IBA Java Workshop VER.2.0
Week #4 Activities (07.02.2023 - 14.02.2023)

author: Mikalai Zaikin (mzaikin@ibagroup.eu)

Project description:

During week #4 we complete all the functionality of the “Registration at IBA” application. It will contain 3 modules:
- Common part
- Web application (must be reachable via browser inside IBA intranet to confirm user email)
- Telegram Bot (does not listen to any ports, accesses Telegram servers, requires internet access)

We will extend Telegram Bot by adding a new command which will reset bot to the default state.

Web Application and Telegram Bot microservices will be packaged in two Docker images and run as two containers.

We will use Docker Compose to run a multi container application, start and stop web and bot parts simultaneously.

We will use “env files” to externalize environment specific properties. There will be two environments – local machine and shared Linux virtual machine in IBA data center.

Week completes by running “Registration at IBA” on the remote Linux VM.

IMPORTANT NOTE: Since we will be using a single shared Linux virtual machine (CentOS), during this week when deploying on the remote machine each student shall:
- give unique Docker image names when creating images (details will be provided)
- use unique HTTP and HTTPS ports to avoid binding conflicts (details will be provided)

Goals:

1) Fork a project from GitLab.
2) Extend Telegram bot functionality by adding a new `/cancel` command.
3) Running application with Docker Compose on local Windows machine
4) Running application with Docker Compose on remote Linux machine
5) Share GitLab project with another developer.
6) Learn some basic Linux shell 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-4`

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-wee
   k-4.git
   ```

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

   ```
   git@code.iby.scdc.io:mzaikin/registration-at-iba-week-4.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 now consists of three 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. The structure of the **`registration-common`** project has not changed.

   - The project structure and number of classes did not change.
   - The main difference from the previous week – the bot now supports different states, e.g. when a user registers for an event, the bot enters the state when it expects the event ID.
   - Get familiar with the `eu.ibagroup.common.mongo.collection.State` enum to understand which states the bot supports.
   - Another major difference – we externalized properties in `registration-common\src\main\resources`. There are
   `.env` – it contains MongoDB connection attributes. Both local and remote applications will connect to the same DB instance, so in our project this file will be the same for

different environments (however in real projects DEV and PROD DB will differ).
`local.env` – it contains a registration URL (i.e. where user confirms his/her email) for locally running application (simply – `http://localhost`)
`prod.env` – it contains a registration URL for the application when running on the remote Linux VM.

9. **ACTIVITY #1. Update the `prod.env` and `.env` file for the `registration-common` project**

   - Goto `registration-common/src/main/resources/`
   - Update the HTTP (unsecured) port in the `prod.env` file **according the the spreadsheet pinned in the course telegram chat** (e.g. `ruser1` shall use port `9081`, `ruser2` shall use port `9082`, and so on)
   - Update MongoDB connection attributes in `.env` file (database name and user ID are provided in the spreadsheet, password was shared in private message in Telegram)

10. Get familiar with **`registration-bot`** project:

   - The bot now supports 2 sets of commands: for anonymous users and for authenticated users. Check the `Command.isAnonymous()` method, also `Session` supports status of the user (authenticated or not), check `SessionService.isUserConfirmed()` method.
   - The initial bot state is `State.DEFAULT`. In this state it waits for a command (e.g. `/help`, `/about`, `/authenticate`, etc.).
   - The state is persisted in MongoDB: `Session.state` field.
   - There are commands without state: they don't change state and the bot does not exit from the `State.DEFAULT` (example of such commands: `/status`, `/about`, `/events`, etc.)
   - The bot also supports commands with multiple states (one or two): once you start such command (e.g. `/register`), it does the following:
   a) changes `Session.state` and persists in MongoDB
   b) waits additional user input specifically for this command (e.g. event ID, etc.), so a regular command (e.g. `/about`) will be treated as a simple string and most likely the bot will fail.
   c) The example of such a state is `State.REGISTER_ENTER_ID` – bot awaits from user event ID, if user types `/about`, most likely such event does not exist and error returned. We will deal with this situation during this week.
   - The bot needs to understand which command is in progress, it uses a map where the key is state (and the state is retrieved each time from the session), and the value is a command, see `BotConfiguration` class and its maps for more details. Make sure you understand the `BotConfiguration.init()` method.
   - Registration bot project also has the `.env` file to store bot name and token.

