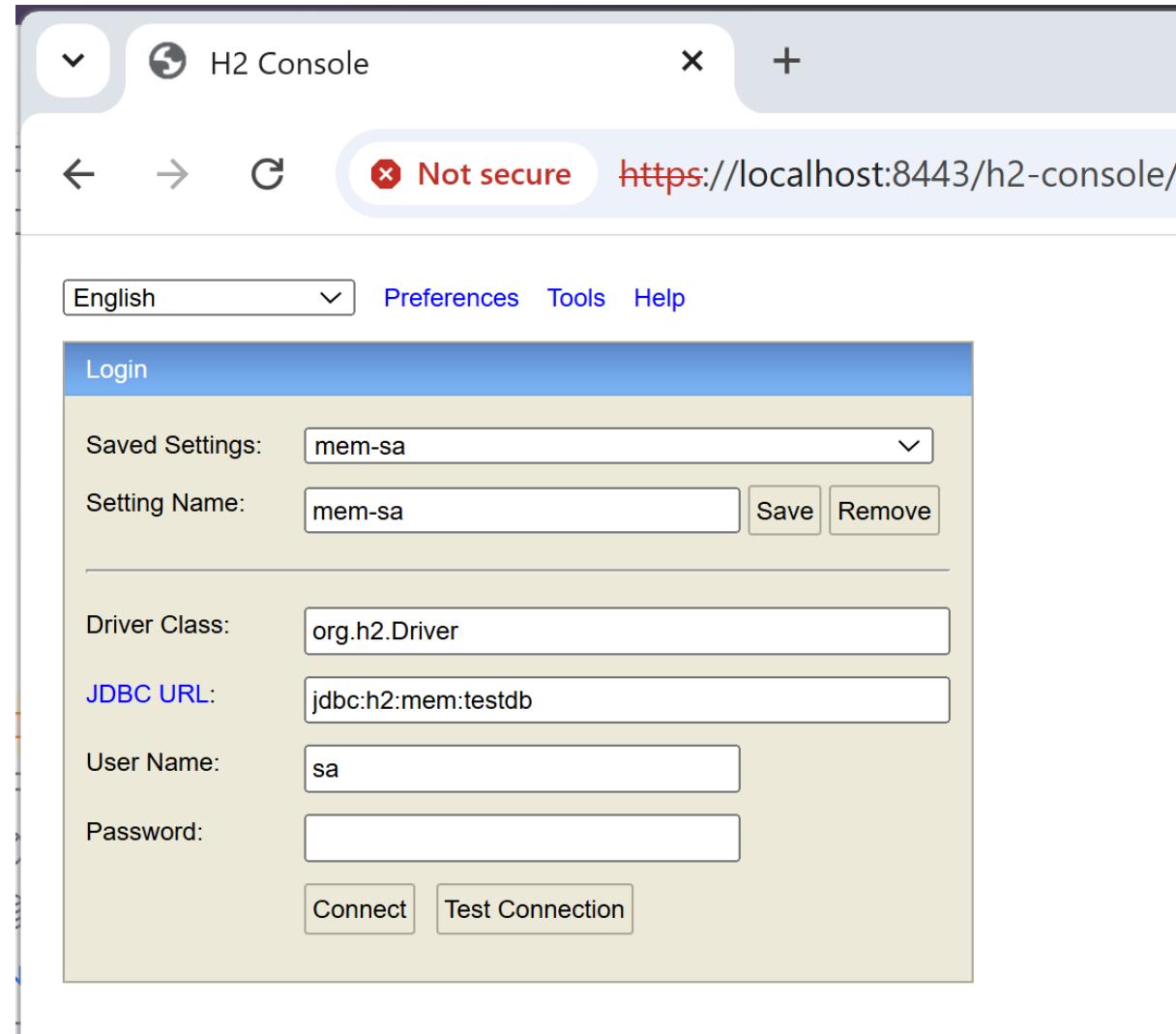
```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.provisioning.JdbcUserDetailsManager;
import org.springframework.security.provisioning.UserDetailsManager;
import javax.sql.DataSource;

@Configuration
public class SecurityDbConfiguration {

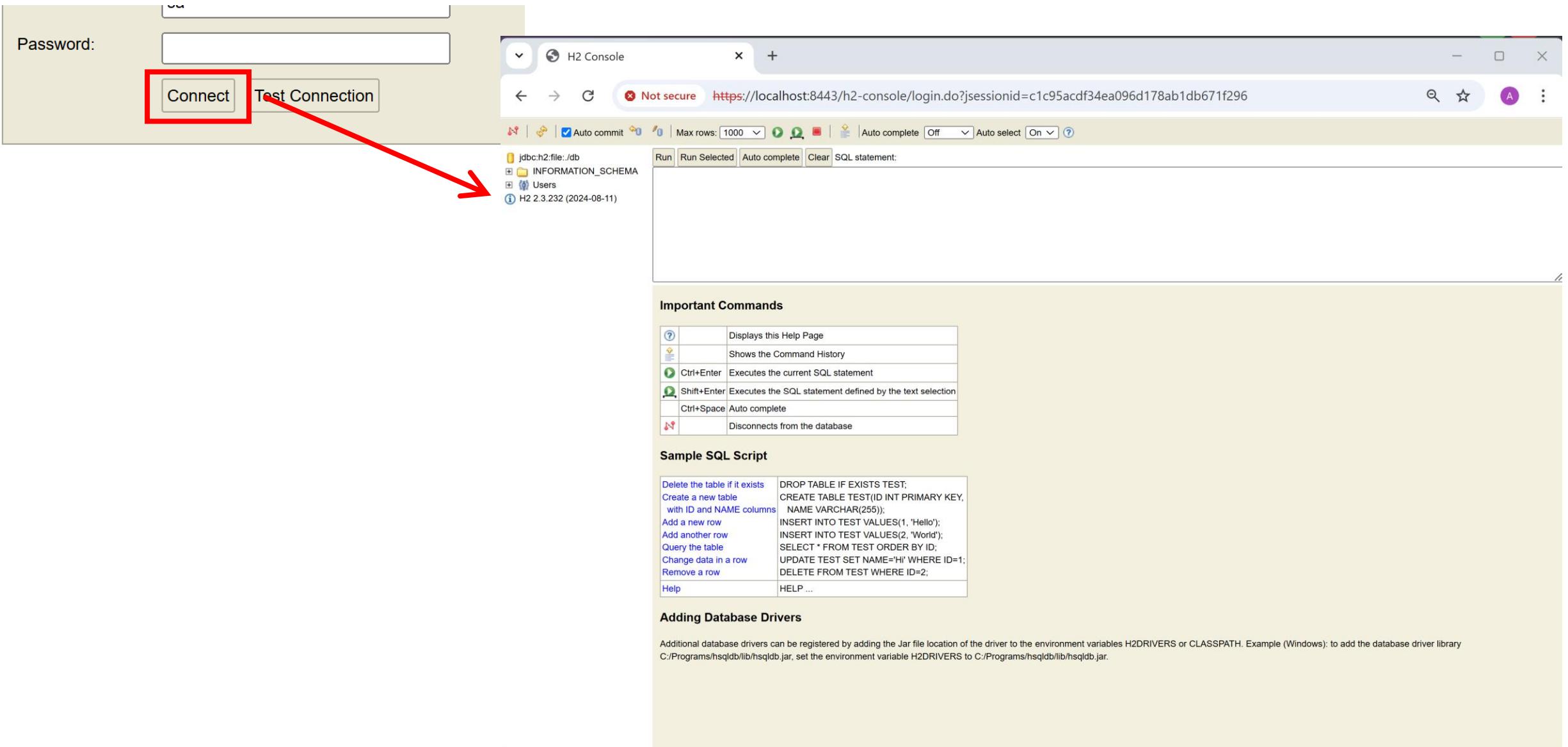
    @Bean
    UserDetailsManager users(DataSource dataSource) {
        JdbcUserDetailsManager users = new JdbcUserDetailsManager(dataSource);
        return users;
    }

}
```

