

Security & Privacy

# TD2 (Password Management)

Esilv 2025

[arnaud.nauwynck@gmail.com](mailto:arnaud.nauwynck@gmail.com)

# Outline ( = Same as Course)

Http Authorization

Transport: Need For confidentiality

Backend-end: storing password in Database ?

Cryptographic "Hash" functions, Salt

Who is "John The Ripper" ?

2FA & MFA (Multi Factor Authentication)

# Outline



## Http Authorization

Transport: Need For confidentiality

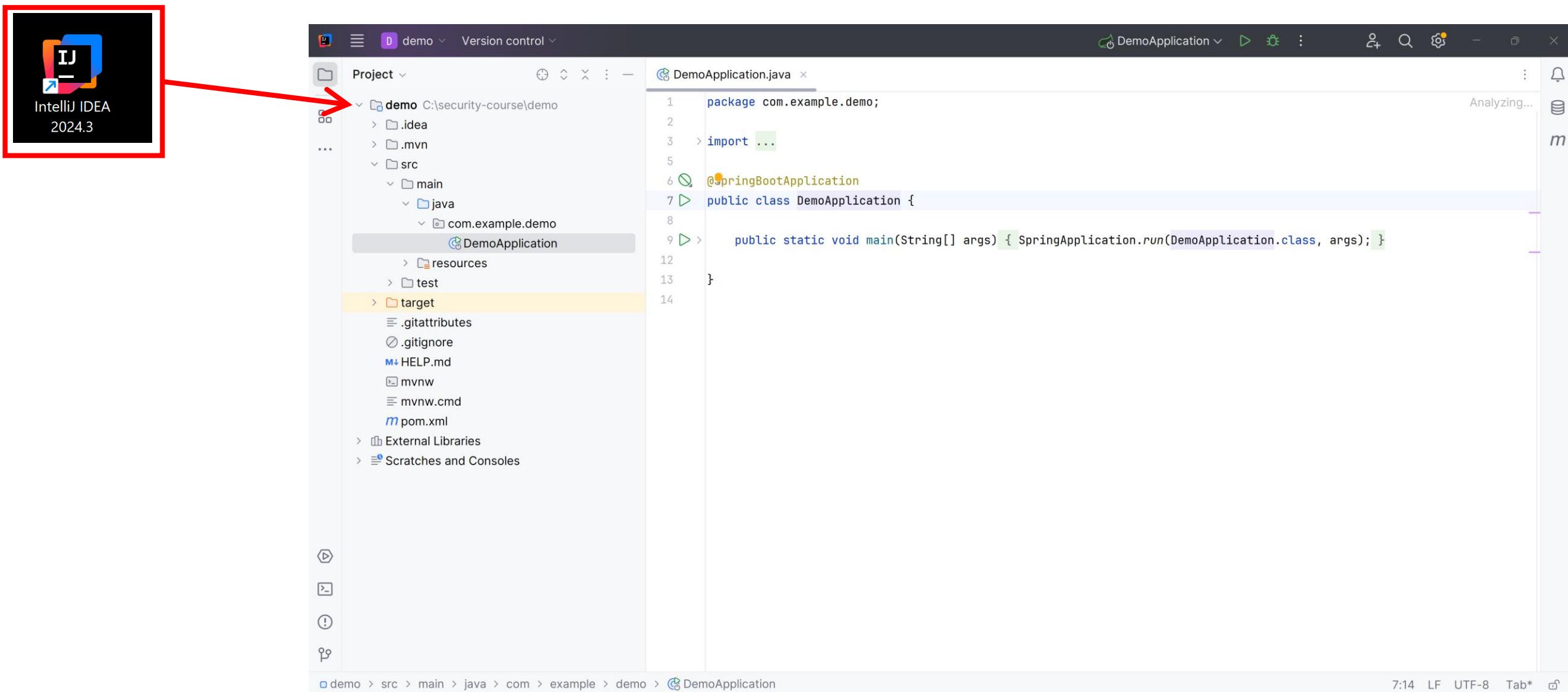
Backend-end: storing password in Database ?

Cryptographic "Hash" functions, Salt

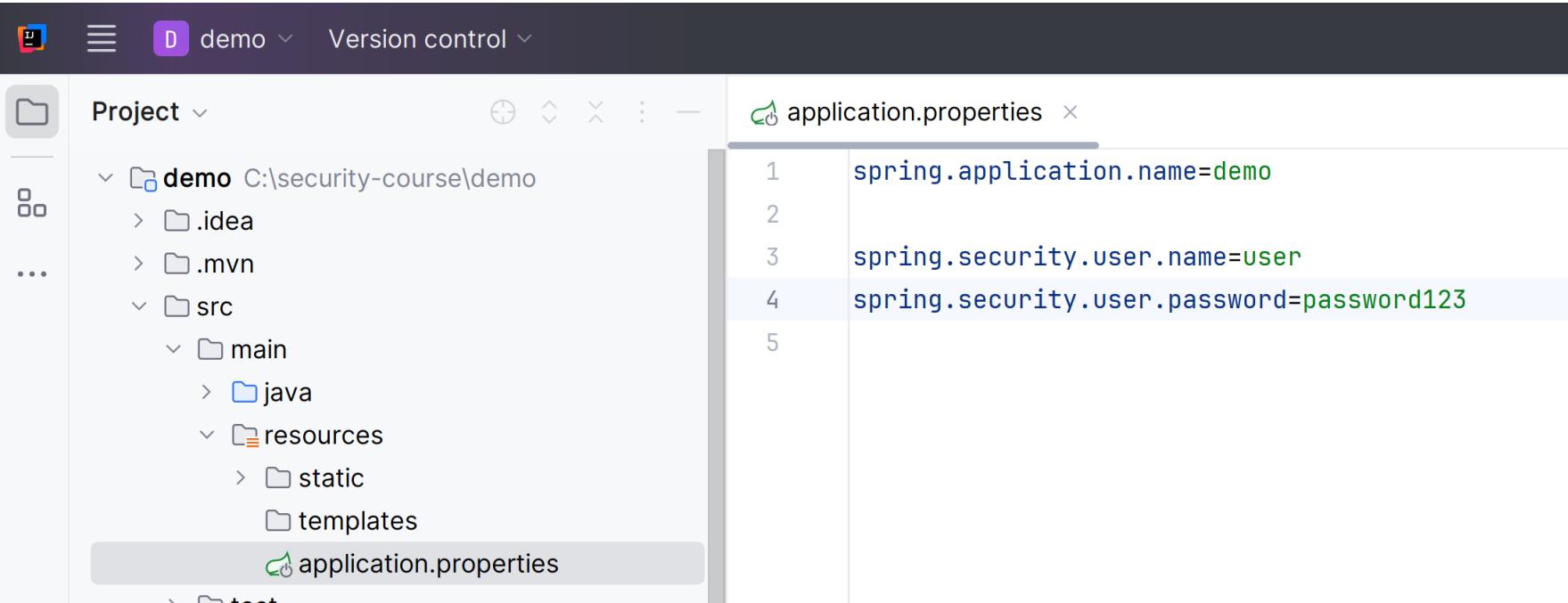
Who is "John The Ripper" ?

2FA & MFA (Multi Factor Authentication)

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



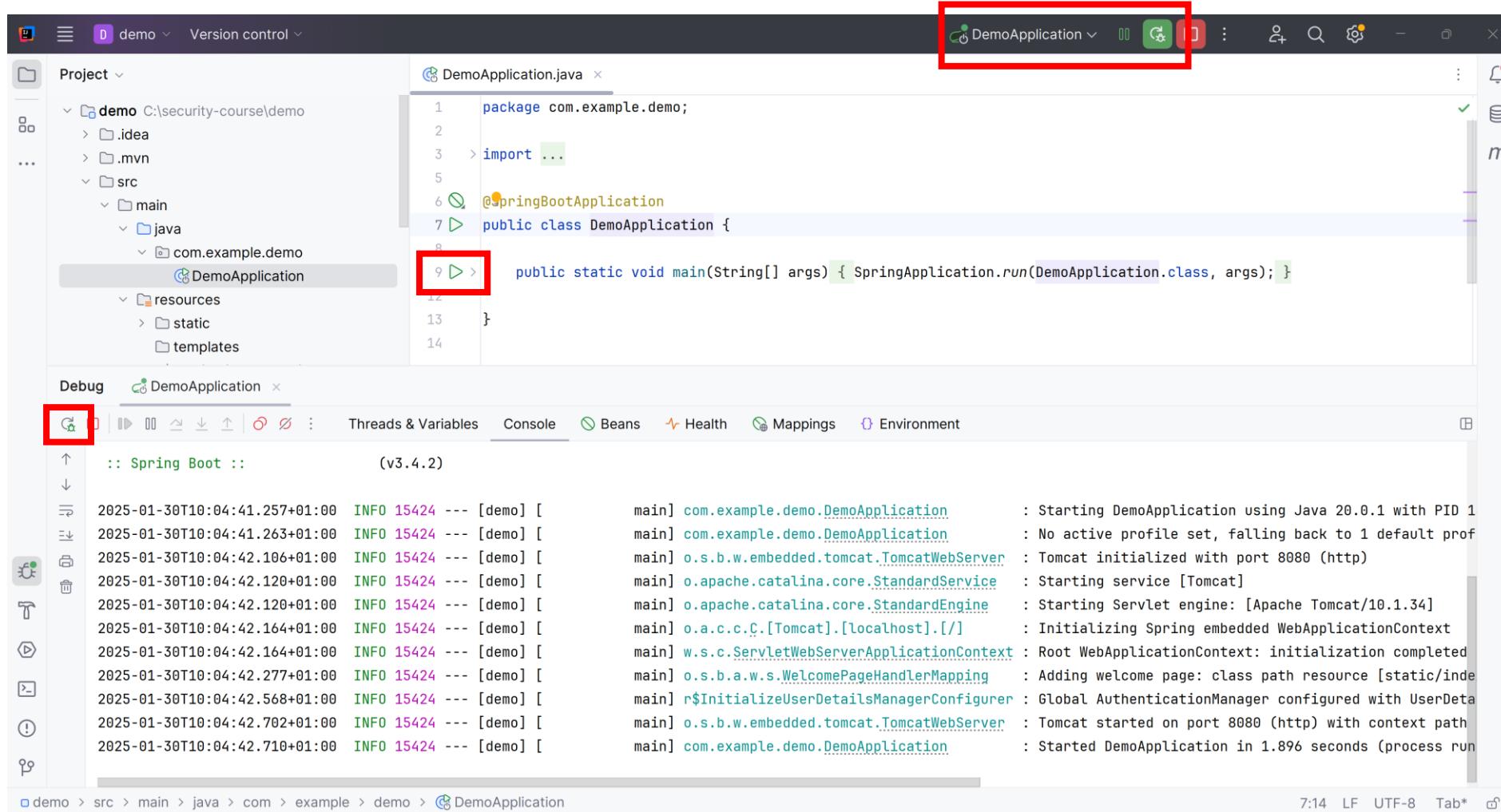
# Pre-Requisite Step 2: check file src/main/resources/application.properties for "user" / "password123"



