Software-Architektur: Praktikumsaufgabe 4

ShareIt Teil III

Prof. Dr.-Ing. Axel Böttcher

1 Erweiterungen zu "ShareIt"

Arbeiten Sie für diese Aufgabe mit Ihrem aktuellen Stand der ShareIt-Medienverwaltung weiter. Bauen Sie also alles, was in dieser Aufgabenstellung gefordert wird, in den Service zur Medienverwaltung ein, der mittlerweile die von Ihnen gebaute Authentisierung nutzt.
Im Moodle finden Sie ein zip-Archiv, das einen Teil des in dieser Aufgabenstellung angegebenen Quellcodes enthält. Den Code dürfen Sie für Ihre Lösung verwenden – Sie sollten allerdings unbedingt prüfen, wo Sie ggfs. Anpssungen für Ihr jeweiliges Projekt vornehmen müssen (z.B. Packages, Entity-Klassennamen etc).

1.1 Bug Fixing

Beseitigen Sie ggfs. Fehler in der bisherigen Lösung:

- Die Darstellungen der ISBN mit bzw. ohne Bindestriche sind äquivalent. Normieren Sie die ISBN vor dem check auf Korrektheit bzw. vor dem Speichern und speichern Sie eine Variante ohne Bindesrtriche.
- Wenn keine Medien gespeichert sind, dann ist das kein Fehler, sondern es wird eine leere Liste zurück gegeben.
- **Vierergruppen zusätzlich:** Schreiben Sie Integration-Tests, um zu prüfen, dass auch das Ändern in allen denkbaren Fällen zum korrekten Ergebnis führt[1].

Im Zweifelsfall halten Sie bitte Rücksprache mit mir.

1.2 `JsonMappingExceptions` behandeln (nur Dreier und Vierer-Teams)

Auf unterschiedliche Fehler, die bei der Verarbeitung auftreten können, sollten REST APIs möglichst einheitlich reagieren. Was Sie bisher noch nicht berücksichtigt haben, sind Fehler, die Jackson beim Parsen der JSON-Daten aufdeckt. Wenn beispielsweise das an Ihr API geschickte JSON statt des Attributs `title` ein Attribut `titel` enthält, dann bekommt Ihr eigener Code das bislang nicht mit und es wird eine für Anwender verwirrende/schwer interpretierbare Fehlermeldung aus der JSON-Mapping-Exception an den Client geschickt.
Um solche Fälle zu behandeln, müssen wir uns in die Fehlerbehandlung von Jackson einklinken (man spricht von *Interception*).
Dazu müssen Sie eine Handler-Klasse erstellen, die `com.fasterxml.jackson.databind.exc.ExceptionMapper<PropertyBindingException>` implementiert[2]. Die Implementierung der Methode

```
public Response toResponse(PropertyBindingException exception)
```

ist straightforward. Sie müssen nur noch den Namen Ihrer Handler-Klasse als zusätzlichen Provider im entsprechenden init-Parameter für das Jersey-Servlet in die `web.xml` eintragen:

[1] Eine Library, die Sie zum Testen verwenden können, ist restassured
[2] leider kann man bei der Typangabe der zu behandelnden Exceptions nicht allgemeiner werden; gerne würde man gleich `JsonMappingException` angeben.

```
<init-param>
<param-name>jersey.config.server.provider.classnames</param-name>
<param-value>org.glassfish.jersey.jackson.JacksonFeature;*IHR HANDLER*</param-value>
</init-param>
```

Schreiben Sie Integration-Tests, welche die Behandlung möglicher Fehlerfälle im laufenden Betrieb testen (z.B: laufender Service auf localhost).

1.3 Dependency Injection einbauen (alle Teams)

Der Einbau von Dependency Injection mit Guice (DI) ist leider nicht ganz geradeaus möglich. Jersey verwendet eine andere DI-Implementierung namens HK2, die wir aber nicht verwenden wollen. Führen Sie die folgenden Schritte durch. Um die DI zum Laufen zu bekommen, passen Sie **package**-Deklarationen und Kommentare entsprechend Ihrer eigenen Bedürfnissen an:

1.3.1 Dependencies

Als Erstes brauchen wir ein paar weitere Dependencies, nämlich für Google-Guice selbst, für dessen Servlet-Integration und für eine Brücke von HK2 nach Guice. Das können Sie direkt im `pom.xml` ergänzen:

```
<dependency>
<groupId>com.google.inject</groupId>
<artifactId>guice</artifactId>
<version>4.1.0</version>
</dependency>
<dependency>
    <groupId>com.google.inject.extensions</groupId>
    <artifactId>guice-servlet</artifactId>
    <version>4.1.0</version>
</dependency>
<dependency>
    <groupId>org.glassfish.hk2</groupId>
    <artifactId>guice-bridge</artifactId>
    <version>2.5.0-b32</version>
</dependency>
```

1.3.2 Injector

Das Projekt wird ja letztendlich auf einem Servlet-Container deployed. Wir haben dort keinen Zugriff auf eine exponierte `main`-Methode, in der wir mit einem Guice-Module die Anwendung konfigurieren könnten[3].
Wir brauchen deshalb zunächst eine Instanz von `GuiceServletContextListener`. Allgemein sind `ServletContextListener` Objekte, die als *Observer* über die Ereignisse von Initialisierung