Relaunch, Open <http://localhost:8443/h2-console>



Click "Connect"



Copy&Paste Create Tables DDL

```
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(500) not null,
    enabled boolean not null
);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fkAuthoritiesUsers foreign key(username) references
users(username)
);
create unique index ix_auth_username on authorities (username,authority);
```

cf doc

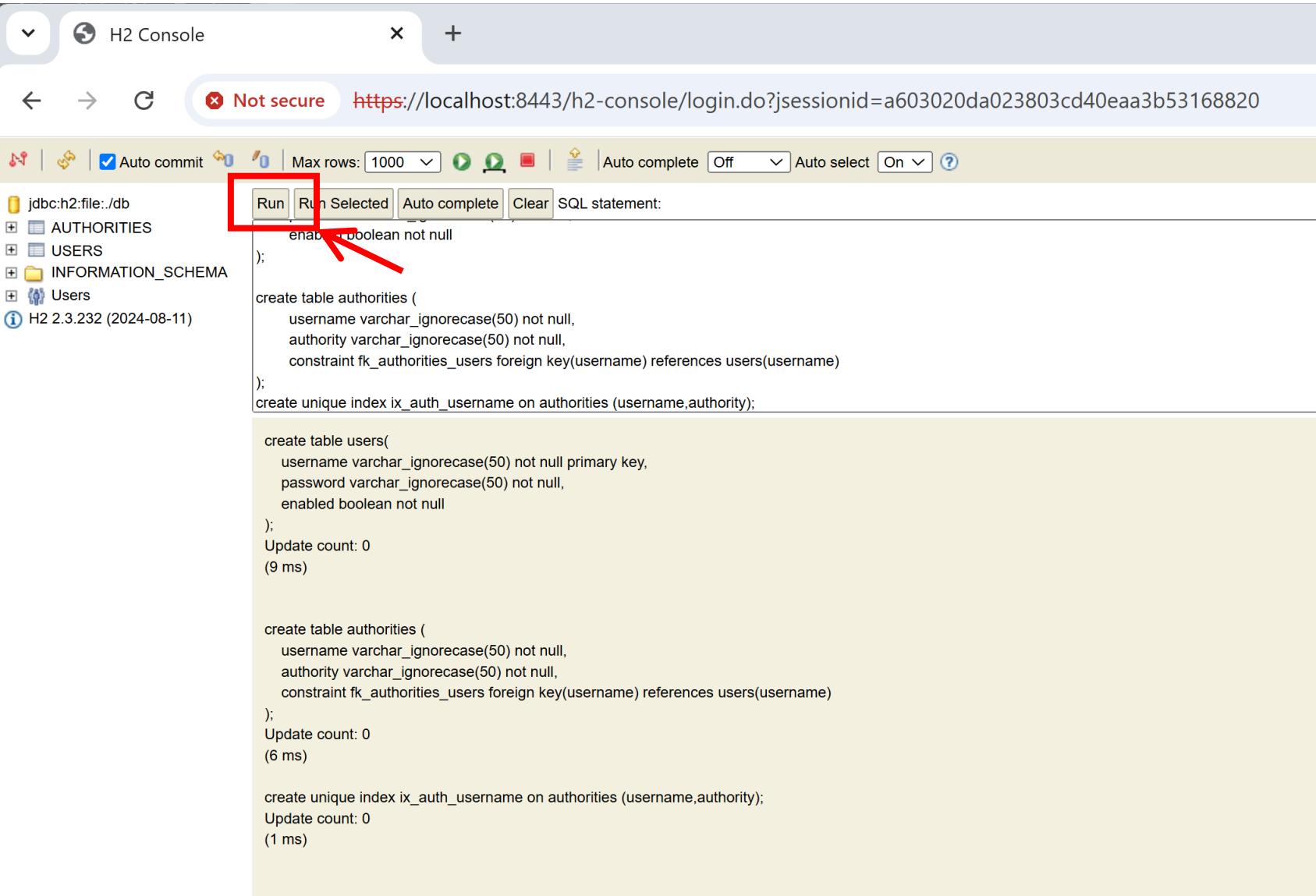
<https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/jdbc.html>