The screenshot shows a Java project structure in an IDE. The project is named 'demo' and is located at 'C:\security-course\demo'. It contains .idea, .mvn, src, main, java, resources, static, templates, and application.properties files. The application.properties file is open in the editor, displaying the following configuration:

```
spring.application.name=demo
spring.security.user.name=user
spring.security.user.password=password123
```

# Pre-Requisite Step 3: relaunch webapp server



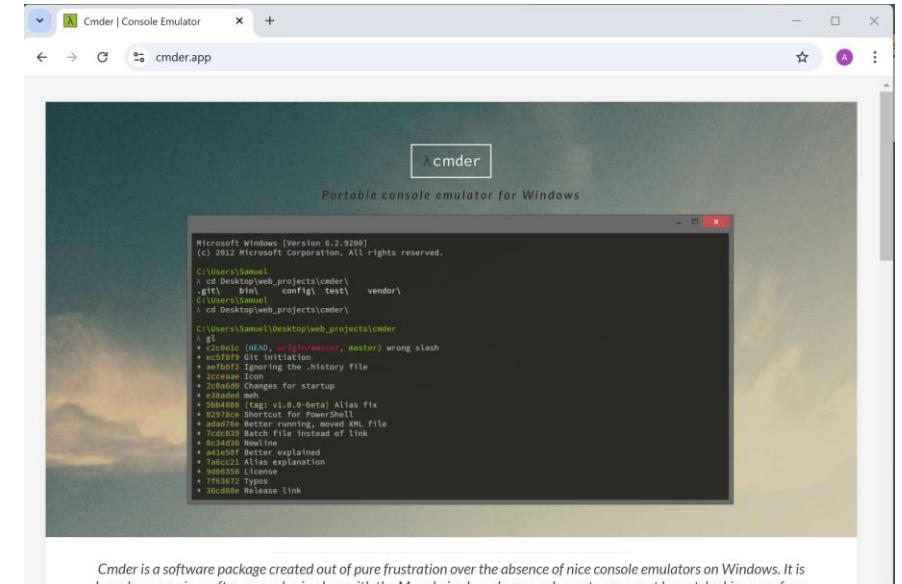
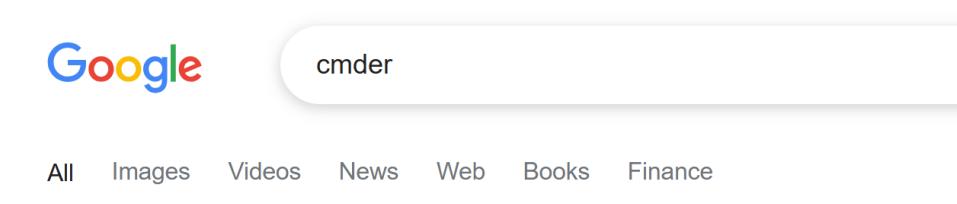
# Pre-Requisite : Install (unix) shell tools "curl" and "base64"

on linux => built-in tools on system

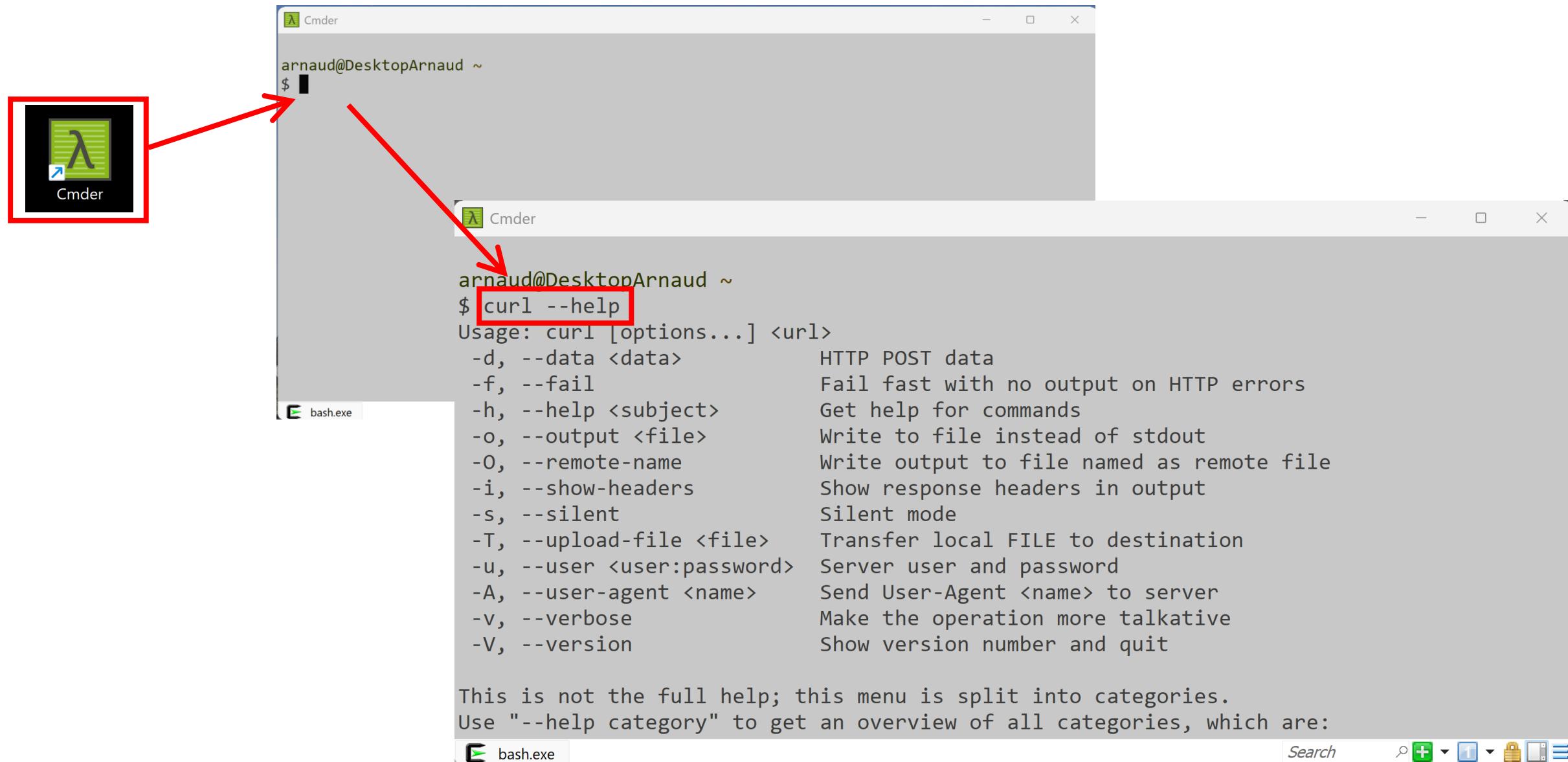
on mac => same (or use "brew install" ?)

