



Fernandès • Hombergs • Jerat • Pfeleiderer

# Spring Boot 3

So geht moderne Java-Entwicklung –  
Konzepte und Anwendungen

**dpunkt.verlag**



**François Fernandès** ist Senior Solution Architect bei Digital Frontiers und hat fast 20 Jahre Erfahrung in der Java-Softwareentwicklung und dem Java-Ökosystem. Seine Erfahrungen reichen dabei von der Entwicklung von Java-Desktop-Anwendungen über verteilte Applikationen bis hin zu Softwarearchitekturen in hybriden Cloud-Umgebungen. Den ersten Kontakt mit Spring hatte François mit dem Spring Framework 2.0. Seither begleiten ihn das Spring Framework, Spring Boot sowie diverse weitere Spring-Projekte in den unterschiedlichsten Einsätzen.



**Tom Hombergs** ist Principal Developer bei Atlassian in Sydney und dort für den Java Tech Stack verantwortlich, der von hunderten Teams weltweit zur Entwicklung der Atlassian-Produkte genutzt wird. Spring Boot ist ein zentraler Bestandteil dieses Tech Stacks. Er arbeitet seit vielen Jahren mit Spring und Spring Boot und trieb bei Atlassian unter anderem die unternehmensweite Migration von Spring Boot 2 zu Spring Boot 3 für über 800 (Micro-)Services voran. Tom ist Autor von »Get Your Hands Dirty on Clean Architecture« (deutsche Ausgabe: »Clean Architecture Praxisbuch«) und »Stratospheric – from Zero to Production with Spring Boot and AWS« und schreibt gelegentlich auf [reflectoring.io](https://reflectoring.io) über Themen der Softwareentwicklung.



**Benedikt Jerat** ist Senior Consultant bei Digital Frontiers. Seine Schwerpunkte liegen auf der Softwareentwicklung im Umfeld von JVM-Sprachen wie Java, Kotlin und Scala, wobei er besonders auf Spring und Spring Boot setzt. In seinen Projekten fokussiert er sich auf die funktionale Programmierung und die Prinzipien des Software Craftsmanship. Benedikt ist Autor zahlreicher Blog-Posts und Artikel in Fachzeitschriften und regelmäßiger Speaker auf Konferenzen zu aktuellen Themen.



**Florian Pfeiderer** ist Co-Founder und Senior Consultant bei Digital Frontiers. In dieser Rolle beschäftigt er sich vor allem mit den Themen Architektur, Microservices und Software Craftsmanship und unterstützt seine Kunden auf ihrem Weg in die Cloud. Das Spring Framework benutzte er zum ersten Mal vor über zehn Jahren, bei Spring Boot war er Nutzer der ersten Stunde. Trotz oder gerade wegen einiger Ausflüge in andere Ökosysteme kehrt er immer wieder gerne zu Spring Boot zurück und empfiehlt es seinen Kunden mit Begeisterung und Überzeugung. Außerhalb von Projekten gibt er sein Wissen regelmäßig auf Konferenzen oder in Fachzeitschriften weiter.

#### Coypright und Urheberrechte:

Die durch die dpunkt.verlag GmbH vertriebenen digitalen Inhalte sind urheberrechtlich geschützt. Der Nutzer verpflichtet sich, die Urheberrechte anzuerkennen und einzuhalten. Es werden keine Urheber-, Nutzungs- und sonstigen Schutzrechte an den Inhalten auf den Nutzer übertragen. Der Nutzer ist nur berechtigt, den abgerufenen Inhalt zu eigenen Zwecken zu nutzen. Er ist nicht berechtigt, den Inhalt im Internet, in Intranets, in Extranets oder sonst wie Dritten zur Verwertung zur Verfügung zu stellen. Eine öffentliche Wiedergabe oder sonstige Weiterveröffentlichung und eine gewerbliche Vervielfältigung der Inhalte wird ausdrücklich ausgeschlossen. Der Nutzer darf Urheberrechtsvermerke, Markenzeichen und andere Rechtsvorbehalte im abgerufenen Inhalt nicht entfernen.