The screenshot shows a web browser displaying the Spring Security JDBC Authentication documentation. The URL in the address bar is docs.spring.io/spring-security/reference/servlet/authentication/passwords/jdbc.html. The page title is "JDBC Authentication :: Spring Security". The left sidebar contains a navigation menu for "Spring Security 6.4.2" with sections like Overview, Prerequisites, Community, What's New, Preparing for 7.0, Migrating to 6.2, Getting Spring Security, Javadoc, Features, Project Modules, Samples, and Servlet Applications. The main content area is titled "User Schema" and discusses the requirements for tables used by `JdbcDaoImpl`. It includes a "NOTE" section about a classpath resource named `org/springframework/security/core/userdetails/jdbc/users.ddl`. Below this is a "Default User Schema" section with two SQL snippets:

```
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(500) not null,
    enabled boolean not null
);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fkAuthorities_users foreign key(username) references users(username)
);
```

Execute Create Table DDL



The screenshot shows the H2 Console interface with the following details:

- Title Bar:** H2 Console
- Address Bar:** Not secure https://localhost:8443/h2-console/login.do?jsessionid=a603020da023803cd40eaa3b53168820
- Toolbar Buttons:** Auto commit, Max rows: 1000, Auto complete: Off, Auto select: On.
- Tool Buttons:** Run, Run Selected, Auto complete, Clear, SQL statement: (text input field).
- Left Sidebar:** Database structure tree with nodes: jdbc:h2:file:/db, AUTHORITIES, USERS, INFORMATION_SCHEMA, Users, and H2 2.3.232 (2024-08-11).
- SQL Statement Area:** Displays the execution of three Create Table DDL statements:
 - First statement: create table authorities (username varchar_ignorecase(50) not null, authority varchar_ignorecase(50) not null, enabled boolean not null);
 - Second statement: create table users(username varchar_ignorecase(50) not null primary key, password varchar_ignorecase(50) not null, enabled boolean not null);
 - Update count: 0 (9 ms)
 - Third statement: create table authorities (username varchar_ignorecase(50) not null, authority varchar_ignorecase(50) not null, constraint fkAuthorities_users foreign key(username) references users(username));
 - Update count: 0 (6 ms)
- Bottom Status:** create unique index ix_auth_username on authorities (username,authority);
Update count: 0
(1 ms)

(Optionnal) DDL for Groups & Members

```
create table groups (
    id bigint generated by default as identity(start with 0) primary key,
    group_name varchar_ignorecase(50) not null
);

create table groupAuthorities (
    group_id bigint not null,
    authority varchar(50) not null,
    constraint fk_groupAuthorities_group foreign key(group_id) references groups(id)
);

create table groupMembers (
    id bigint generated by default as identity(start with 0) primary key,
    username varchar(50) not null,
    group_id bigint not null,
    constraint fk_groupMembers_group foreign key(group_id) references groups(id)
);
```

Insert users in database

Using H2 "New Row", edit, commit

The screenshot illustrates the process of inserting users into the H2 database using the "New Row" feature.

Initial State: The left panel shows the database structure with the **USERS** table selected. A red box highlights the **USERS** table in the tree view. The right panel displays the result of the query `SELECT * FROM USERS;`, which returns two rows: `user1` and `user2`. An **Edit** button is highlighted with a red box.

Editing Phase: A red arrow points from the initial state to the second screenshot. In the second screenshot, the `USERS` table is shown in edit mode. The `user2` row has been selected for modification. A red box highlights the `+ New Row` button at the bottom-left of the table.