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

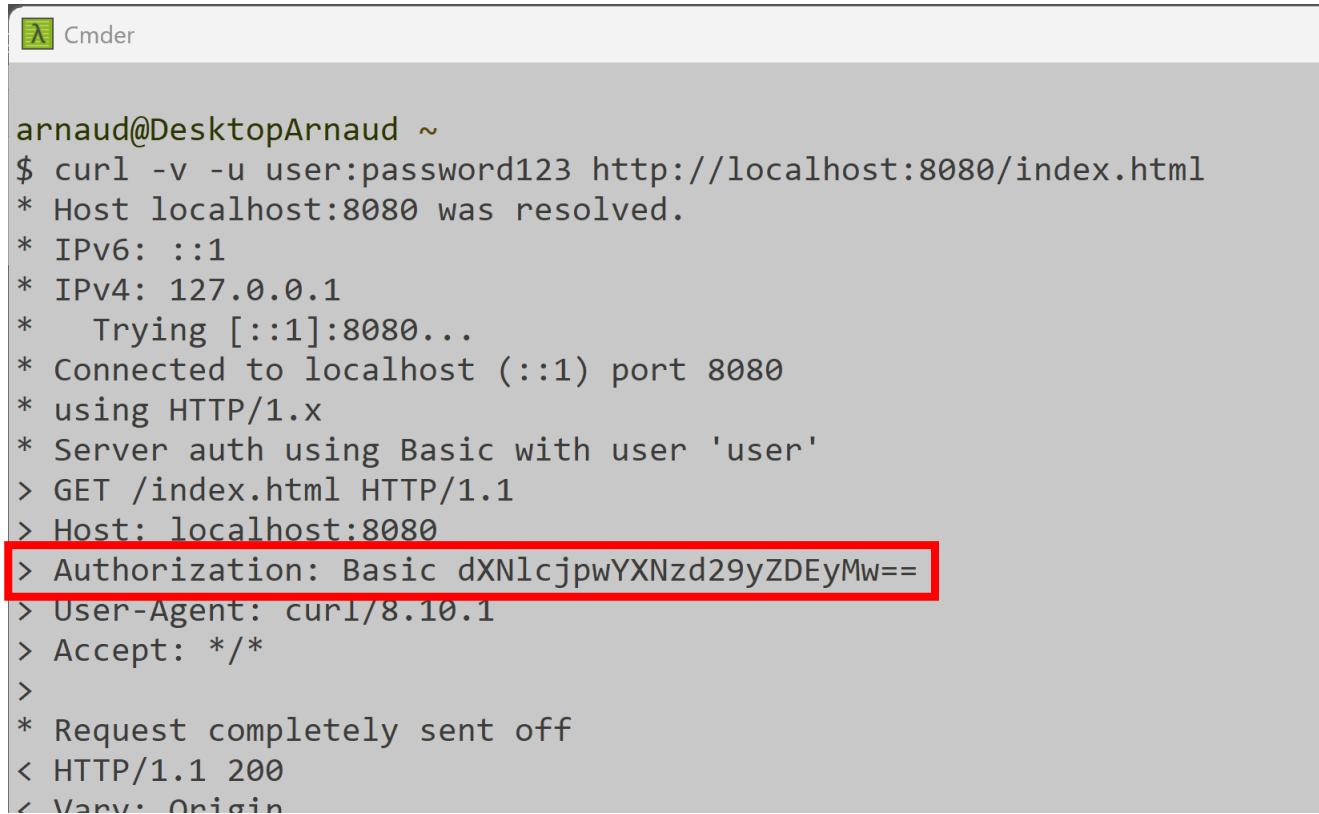
## 200, 301, 400, 403, 404, ...

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
<
```



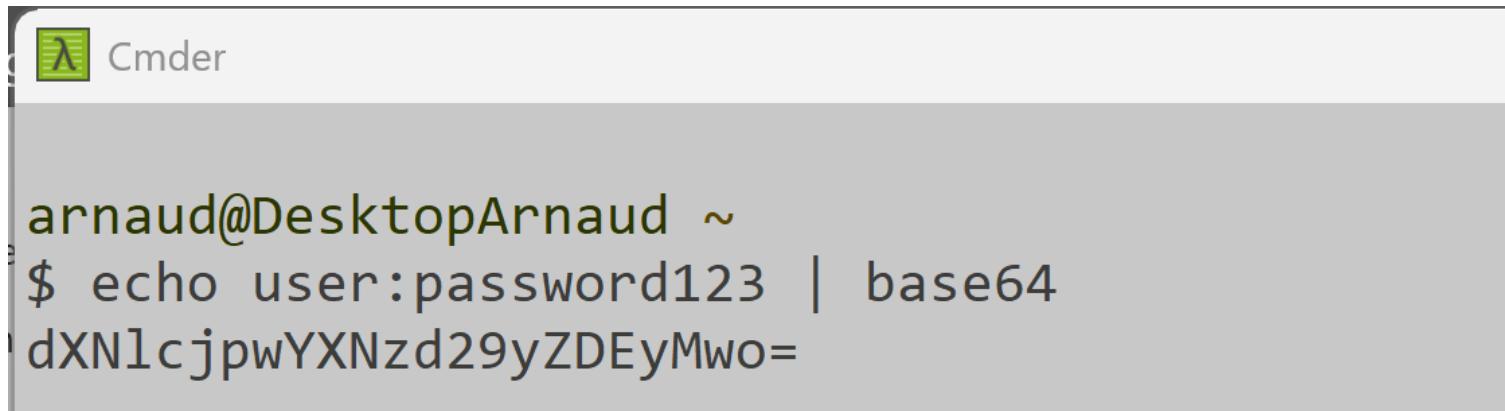
```
curl -v -u "user:password123"
=> see Http Header "Authorization: Basic ..."
```



```
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
```

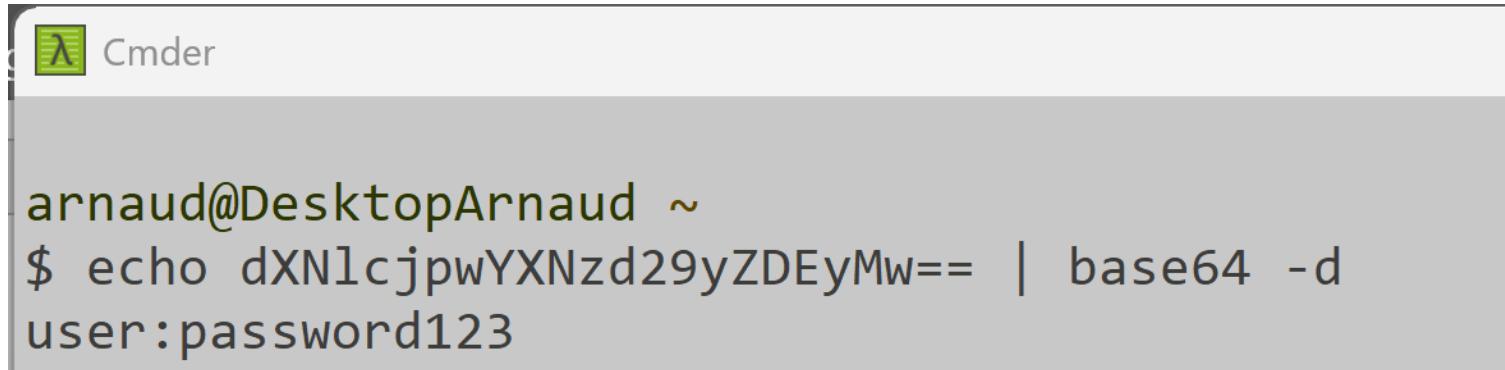
# Encode / Decode base64 password, using "base64 -d"

```
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

Http Authorization



**Transport: Need For confidentiality**

Backend-end: storing password in Database ?

Cryptographic "Hash" functions, Salt

Who is "John The Ripper" ?

2FA & MFA (Multi Factor Authentication)

