Introduction

Goals

To obtain some theoretical and practical knowledge related to the following concepts:

Concept	Theoretical	Practical
Dependency Injection	X	X
Spring Boot Setup	X	X
Spring Boot Configurations	X	X
Spring Boot Profiles	X	
Spring JDBC Templates	X	X
Spring Row Mapper	X	X
JPA		X
Spring Data JPA	X	X
Spring REST Controllers	X	X
Spring HTTP Message Converters	X	X
Spring Security – Basic	X	X
Spring REST Templates	X	X
Spring Task Scheduler	X	X
Apache Olingo – Vanilla	X	X
Apache Olingo – JPA Processor	X	X
Flyway	X	X

Working mode

The roadmap consists of several steps. In each step, a set of theoretical concepts are explored, supported by reference documentation, book chapters, tutorials and videos.

In parallel, a simple application will be built with the learned concepts: the “Online Shop” application.

After all learning material for a given step was completed, either some new functionality will be added to this application or old functionality will be refactored.

The application will have little-to-no user interface. Developers are expected to:

- Write tests to cover the implemented logic.
- Test the REST interfaces manually with Postman.

All the code written should be published on GitHub. Push + commits should be done when each individual chapter is finished. A dedicated repository should be created for the Online Store.

For doing static code quality checks, two separate mechanisms will be used:

- For all projects, the plugin for SonarLint should be used to fix code issues.
- For the Online Shop, https://www.codacy.com/ should be enabled (default settings) and developers should check the detected issues periodically.

Environment

You can work using your local environment:

- You need to install git, Postman, IntelliJ (Community) / Eclipse, SonarLint plugin and Lombok plugin for your respective IDE.
- If maven does not work because of SSL errors, setup the Zscaler certificate in your trust store (for Java).

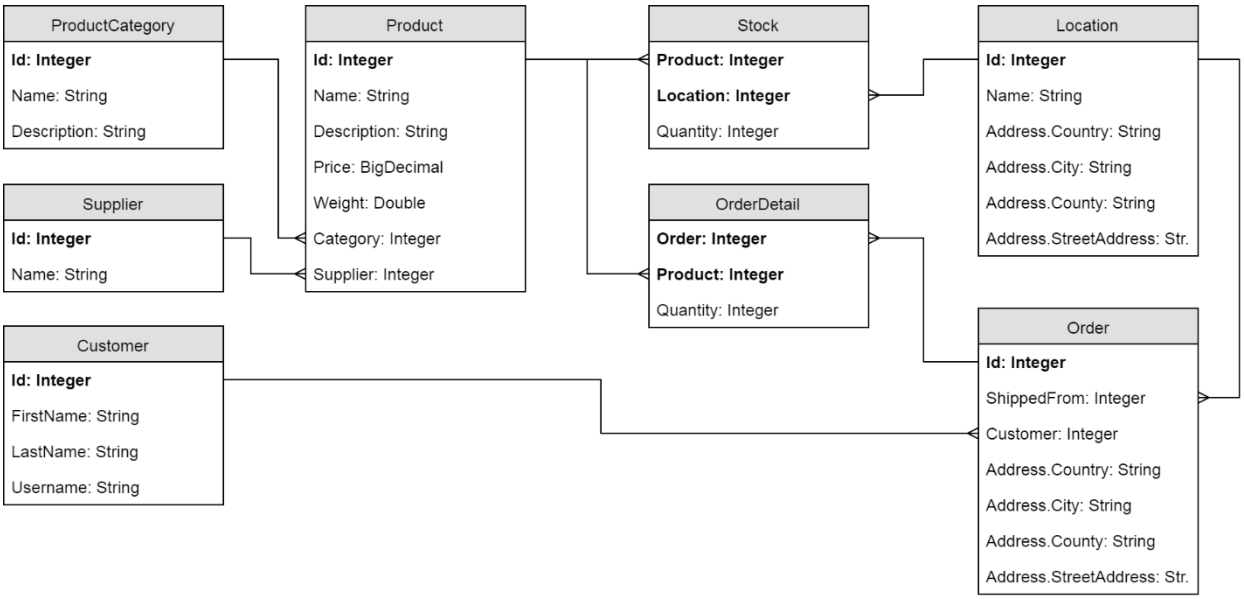
Online Shop

General Description

The application will deal with the management and daily functioning of a small shop. Business processes:

- Order creation: an end customer places an order to buy several products (based on the availability of the products in the stock).
- Stock management: the existing product stocks are updated automatically based on the orders placed by customers.
- Shop analytics: the management must be able to view the evolution of the daily revenue for each individual location of the shop.

