Fachhochschule Nordwestschweiz
Hochschule für Technik

Developer Guide

Mächler Markus und Stucki Melanie

IP-516bb, Studiengang Informatik und Informatik IComptence
FHNW

Brugg, 23.08.2016

Markus.Maechler@students.fhnw.ch
Melanie.Stucki@students.fhnw.ch
www.fhnw.ch

Inhaltsverzeichnis

1 Abstract

Dieses Dokument ist als Ergänzung zur Projektdokumentation (Projektdokumentation IP-516bb.docx) zu sehen. Es sollte unbedingt zuerst die Projektdokumentation gelesen werden, da in diesem Dokument nur Ergänzungen zu Details der Implementierung enthalten sind.

2 Module und Source Code

2.1 Ordnerstruktur

Die Projektstruktur folgt der Standardstruktur[1], wie sie von Symfony definiert wird. Folgende Tabelle zeigt eine Übersicht.

Ordner / Datei	Beschreibung
app/config	Enthält Konfigurations-Dateien wie z.B. die Service Konfiguration.
app/Resources	Enthält applikations-spezifische Ressourcen wie z.B. die Templates für die HTML Serialisierung.
bin	Enthält Symfony Skripte für die Interaktion mit dem Framework über die Konsole.
features	Enthält die Testszenarien für die Integrationstests mit dem Behat Framework.
src	Enthält den gesamten Source Code der projektspezifischen Implementierungen.
tests	Enthält die Unit Tests.
var	Enthält applikations-spezifische Log-, Session- oder Cache-Dateien.
vendor	Enthält alle Abhängigkeiten, die von Composer verwaltet werden.
web	Dieser Ordner ist der eigentlich Web-Root und sollte auch als solcher im Web-Server konfiguriert werden. Nur Dateien in diesem Ordner sind direkt öffentlich zugänglich.
.gitignore	Enthält Dateien- und Ordner-Pfade, die nicht mit Git versioniert werden sollten.
.travis.yml	Enthält die Konfiguration für den Build Server Travis CI.
behat.yml	Enthält die Konfiguration für die Integrationstests mit Behat.
composer.json	Enthält die Konfiguration für die von Composer verwalteten Abhängigkeiten.
composer.lock	Lock-Datei von Composer um die exakt installierten Versionen für alle Entwickler gleich zu haben.[2]
development.md	Enthält Informationen für Entwickler für die Installation sowie Performance-Optimierung der Applikation.
LICENCE	Die Lizenzierung des Source Codes.
phpunit.xml	Die Konfiguration für das Ausführen der PHPUnit Tests.
README.md	Readme Datei des Projekts.

[1] http://symfony.com/doc/current/quick_tour/the_architecture.html

[2] https://getcomposer.org/doc/01-basic-usage.md#composer-lock-the-lock-file

2.2 API-Platform

An diversen Stellen mussten wir die Standardfunktionalitäten der API-Platform mit unserer eigenen Implementierung erweitern. Alle Änderungen, bei denen wir API-Platform Services erweitern, befinden sich im Namespace *LinkedSwissbibBundle\ApiPlatform*.
Eine ausführliche Dokumentation des Frameworks gibt es auf GitHub:
https://github.com/api-platform/docs

2.3 PathResolver

Die Klasse *LinkedSwissbibBundle\PathResolver\CamelCaseResourcePathGenerator* ist dafür verantwortlich die Form der URLs für Entitäten zu definieren. Sie erstellt aus einem Entität-Namen den entsprechenden Pfad inklusive Formate.
Genauere Informationen dazu können in der Dokumentation der API-Platform gefunden werden:
https://github.com/api-platform/docs/blob/master/core/operation-path-naming.md

