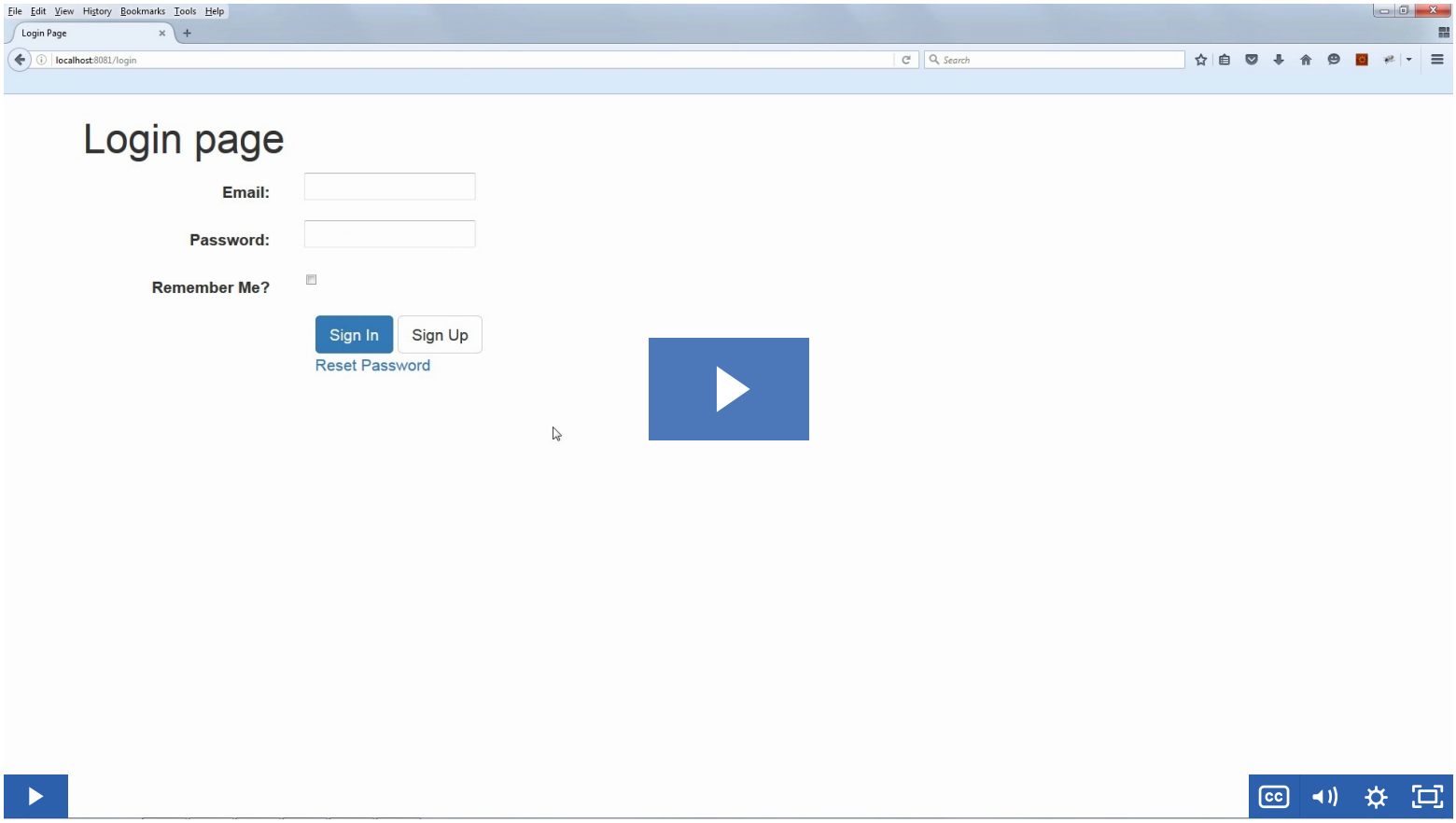
Lesson 1: A Simple Remember Me Flow



1. Main Goal

The focus here is to implement a very simple remember-me flow for our login page.

2. Lesson Notes

If you're using the git repository to get the project - you should be using the module3 branch.

The relevant module you need to import when you're starting with this lesson is: m3-lesson1

If you want to skip and see the complete implementation, feel free to jump ahead and import: m3-lesson2

The credentials used in the code of this lesson are: test@email.com/pass (data.sql).