**François Fernandès · Tom Hombergs ·  
Benedikt Jerat · Florian Pfeiderer**

# **Spring Boot 3**

**So geht moderne Java-Entwicklung –  
Konzepte und Anwendungen**

François Fernandès · Tom Hombergs · Benedikt Jerat · Florian Pfeleiderer

Lektorat: Dr. Benjamin Ziech  
Projektkoordinierung: Julia Griebel  
Copy-Editing: Annette Schwarz, Ditzingen  
Satz: inpunkt[w]o, Wilnsdorf ([www.inpunktwo.de](http://www.inpunktwo.de))  
Herstellung: Stefanie Weidner  
Umschlaggestaltung: Eva Hepper, Silke Braun

Bibliografische Information der Deutschen Nationalbibliothek  
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;  
detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN:  
Print 978-3-86490-994-8  
PDF 978-3-98890-125-5  
ePub 978-3-98890-126-2

1. Auflage 2024  
Copyright © 2024 dpunkt.verlag GmbH  
Wieblinger Weg 17  
69123 Heidelberg

*Schreiben Sie uns:*  
Falls Sie Anregungen, Wünsche und Kommentare haben,  
lassen Sie es uns wissen: [hallo@dpunkt.de](mailto:hallo@dpunkt.de).

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autoren noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

---

# Inhaltsübersicht

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>Teil I</b>	<b>Grundlagen</b>	<b>5</b>
<hr/>		
<b>2</b>	<b>Hallo, Spring Boot</b>	<b>7</b>
<b>3</b>	<b>Spring-Grundlagen</b>	<b>15</b>
<b>4</b>	<b>Spring-Boot-Grundlagen</b>	<b>27</b>
<b>5</b>	<b>Konfiguration</b>	<b>35</b>
<b>6</b>	<b>Build Management mit Spring Boot</b>	<b>53</b>
<b>7</b>	<b>Einführung ins Testen</b>	<b>61</b>
<b>8</b>	<b>Troubleshooting einer Spring-Boot-Anwendung</b>	<b>67</b>
<b>Teil II</b>	<b>Anwendungsfälle</b>	<b>85</b>
<hr/>		
<b>9</b>	<b>Einen REST-Service entwickeln</b>	<b>87</b>
<b>10</b>	<b>Das Reactor-Framework verwenden</b>	<b>91</b>
<b>11</b>	<b>Eine GraphQL-API entwickeln</b>	<b>95</b>
<b>12</b>	<b>Integration einer SPA mit Spring Boot</b>	<b>105</b>
<b>13</b>	<b>Ein serverbasiertes Web-Frontend entwickeln</b>	<b>119</b>
<b>14</b>	<b>Eine Datenbank anbinden</b>	<b>129</b>
<b>15</b>	<b>Eine CLI-Anwendung entwickeln</b>	<b>133</b>
<b>16</b>	<b>Architektur-Governance mit Spring Boot</b>	<b>147</b>

<b>Teil III</b>	<b>Referenz</b>	<b>157</b>
17	Testing	159
18	Spring Reactive	189
19	Spring Web MVC	211
20	HTTP-Clients mit Spring	225
21	GraphQL	231
22	Spring Boot Developer Tools	255
23	Events	259
24	Caching	269
25	Messaging	275
26	Spring Data	289
27	Spring Cloud Config	315
28	Spring Security	323
29	Observability	339
30	Docker-Images mit Spring Boot	377
31	Native Images mit Spring Boot	385
32	Spring Boot erweitern	393
33	Coordinated Restore at Checkpoint (CRaC)	407
34	Migration von Spring Boot 2 zu Spring Boot 3	413
35	Ausblick	425