2.4 Entitäten

Die Entitäten sind die Datenbehälter jeder Ressource, die über die API zur Verfügung gestellt wird. Sie müssen alle im Namespace *LinkedSwissbibBundle\Entity* erstellt werden. Über Annotationen können Operationen, die auf einer Entität möglich sind, festgelegt werden sowie RDF- oder Hydra- spezifische Informationen hinterlegt werden.
Um Entitäten aus einer Antwort des Elasticsearch Servers zu erstellen gibt es die Klasse *LinkedSwissbibBundle\Entity\SimpleEntityBuilder*.
Mehr Informationen zu den Entitäten findet man in der offiziellen Dokumentation von API-Platform:
https://github.com/api-platform/docs/blob/master/schema-generator/getting-started.md

2.5 ContextMapper

Wir haben die Herausforderung, dass die Daten im Elasticsearch Server mit Namespace-Präfixen versehen sind, unser Framework das so aber nicht unterstützt. Das heisst ein Feld für den Titel einer bibliographischen Ressource heisst im Elasticsearch Server *dct:title* bei unserer API aber nur *title*. Das führt dazu, dass Benutzer unserer Schnittstelle z.B. eine Suche im Feld *title* machen möchten, wir die Suche für den Elasticsearch Server aber so aufbereiten müssen, dass das Feld dann *dct:title* heisst.
Für diese Aufgabe haben wir die Klasse *LinkedSwissbibBundle\Elasticsearch\ContextMapper* ins Leben gerufen. Anhand von den definierten Context-Informationen im Elasticsearch Server kann diese Klasse alle Felder-Namen unserer Entitäten hin und her konvertieren.
Der Context des Elasticsearch Servers wird aus Gründen der Performance lokal bei der API gecached. Wenn Änderungen am Context vorgenommen werden, muss zwingend der Applikations-Cache geleert werden.

2.6 DataProvider

Die Klasse *LinkedSwissbibBundle\DataProvider\ElasticsearchDataProvider* ist verantwortlich um für einen gegebenen Typ von Entität entweder eine einzelne Entität, oder eine ganze Liste von Entitäten zur Verfügung zu stellen.
In unserem Fall benützt der DataProvider unsere eigene ElasticsearchAdapter-Implementierung um Abfragen auf den Elasticsearch Server zu machen.
Der DataProvider verbindet dabei die Antwort des Adapters mit dem ContextMapper und des EntityBuilders um aus einem Request eine entsprechende Liste von Entitäten zu erstellen.
Mehr Informationen zu den DataProvider findet man in der Dokumentation von API-Platform: https://github.com/api-platform/docs/blob/master/core/data-providers.md

2.7 Normalizer

Im Framework API-Platform können Normalizer-Klassen dazu verwendet werden um die API-Antworten mit spezifischen Informationen anzureichern oder zu filtern. In unserem speziellen Fall verwenden wir die Anreicherung mit Hydra Konzepten und die Filterung von nicht gesetzten Attributen (Wert ist *NULL*).
API-Platform unterstützt die Hydra-Konzepte standardmässig nur für das Format JSON-LD. Da alle unsere Ausgabeformate spezifische Serialisierung von RDF sind, können wir die Hydra-Konzepte in allen Serialisierungen anbieten. Dazu erweitern wir die Klassen von API-Platform und heben die Einschränkung für JSON-LD auf.

2.8 Encoder

Nach den Normalizern kommen verschiedene Encoder Klassen ins Spiel, die die normalisierte Antwort noch in das spezifisch verlangte Ausgabeformat konvertieren. In unserem Fall erhalten die Encoder die Daten immer im Format JSON-LD und erstellen dann daraus mit Hilfe der Bibliothek EasyRDF[3] die entsprechende Antwort.
Eine Ausnahme bildet dabei die Serialisierung zu HTML. Da wir auch bei der HTML-Serialisierung RDF Konzepte unterstützen möchten, reichern wir die Antwort mit RDFa[4] an. Dazu haben wir einen eigenen Serialisierungsmechanismus implementiert mit der Template-Library Twig[5].

[3] http://www.easyrdf.org/

[4] https://rdfa.info/

[5] http://twig.sensiolabs.org/