IBA Java Workshop VER.2.0
Week #2 Activities (24.01.2023 - 31.01.2023)

author: Mikalai Zaikin (mzaikin@ibagroup.eu)

Project description:

For week #2 we have a project which connects to the remote MongoDB database with persistent volume, running on an IBA Cloud VM. The project structure is the same – Maven-based and consists of 2 modules (common and web), but class packages structure improved: now there is stronger package separation between modules – this is also with compliance with Java 9 Platform Module System (JPMS) where a package may not be shared by two modules.

We will learn how to run a web project as a Docker container and how to make an image environment-independent. The idea is to use the same Docker image in the development environment, test environment, and production environment.

Web project demonstrates some extended REST functionality. You will need to 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), use either SSH keys from previous week, or recreate new SSH keys for this week using provided command line scripts.

Goals:

1) Fork a project from GitLab
2) Get familiar with MongoDB Compass utility to handle collections via GUI interface.
3) Create a new REST endpoint for Spring web application, the endpoint should distinguish which type of content (JSON or plain text) client prefers and return payload in proper format.
4) Externalize environment-specific properties to “env file”
5) Get familiar with the multi stage Docker image build process.
6) Create a Docker image with the application and run the application locally in a Docker container.
7) Customize networking ports used by the Docker container.
8) Share GitLab project with another developer
9) Learn essential Docker terminology and commands

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-2`

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-2.git
   ```

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

   ```
   git@code.iby.scdc.io:mzaikin/registration-at-iba-week-2.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

   - the project still consists of 2 modules.
   - if needed, 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:

   - it contains the `eu.ibagroup.common.mongo.collection` package with **Document** classes for MongoDB collections.
   - it contains the `eu.ibagroup.common.mongo` package with **MongoDB Repository** interfaces and a utility class.
   - the `application-common.properties` file contains properties to connect to MongoDB, properties to connect to mail server and web application URL (added to email notification text).**Hint**: you can use **Ctrl+N** to quickly to **n**avigate to class or file in IntelliJ

IDEA.
- MongoDB in future will be accessed by both microservices (web and TG bot), so related classes and interfaces are placed into the common module.
- it contains the `eu.ibagroup.common.service` package with **Spring Service** classes.
- the `pom.xml` contains dependency on MongoDB libraries

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

   - it contains 2 REST endpoints (`ConfirmationController` and `EventController`)
   - you will be adding new REST endpoint (`RegistrationController`) in this week
   - since the main application class `RegistrationWebApplication` is in `eu.ibagroup.web` package and the `@SpringBootApplication` will perform automatic beans scan in subpackages only; therefore we need to define additional annotations `@ComponentScan` and `@EnableMongoRepositories` for explicit scan and to allow Spring beans and repositories be autowired (injected) properly.

10. Create new (or reuse old) SSL keystores for mutual authentication using the provided **`generate-all.cmd`** script (create missing `client` and `server` directories) and place the **`.p12`** keystores as follows:

    - server-side keystore into `registration-web/src/main/resources/keystore`
    - client-side keystore into `registration-web/src/test/resources` (this is where you keep `curl` scripts and JSONs)

    NOTE: you can reuse the keystore pair from week №1.

11. Populate database name, user ID, and password to connect to MongoDB in `registration-common/src/main/resources/application-common.properties`

    - MongoDB database name and user ID are provided in the spreadsheet
    - MongoDB password will be shared in private message in Telegram

12. Run the project in Intellij IDEA IDE

    - Click **Add Configuration…** in toolbar
    - Click the **+** button
    - Select **Application**
    - Enter name: **`RegistrationWebApplication`**
    - You can use **`-cp <no module>`**
    - Select Main class: `eu.ibagroup.web.RegistrationWebApplication`
    - Click **Apply** and **OK**

- Run the project using green triangle icon or click **Shift + F10** (select **Continue Anyway** if asked)

13. Make sure there are no errors in the console log in IntelliJ IDEA.

14. Make sure application listens 8080 (HTTP) and 8443 (HTTPS):

```
Tomcat started on port(s): 8443 (https) 8080 (http)
```

15. Use files in **`registration-web/src/test/resources/`** folder to create events, list events, edit events, delete events.

    NOTE: this activity is assumed to be performed by application admin, so HTTPS endpoint (with mutual authentication) must be used, not HTTP.

16. Stop the web application in the IDE (use red square icon).

17. **ACTIVITY №1**: **Get familiar with MongoDB Compass utility and create collection**

    - Start the MongoDB Compass utility
    - Create connection to your database: click the **New Connection** button
    - Click the “**Advanced Connection Options > Authentication > Username / Password**”
    - Fill the values individually using your `application-common.properties` file
    - First check existing `events` collection, browse events you created via REST endpoint
    - Click the “ **+** “ button and create manually `registrations` collection in your DB
    - Import the provided `registration-web/src/test/resources/registrations.json` file into the `registrations` collection.

18. At the moment you have 2 REST endpoints:

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

19. **ACTIVITY №2**: **Create new REST endpoint**

    - create a new REST endpoint (REST controller) – class `eu.ibagroup.web.controller.RegistrationController`
    - this endpoint expected to be used by application administrator to browse attendants registered for the particular event
    - the class should have 2 methods (select any name) which respond on GET HTTP request

- both methods must be mapped to `/registration` path
- both methods must accept optional request parameter (query parameter) `id` which contains event ID for the registrations to return
- one of the two methods returns list of registrations in JSON format, in case of missing parameter or if no registrations found – blank list (`[]`)
- other of the two methods produces list of registrations in plain text format (string), one line per registration (use provided method `Registration.asRestString()` method), in case of missing parameter or if no registrations found the method returns an empty string
- methods are selected by Spring Boot runtime automatically depending on HTTP headers sent by user in request, i.e. if user sends header that s/he prefers JSON content, the JSON output must be returned
- you are provided 2 scripts – **`get.registrations.plain.cmd`** and **`get.registrations.json.cmd`** which can be used to test the endpoint you create

20. Once you successfully tested the registrations REST endpoint with the provided scripts, stop the application.

21. At this moment you have all properties located in `*.properties` files. When creating a Docker image, these properties are placed forever inside the image. This approach have drawbacks:

    - sensitive information (login/password) stored inside the image and published to the container registry
    - the same image may not be deployed in different environments, as the database name, database location, and other resources are hardcoded, and the development database will be different from the production one, so this information should not be placed inside the Docker image.