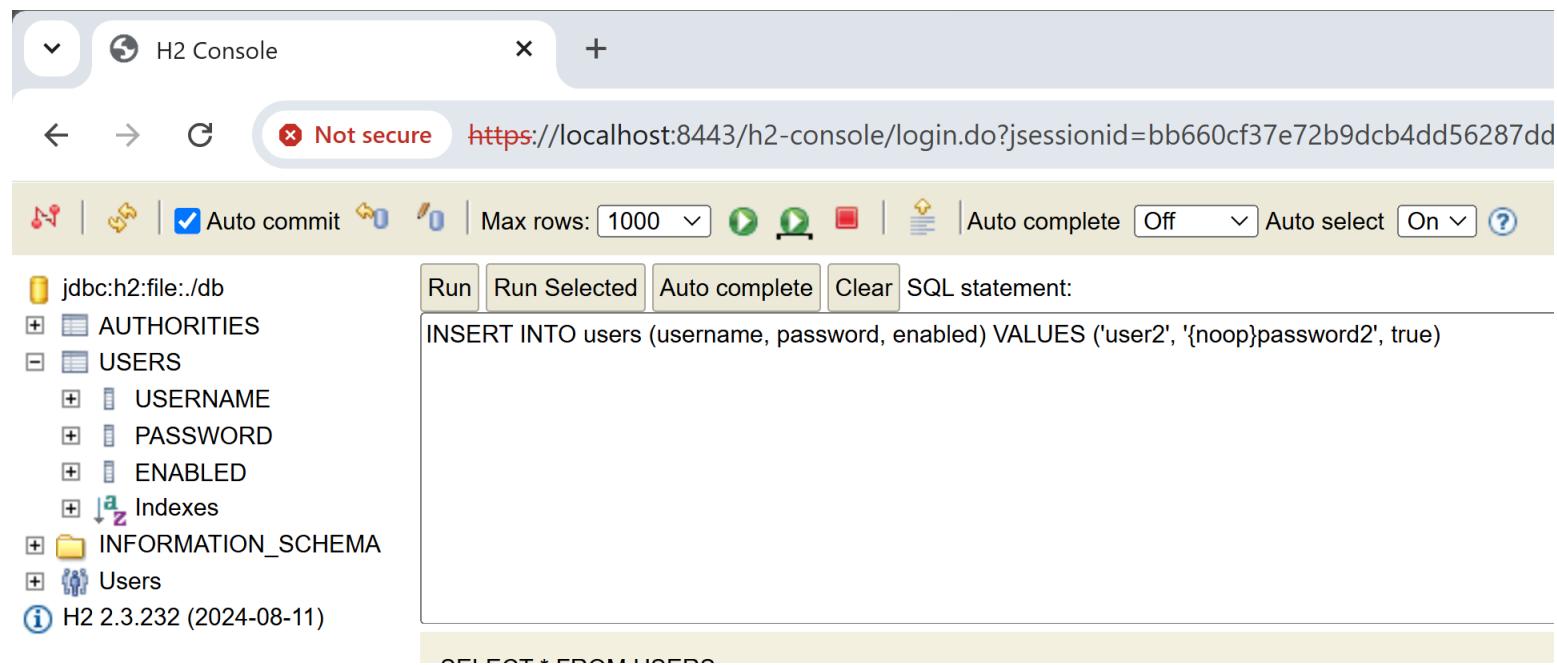
Final State: A red arrow points from the editing phase to the final screenshot. The `USERS` table now contains three rows: `user1`, `user2`, and a new row added via the `+ New Row` button. The new row has the following values: `Action` (checkbox checked), `USERNAME` (`user3`), `PASSWORD` (`{noop}password3`), and `ENABLED` (`TRUE`). The `{noop}password3` value is highlighted with a red box.



NOTICE: should prefix password with `{noop}`
(see next lesson for Hash and Salting)

INSERT INTO users... using SQL

INSERT INTO users (username, password, enabled) VALUES ('user2', '{noop}password2', true)



Also add users a "role" (= AUTHORITY)

The screenshot shows the H2 Database Console interface. The left sidebar lists databases (jdbc:h2:file:./db), schemas (AUTHORITIES, USERS, INFORMATION_SCHEMA), and tables (Users). The right side has a toolbar with Run, Run Selected, Auto complete, Clear, and SQL statement: fields. Below is a query window with the SQL command: SELECT * FROM AUTHORITIES. At the bottom, there is a table view with the following data:

Action	USERNAME	AUTHORITY
	user1	ROLE_USER
	user2	ROLE_MANAGER

(2 rows, 1 ms)

```
INSERT INTO authorities (username, authority) VALUES ('user1', 'ROLE_USER')
```

```
INSERT INTO authorities (username, authority) VALUES ('user2', 'ROLE_MANAGER')
```



If NO role => user treated as 'Not Found' !

cf source code

<https://github.com/spring-projects/spring-security/blob/main/core/src/main/java/org/springframework/security/core/userdetails/jdbc/JdbcDaoImpl.java#L200>

The screenshot shows a GitHub code editor interface. The left sidebar displays the project structure under 'spring-security/core/src/main/java'. The main panel shows the file 'JdbcDaoImpl.java' with line numbers 181 to 206. Line 200 is highlighted with a yellow background. The code snippet is as follows:

```
181  
182     @Override  
183     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
184         List<UserDetails> users = loadUsersByUsername(username);  
185         if (users.isEmpty()) {  
186             this.logger.debug("Query returned no results for user '" + username + "'");  
187             throw new UsernameNotFoundException(this.messages.getMessage("JdbcDaoImpl.notFound",  
188                     new Object[] { username }, "Username {0} not found"));  
189         }  
190         UserDetails user = users.get(0); // contains no GrantedAuthority[]  
191         Set<GrantedAuthority> dbAuthsSet = new HashSet<>();  
192         if (this.enableAuthorities) {  
193             dbAuthsSet.addAll(loadUserAuthorities(user.getUsername()));  
194         }  
195         if (this.enableGroups) {  
196             dbAuthsSet.addAll(loadGroupAuthorities(user.getUsername()));  
197         }  
198         List<GrantedAuthority> dbAuths = new ArrayList<>(dbAuthsSet);  
199         addCustomAuthorities(user.getUsername(), dbAuths);  
200         if (dbAuths.isEmpty()) {  
201             this.logger.debug("User '" + username + "' has no authorities and will be treated as 'not found'"  
202             throw new UsernameNotFoundException(this.messages.getMessage("JdbcDaoImpl.noAuthority",  
203                     new Object[] { username }, "User {0} has no GrantedAuthority"));  
204         }  
205         return createUserDetails(username, user, dbAuths);  
206     }
```