11. **ACTIVITY #2. Update the `.env` file for the `registration-bot` project**

- goto `registration-bot/src/main/resources/`
- populate with your bot token and bot name.

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

    - it contains 3 REST endpoints (`ConfirmationController`, `EventController`, and `RegistrationController`)
    - we do not modify web project code this week

13. 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.

14. **ACTIVITY №3: Run web and bot microservices from IDE and test**

    NOTE: if you are experiencing difficulties because you are using another IDE (e.g. VS Code or Eclipse), you can skip this activity and proceed to "ACTIVITY №4 – Run web and bot microservices in Docker Compose locally and test".

    - Create 2 Run Configurations in IntelliJ IDEA:
    - **RegistrationBotApplication** configuration:
        - The main class `eu.ibagroup.bot.RegistrationBotApplication`
        - In the **Build and run** section in the drop down select `-cp registration-bot`
        - check the **EnvFile** checkbox and add env files:
            - `registration-bot/src/main/resources/.env`
            - `registration-common/src/main/resources/.env`
            - `registration-common/src/main/resources/local.env`
    - Run the configuration, make sure no errors in log and open the bot using `https://t.me/<BOT_USERNAME>` link
    - Try the `/help` command and make sure commands are listed and some of them require confirmed email
    - Try the `/authenticate` command and make sure you receive an email with a link (the link should point to `http://localhost`).

    - **RegistrationWebApplication** configuration:
        - The main class `eu.ibagroup.web.RegistrationWebApplication`

- In the **Build and run** section in the drop down select `-cp registration-web`
- Check the **EnvFile** tab and add env files:
  - `registration-common/src/main/resources/.env`
  - `registration-common/src/main/resources/local.env`
- Run the configuration, make sure no errors in log and click the link you received in email, web application should return success message to browser
- Try the `/status` command in the bot and make sure you have `Authenticated: yes`

- Stop the both configurations

15. **ACTIVITY №4: Run web and bot microservices in Docker Compose locally and test**

- You are provided two multistage Dockerfile: `Dockerfile.bot` and `Dockerfile.web`
- Multistage Dockerfile saves size of the image, as it does not contain intermediate build layers, and only contains base image and ready to start application.
- Make a note that the web docker file exposes HTTP (8080) and HTTPS (8443) ports. You do not need to edit them, as host ports will be different.
- You are provided three scripts to run application locally (on Windows machine):
a) `build.local.cmd` – creates Docker images
b) `up.local.cmd` – starts Docker containers (bot and web)
c) `down.local.cmd` – stops Docker containers (bo and web)
- The most important file for the deployment: **`docker-compose.yaml`**
- It contains a list of services (containers) the application consists of, details how to create images, order of start, and other information.
- Edit the **`docker-compose.yaml`** file:
a) change in two places **`image`** names – **append your unique number** from the spreadsheet (to avoid conflicts later on remote machine)
b) for the **`web-service.ports`** section, change the ports which Docker will expose your application at, again these must be unique ports to avoid binding conflict in future, **use ports number from the spreadsheet**.
- From the project directory (`registration-at-iba-week-4`) start the `build.local.cmd`, make sure there are no errors (ignore warnings). It may take a few minutes as all Maven dependencies are downloaded from scratch.
- You can check images are created (make sure image names contain your unique number in the end):

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

- Start the application using `up.local.cmd` script. Docker Compose joins two container logs into a single console for your convenience.
- Try to register for an event using the `/register` command in the TG bot.

- Imagine you already registered for all the events, no available events exist, or you forgot the event ID and need to return and use the `/events` command. You cannot return to the initial menu, the bot is stuck inside the registration command. You can check your session status in MongoDB Compass (the `sessions` collection) – state is `REGISTER_ENTER_ID`, restart of the bot in TG on the client won't help. You need a new command to reset the bot to the initial `DEFAULT` state (so the top level menu is available).
- Stop application using **Ctrl + C** or `down.local.cmd` script.