Business Object Model



We assume that prices are in EUR and weights are in KG.

Example

You can find a very good example implementation here:
https://github.com/denisadaniella/javaAusbildungShop

1. Spring Basics

Learning Material

Spring in Action – Part 1

Dependency injection.

Spring Reference chapters I, III-7, III-8, III-9, IV-13, IV-14

Java Component Design

Spring Java 8

Maven in 5 Minutes

Getting started with Maven

Git Basics

Tutorial: GitHub Tutorial

Online Shop

Nothing to do right now. Make sure to read a little about the .gitignore, .editorconfig and pom.xml files.

2. Spring Boot Basics

Learning Material

Spring Boot Reference chapters I, II-8, III, IV-23, IV-24, IV-25

Spring Boot Into

Spring Boot Cloud

Spring Boot Apps

Spring in Action – Chapter 21.1

Tutorial: Spring Boot Reference chapter II-11

Tutorial: Spring Boot Embedded Tomcat

Sample: spring-boot-sample-tomcat

Online Shop

Go to „Spring Initializr“ and generate a new project:

- Group: ro.msg.learning
- Artifact: shop
- Dependencies: Web, Security, JPA, JDBC, H2

Import this project into your IDE (you can delete the mvnw, mvnw.cmd and .mvn files / folders as you have maven in the IDE anyway).

Create a new .editorconfig file (can be done automatically with IntelliJ).

Enable the H2 console for your application (see this link) and configure H2 to use a file-based storage somewhere on your computer (see this S.O. answer).

3. JDBC

Learning Material

Spring Reference chapter V

Spring Boot Reference chapter IV - 29.1, 29.2, Spring Boot Reference chapter IV - 41.3.9

Spring in Action – Chapter 10

Tutorial: relational-data-access

Tutorial: spring-boot-jdbc-mysql-hikaricp-example

Tutorial: springboot-working-with-jdbctemplate/

Online Shop

Create a data model for your application. You can refine the model by adding unique indexes (e.g. supplier name), introducing artificial PKs instead of composite ones (+ also add a unique index).

Translate this data model into Java entity classes. Use Lombok to annotate these classes with @Data annotations (to generate getters, setters, equals and hashCode).

Create a database initialization script; write RowMappers and Repositories with CRUD support for these tables: Product and Product Category.

Make a new Git branch (called jdbc) to commit the jdbc repositories and the database initialization script. The data model itself can be committed directly to master (first commit the data model classes, then create the jdbc branch, and commit the repositories and database script).

4. Data JPA and Flyway

Learning Material

Reference Documentation

Intro to Spring Data JPA

Introduction to Data JPA

Spring in Action – Chapter 11

Flyway: https://flywaydb.org/, Spring Boot Reference, Sample Application

Tutorial: accessing-data-jpa

Tutorial: spring-data-jpa-tutorial

Online Shop

Switch back to the master branch.

Annotate all the entities with the proper JPA annotations (and adjust the entities if necessary). Write Spring Data JPA repositories to replace the JDBC repositories (and additionally write repositories for all the other entities).

Create a database initialization script using hibernate (see this SO question and the reference). Compare this script with the one you have built previously for JDBC. Also, add a script for creating some simple mock data. Enable flyway and use it to run these scripts automatically on application start (you will have to clean your schema first).