First, we're going to enable the basic remember-me functionality in the security config:

```
.rememberMe().key("lssAppKey")
```

We are then going to add the remember-me checkbox on the login page:

```
<input id="remember" type="checkbox" name="remember-me" value="true" />
```

Finally, we are going to test by following two scenarios:

Scenario 1:

- log in without remember-me
- remove the JSESSIONID cookie manually
- refresh the page => we should be redirected to login

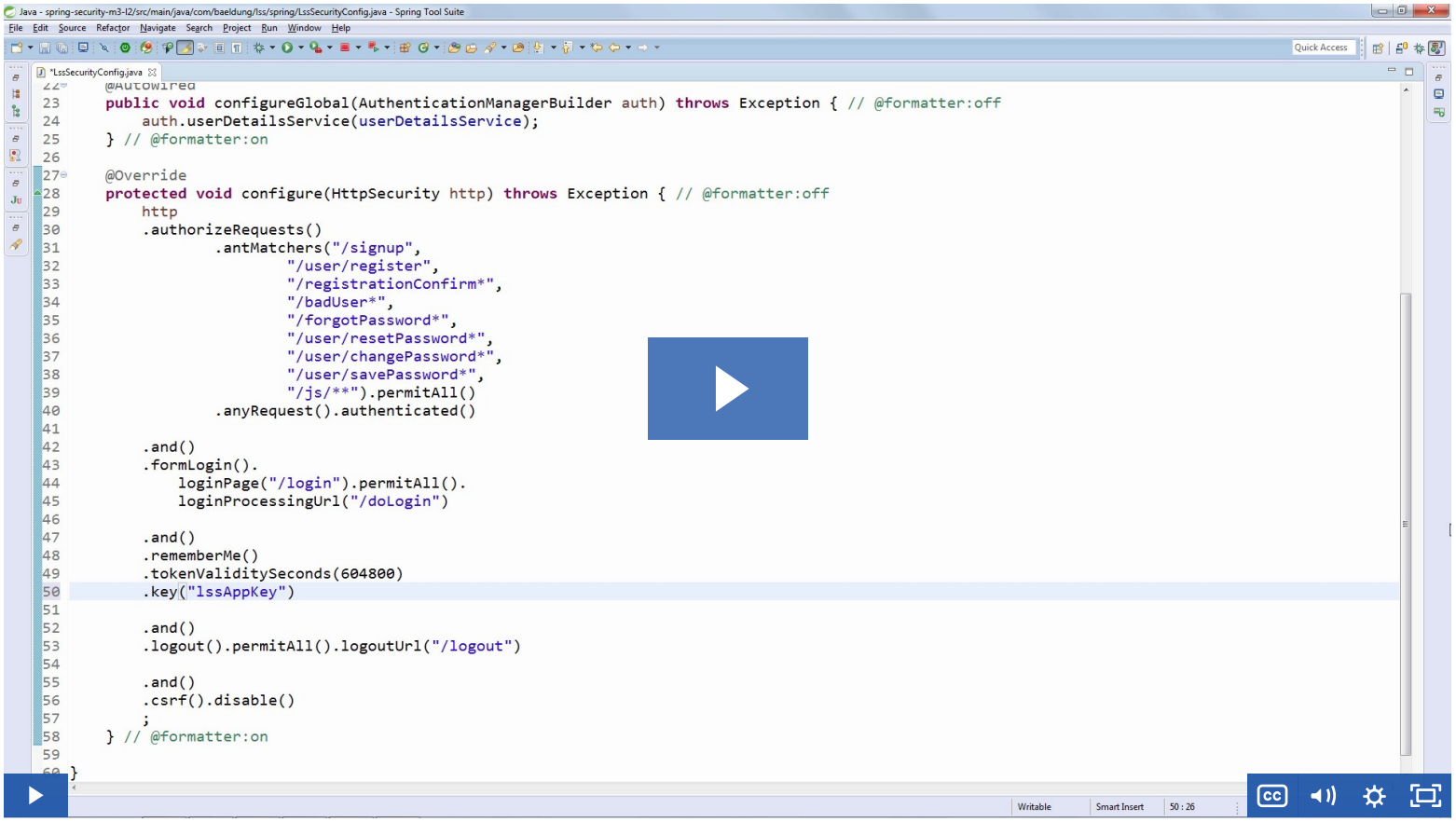
Scenario 2:

- log in with remember-me
- remove the JSESSIONID cookie manually
- refresh the page => we should now remain logged in

3. Resources

- Remember Me in the Official Reference

Lesson 2: Remember Me with Cookie



1. Main Goal

The focus here is on changing the defaults of the remember-me flow we implemented previously.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m3-lesson2

If you want to skip and see the complete implementation, feel free to jump ahead and import: m3-lesson3

The credentials used in the code of this lesson are: test@email.com/pass (data.sql).