Springboot refuse password without "Encoder" ... or use explicitly "{noop}"

Screenshot of a Java IDE (IntelliJ IDEA) showing a stack trace in the Debug tool window. The stack trace indicates a failure due to a missing password encoder.

```
at org.springframework.security.config.annotation.web.configuration.WebMvcSecurityConfiguration$CompositeFilterChainProxy.doFilter(WebMvcSecurityConfiguration.java:413)
2025-01-30T16:13:13.137+01:00 ERROR 21412 --- [demo] [io-8443-exec-10] o.a.c.c.C.[.].[dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in
java.lang.IllegalArgumentException Create breakpoint : Given that there is no default password encoder configured, each password must have a password encoding prefix. Please
at org.springframework.security.crypto.password.DelegatingPasswordEncoder$UnmappedIdPasswordEncoder.matches(DelegatingPasswordEncoder.java:307) ~[spring-security-crypto-6.4.2.jar:6.4.2]
at org.springframework.security.crypto.password.DelegatingPasswordEncoder.matches(DelegatingPasswordEncoder.java:248) ~[spring-security-crypto-6.4.2.jar:6.4.2]
at org.springframework.security.authentication.dao.DaoAuthenticationProvider.additionalAuthenticationChecks(DaoAuthenticationProvider.java:90) ~[spring-security-core-6.4.2.jar:6.4.2]
at org.springframework.security.authentication.dao.AbstractUserDetailsAuthenticationProvider.authenticate(AbstractUserDetailsAuthenticationProvider.java:147) ~[spring-security-core-6.4.2.jar:6.4.2]
at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:182) ~[spring-security-core-6.4.2.jar:6.4.2]
at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:201) ~[spring-security-core-6.4.2.jar:6.4.2]
at org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter.attemptAuthentication(UsernamePasswordAuthenticationFilter.java:85) ~[spring-security-web-6.4.2.jar:6.4.2]
```

Given that there is no default password encoder configured, each password must have a password encoding prefix.
Please either prefix this password with '{noop}' or set a default password encoder in `DelegatingPasswordEncoder`.



Check (update) password to be "{noop}password"

Screenshot of a database management tool showing the 'USERS' table.

Toolbar: Auto commit checked, Max rows: 1000, Run, Run Selected, Auto complete, Clear, SQL statement.

Table structure:

Action	USERNAME	PASSWORD	ENABLED
X	user1	{noop}password1	TRUE
X	user2	{noop}password2	TRUE

(2 rows, 0 ms)

Screenshot of a database management tool showing the 'USERS' table.

Toolbar: Run, Run Selected, Auto complete, Clear, SQL statement.

Table structure:

Action	USERNAME	PASSWORD	ENABLED
X	user1	{noop}password1	TRUE
X	user2	{noop}password2	TRUE

Screenshot of a database management tool showing the 'USERS' table.

Toolbar: Run, Run Selected, Auto complete, Clear, SQL statement.

Table structure:

Action	USERNAME	PASSWORD	ENABLED
✓ X	user1	{noop}password1	TRUE
✓ X	user2	{noop}password2	TRUE

(2 rows, 0 ms)

Relaunch & Test ..

https://localhost:8443/login

Please sign in

user1

.....

Sign in

next lesson ... for activating the "HASH" + "Salting"

Outline

Authentication

Authorization - Permission

Confidentiality/Encryption

Backend-end: storing password in Database



Cryptographic "Hash" function, Salt

Who is "John The Ripper" ?

Re-Salting an already saved {noop}password

by default, springboot "BCryptPasswordEncoder" already generates a random Salt for each encoding

It encode both salt + hashed password in field "password"

(no need to save in a different column in database table)

Adding java class for Updating Password [1/2]

```
package com.example.demo;

import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.UserDetailsManager;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/v1/users")
public class UserUpdateRestController {

    private final UserDetailsManager userDetailsManager;

    private final PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    private static final String ENCODE_PREFIX = "{bcrypt}"; // cf PasswordEncoderFactories
```

(...next)

```
public UserUpdateRestController(UserDetailsManager userDetailsManager) {  
    this.userDetailsManager = userDetailsManager;  
    // for debug (uncomment to execute once, or use curl  
    // updateResaltPassword(new UserPasswordRequestDTO("user2", "password2"));  
}  
  
record UserPasswordRequestDTO(String username, String newPassword) {}  
  
@PostMapping("/update-password")  
public void updatePassword(@RequestBody UserPasswordRequestDTO req) {  
    UserDetails userDetails = userDetailsManager.loadUserByUsername(req.username);  
    String oldPassword = userDetails.getPassword();  
  
    String hashedPassword = ENCODE_PREFIX + passwordEncoder.encode(req.newPassword);  
  
    System.out.println("updating user password: old hashed '" + oldPassword + "' => new '" + hashedPassword + "'");  
    UserDetails updatedUser = User.withUserDetails(userDetails).password(hashedPassword).build();  
    userDetailsManager.updateUser(updatedUser);  
}  
}
```