# 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"

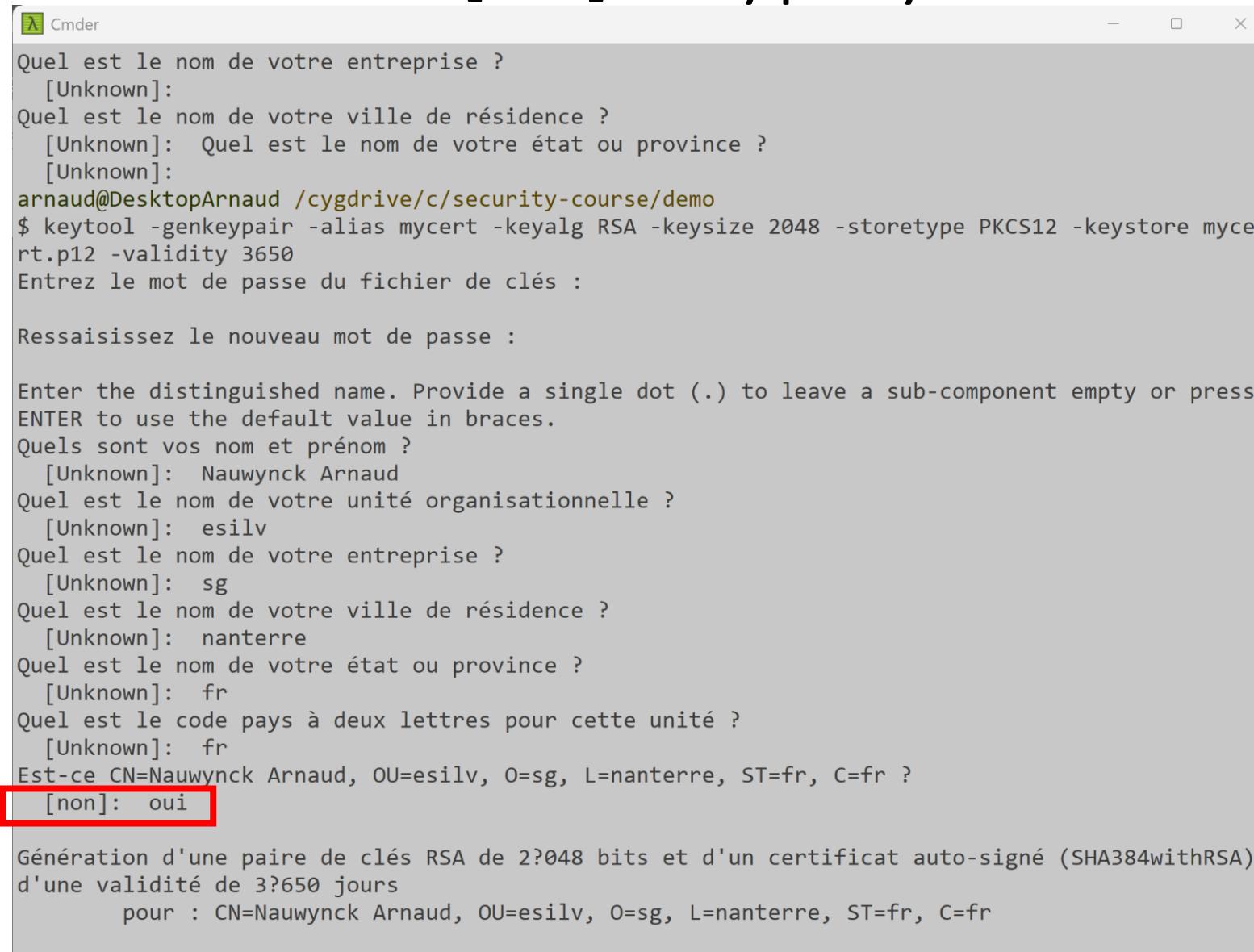
... to fix, use windows "env" to edit your environement variables, or export PATH="\$PATH:\$JAVA\_HOME/bin"



```
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 !!



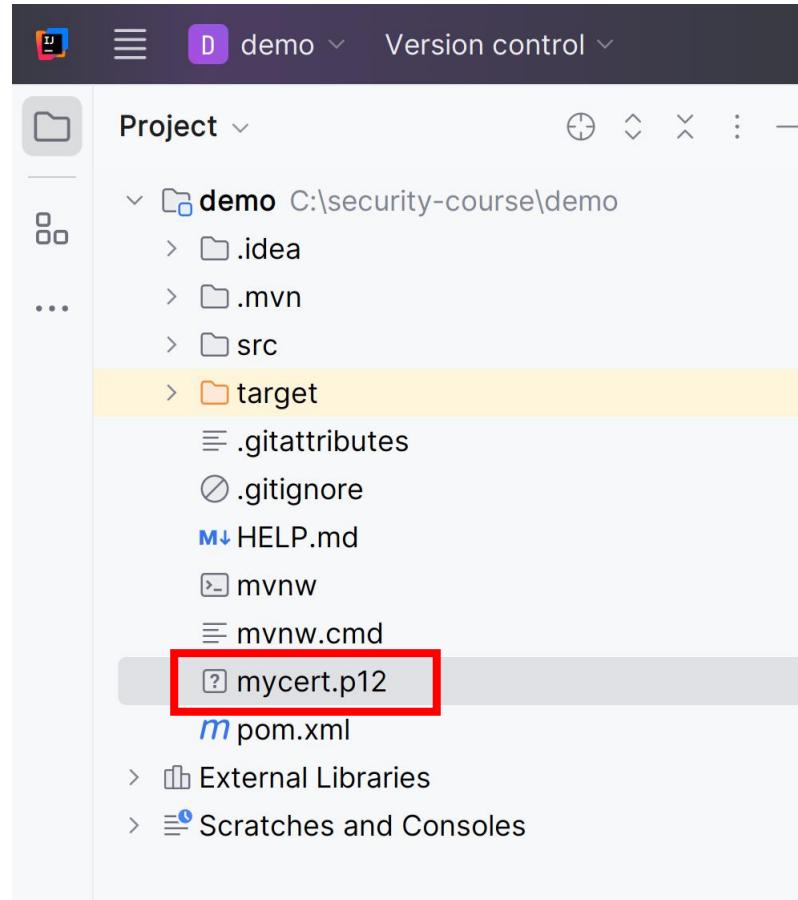
```
Quel est le nom de votre entreprise ?
[Unknown]:
Quel est le nom de votre ville de résidence ?
[Unknown]: Quel est le nom de votre état ou province ?
[Unknown]:
arnaud@DesktopArnaud /cygdrive/c/security-course/demo
$ keytool -genkeypair -alias mycert -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore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

Ressaisissez le nouveau mot de passe :

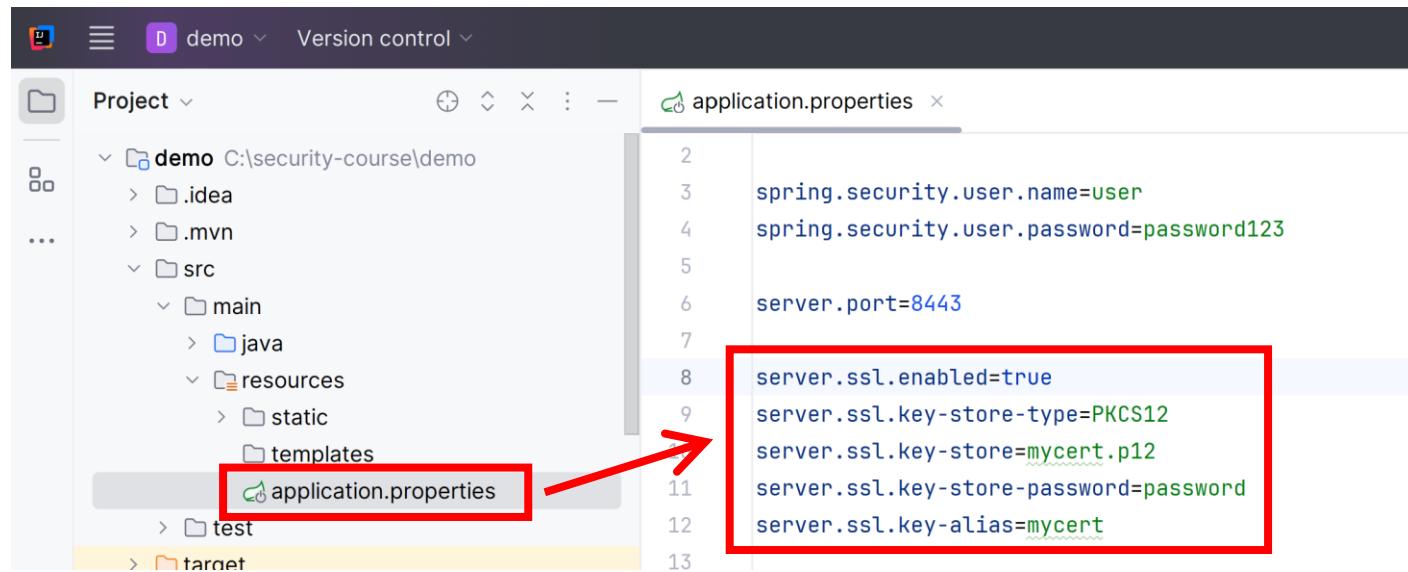
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press
ENTER to use the default value in braces.
Quels sont vos nom et prénom ?
[Unknown]: Nauwynck Arnaud
Quel est le nom de votre unité organisationnelle ?
[Unknown]: esilv
Quel est le nom de votre entreprise ?
[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 are 'java', 'resources', 'static', and 'templates' folders. The 'application.properties' file is located in the 'resources' folder. A red box highlights the 'application.properties' file in the file tree. A red arrow points from this highlighted file to the content of the file in the main editor area. The file content is as follows:

```
spring.security.user.name=user
spring.security.user.password=password123
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
```

