IBA Java Workshop VER.2.0
Week #1 Activities (17.01.2023 - 24.01.2023)

author: Mikalai Zaikin (mzaikin@ibagroup.eu)

Project description:

For week #1 we have a project which does not use any database (entities are stored in memory in `List`s), and runs locally only. The project is Maven-based and consists of 2 modules (do not mix with Java Platform Modules added in Java SE 9). The idea of modules in Maven is to create a top level project, with sub-projects, and subprojects (the modules) may depend on others. E.g. common model, and web project depends on common classes. For better maintainability it is reasonable to divide projects logically in smaller ones.

Week #1 project contains 2 modules (sub-projects): **common** and **web**. The web project depends on the common project.

Web project demonstrates basic REST functionality in Spring. You will test endpoints using command line `curl` utility which emulates web browser's HTTP/HTTPS request (make sure you use `curl` from `Git` installation, built-in Windows `curl` does not work properly with SSL certificates), extend the existing web project by adding new methods to endpoints, implement HTTPS security, create self-signed SSL certificates for server side and client side, and set up mutual SSL authentication for Spring web application.

The idea is to have some endpoints (resources) unprotected and available via HTTP protocol (used by any person), and some endpoints will be secured, protected and accessed only after mutual SSL authentication (used by application admin only).

Goals:

1) Fork a project from GitLab
2) Get familiar with multi module Maven project
3) Use `curl` command to access REST endpoints
4) Extend application functionality: add to existing REST endpoints new operations.
5) Create server side and client side SSL certificates, add public certificates to truststores, list keystore content.
6) Configure the Spring web application to use both HTTPS and HTTP at the same time.
7) Configure mutual SSL authentication for the Spring web application.
8) Protect one endpoint and open another endpoint.
9) Share GitLab project with another developer
10) Learn essential classes and annotations of Spring Framework

Steps:

1. Login to IBA GitLab at:

   ```
   https://code.iby.scdc.io/
   ```

2. Open **Java Workshop 2.0 Stream 1** group using Groups menu. Find the shared project `registration-at-iba-week-1`

3. Create a project fork using the **Fork** button at the top right corner, select your own account as the target namespace.

4. Start IntelliJ IDEA IDE.

5. Select menu: **File > New > Project from Version Control...**

6. Use the URL to import from as follows:

   ```
   git@code.iby.scdc.io:<YOUR_GITLAB_USER>/registration-at-iba-week-1.git
   ```

   NOTE, if you will use URL as original source project:

   ```
   git@code.iby.scdc.io:mzaikin/registration-at-iba-week-1.git
   ```

   You will be able to clone the project, but won't be able to push your code. Use **your own username** (namespace) in the URL.

7. Get familiar with the top level `pom.xml` file

   - it declares 2 sub-modules (**registration-common** and **registration-web**)
   - it declares own format: `<packaging>pom</packaging>`
   - **NOTE**: the project uses Java 17, if you have not defined system `JAVA_HOME` environment variable pointing to Java 17, or if it already exists on your machine and points to other JDK, edit **mwnw.cmd** and add as the first line:

   ```
   SET JAVA_HOME=C:\Program Files\Eclipse Adoptium\jdk-17.0.2.8-hotspot\
   ```

   (use your path to Java 17)

8. Get familiar with **registration-common** project

   - the POM file declares JAR packaging format: `<packaging>jar</packaging>`
   - it contains common POJO entities, and some Spring services
   - does not use database, all entities stored in memory
   - this is not a runnable application
   - it stores properties in **`src/main/resources/application-common.properties`** file

9. Get familiar with **`registration-web`** project

   - the POM file declares JAR packaging format: `<packaging>jar</packaging>`
   - it contains 2 REST endpoints
   - the POM file also declares dependency on **`registration-common`** project
   - it's a runnable application (entry point class: **`RegistrationWebApplication`**)
   - it keeps properties in **`src/main/resources/application.properties`**
   - global servlet context path is **`/registration-web`** (defined in **`application.properties`**)
   - note the project uses **`spring-boot-maven-plugin`** plugin to prepare a single "FAT JAR" with the application classes and all dependencies packaged in a single archive.

10. Run the project in Intellij IDEA IDE via toolbar button **Edit Configuration > Add New Configuration > Application**.

11. Select Main class: `eu.ibagroup.RegistrationWebApplication`

12. Ignore the **Change Configuration Settings** dialog.

13. Make sure there are no errors in logs.

14. Application listens on 8080 port by default:

    ```
    Tomcat started on port(s): 8080 (http)
    ```

15. Use files in **`registration-web/src/test/resources/`** folder

16. At the moment you have 2 unprotected REST endpoints:

    - `/registration-web/`**`event`** (defined by **`EventController`** class)
    - `/registration-web/`**`confirmation`** (defined by **`ConfirmationController`** class)

17. Use the **`list.events.cmd`** script to see events, it produces HTTP 200 (OK) status code, and returns a blank list: **`[]`**

    NOTE: all shell scripts use the `curl` in verbose mode (`-v`), you can turn it off to reduce output (headers, HTTPS negotiation, etc.)