Execute update password

Either Launch ONCE in debugger with un-commented line

```
19 public UserUpdateRestController(UserDetailsManager userDetailsManager) {  
20     this.userDetailsManager = userDetailsManager;  
21     // for debug (uncomment to execute once, or use curl  
22     //updatePassword(new UserPasswordRequestDTO("user2", "password2"));  
23 }  
  
19 public UserUpdateRestController(UserDetailsManager userDetailsManager) {  
20     this.userDetailsManager = userDetailsManager;  
21     // for debug (uncomment to execute once, or use curl  
22     updatePassword(new UserPasswordRequestDTO(username: "user2", newPassword: "password2"));  
23 }
```



Or using Rest api

```
curl -v -k -u "user1:password1" -H 'Content-Type: application/json' \  
-X POST https://localhost:8443/api/v1/users/update-resalt-password \  
-d '{"username": "user1", "newPassword": "password1"}'
```

check HTTP 200 + Check in Debug Console :

Showing encoded password

The screenshot shows the H2 Database Browser interface. The left sidebar displays database objects: 'jdbc:h2:file:./db' (selected), 'AUTHORITIES', 'USERS', 'INFORMATION_SCHEMA', and 'Users'. A note at the bottom says 'H2 2.3.232 (2024-08-11)'. The main area has a toolbar with various icons and dropdowns for 'Auto commit' (checked), 'Max rows: 1000', 'Auto complete' (Off), and 'Auto select' (On). Below the toolbar is a SQL statement input field containing 'SELECT * FROM USERS'. The results pane shows the output of the query:

USERNAME	PASSWORD	ENABLED
user2	{bcrypt}\$2a\$10\$BNAhHEMM9ihsq/SKTdmX7.yR9vXA2ZKzxhOYvHbkNohskr3BOstnC	TRUE
user1	{noop}password1	TRUE

(2 rows, 3 ms)

[Edit](#)

ANNEXE... Screenshot Debugging "encoders"

The screenshot shows a Java debugger interface with the following details:

- Structure** tab selected.
- Debug DemoApplication** session.
- Threads & Variables** tab selected.
- Evaluate expression (Entrée) or add a watch (Ctrl+Maj+Entrée)**:
 - Selected item: `this = {DelegatingPasswordEncoder$UnmappedIdPasswordEncoder@11080}`
 - Sub-items:
 - `this$0 = {DelegatingPasswordEncoder@11081}`
 - `idPrefix = "{"`
 - `idSuffix = "}"`
 - `idForEncode = "bcrypt"`
 - `passwordEncoderForEncode = {BCryptPasswordEncoder@11088}`
 - `idToPasswordEncoder = {HashMap@11089} size = 14`
 - `"argon2" -> {Argon2PasswordEncoder@11107}`
 - `"pbkdf2" -> {Pbkdf2PasswordEncoder@11109}`
 - `"SHA-1" -> {MessageDigestPasswordEncoder@11111}`
 - `"sha256" -> {StandardPasswordEncoder@11113}`
 - `"pbkdf2@SpringSecurity_v5_8" -> {Pbkdf2PasswordEncoder@11115}`
 - `"scrypt@SpringSecurity_v5_8" -> {SCryptPasswordEncoder@11117}`
 - `"bcrypt" -> {BCryptPasswordEncoder@11088}`
 - `"noop" -> {NoOpPasswordEncoder@11119}`
 - `"ldap" -> {LdapShaPasswordEncoder@11121}`
 - `"SHA-256" -> {MessageDigestPasswordEncoder@11123}`
 - `"argon2@SpringSecurity_v5_8" -> {Argon2PasswordEncoder@11125}`
 - `"MD4" -> {Md4PasswordEncoder@11127}`
 - `"scrypt" -> {SCryptPasswordEncoder@11129}`
 - `"MD5" -> {MessageDigestPasswordEncoder@11131}`

Bottom status bar: `spring-security-crypto-6.4.2-sources.jar > org > springframework > security > crypto > password > DelegatingPasswordEncoder`, `299:1 LF UTF-8 4 spaces`.

ANNEXE Screenshot Debugging authentication

The screenshot shows the IntelliJ IDEA interface during a debugging session of a Spring Security application.

Project Structure: The left sidebar shows the project structure, with the `spring-security-core-6.4.2.jar` library root expanded to show `META-INF`, `org.springframework.security` (containing `access`, `aot.hint`, `authentication`), and `dao` (containing `AbstractUserDetailsAuthenticationProvider`, `DefaultPostAuthenticationCheck`, `DefaultPreAuthenticationCheck`, and `DaoAuthenticationProvider`). The `AbstractUserDetailsAuthenticationProvider` class is selected in the structure view.

Code Editor: The main editor window displays the `AbstractUserDetailsAuthenticationProvider.java` code. The cursor is at line 123, which contains the `@Override` annotation and the `authenticate` method. A tooltip for the `authenticate` method is visible, stating: `Assert.isInstanceOf(UsernamePasswordAuthenticationToken.class, authentication, "UsernamePasswordAuthenticationToken [Principal=use... View")`.