[3] Die `main`-Methode zum Starten von Jetty dient lediglich zum Starten der Anwendung aus der IDE heraus.

und Beendigung des Servlet-Kontexts informiert werden. Leider ist es erforderlich, den Injector über eine statische Instanzvariable/Methode an eine HK2-zu-Guice-Bridge (siehe unter Punkt 1.3.3) zur Verfügung zu stellen, was den Code unschön macht[4]. Die Sichtbarkeiten sind entsprechend eingeschränkt:

```
package edu.hm;

import com.google.inject.Guice;
import com.google.inject.Injector;
import com.google.inject.servlet.GuiceServletContextListener;
import com.google.inject.servlet.ServletModule;
import edu.hm.shareit.services.MediaService;
import edu.hm.shareit.services.MediaServiceImpl;

/**
 * Context Listener to enable usage of google guice together with jersey.
 */
public class ShareitServletContextListener
                 extends GuiceServletContextListener {

    private static final Injector injector
       = Guice.createInjector(new ServletModule() {
            @Override
            protected void configureServlets() {
                bind(MediaService.class).to(MediaServiceImpl.class);
            }
    });

    @Override
    protected Injector getInjector() {
        return injector;
    }

    /**
     * This method is only required for the HK2-Guice-Bridge in the
     * Application class.
     * @return Injector instance.
     */
    static Injector getInjectorInstance() {
        return injector;
    }
}
```

[4] Vielleicht hat jemand eine Idee, wie man das schöner hin bekommen kann.

Diese Listener-Klasse muss in der `web.xml` als Listener explizit angegeben werden. Dort müssen Sie also sinngemäß folgendes ergänzen:

```
<web-app>
    ......
    <listener>
        <listener-class>
            edu.hm.ShareitServletContextListener
        </listener-class>
    </listener>
</web-app>
```

1.3.3 Guice-HK2-Bridge

Die Verbindung zwischen Jersey/HK2 und Guice kann in einer `ResourceConfig`-Klasse folgendermaßen stattfinden:

```
package edu.hm;

import javax.inject.Inject;

import org.glassfish.hk2.api.ServiceLocator;
import org.glassfish.jersey.server.ResourceConfig;
import org.jvnet.hk2.guice.bridge.api.GuiceBridge;
import org.jvnet.hk2.guice.bridge.api.GuiceIntoHK2Bridge;

/**
 * Application class to enable guice within jersey.
 */
public class ShareItApplication extends ResourceConfig {

    @Inject
    public ShareItApplication(ServiceLocator serviceLocator) {
        GuiceBridge.getGuiceBridge().initializeGuiceBridge(serviceLocator);
        GuiceIntoHK2Bridge guiceBridge
            = serviceLocator.getService(GuiceIntoHK2Bridge.class);
        guiceBridge.bridgeGuiceInjector(
            ShareitServletContextListener.getInjectorInstance());
    }

}
```

Die Application muss in der `web.xml` noch als init-Parameter für das Jersey-Servlet angegeben werden:

```
<init-param>
 <param-name>javax.ws.rs.Application</param-name>
 <param-value>edu.hm.ShareItApplication</param-value>
</init-param>
```

1.3.4 Unit Tests

Erweitern Sie die Tests, sodass zwischen Resourcen- und Service-Klassen IoC und DI vorhanden ist und testen Sie die Resourcen-Klassen mittels Mockito-generierten, un dim `Module` verdrahteten Mock-Objekten.

1.4 Persistenzierung mit Hibernate

Für diese Teilaufgabe müssen Sie mehrere externe Libraries in das Projekt einbinden:

- Log4j – Hibernate nutzt ausgiebig Logging
- Hibernate selbst
- Treiber für eine Datenbank. Sie dürfen eine Datenbank Ihrer Wahl verwenden. Eine meiner Meinung nach gute Variante zum Entwickeln und Testen ist HSQLDB. Diese kann entweder in-memory laufen oder dateibasiert (Daten werden als SQL-inserts in eine Datei geschrieben). Mit dem (etwas veralteten) Tool schemaspy lässt sich die in der Datenbank durch Hibernate erzeugte Struktur visualisieren. Ein Kommandozeilentool zum Aufruf von schemaspy ist im zip-Archiv zur Aufgabe enthalten.

```
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.8</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.8</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.10.Final</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>5.2.10.Final</version>
```

```
20    </dependency>
      <dependency>
        <groupId>org.hsqldb</groupId>
        <artifactId>hsqldb</artifactId>
        <version>2.3.0</version>
25    </dependency>
```

Ferner müssen Sie Log4j konfigurieren mit einer geeigneten `log4j2.xml` (nicht im zip-Archiv enthalten).
Halten Sie die Vorgabe der geschichteten Architektur ein. Das bedeutet, dass die Persistenzschicht durch Dependency Injection mit der darüber liegenden Schicht verbunden wird.

1.5 Deployment

Deployen Sie Ihre Anwendung weiterhin auf Heroku.