18. Use **`create.event.cmd`** script to create an event (values taken from external JSON file)
    NOTE: it returns HTTP 201 (CREATED) and returns the new entity URL in the

`Location` header.

19. Use the **`list.events.cmd`** script to see events now, it produces a list with 1 event.

20. **ACTIVITY №1**: **Add new functionality to work with confirmations**
    - stop the web application
    - add the **`create.confirmation.cmd`** script, by analogy to **`create.event.cmd`**
    - add the **`confirmation.json`** by analogy to **`workshop1.json`**, use your corporate email address, see `Confirmation` class for JSON attribute names, for creation time use format as follows: "`2000-01-01T00:00:00`"
    - create the **`ConfirmationController.createConfirmation`** method by analogy to **`EventController.createEvent`** method which will create and store an instance of confirmation in memory on POST request. Make sure the `Location` HTTP header returns the URL to the new instance.
    - start the web application
    - create new confirmation instance using your new script, the HTTP response must return URL of newly created confirmation inside header
    - use the **`get.confirmation.cmd`** script to view your confirmation, make sure it uses the correct ID, if you did all correctly, you should receive email to your IBA Lotus mailbox.

21. **ACTIVITY №2**: **Create SSL certificates and keystores for client and server**
    - stop the web application
    - go to **`registration-web/util/certificates`** directory
    - carefully inspect all 4 `.cmd` scripts
    - optionally edit properties in the **`common.properties.cmd`**: `ORGUNIT, CITY, STATE, COUNTRY, PASSWORD` to your preferred values
    - create blank **`server`** and **`client`** directories inside **`registration-web/util/certificates`**
    - run the **`generate-all.cmd`** script, review output, press any key to continue further
    - review new files created in **`./server`** and **`./client`** directories, you will need only **`.p12`** keystores.
    - use **`list.client.keystore.cmd`**, **`list.server.keystore.cmd`** scripts to inspect the contents of client and server keystores (make sure you understand which 2 entries contains each of the **`.p12`** files)

22. Copy the **`client.keystore.p12`** file in the **`registration-web\src\test\resources`** directory

23. Copy the **`server.keystore.p12`** file in the **`registration-web\src\main\resources\keystore`** directory (create the

missing paths)

24. **ACTIVITY №3**: **Enable HTTPS and security in web application, and configure mutual authentication**
    - in the **web-registration** application, replace content of **application.properties** with the content of **application.properties.secured**
    - in `RegistrationWebApplication` class: uncomment all commented out code, carefully review new code.
    You can use **Ctrl + N** to quickly navigate to a class in IntelliJ IDEA. You can highlight a fragment and use **Ctrl + /** in IntelliJ IDEA to quickly uncomment code fragments.
    - in `SecurityConfiguration` class: uncomment all commented out code (including annotations), carefully review the new code. You can use **Ctrl + N** to quickly navigate to a class in IntelliJ IDEA. You can highlight a fragment and use **Ctrl + /** in IntelliJ IDEA to quickly uncomment the code.

25. Start the web application. Application listens now 8080 (HTTP) and 8443 (HTTPS) ports

    ```
    Tomcat started on port(s): 8443 (https) 8080 (http) with
    context path '/registration-web'
    ```

26. **ACTIVITY №4**: **Test protected resources**
    - goto **registration-web/src/test/resources** directory
    - use the **list.events.cmd** script, notice code HTTP 302 (FOUND) and proposed URL with HTTPS schema
    - use the **create.event.cmd** script, it also fails, does not return HTTP 201 (CREATED).
    - use the **list.events.https.cmd** it should succeed, and observe the empty list.
    - use the **create.event.https.cmd** it should succeed, and create a new event
    - use the **list.events.https.cmd** and observe non-empty list
    - use the **create.confirmation.cmd** script and make sure it still works via HTTP
    - use the **get.confirmation.cmd** script and make sure it still works via HTTP

27. Stop the web application.

28. In IntelliJ IDEA open Git panel (click **Git** tab on the bottom left)

29. Open **Local Changes** tab, expand the "**Changes**" section and "**Unversioned Files**" section, and review the changes.

    Select all changed and unversioned files, and right click and select "**Commit Files…**".

    NOTE: do not commit locally generated files:
    `.gitignore` file,
    `.idea` folder
    `target` folders in each project

`*.class` files

Commit and push the files.

30. Open your project in GitLab UI at:

    `https://code.iby.scdc.io/<YOUR_GITLAB_USER>/registration-at-iba-week-1`

    Make sure your commit is visible in the web UI.

31. On the left side use menu **Settings > Members** to add instructor as **Reporter** role member to your week #1 project.

Self education activities (not optional), use Google, online articles, JavaDocs, sample code online, etc..

1. Spring annotation `@Component / @Service`
2. Spring annotation `@Autowired`, difference constructor dependency injection v.s. fields dependency injection.
3. Spring annotation `@SpringBootApplication`
4. Spring annotations `@PropertySource, @Value`
5. `SecurityFilterChain` interface and how to configure filters.

Once you complete all steps notify the instructor.

Any questions or problems: ask in the workshop TG group chat or in personal messages to the instructor.

Have fun!