Debug Bar: The bottom navigation bar includes tabs for `Debug` (selected) and `DemoApplication`. Other tabs include `Threads & Variables`, `Console`, `Beans`, `Health`, `Mappings`, and `Environment`.

Breakpoint: A red breakpoint icon is present on line 123.

Variables: The `Threads & Variables` tab shows the current thread: `"https-jsse-nio-8443-exec-45 in group "main": RUNNING`. The variable `this` is shown as `{DaoAuthenticationProvider@11037}`. The variable `authentication` is shown as `{UsernamePasswordAuthenticationToken@11036} "UsernamePasswordAuthenticationToken [Principal=use... View`. The variable `this.messages` is shown as `{MessageSourceAccessor@11039}`.

Bottom Status: The status bar at the bottom shows the file path: `spring-security-core-6.4.2.jar > org > springframework > security > authentication > dao > AbstractUserDetailsAuthenticationProvider > authenticate`, and the terminal settings: `124:1 LF UTF-8 4 spaces`.

Outline

Authentication

Authorization - Permission

Confidentiality/Encryption

Backend-end: storing password in Database

Cryptographic "Hash" function, Salt



Who is "John The Ripper" ?

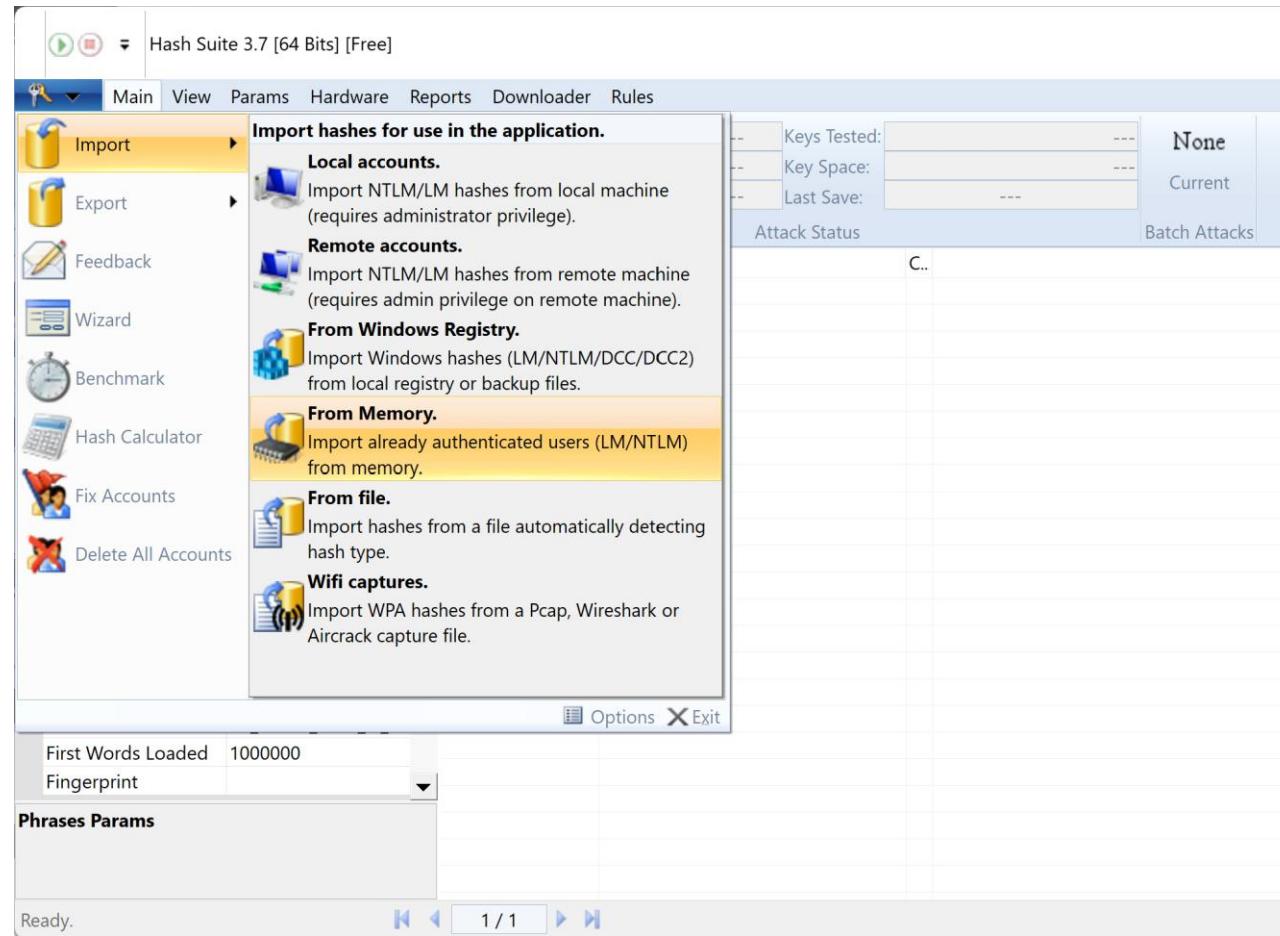
Download John The Ripper / HashSuite

- John the Ripper Pro for Linux
- John the Ripper Pro for macOS
- On Windows, consider Hash Suite (developed by Openwall)
- On Android, consider Hash Suite Droid

