Spring Cloud

Sample Case

Gruppe Obelix

Urs Mezenen
Simon Herrmann
Yannis Biasutti

Version 1.0, 31.01.2021

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Moduls “Java Software Entwicklung mit Open Source 1” (BTI7515p) wurde eine Applikation entwickelt, welche Woche für Woche durch ein neues Thema der Open Source Entwicklung erweitert wurde. Die Umsetzung wurde mit Spring Boot und Tools aus Spring Cloud realisiert. Ziel war es, dass am Ende des Moduls die Applikation als einzelne Microservices über Docker in einem Kubernetes Umfeld bereitgestellt werden kann. In diesem Dokument werden die Arbeiten der jeweiligen Themen, unsere Implementierung sowie die Problemstellungen beschrieben.

2 Git / Maven

Als erste Übung wurde ein Repository für die zu erledigenden Arbeiten erstellt und der Zugang für alle Teammitglieder, inkl. Dozent, sichergestellt. Dadurch konnte die Grundlage für eine saubere Codeentwicklung, welche auch geräteunabhängig und parallel von mehreren Teammitgliedern funktioniert, gelegt werden. Git-Plattformen werden von diversen Anbietern zur Verfügung gestellt. In Rahmen dieses Projekts verwendeten wir die Plattform des Anbieters "GitHub".

Als Buildtool wird Maven eingesetzt. Maven übernimmt die Aufgabe des Dependency Management.

2.1 Probleme/Lösungen

Wir haben alle bereits mit Git gearbeitet und benutzen es regelmässig. Dadurch entstanden hier keine Probleme. Bei Maven hingegen hatten wir teilweise Mühe zu entscheiden, welche Konfiguration wir nun in unserem "Root pom" haben wollen und welche in den projektspezifischen. Wir haben dann eine Lösung gefunden, bei welcher wir die Versionen unserer Dependencies auf Root-Ebene definieren können und auf die Submodule, welche ebenfalls diese Dependencies verwenden, nur noch referenzieren müssen.

3 Spring Dependency Injection

Um ein Objekt nicht immer und immer wieder neu instanziieren zu müssen sowie deren Implementation zu abstrahieren, können in Spring sogenannte Beans definieren werden. Mithilfe dieser Beans können mit Spring "Magic" und dessen Annotationen die Objekte in anderen Klassen wiederverwendet werden.

3.1 Probleme/Lösungen

Da wir bisher alle immer direkt Spring Boot verwendet haben, war uns diese XML-Konfiguration nicht bekannt. Wir verstehen, dass dies aus historischer Sicht aber durchaus Sinn macht. Implementiert haben wir diesen Lösung Ansatz aber nicht, da uns zu dieser Zeit bereits eine bessere Lösung bekannt war.

4 Spring Data

Mittels Spring Data können Data Access Objects erstellt werden. Diese übernehmen die automatische Abbildung einer Java Klasse auf Tabellenebene. Die Konfiguration wird über Spring gesteuert. Dies geschieht wie fast alles in Spring mithilfe von Annotationen der jeweiligen Properties oder über einen Eintrag im "application.yml"- bzw. "application.properties"-File. Spring Data vereinfacht den Zugriff auf die Datenbank und dadurch muss sich der Entwickler nur im Java Code bewegen. Als Datenbank haben wir eine H2 in Memory Datenbank verwendet. Dies kann schnell aufgesetzt und auch die Konfiguration im "application.properties" ist trivial.

4.1 Probleme/Lösungen

Wir haben unsere Heros problemlos in der Datenbank speichern können. Leider wollte aber die Applikation anschliessend nicht mehr starten. Wir haben unsere Methoden auf dem Interface des jeweiligen Entitys definiert, um bereits Funktionen von späteren Übungen abzubilden. Dies hat uns dazu verleitet, nicht die Syntax des «Auto-Querys» zu verwenden. Durch diese falschen Methoden-Bezeichnungen wurde die ganze Spring Applikation blockiert. Durch Entfernen dieser falschen Methoden auf Interface-Ebene konnte das Problem aber gelöst werden.

5 Spring Boot & MVC/REST

Mittels Spring Boot soll das aufsetzten von Spring Applikationen innert weniger Minuten ermöglicht werden. In der Annotation "@SpringBootApplication" sind drei wesentliche Annotationen enthalten.

1. @Configuration, welches schon vorher mit Spring Data verwendet worden ist.
2. @EnableAutoConfiguration, sagt Spring, es sollen Beans basierend auf classpath Einstellungen, anderen beans- und properties-Einstellungen hinzugefügt werden.
3. @ComponentScan, scannt das Packet nach anderen Komponenten.

