Spring Boot Labs

Lab 1 – Basic Spring

- In this Lab, you will finish the wiring up of a Spring application. You will use only 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 many changes to the the application code – maybe some minimal adjustments.
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 2a – Spring Boot Command Line 

1. Create a new Spring Boot Application.
    1. Use either your IDE or **start.spring.io**
    2. Choose your own values for artificat, group and package.
    3. Do not choose any dependencies right now because we are going to be making a command line application.
        1. If it insists you choose something, choose Developer Tools → Spring Boot DevTools
    4. Make sure you **do not** call your Application sbdemo. That may clash with my example in the repository.
    5. Create a **CommandLineRunner** bean by implementing the interface. Print out some pithy message from the runner.
    6. Run your application and make sure all is good.

Lab 2b – 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/CmdLineLabStarter.**
4. Copy the code from the **CmdLineLabStarter** project to your new project. Copy everything under the '**ttl'** directory.
5. The application is a 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.

Lab 2b – Spring Boot Cmd Line contd.

6. 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
7. 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 Spring Boot application to be able to call methods in the TrackService class
8. You will put the code into the **CommandLineRunner** you created in the last lab
9. Refer to the live code for an example of how to set up the CommandLineRunner

Lab 2b – Spring Boot Cmd Line contd.

10. You should set up the project so that you don’t need to specify any ComponentScan packages, i.e. the default SpringBoot project structure.
11. This may involve changing some package names.
12. Also try to change the log level for Spring, e.g
    1) **logging.level.org.springframework=debug**

Lab 2c – Spring Boot Tests

10. Copy the tests from the **CmdLineStarter** project to your Spring Boot project.
11. Make the tests run in your new application using **@SpringBootTest**.

Lab 3a – 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) Creating a new Track

Lab 3a – 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 3b – Spring Boot Controller

1) You will finish your REST Controller. Make sure you are returning appropriate Http response codes. And fix and/or complete all of these use cases.
   a) Getting all Tracks
   b) Getting a Track by Id
   c) Creating a new Track
   d) Deleting a Track by Id
   e) Updating a Track

Lab 3c – Spring Boot Controller Test 

1. Write some Unit tests for your controller using **Mockito**. Use the StudentControllerUnitTest in the **sbdemo** application as an example. Write tests for at least the following use cases:
   - get all Tracks
   - get one Track
   - add a Track
2. Copy the **TrackControllerTest** class from the **Labs/ControllerLabStarter/src/test/java/controller** directory and make sure it runs.

Lab 4 – Database setup

Connect your 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.
2. While you are there, also add a dependency for the **h2** database.
3. Set up **DataSource** properties for an embedded h2 database.
   a) DataSource properties are declared in the *application.properties* file. You can copy the one in **Labs/DBLabStarter/src/main/resources.**
   b) This file also sets properties to tell hibernate not to make the database.
   c) And makes the h2-console available at **localhost:8080/h2-console**

Lab 4 – Database setup (contd.)

4. Make sure you have **schema.sql** and **data.sql** files in the resources directory to have Spring Boot create and populate your schema on startup. You can copy them from the **Labs/DBLabStarter/src/main/resources** directory.

5. You should also copy the file **Labs/DBLabStarter/src/main/java/ttl/larku/domain/JPADurationConverter** to your ttl.larku.domain source directory.

   This file has a JPA Converter to convert to/from a Java Duration object to a String representation for the database.

Lab 4 – Database setup (contd.)

6. Set the logging level for org.springframework and org.hibernate to debug
7. Start your application and search in the console log to make sure your schema.sql and data.sql scripts have run.
8. Connect to your database through the h2 console using the properties from your data source. Careful about case!!
9. Confirm that your database has been created and populated with the correct data.

Lab 5a – Spring Data Repositories

Create a Spring Repository to use as a DAO

1. Convert the **Track** class into a JPA Entity
    1. **@Entity** on the class
    2. **@Id** and **@GeneratedValue(strategy = GenerationType.IDENTITY)** on the id field
    3. **@Enumerated** on any enum types
    4. Look at sbdemo or SpringDB for examples
2. Create an interface for your Repository by extending **JpaRepository<Track, Integer>.**

    a) Remember to annotate it with @**Repository**
3. Inject your new Repository into your TrackService. You may want to make a copy of the TrackService.
4. Check profile settings etc. to make sure your new repository is coming into play.

Lab 5a – Spring Data Repositories

5. Make sure your controller is using the new service and repository. Again, you might want to create a second controller
6. Run the app and make sure you can get to the data through the new repository.

Lab 5b – Repository testing

1. Write Unit Tests for the new repository:
    1. Copy the existing **InMemoryTrackDAOTest** class to **TrackRepositoryTest.**
2. Change the new test class appropriately to use your new repository:
    a) Inject the repository.
    b) Add annotations to run the test as a Spring Boot test.
    c) Change the calls to the old DAO into calls to the repository.
    d) Make sure your database will get initialized for each test.
        1. Either with **@Sql** scripts and/or
        2. @Transactional
    e) Any other changes you think necessary.

Lab 5c – Custom Methods

1. Create custom methods in your repository
    1. *findByName*
    2. *findByNameContainingIgnoreCase*
2. Add tests for the new methods to your test class

Lab 6 – EntityManager

1. Create a DAO using using an EntityManager.
2. You can start by copying either of your existing DAOs.
3. You can inject an EntityManager by using the **@PersistenceContext** annotation. Check the DAOs in the sbdemo project or the SpringDB project for an example.
4. Use the EntityManager appropriately in the DAO methods. Again, look at the sbdemo or SpringDB projects for examples.
   a) persist – to insert a new entity
   b) merge – to update an existing entity
   c) remove – to delete an existing entity
   d) find – to find by id
   e) query – to find by anything else, e.g. to find all entities

Lab 6 – EntityManager

5. Make sure that your application actually uses the *JPATrackDAO*. This may require changes to either your profile setting, and/or your Spring configuration.
6. Copy the **InMemoryTrackDAOTest** to **JPATrackDAOTest**.
7. Change JPATrackDAOTest appropriately to test the new JPA DAO.
   a) Inject the DAO. This is necessary because the DAO itself needs the PersistenceContext injected into it by Spring.
   b) call the appropriate methods of the dao.
   c) As with the Repository, make sure that your database will be initialized properly for each test.

The End