Workshop #1 - Building microservices with Spring Boot

Welcome to our Spring Boot Workshop!
During this session you will be practicing the following topics:

1. Initializing a new **Spring Boot** Project using the Spring Boot Initializer
2. Configuring H2 in-memory **database**
3. Configuring **Flyway** migration tool
5. Building a **Microservice** from scratch using **Spring Data JPA (Hibernate)**

Overview:
We are about to build a new Spring Boot microservice from Scratch together (excited?).
Our service will be used to persist Player information in the database by accepting REST calls.
And will be able to connect to a H2 database and create/read/update/delete players data.

Let's Start, happy coding!

Initializing a New Spring Boot Project:

1. Open Spring Boot Initializer website

2. Select a new **Maven** with **Java** and **Spring Boot 2.1.1** project

3. Select desired Maven **Group ID** (project unique identifier) and **Artifact** (the name of the project)
4. Select the following Maven dependencies: **Web, JPA, Flyway and H2 (database)**

5. Click on 'Generate Project' and save the project to a desirable folder in your PC having the same name as the Artifact name (this is recommended but not mandatory)
   In this session I'll call it by the generic name: {project name}

6. Open Eclipse IDE-> new workspace and point it to {project name}/workspace
   (the `workspace` folder will be created automatically for you)

7. In Project Explorer tab of Eclipse IDE, right click and choose Import-> Import (yes, twice)

8. In the Import popup, select Maven->Existing Maven project (our project is Maven based, remember?) -> Next

9. In the 'Import Maven Project' popup, in the root directory navigator, navigate to the folder you created in section 5, {project name} and click 'OK'

10. Verify that the pom.xml file is selected and click ‘Finish’.
    Now will be a good time to take a few moments and browse our new project.
    What does it include? (Hint: pom.xml, Application.java file, application.properties file, mvnw.cmd)
    What doesn’t it include? (Hint: web.xml)
    Observe the pom.xml file, it already includes all we asked for and more, and it’s pretty arranged as well.

    That’s it! you have a new Spring boot Maven project created from Scratch
    (Quite easy, isn’t it?)

<u>Configuring H2 in-memory database</u>

1. In your project, open src/main/resources/application.properties.
   As you see, the file is currently empty, but it won’t remain this way for too long.

2. Let’s first enable H2 database in our project.
   Add the following line to the application.properties file:
   spring.h2.console.enabled=true

3. Now let’s configure the H2 console path so we’ll be able to access it from the browser.
   Add the following line to the file:
   spring.h2.console.path=/h2-console

4. We still need to specify Spring the JDBC value, the name of our database, the name of the schema and a few other database specific configuration of our choice.
   Add the following line to the file:
   spring.datasource.url=jdbc:h2:mem:testdb;SCHEMA_SEARCH_PATH=mySchema;
   DB_CLOSE_DELAY=-1;DATABASE_TO_UPPER=false;IGNORECASE=TRUE

5. Let’s configure the desired username and password for the database, we’ll use the default:
   spring.datasource.username=sa
   spring.datasource.password=

   That’s it, our database should be ready now.

<u>Configuring Flyway migration tool</u>

1. Add the following line to the configuration file:
   spring.jpa.hibernate.ddl-auto=none
   This property influences on how Flyway will manipulate the database schema at Startup.
   The property acccepts the following values: create, create-drop, validate, update and none.
   More on this property can be found here.

2. We need to override the Spring Boot default Naming Strategy:
   Add the following line to the configuration file:
   spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

   This property is quite technical, we need it in order to override H2 database specific configuration, but if you still want to understand more about it, feel free to ask me.

3. Let's create the migration scripts folder structure:
   (src/main/resources)**/db/migration**

4. We are now ready to create our first Flyway migration script:
   Inside \`migration\` folder, create the file : *V1.0.1__create_schema.sql.*
   Add the following sql line inside it:
   CREATE SCHEMA IF NOT EXISTS "mySchema";

Now, let's take a moment to realize what just happened here.
We configured two Spring **external** dependencies (H2 Database and Flyway) using the **internal** Spring configuration mechanism (application.properties file).
We didn't have to create any dedicated Bean for any of them (remember DataSourceConfig.java which includes FlywayConfig bean class inside it we have in ng-admin, ng-player etc..?)
Spring Boot did all this by itself (created Beans behind the scene, registered them in it's IoC container will inject them in run time).
This is all part of the 'magic' called @EnableAutoConfiguration (which is part of @SpringBootApplication annotation)

Now let's fire up our brand new microservice for the first time! :)

December 2018, Itzik Shachar

1. In Eclipse IDE, right click on the Project and select 'Run As' -> 'Java Application'
   Observe the console log, if it finishes by printing the line:
   'Completed initialization in x ms' than you are ready to go
   (take a coffee break, you deserve it!).
   Otherwise, please let me know, I can save you some troubleshooting time.
2. Open a web browser and navigate to: http://localhost:8080/h2-console/
3. Click 'Connect'.
   This is the H2 in-memory database console in which we'll be able to observe
   in order to see the database content we will create.
   Verify that the schema `mySchema` was successfully created in the database.
4. You can stop the server for now, we haven't done yet.
   We still need to make our microservice do something interesting.

Building a Microservice from scratch using Spring Data JPA (Hibernate)
Now comes the artistic part of our session!
We are about to turn our project into a full functioning microservice having the ability to create, read, update and delete players information from the H2 database using the popular JPA implementation - Hibernate.

1. Create a new Player Entity, design it as you wish but just make sure it includes:
   private long id annotated with @Id and
   @GeneratedValue(strategy = GenerationType.IDENTITY) annotations.
2. Create a new *PlayerRepository* Class which extends JpaRepository<Player, Long>
3. Create a *PlayerController* which exposes desired service operations (create, read, update and delete).
   Make your *PlayerController* @Autowire the *PlayerRepository* so you will be able to use it's methods.
4. Create another database migration script for the Player table which corresponds with the Player entity you created in section 1.

Important: Pay attention to the location of the classes you have just created.
In case you decided to place them in a package different than the default (GroupId), make sure the package is a sub-package of it ({GroupId}.x.y.z for example).
Otherwise, Spring won't be able to automatically detect your classes and to create a Bean out of them.

That's it, it's time to kick start our microservice and start playing!
Start the service and Using a REST API tool of your choice (PostMan would be a good choice), create a new Player in the H2 database, read it, update it and then delete it.

I hope you enjoyed the workshop.

December 2018, Itzik Shachar