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Warum Spring Boot? .....	1
1.2	Für wen ist dieses Buch? .....	2
1.3	Aufbau des Buchs .....	2
1.4	Codebeispiele .....	3
<b>Teil I</b>	<b>Grundlagen</b>	<b>5</b>
<b>2</b>	<b>Hallo, Spring Boot</b>	<b>7</b>
2.1	Einleitung .....	7
2.2	JDK installieren .....	7
2.3	Kickstart mit dem Spring Initializr .....	8
2.4	Projektstruktur .....	9
2.5	Die Anwendung bauen .....	10
2.6	Die Anwendung starten .....	11
2.7	Einen REST-Controller bauen .....	12
<b>3</b>	<b>Spring-Grundlagen</b>	<b>15</b>
3.1	Dependency Injection und Inversion of Control .....	15
3.1.1	Inversion of Control .....	15
3.1.2	Dependency Injection .....	16
3.2	Der Spring Application Context .....	19
3.3	Application Context mit XML konfigurieren .....	20
3.4	Java-Konfiguration im Detail .....	21
3.4.1	@Configuration und @Bean .....	21
3.4.2	@Component und @ComponentScan .....	22
3.4.3	@Configuration und @ComponentScan kombinieren ...	23
3.5	Was haben wir vom Spring Container? .....	25

<b>4</b>	<b>Spring-Boot-Grundlagen</b>	<b>27</b>
4.1	Bootstrapping	27
4.2	Den Application Context beeinflussen	29
4.3	Embedded Webserver	31
4.4	Dependency Management	32
4.5	Integrationen	32
4.6	Produktionsbetrieb	33
<b>5</b>	<b>Konfiguration</b>	<b>35</b>
5.1	Warum Konfiguration?	35
5.2	Konfigurationsparameter	36
5.2.1	Konfigurationsparameter definieren	36
5.2.2	Parameter als String injizieren	38
5.2.3	Parameter typischer injizieren	39
5.2.4	Parameter validieren	41
5.2.5	Default-Werte definieren	41
5.3	Profile	42
5.3.1	Konfigurationsparameter für ein Profil definieren	42
5.3.2	Ein Profil aktivieren	44
5.3.3	Konfigurationswerte aus unterschiedlichen Quellen kombinieren	45
5.4	Das Environment-Bean	45
5.5	Best Practices für Konfigurationsmanagement	46
5.5.1	Konfigurationsdateien im JAR	46
5.5.2	Secrets als Umgebungsvariablen definieren	47
5.5.3	Umgebungsvariablen explizit definieren	47
5.5.4	Profile nur für Umgebungen	48
5.5.5	Konfiguration von Default-Werten	51
<b>6</b>	<b>Build Management mit Spring Boot</b>	<b>53</b>
6.1	Überblick	53
6.2	Gradle oder Maven?	53
6.3	Spring Boot »Fat JAR«	54
6.4	Das Gradle Plugin	56
6.5	Das Maven Plugin	57
6.6	Dependency Management mit Spring Boot	58
6.6.1	Spring-Boot-BOM mit Gradle konsumieren	59
6.6.2	Spring-Boot-BOM mit Maven konsumieren	59



<b>7</b>	<b>Einführung ins Testen</b>	<b>61</b>
7.1	Testen – wieso, weshalb, warum? . . . . .	61
7.2	Spring Boot Starter Test . . . . .	62
7.3	Unit-Tests . . . . .	62
7.4	SpringExtension (JUnit 5) . . . . .	63
7.5	Integrationstests mit Spring . . . . .	63
	7.5.1 @SpringBootTest . . . . .	65
	7.5.2 Ausblick auf Slice-Annotationen . . . . .	66
<b>8</b>	<b>Troubleshooting einer Spring-Boot-Anwendung</b>	<b>67</b>
8.1	Spring-Boot-Magie . . . . .	67
8.2	Troubleshooting-Werkzeuge . . . . .	67
	8.2.1 Lokal reproduzieren . . . . .	68
	8.2.2 Debug-Modus . . . . .	69
	8.2.3 Actuator-Endpoints . . . . .	69
	8.2.4 Logging . . . . .	69
	8.2.5 Hooks . . . . .	70
8.3	Troubleshooting-Anwendungsfälle . . . . .	71
	8.3.1 Welche Beans stehen mir zur Verfügung? . . . . .	71
	8.3.2 Wo kommt eine Bean her? . . . . .	72
	8.3.3 Warum ist meine Bean nicht im Application Context? . . .	73
	8.3.4 Warum ist eine Bean doppelt im Application Context? . . .	75
	8.3.5 Wie ist meine Anwendung konfiguriert? . . . . .	76
	8.3.6 Welche Konfigurationsparameter werden (nicht) genutzt? . . . . .	77
	8.3.7 Welche Endpoints stellt meine Anwendung zur Verfügung? . . . . .	81
	8.3.8 Warum startet meine Anwendung nicht? . . . . .	83
<b>Teil II</b>	<b>Anwendungsfälle</b>	<b>85</b>
<b>9</b>	<b>Einen REST-Service entwickeln</b>	<b>87</b>
9.1	Was ist REST? . . . . .	87
9.2	Codebeispiel . . . . .	88
9.3	Endpoints . . . . .	89
9.4	Request Body und Parameter . . . . .	89
9.5	Fehlerbehandlung . . . . .	90