**server.port=8443**

**server.ssl.enabled=true**

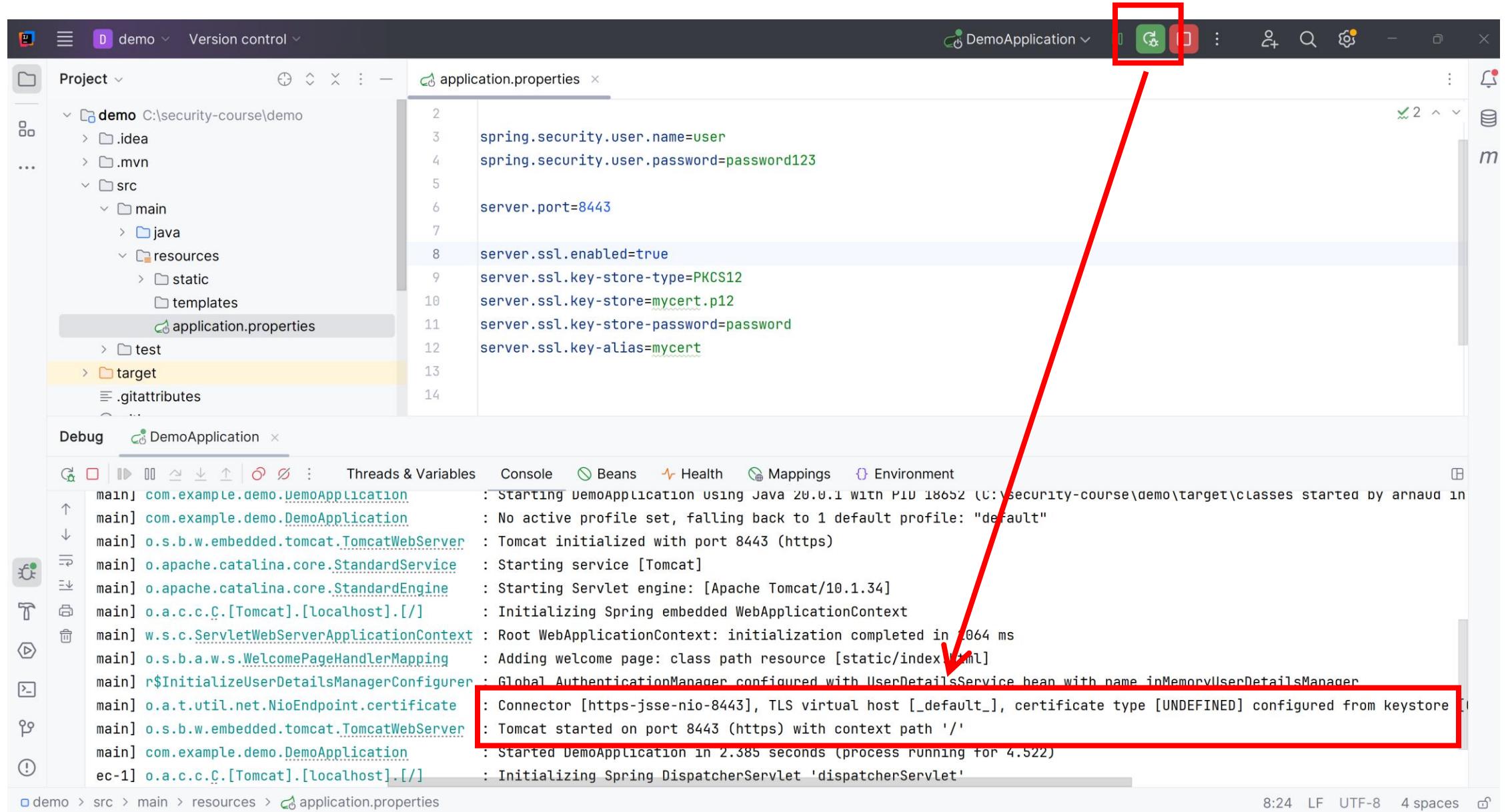
**server.ssl.key-store-type=PKCS12**

**server.ssl.key-store=mycert.p12**

**server.ssl.key-store-password=password**

**server.ssl.key-alias=mycert**

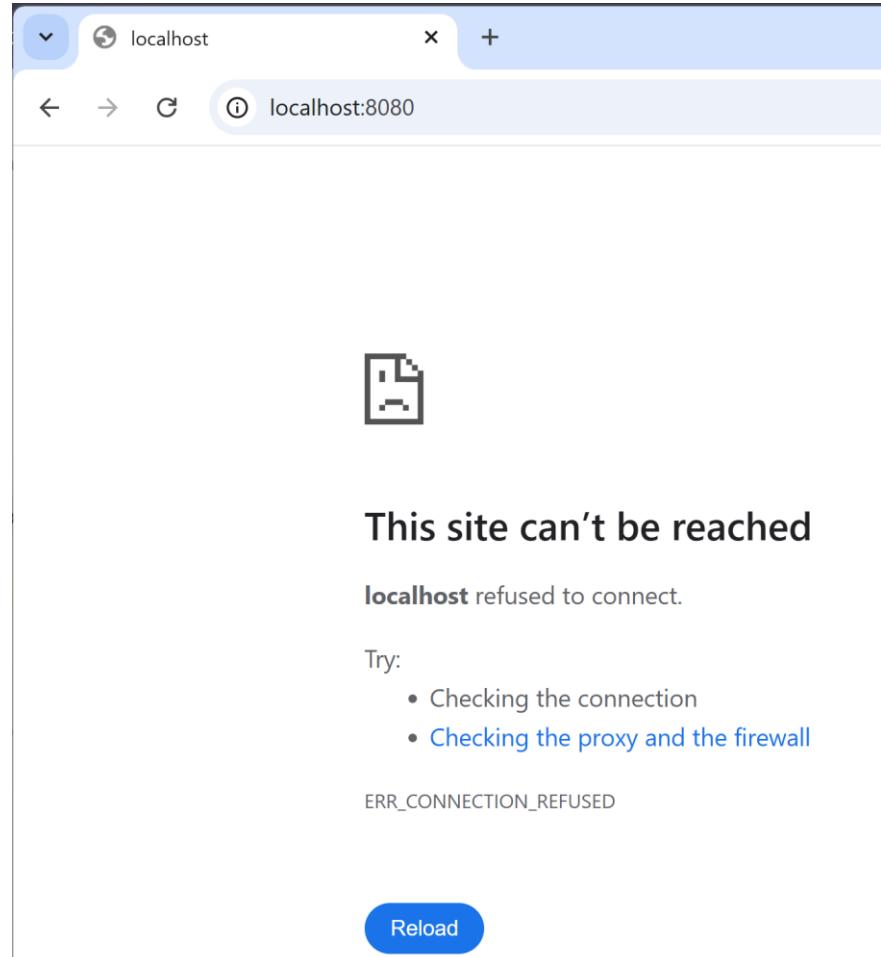
# Relaunch server, check new console log



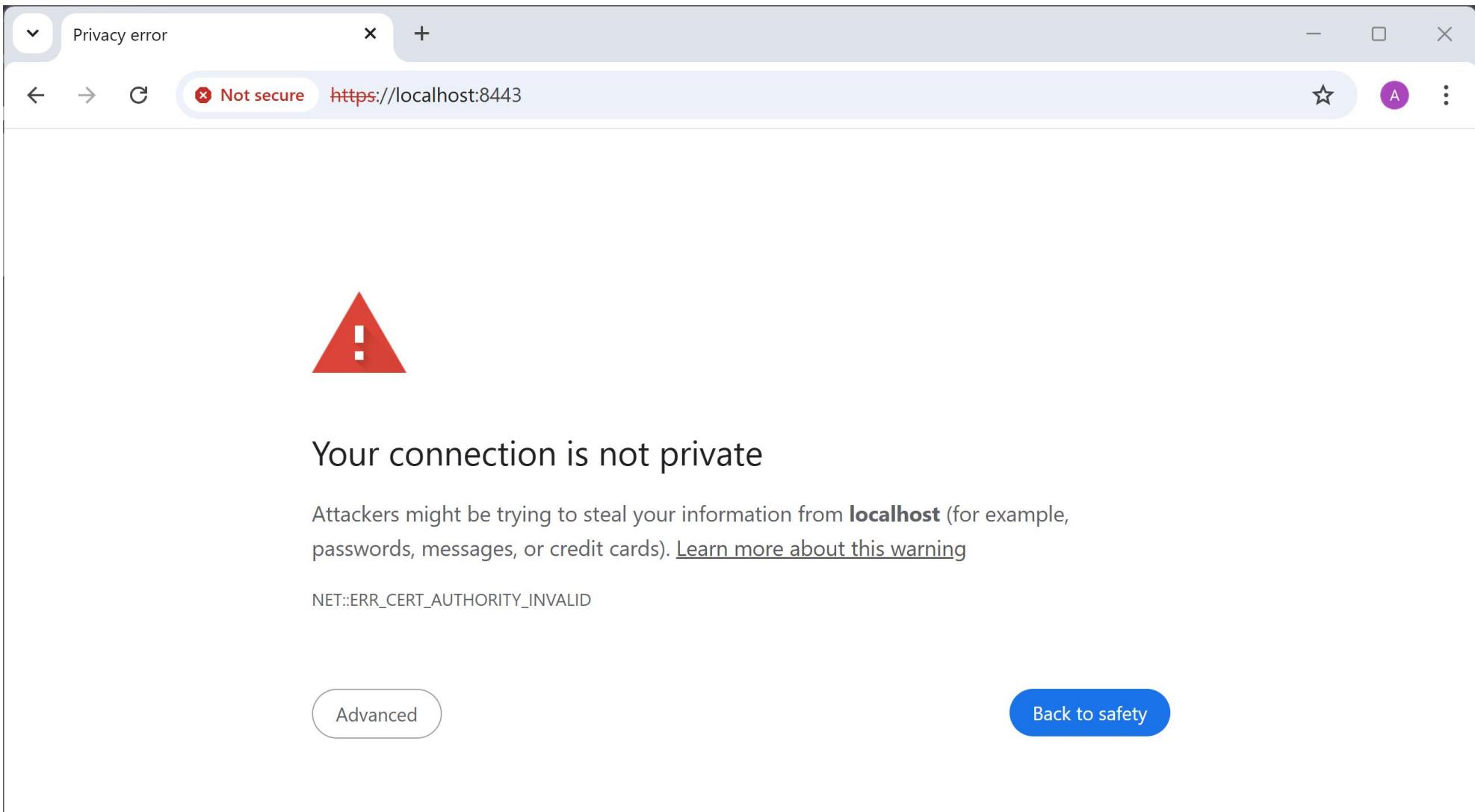
port changed 8080 (default for http)  
-> 8443 (default for https)