<https://hashsuite.openwall.net>

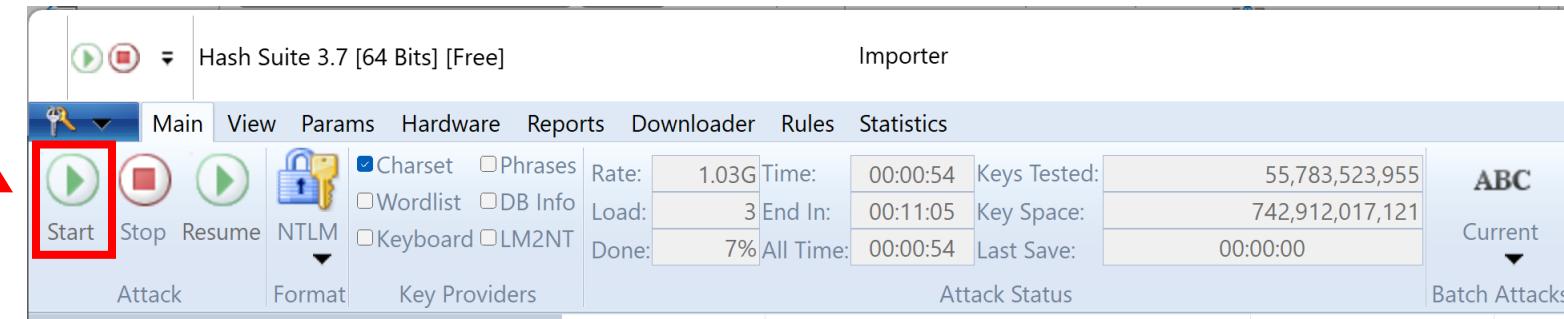
The screenshot shows the Hash Suite website at <https://hashsuite.openwall.net>. The main navigation bar includes Home, FAQ, Tutorial, Performance, Comparison, Download, Android, Changelog, License, and Contact. A banner on the right encourages following @HashSuite on Twitter for new release announcements. Below the banner, a sub-header reads "Fast, powerful, simple". The central content area features a "Home" section with a brief description of Hash Suite's objectives and a list of its features: Fast, Simple and modern, Smart, Powerful, and Scalable. To the right, a "News" sidebar highlights two recent releases: "Hash Suite Droid 1.6" (October 28, 2021) and "Hash Suite 3.7, Droid 1.5.1" (March 19, 2021). Both news items mention the addition of support for Android 11 scoped storage enforcement. Below the news, a link states that Hash Suite 3.7 and Hash Suite Droid 1.5.1 are available for download with SHA256CRYPT and SHA512CRYPT support. A screenshot of the Hash Suite application interface is displayed, showing the main window with tabs for Main, View, Params, Hardware, Reports, Downloader, and Rules. The Main tab is active, showing various attack parameters like Charset, Phrases, Wordlist, DB Info, Keyboard, LM2NT, and attack metrics like Rate, Load, and Keys Tested. Below the interface, a table titled "Key Provider Params" lists "Charset Params" for "hcookis" with values Minimum Size 0 and Maximum Size 6, and "Hash" and "Cleartext" columns showing the hash F83C01861FDD23B4354465FE6D7F6402 and the cleartext nurse.

Step 1/ Import Your Windows Hashed Password From Memory



Step 2/ Test your password strength

Run 5-10mn (default settings)



Step 3/ Edit "word.lst" containing some words (maybe known internet password databases) RE-Run 5-10mn with custom rules + wordlist

Restrict searches
with hypothesis

Add known words
in dictionary

The screenshot shows the Hash Suite 3.7 [64 Bits] [Free] application. The main window has tabs for Main, View, Params, Hardware, Reports, Downloader, Rules, and Statistics. The Statistics tab is selected. Below the tabs, there are summary counts: Users Added: 5, Lines Skipped: 0, Completion: 100%, Added: 0, Disable: 5, Exist: 0, and a status message 'Imported' with a red X icon.

The central area displays a table of hashes with columns for Username, Hash, and Cleartext. The Hash column for several entries is highlighted in yellow. The table includes rows for Administrateur, arnaud, DefaultAccount, Invité, and WDAGUtilityAcc... with their respective hash lengths (e.g., 31, 49, 31, 31, 26) and cleartext representations.

On the left, the 'Key Provider Params' section is expanded, showing:

- Charset Params**: Minimum Size: 0, Maximum Size: 6, Use rules (highlighted with a red box).
- Charsets**: Lower, Upper, Digit, Symbol (all checked). A red arrow points from the 'Wordlist Params' section below to the 'Symbol' setting.
- Wordlist Params** (highlighted with a red box).
- Keyboard Params**, **Phrases Params**, **DB Info Params**, **LM2NT Params**, **Rules**.

On the right, another 'Key Provider Params' section is shown, with the 'Wordlist Params' section expanded:

- Symbol**: !@#\$%^&*()_-+=~`[]{};:"...
- Wordlist Params**: Minimum Size: 1, Maximum Size: 6, Use rules (highlighted with a red box), Wordlist: wordlist_small.lst (21.8 KB).
- Keyboard Params**, **Phrases Params**, **DB Info Params**, **LM2NT Params**, **Rules**.

Summary

Authentication	 Transmit Password to Http
Authorization - Permission	password=>authentication=>role
Confidentiality/Encryption	password not in clear over internet
Backend-end: storing password in Database	storing password ...
Cryptographic "Hash" function, Salt	... storing hashed+salted password
Who is "John The Ripper" ?	finding password from hashed

Questions ?

arnaud.nauwynck@gmail.com