5. Spring MVC

Learning Material

Spring Reference – Chapter VI, Chapter IV–15.6

Spring MVC 4.2

Tutorial: serving-web-content

Tutorial: creating-web-application-with-spring-boot

http://static.olivergierke.de/lectures/ddd-and-spring/

Additional: Transactional

Online Shop

Time to create some business logic:

1. Create a service (annotated with @Service) class that handles the creation of orders. The following constraints apply:
   - You get a single java object as input. This object will contain the order timestamp, the delivery address and a list of products (product ID and quantity) contained in the order.
   - You return an Order entity if the operation was successful. If not, you throw an exception.
   - The service has to select a strategy for finding from which locations should the products be taken. See the strategy design pattern. The strategy should be selected based on a spring boot configuration. The following initial strategy should be created:
     - Single location: find a single location that has all the required products (with the required quantities) in stock. If there are more such locations, simply take the first one based on the ID.
   - The service then runs the strategy, obtaining a list of objects with the following structure: location, product, quantity (= how many items of the given product are taken from the given location). If the strategy is unable to find a suitable set of locations, it should throw an exception.
   - The stocks need to be updated by subtracting the shipped goods.
   - Afterwards the order is persisted in the database and returned.
2. Create a service (annotated with @Service) class that handles the export of stocks. It has one method for exporting the stock of a given location (input = location ID, output = list of stocks).
3. Create Rest Controllers for all the above operations:
   - Create order will be a POST mapping, JSON body and JSON response.
   - Export stock will be a GET mapping, CSV response.
4. Create message converters for CSV handling. You should use the Jackson CSV library. Please firstly create a utility class that has the following methods (and then wrap it into a subclass of AbstractGenericHttpMessageConverter):
   - fromCsv:
     - Has a generic type parameter <T> = the type of the POJOs stored in the CSV.

    - Returns a List<T>
    - Has a parameter Class<T>.
    - Has an input stream parameter.
- toCsv:
    - Has a generic type parameter <T> = the type of the POJOs stored in the CSV.
    - Returns void.
    - Has a parameter Class<T>.
    - Has a parameter List<T> = the list of POJOs to be written in the CSV.
    - Has an output stream parameter.

5. Create Rest Controller Advices for handling the exceptions raised by the services.

Write automatic unit tests for the CSV conversion utility class.

6. Spring Security

Learning Material

Reference

Spring in Action – Chapter 9

Guide: helloworld-boot

Tutorial: Spring boot with Basic Authentication

Tutorial: Building REST services with Spring (includes concepts from all the previous steps)

Tutorial: spring-security-spring-data-jpa

Extra: OAuth 2 Basics, Comparison

Online Shop

Secure the application with (switch between them):

- Basic HTTP Authentication
- Form Based Authentication

There should only be one type of user: customer. Customers may do anything (if the user is not logged in, then he should not be able to do anything).

Users will be stored in the database. Create user / authority tables. Configure the authentication manager to read the users and authorities from these tables.

7. REST Templates

Learning Material

Tutorial: Spring RESTful Client

Tutorial: rest-template

Online Shop

Create a Google Distance Matrix API Key: Getting Started. You can use this API key: ***REDACTED*** if you cannot create one (you would need to put your credit card details in there).

Using this Google API, create an order creation strategy that selects locations based on the proximity to the delivery address (very similar to the single location strategy, but instead of selecting the first one by ID, you need to select the first one by distance to the order location).

8. Task Scheduler

Learning Material

Documentation: scheduling

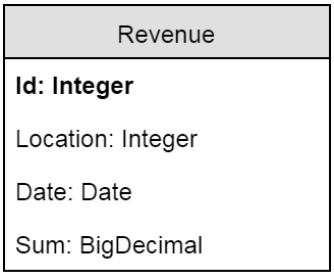
Tutorial: Getting started

Tutorial: A guide to Spring Task Scheduler

Tutorial: The @Scheduled annotation

Online Shop

Create a periodical job that runs at the end of each day. It should aggregate all the sales revenues for each location for that given day and store the result into a dedicated database table.



9. Apache Olingo

Learning Material

Documentation and tutorials: https://olingo.apache.org/doc/odata2/index.html

Bonus: SAP SDK for OData service creation: https://blogs.sap.com/2018/03/01/sap-cloud-platform-sdk-for-service-development-overview-of-blogs/

Online Shop

Expose the following entities through a JPA-based OData service:

- Order
- OrderDetails
- Product

Use the JPA-EDM XML mapping to obtain a service metadata as similar as possible to the following: metadata.xml.

Create another OData based service using the core processor. This service should expose the “create order” service (using a deep create).

Check out the spring-training-odata-v2-examples repository for some small examples using spring boot and Apache Olingo.

Bonus: Do the same, but using the SAP SDK instead of vanilla Olingo.