22. **ACTIVITY №3**: **Externalize sensitive information to "env file"**

    - for this week we only externalize (1) DB name, (2) DB user ID, and (3) password, but you can externalize as many properties as needed
    - edit the `registration-common\src\main\resources\application-common.properties` file where the MongoDB information stored
    - replace database, username, and password with environment variable names, use format like that (select own variable names):
    `prop.name=${ENV_VAR_NAME}`
    - create new **`.env`** file (in future you can use names like `dev.env`, `ft.env`, and `prod.env`) in the same folder which will contain values for these environment variables (for specific environment), use format like that:

```
ENV_VAR_NAME=<value>
```

23. Edit the IntelliJ IDEA **Run Configuration** to use the new env file.

    - Click **Edit Configurations…**
    - In the **Run/Debug Configuration** dialog click on the **RegistrationWebApplication** on the left side
    - Click the **Enable EnvFile** checkbox
    - Click the **+** icon and select the new `.env` file created in previous activity step
    - Click **Apply** and **OK**
    - Run the application
    - Test registration endpoint using command line scripts (`get.registrations.json.cmd` and `get.registrations.plain.cmd`)
    - Stop the application in the IDE.

24. **ACTIVITY №4**: **Prepare Dockerfile for web microservice**

    - you are given `Dockerfile` file template (in the week #2 project's root directory), it is a multi stage docker file; get familiar with the file
    - you already know that the web microservice listens on `8080` and `8443` ports
    - you need to edit existing Docker file to expose these ports from future container (replace the placeholders with needed Docker instructions)

25. Prepare web microservice Docker image locally:

    - create Docker image (run command from week #2 project's root directory):

```
docker build -t registration-web -f Dockerfile .
```

NOTE:
- in case of problem make sure the `mvnw` file has Unix-style line endings (LF), not Windows-style (CRLF)
- the build process may take a few minutes

- make sure the `registration-web` image created:

```
docker image ls | findstr registration-web
```

26. Run the dockerized web service in the interactive mode with the attached console (should be a single line):

```
docker run --name registration-web -it --rm
--env-file=registration-common/src/main/resources/.env
registration-web
```

Make sure no errors in logs during startup, and MongoDB connected OK.

Explanations:
`--name registration-web` defines the name of the new **container**, all containers must have unique names, otherwise, there will be conflict and command fails. You can remove a container with this command:

```
docker rm <CONTAINER_NAME>
```
or
```
docker container rm <CONTAINER_NAME>
```

`-it` == `-i` and `-t` options, provide interactive (foreground) mode with console, you can use **Ctrl + C** to exit the application.

`--rm` remove the container when it exits, handy so you won't get container name conflict next time

Troubleshooting:
- in case of network problems restart the machine, switch to Windows containers, and back to Linux containers, or try to stop/start CheckPoint VPN. We create the image and run the container in Linux Containers (Selected via the context menu on the Docker icon in the tray).
- in case of port binding conflict other instances of the bot are running (e.g. in IDE)

27. Open another command line terminal and try to test web application using scripts (`get.registrations.json.cmd` and `get.registrations.plain.cmd`)

    - Make sure script **CANNOT** reach application inside container (connection refused)
    - Stop the web application (Ctrl + C)
    - The container will be automatically removed

28. Exposing a port **doesn't** make it available when you run a container. To make the port available, you must **publish** your ports. Now, imagine you have ports `8080` and `8443` already used on your host machine, so you need to map Docker container's ports:
    - `8080` in container to `9080` on host machine
    - `8443` in container to `9443` on host machine

    Run again your web service using command (from week #2 project's root directory, must be a single line):

    ```
    docker run --name registration-web -it --rm
    --env-file=registration-common/src/main/resources/.env ****add
    port mapping options here**** registration-web
    ```

- edit `get.registrations.json.cmd` and `get.registrations.plain.cmd` to use `9443` port on `localhost` and test containerized web application is responding
- Stop the web application (Ctrl + C)
- The container will be automatically removed

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

30. Open **Local Changes** tab, expand the “**Changes**” section and “**Unversioned Files**” section, and review the changes.

    Select all changes 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.

31. Open your project in GitLab UI at:

    ```
    https://code.iby.scdc.io/<YOUR_GITLAB_USER>/registration-at-iba
    -week-2
    ```

    Make sure your commit is visible in the web UI.

32. On the left side use menu **Settings > Members** to add instructor as **Reporter** role member to your week #2 project.

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

1. Learn main docker terminology:
   - what is a Docker image
   - what is a Docker container
   - how to declare port mapping between container and host machine

2. Main Docker commands meaning and syntax (or google online docs):

   ```
   docker build --help
   docker run --help
   docker ps --help
   docker stop --help
   ```

```
docker rm --help
```

3. Spring Boot annotation `@RequestMapping`

4. Spring Boot enum `MediaType`

5. Spring Boot annotation `@RequestParam`

Once you complete all steps notify the instructor.

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

Have fun!