.tokenValiditySeconds(604800)

First, we are changing the default expiration time of the remember-me cookie.

We're going from the default value of 2 weeks to 1 week (604800 seconds)

.key("lssAppKey")

Next, we are configuring the secret that identifies the tokens generated by our application.

The framework uses this secret to if tokens are valid.

.useSecureCookie(true)

Next, we are configuring the cookie to be secured.

It's important to understand that this change will mean the cookie is no longer being sent for unsecured connections.

So, in our case during local development - where we're not using HTTPS - the cookie will existing but will simply be ignored and have no effect.

.rememberMeCookieName("sticky-cookie")

Next, we are simply going to change the name of the cookie, from the default value of *remember-me* to any other value (in our case here - *sticky-cookie*)

The main reason we want to do that is to make sure we are not exposing any of the underlying details of the framework we are using to secure our application.

A quick side-note is that the old cookie - *remember-me* - won't be automatically cleared; it will still exist but of course it will have no impact on the authentication process.

.rememberMeParameter("remember")

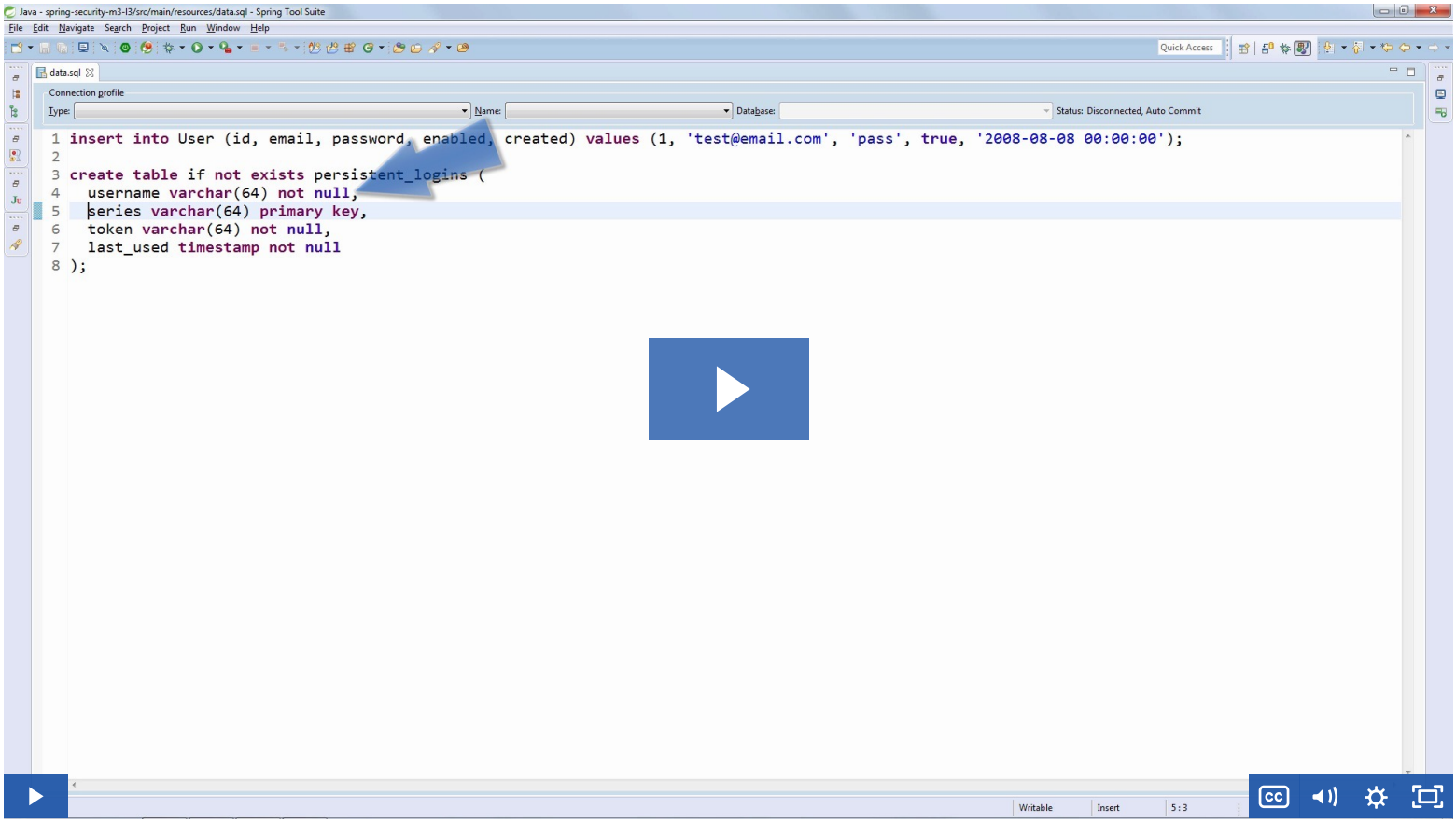
Finally we are going to change yet another default value provided by Spring Security - the parameter used by the *remember-me* checkbox (on the login page)