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)

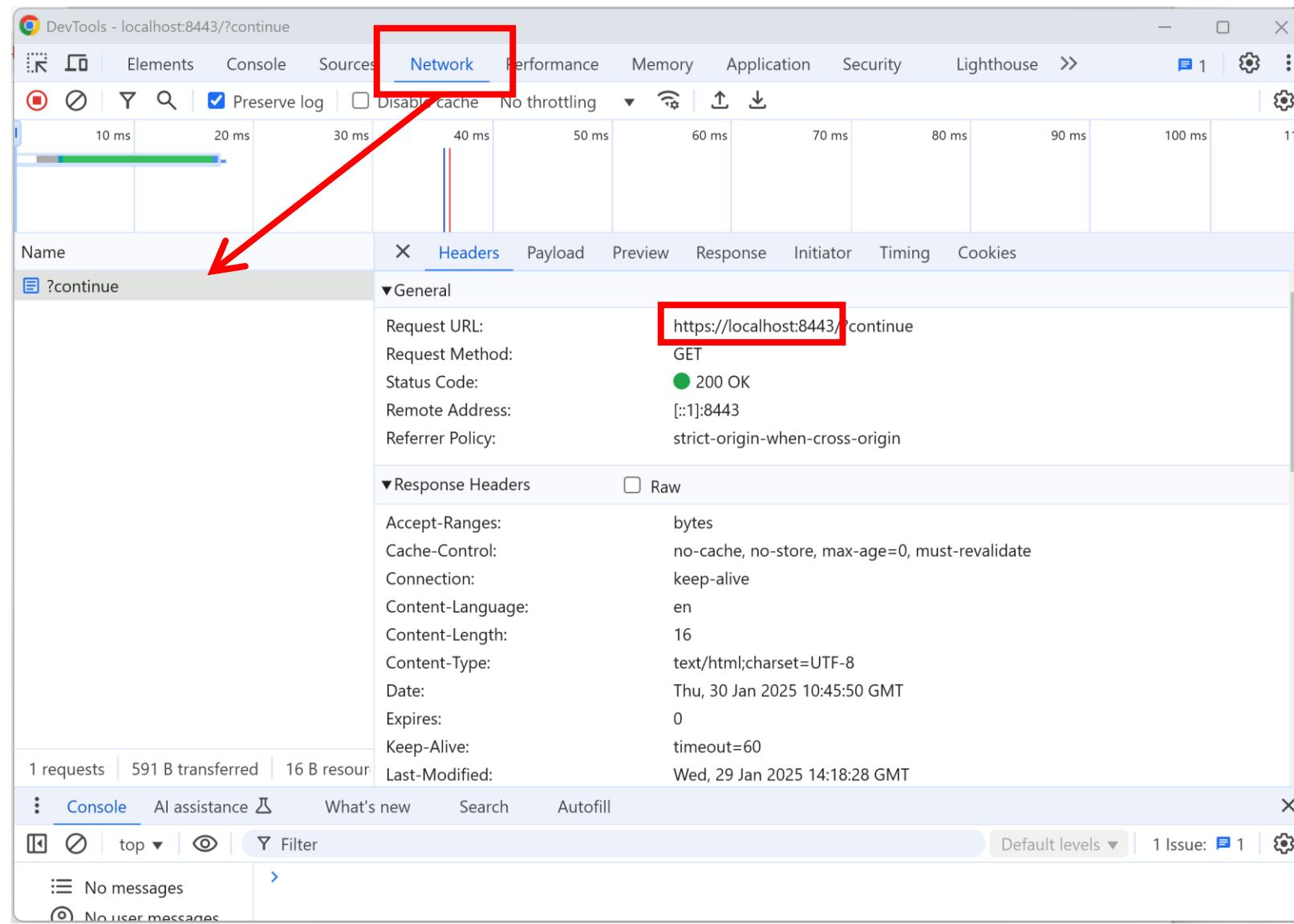
The image shows a web browser window with two tabs and a separate sign-in form.

**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:

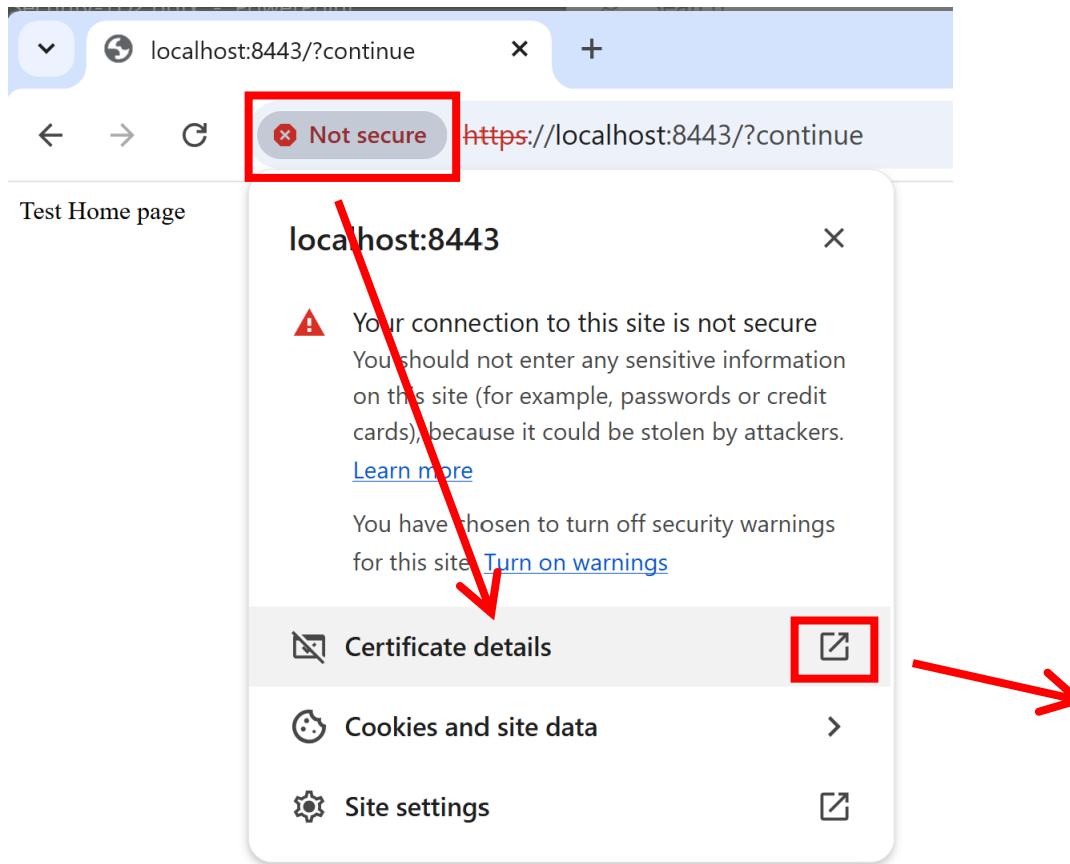
- ⚠ Certificate - Subject Alternative Name missing**: The certificate for this site does not contain a Subject Alternative Name extension containing a domain name or IP address. A "View certificate" button is provided.
- ⚠ Certificate - missing**: This site is missing a valid, trusted certificate (net::ERR\_CERT\_AUTHORITY\_INVALID).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

At the bottom, the DevTools footer includes links for Console, AI assistance, What's new, Search, Autofill, and a message center showing "No messages" and "No user messages".

# See Also Chrome -> Certificate Details



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

# Outline

Http Authorization

Transport: Need For confidentiality



**Backend-end: storing password in Database ?**

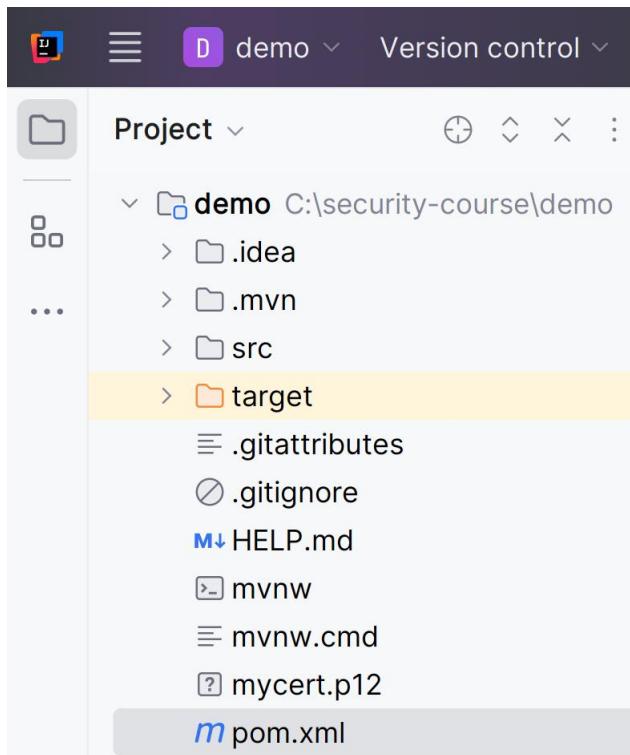
Cryptographic "Hash" functions, Salt

Who is "John The Ripper" ?

