Spring Boot Labs

Lab 1 – Basic Spring

- In this Lab, you will finish the wiring up of a Spring application. You will use both XML and annotation based configuration. The end goal is to make a suite of Junit tests run successfully.
- Instructions start on the next page

Lab 1 – Basic Spring

1. Do your work in **Labs/BasicSpringLab.** You may need to import it into your workspace.
2. You may need to set up or configure some Libraries. If you are unsure about how to do this, ask your Instructor.
3. Examine the code. Source code is in **src/main/java**, configuration resources are in **src/main/resources**, and Junit tests are in **src/test/java**.
4. Run any of the **service** tests in **src/test/java** (right click and choose **Run As → Junit Test**)
5. You will find see a whole bunch of errors in the Junit console.
6. Your job is to fix the errors for all the service tests.

Lab 1 – Basic Spring

8. You will **NOT** need to make any changes to the code itself.
9. You will need to make changes to the Spring configuration. The config class you should use is **ttl.larku.jconfig.LarkUConfig.**
10. You will also need to make annotation based changes to some of the Junit test cases.
11. There are some **TODO** comments in various source files that provide hints about what needs to be done.
12. You will probably need to iterate through a sequence of changes, fixing errors one at a time. In some cases, one fix will cause a bunch of errors to go away.
13. Your goal is to see that lovely green bar indicating a successful Junit test run.
14. A good strategy would be to proceed a test at a time.

Lab 2 – Spring Boot Command Line

1) Expose an existing application as a Spring Boot application.
2) The objective here is to understand the structure of a Spring Boot application. This one is going to be command line application – no web component.
3) You have a working Spring application in **Labs/SpringBootStarter.**
4) The application is very simple music playlist manager. The only classes you have right now are **Track, TrackDAO,** and **TrackService**. These are used to manage tracks in a playlist.
5) There is an "application" in **ttl.larku.app.Playlist.java,** and some unit tests. Examine and run the app and the tests so you know how it works

Lab 2 – Spring Boot Cmd Line contd.

6) The Track class implements a Builder pattern so you can create Track objects like this:
   Track.title(“Sunrise”).artist(“Bill Taylor”).build();
7) Your job here is to write a command line application to be able to call methods in the TrackService class
8) You will need to create a **CommandLineRunner** in your application to use as the “main” method of your application.
9) Refer to the live code for an example of how to set up the CommandLineRunner

Lab 2 – Spring Boot Cmd Line contd.

10)The easiest way to do this would be to create a new Spring Boot application using the Spring Initializer. For this stage, you don't need any starter dependencies at all.

11)You can then copy the code from the **SpringBootStarter** project to your new project.

12)You should set up the project so that you don't need to specify any ComponentScan packages, i.e. the default SpringBoot project structure.

13)This may involve changing some package names.

14)Part of your job is to convert the tests to be Spring Boot Tests. For now, just fix the tests for the service and the dao.

Lab 3 – Spring Boot Controller

1) Onwards to adding a web feature to your application. Your job is to write a REST application which will allow for the following:
   a) Getting all Tracks
   b) Getting a Track by Id
   c) Getting a Track by name – may require changes to the business layer.
   d) Creating a new Track
   e) Deleting a Track by Id
   f) Updating a Track

Lab 3 – Spring Boot Controller

2) You need to add the ‘web’ starter to your Spring Boot application.
3) The easiest way to do this is to go to **start.spring.io** and set up an application with the web starter.
4) Then click on ‘Explore’ and copy and paste the web starter dependency into your pom file.

Lab 4 - Database

Connect your track application to a database.

1) First thing your will need to do is add a dependency for the **spring-boot-starter-data-jpa** in your pom.xml file. You will also need a dependency for the **h2** database.
2) Then you have to convert your Track class into a JPA entity. The two most important annotations for doing that are **@Entity** and **@Id.** Apply them appropriately to your class.
3) Set up **DataSource** properties for an embedded h2 database.
4) Create **schema.sql** and **data.sql** files in the resources directory to have Spring Boot create and populate your schema on startup. Look at examples in the SpringDB module if you need help.

Lab 4 – Database contd

5) Make sure that you **turn off** Hibernate DDL generation with
    1) spring.jpa.generate-ddl=false, and
    2) spring.jpa.hibernate.ddl-auto=none
6) You will need to change the JPATrackDAO to use an **EntityManager** to interact with the database.
    1) You can inject an EntityManager by using the **@PersistenceContext** annotation. Check the DAOs in the SpringDB project for an example.
7) Make sure that your application actually uses the JPATrackDAO. This may require changes to either your profile setting, and/or your Spring configuration.

Lab 5 – Spring Data Repositories

Replace the JPATrackDAO with a Spring Data Repository

1) Create an interface for your Repository by extending **JpaRepository<Student, Integer>**
2) Write Unit Tests for the new repository:
    1) Copy the existing **InMemoryTrackDAOTest** class to **TrackRepositoryTest.**
3) Change the new test class appropriately to use your new repository:
    1) Inject the repository.
    2) Add annotations to run the test as a Spring Boot test.
    3) Change the calls to the old DAO into calls to the repository.
    4) Any other changes you think necessary.

The End