The reason for changing this is similar - we want to make sure there's no public information that can identify the exact framework we are using to secure our application.

3. Resources

- Remember Me - Simple Hash-Based Token Approach (in the Official Reference)

Lesson 3: Remember Me with Persistence



1. Main Goal

The focus of this lesson is to switch from the cookie based remember-me implementation to the persistence backed implemented that the framework provides.

2. Lesson Notes

The relevant module you need to import when you're starting with this lesson is: m3-lesson3

If you want to skip and see the complete implementation, feel free to jump ahead and import: m3-lesson4

A quick note is that this lesson is using MySQL, so you'll need to naturally have a MySQL Server running locally.

The credentials used in the code of this lesson are: test@email.com/pass (data.sql).

2.1. The Theory

The persistence backed implementation of remember-me is more secure than its cookie based counterpart with a mechanism that no longer relies on both *username* and *passsword*.

Now, only the *username* is exposed to the client and the password is never present in the cookie.

And as a direct consequence, if a cookie gets compromised, it's enough the delete the persisted tokens and the user no longer needs to change their password to invalidate issued tokens.

Next - the new remember me mechanism no longer validates the signature being present in the cookie - but the fact that the token is present in the DB.

2.2. The Implementation

Let's start by switching our persistence over to using MySQL just so that we can easily check the DB and see these tokens.

We're first going to define the MySQL dependency in Maven:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

We are going to make use of Boot and the *application.properties* file to configure our persistence to use MySQL:

```
spring.datasource.url=jdbc:mysql://localhost/lss33?createDatabaseIfNotExist=true
spring.datasource.username=restUser
spring.datasource.password=restmy5ql
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Now, over in our security configure, we are going to wire in the data source and start using it when defining a persistence repository for our tokens:

```
@Autowired
private DataSource dataSource;
@Bean
public PersistentTokenRepository persistentTokenRepository() {
    JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
    jdbcTokenRepository.setDataSource(dataSource);
    return jdbcTokenRepository;
}
```

Finally, we're going to make the switch from the cookie based implementation to the persistence backed one:

```
.tokenRepository(persistentTokenRepository())
```

2.3. The DB Structure

Before we run everything, we need to make sure the DB structure necessary for storing our tokens is created as well:

```
create table if not exists persistent_logins (
  username varchar_ignorecase(100) not null,
  series varchar(64) primary key,
  token varchar(64) not null,
  last_used timestamp not null
);
```

We can make use of the *data.sql* file that Spring Boot uses by default for setting up data on startup here.

But, keep in mind that there are a lot of options to set up an initial DB structure when a project starts up - besides Spring Boot.

2.4. The Series

Finally, one quick note - I touched on the concept of the *series* in the video.

This *series* field identifies the login of the user and doesn't change for duration of the entire persistent session.

It also contains a random token, regenerated each time a user logs in via the persisted remember-me functions.

The goal is to make brute force attacks impossible in practice.

3. Resources

- Remember-Me - Persistent Token Approach (in the Official Reference)

- Persistent Remember-Me on Baeldung

- Improved Persistent Login Cookie Best Practice

Lesson 4: Remember Me - The Advanced Scenarios (Bonus Material To Be Released)

All lessons marked as "bonus material" will be released after the go-live of this Class.