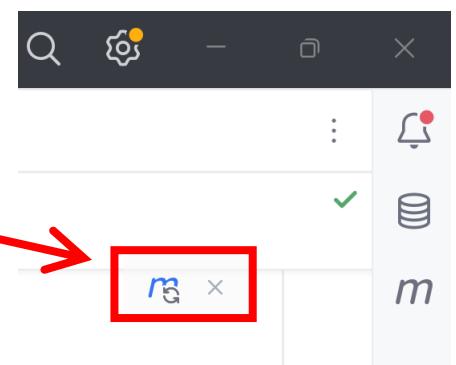
2FA & MFA (Multi Factor Authentication)

# 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 and highlighted with a grey background. The main editor area shows the content 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
```

```
# 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;
    }

}
```

# Allow access to "/h2-console"

Add Java Class "SecurityConfiguration"

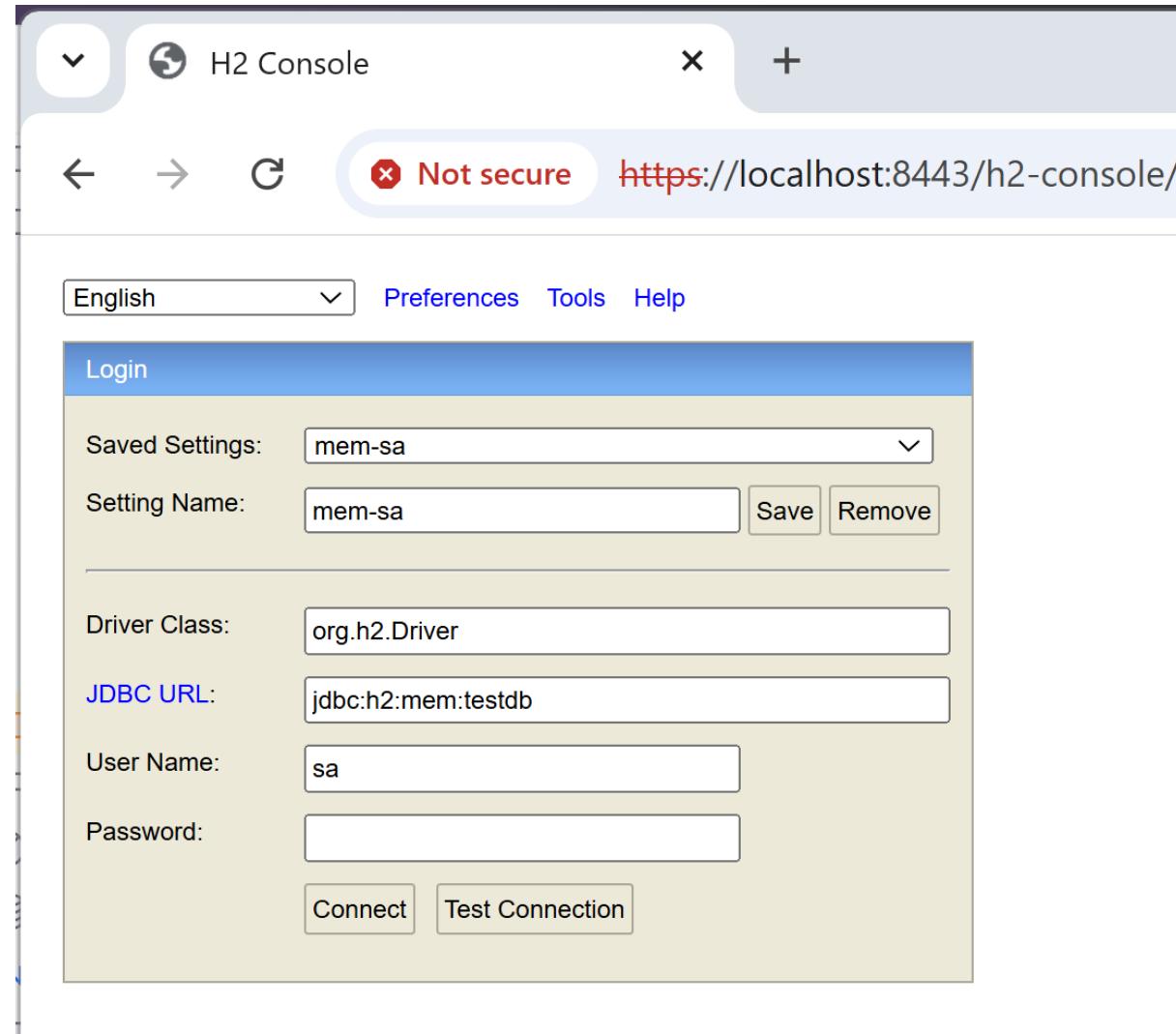
```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.configuration.WebSecurityCustomizer;

@Configuration
public class SecurityConfiguration {

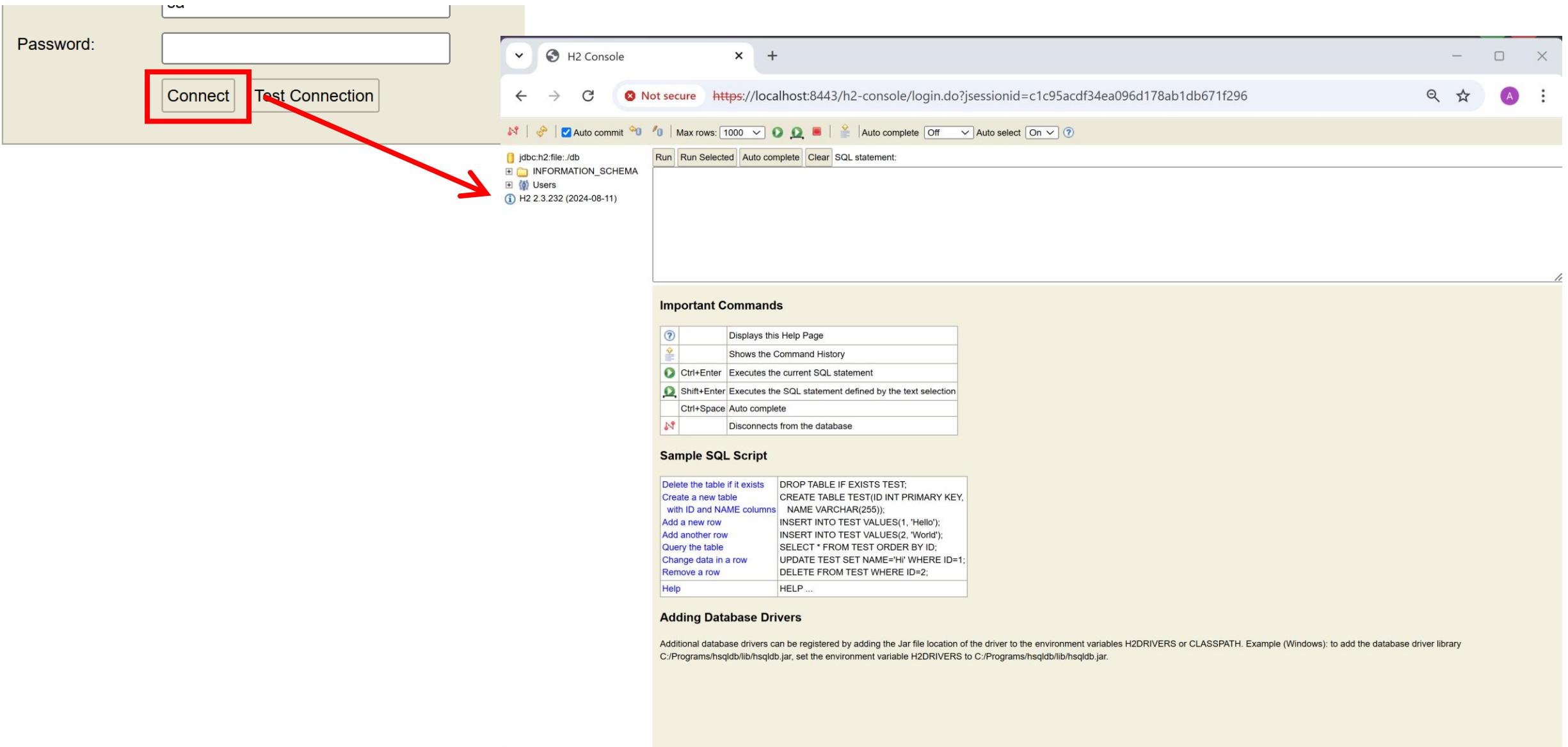
    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        return (web) -> web.ignoring()
            .requestMatchers("/h2-console/**");
    }

}
```

# 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](https://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. The title bar says "H2 Console". The address bar shows "Not secure https://localhost:8443/h2-console/login.do?jsessionid=a603020da023803cd40eaa3b53168820". The toolbar includes "Run" (highlighted with a red box and arrow), "Run Selected", "Auto complete", "Clear", and "SQL statement:". The left sidebar lists database objects: "jdbc:h2:file:/db", "AUTHORITIES", "USERS", "INFORMATION\_SCHEMA", and "Users". The main area displays the following SQL code:

```
enable 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)
);
create unique index ix_auth_username on authorities (username,authority);

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)

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)

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 a database using the H2 "New Row", edit, commit feature. It consists of three panels:

- Panel 1 (Top Left):** Shows the database structure with the **USERS** table selected. A red box highlights the **USERS** table in the tree view. A red arrow points from this panel to the **Edit** button in Panel 2.
- Panel 2 (Bottom Left):** Displays the result of the `SELECT * FROM USERS;` query. It shows two rows: `user1`, `password1`, `TRUE` and `user2`, `password2`, `TRUE`. A red box highlights the **Edit** button at the bottom left of the results table.
- Panel 3 (Bottom Right):** Shows the `@edit SELECT * FROM USERS;` command with the results table. The first row (`user1`) has a delete icon (red X) in the Action column. The second row (`user2`) has a green plus icon in the Action column. A red box highlights the green plus icon. A red arrow points from the green plus icon in Panel 3 to the green plus icon in Panel 2.