16. **ACTIVITY №5: Create the `/cancel` command**

    - Create code which will respond to the `/cancel` command from the user. It must be a regular anonymous command (e.g. it can be used to interrupt user authentication)
    - The command must update the user's `Session`: assign `DEFAULT` state, and return a message "`The current command was canceled`".
    - Update the **`BotCommandFactory`** class which which returns "command to run":
    a) read Java comments, add the new code as required.
    b) in general the logic must be: if we are not in the `DEFAULT` state, we retrieve the "ongoing command" by its state, this will be "command to run". Then we check the user's input and if the user's input is `/cancel`, then the user wants to cancel the ongoing command, and the command to run must be the cancel command.
    - Run the bot (from IDE or Docker Compose) and try to exit from ongoing `/register` command

17. Stop the application

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

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

    Select all changes and unversioned files you worked on this week, 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.

20. **ACTIVITY №6: Run web and bot microservices in Docker Compose on remote Linux VM and test**

    - Login to Linux server (VM machine):

    ```
    ssh YOUR_USER_NAME@10.26.8.5
    ```

    Answer “Yes
    where `YOUR_USER_NAME` is the name from the course spreadsheet, e.g. `ruser1`, `ruser2`, etc.
    Type in password (the same as the password for MongoDB)

    - On the remote Linux machine: generate a pair of SSH keys (private and public):

    ```
    ssh-keygen
    ```

    Click “**Enter**” three times (default location, no password)

    - Type public key to screen:

    ```
    cat .ssh/id_rsa.pub
    ```

    - Copy the public key to the clipboard and add it to your IBA GitLab account:

    Open `https://code.iby.scdc.io/profile/keys`
    In the text area paste the key
    Click “**Add key**” button

    - In Linux shell try these commands to test access (answer yes for server fingerprint):

    ```
    cd ~/
    git clone git@code.iby.scdc.io:mzaikin/access-test-project.git
    ```

    - If access works, proceed to cloning your Java application (it’s a single command, should be a single line):

    ```
    git clone
    git@code.iby.scdc.io:<YOUR_USER_NAME>/registration-at-iba-week-
    4.git
    ```

    - adjust shell scripts permissions (add the executable bit)

    ```
    cd registration-at-iba-week-4/
    chmod +x *sh
    ```

```
chmod +x mvnw
```

- create docker images:

```
./build.remote.sh
```

- list new Docker images:

```
sudo docker image ls
```

- start the application in detached mode (background mode):

```
./up.remote.sh
```

- list the alive containers:

```
sudo docker container ls
```

NOTE: if you see only one container or zero containers, then list all containers, including crashed and troubleshoot:

```
sudo docker container ls -a
sudo docker logs <CONTAINER_NAME>
```

- test new `/cancel` command in the bot: try to register to an event and then cancel registration in the middle.

- leave the application running on Linux VM.

21. Open your project in GitLab UI at:

    ```
    https://code.iby.scdc.io/<YOUR_GITLAB_USER>/registration-at-iba
    -week-4
    ```

    Make sure your commit is visible in the web UI.

22. On the left side use menu **Settings > Members** to add instructor as **Reporter** role member to your week #4 project.

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

1. Make sure you understand **all bash shell commands** used in this week's activities.

Hint: Google text “`man <command>`”, e.g. “`man sudo`”

2. Get familiar with basic Git commands (NOTE: there are much more)
   - `git init` (https://git-scm.com/docs/git-init)
   - `git add` (https://git-scm.com/docs/git-add)
   - `git clone` (https://git-scm.com/docs/git-clone)
   - `git commit` (https://git-scm.com/docs/git-commit)
   - `git push` (https://git-scm.com/docs/git-push)
   - `git pull` (https://git-scm.com/docs/git-pull)

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.

Congratulations on completing the IBA Java Workshop 2.0 !