Durch diese Annotationen wird das Aufsetzen neuer Applikationen erleichtert. Mit der Verbindung von Paketen wie "spring-boot-starter-web", "spring-boot-starter-hateoas" und "spring-boot-starter-data-rest", wird das Erstellen einer Rest Schnittstelle ermöglicht. In unserem Beispiel wurden für das Hero Model die CRUD-Optionen bereitgestellt. Für das Party Model die Read Option. Mit den Annotationen von Spring kann schnell ein Controller zur Verfügung gestellt werden, welcher üblicherweise JSON-Daten als Input und Output liefert. Der Controller richtet sich nach dem HTTP-Protokoll und enthält hauptsächlich die Methoden GET, POST, PUT und DELETE. Hier ist es Wichtig, dass keine inkonsistenten Daten in die Datenbank geschrieben werden und das Fehlermeldungen, wenn möglich, mit dem passenden HTTP -Status verschickt werden.

5.1 Probleme/Lösungen

Die Umstellung auf Spring Boot stelle keine grossen Probleme dar. Die Konfiguration wurde vereinfacht und Spring Boot wurde bereits von allen Teammitglieder ausserhalb dieses Projekts eingesetzt.

Beim Erstellen der REST-Schnittstelle haben wir von Anfang an versucht, allfällige falsche respektive nicht ausreichende Daten so gut wie möglich abzufangen und eine passende Antwort zurückzugeben.

Für den HAL-Browser haben wir zuerst einen eigenen Controller erstellt, welche sowohl für die Heros wie auch für die Party links erstellt. Dadurch lernten wird die Methoden richtig kennen und es ist uns aufgefallen, dass dafür kein separater Controller notwendig ist. Die Entity-links können direkt im Hero- und Party-Controller hinzugefügt werden.

6 Microservices

In der Aufgabe "Microservices" soll für jede Aufgabe der Applikation ein eigener Service zur Verfügung gestellt werden. Hierzu mussten zuerst weitere Aufgaben hinzugefügt werden, da bis jetzt nur die CRUD-Operationen für unsere Models bestanden. Hierfür musste folgendes umgesetzt werden:

- Einen Arena Service, welcher für das Kämpfen zweier Parties zuständig ist.
- Einen Camp Service, welcher für das Erstellen von Parties übernimmt.
- Einen Promoter Service, durch welchen der Kampf ausgeführt werden kann.

6.1 Probleme/Lösungen

Da für diese Übung bereits viele der verwendeten Klassen zur Verfügung gestellt wurden, bestand ein grosser Teil der Aufgabe darin, neue Module zu erstellen und diese in der POM-Konfiguration richtig zu erfassen. Dank den Hints in den Folien sowie durch die Erfahrung von Yannis Biasutti war schnell klar, was zu tun ist. Die Aufgabe konnte daher schnell und ohne grosse Probleme umgesetzt werden.

7 Spring Cloud

Die Aufgabe "Spring Cloud" hatte zum Ziel, eine Cloud für unsere verschiedenen Microservices aufzubauen. Es sollen Monitoring Möglichkeiten und ein Fallback Mechanismus eingebaut werden. Diese Aufgabe wurde mithilfe der folgenden Tools umgesetzt:

- Reverse Proxy und Single Point of Entry – Zuul
- Discovery Server – Eureka
- Circuit Breaker – Feign
- Monitoring - Hysterix

7.1 Probleme/Lösungen

Bei der Erstellung der verschiedenen Module haben wir nicht auf die Spring Version geachtet und dadurch liefen gewisse Services mit der Version 2.3.5 und andere mit 2.4.0. Dies stellte zu Beginn kein Problem dar. Erst als wir versuchten "Feign" zu implementieren, kamen wir nicht mehr weiter. Seit der Version 2.4.0.RELEASE werden Feign, Hysterix und Zuul nicht mehr unterstützt und müssten dadurch durch neue Module ersetzt werden. Wir hatten nun die Möglichkeit diese neuen Module anzubinden oder alle Services auf 2.3.5 zu downgraden. Letzteres haben wir dann auch getan.

Beim Prüfen des Hysterix-Dashboards haben wir festgestellt, dass wenn der Arena Service nicht verfügbar ist, der Circuit immer noch als geschlossen angezeigt wird. Nach einem klärenden Gespräch mit Roger Villars haben wir erfahren, dass dies auf einem Algorithmus basiert, welcher erst nach mehreren fehlgeschlagenen Requests den Circuit öffnet. Mittels Refresh Spam auf dem nicht verfügbaren Endpoint wurde anschliessend das gewünschte Ergebnis erreicht.

8 Docker / Kubernetes

Wurde nicht implementiert.