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)**

The screenshot shows the H2 Console interface running in a web browser at <https://localhost:8443/h2-console/login.do?jsessionid=bb660cf37e72b9dcb4dd56287dd>. The interface includes a sidebar with database objects like AUTHORITIES, USERS, INFORMATION\_SCHEMA, and Users. The main area contains the SQL statement:

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

The 'Run' button is highlighted in yellow.

# 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 pane contains a SQL editor with the query `SELECT * FROM AUTHORITIES`. Below the editor is a table with two rows:

Action	USERNAME	AUTHORITY
	user1	ROLE_USER
	user2	ROLE_MANAGER

(2 rows, 1 ms)

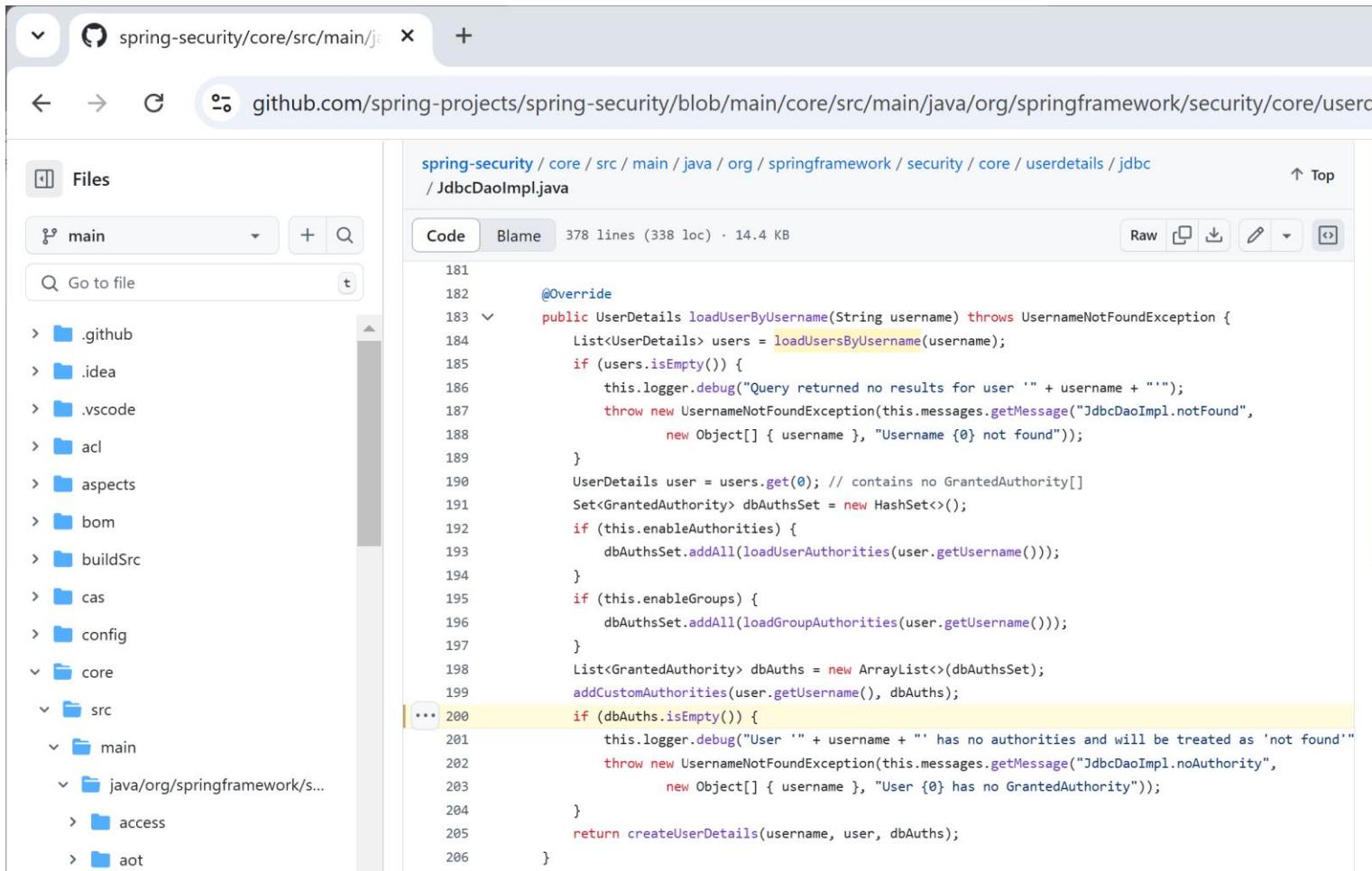
```
INSERT INTO authorities (username, authority) VALUES ('user1', 'ROLE_USER')
```

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

# ⚠ When NO role => user are disabled (!)

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 browser interface. The address bar displays the URL: `https://github.com/spring-projects/spring-security/blob/main/core/src/main/java/org/springframework/security/core/userdetails/jdbc/JdbcDaoImpl.java#L200`. The left sidebar shows the project structure under the `main` branch, including folders like `.github`, `.idea`, `.vscode`, `acl`, `aspects`, `bom`, `buildSrc`, `cas`, `config`, `core`, `src`, `main`, and `java/org/springframework/s...`. The right panel shows the Java code for `JdbcDaoImpl.java`. The code is annotated with several yellow highlights, notably around line 200 and line 203. Line 200 contains the condition `if (dbAuths.isEmpty()) {`. Line 203 contains the throw statement `throw new UsernameNotFoundException(this.messages.getMessage("JdbcDaoImpl.noAuthority", new Object[] { username }, "User {0} has no GrantedAuthority"));`. The code is part of the `loadUserByUsername` method.

```
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$CompositeFilterChainProxy.doFilter(WebMvcSecurityConfiguration.java:414)
2025-01-30T16:13:13.137+01:00 ERROR 21412 --- [demo] [io-8443-exec-10] o.a.c.c.C.[.].[dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in
java.lang.IllegalArgumentException Create breakpoint : Given that there is no default password encoder configured, each password must have a password encoding prefix. Please
at org.springframework.security.crypto.password.DelegatingPasswordEncoder$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

Http Authorization

Transport: Need For confidentiality

Backend-end: storing password in Database ?



Cryptographic "Hash" functions, Salt

Who is "John The Ripper" ?

2FA & MFA (Multi Factor Authentication)

# Adding Rest java class for Re-Salting

```
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;
    // may create using @Configuration + @Bean
    private final PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
```

(...next)

```
public UserUpdateRestController(UserDetailsManager userDetailsManager) {  
    this.userDetailsManager = userDetailsManager;  
    // for debug  
    // updateResaltPassword("user2");  
}  
  
@PutMapping("/update-resalt-password")  
public void updateResaltPassword(  
    @RequestBody String username) {  
    UserDetails userDetails = userDetailsManager.loadUserByUsername(username);  
    String oldPassword = userDetails.getPassword();  
  
    String hashedPassword = passwordEncoder.encode(oldPassword);  
  
    System.out.println("updating user (resalting password): old '" + oldPassword + "' => new '" + hashedPassword + "'");  
    UserDetails updatedUser = User.withUserDetails(userDetails).password(hashedPassword).build();  
    userDetailsManager.updateUser(updatedUser);  
}  
}
```

# Re-Salting an already saved {noop}password

```
2025-01-30T17:36:36.125+01:00  INFO 19156 --- [demo] [           main] o.s.s.p.JdbcUserDetailsManager : No authentication manager set. Reauthentication of us
updating user (resalting password): old '{noop}password2' => new '$2a$10$MInVR3CwkgonDtdxGdAyQeTv0njq89kKFKZLiiH668.ZzlbHiHacm'
2025-01-30T17:34:44.000+01:00  WARN 19156  [demo] [           main] InMemoryConfiguration$InMemoryConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during a view's lifecycle. If this causes issues, it can be disabled by setting spring.jpa.open-in-view=false.
```

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

It saves both salt + hashed password in field "password" (separated with ".")

# Outline

Http Authorization

Transport: Need For confidentiality

Backend-end: storing password in Database ?

Cryptographic "Hash" functions, Salt



**Who is "John The Ripper" ?**

2FA & MFA (Multi Factor Authentication)













# Outline

Http Authorization

Transport: Need For confidentiality



**Backend-end: storing password in Database ?**

Cryptographic "Hash" functions, Salt

Who is "John The Ripper" ?

2FA & MFA (Multi Factor Authentication)



























































