<b>10</b>	<b>Das Reactor-Framework verwenden</b>	<b>91</b>
10.1	Was ist das Reactor-Framework? .....	91
10.2	Codebeispiel .....	92
10.3	Andere reaktive Operatoren .....	93
<b>11</b>	<b>Eine GraphQL-API entwickeln</b>	<b>95</b>
11.1	GraphQL in Kürze .....	95
11.2	Die Beispielanwendung .....	96
11.3	Erstellung der Anwendung .....	96
11.4	Abbildung des Datenmodells in GraphQL .....	98
11.5	Implementierung des Controllers .....	98
	11.5.1 Query Mapping .....	99
	11.5.2 Schema Mapping .....	100
	11.5.3 Mutation Mapping .....	101
11.6	Implementierung einer Subscription .....	102
<b>12</b>	<b>Integration einer SPA mit Spring Boot</b>	<b>105</b>
12.1	Herausforderungen von SPAs .....	105
	12.1.1 Handhabung von URLs .....	105
	12.1.2 Integration unterschiedlicher Toolings .....	106
12.2	Mögliche Varianten für ein Deployment .....	107
	12.2.1 Integration in das Spring-Boot-JAR .....	107
	12.2.2 Unabhängige Deployments des Frontends und Backends .....	108
12.3	Ein Spring Boot Backend mit React Frontend .....	110
	12.3.1 Verzeichnisstruktur .....	110
	12.3.2 Konfiguration des Builds .....	111
	12.3.3 Integration einer REST-API in das Frontend .....	114
	12.3.4 Unterstützung von Deep-Links mit der History-API .....	117
<b>13</b>	<b>Ein serverbasiertes Web-Frontend entwickeln</b>	<b>119</b>
13.1	Warum serverseitig? .....	119
13.2	Die Beispielanwendung .....	120
13.3	»Hello World« mit Thymeleaf .....	120
13.4	Formulardaten verarbeiten .....	121
13.5	Interaktivität mit HTMX .....	124
13.6	Spring Boot Developer Tools .....	128
13.7	Weitere Ressourcen .....	128

---

<b>14</b>	<b>Eine Datenbank anbinden</b>	<b>129</b>
14.1	Datenbanken .....	129
14.2	Codebeispiel .....	130
14.3	Spring Data Repositories .....	131
14.4	Transaktionen .....	132
<b>15</b>	<b>Eine CLI-Anwendung entwickeln</b>	<b>133</b>
15.1	Eine ADR-Management-Anwendung .....	133
15.2	Erstellen des Projekts .....	134
15.2.1	Die ADR-API .....	134
15.3	Ein erstes Kommando .....	136
15.4	Registrieren von Kommandos .....	137
15.4.1	Programmatische Registrierung von Kommandos .....	138
15.4.2	Kommandos mit Annotationen definieren .....	140
15.4.3	AvailabilityProvider .....	142
15.4.4	Anzeige eines eigenen Prompts .....	143
15.5	Ausführung im interaktiven oder Kommando-Modus .....	145
15.6	Alternativen zu Spring Shell .....	146
<b>16</b>	<b>Architektur-Governance mit Spring Boot</b>	<b>147</b>
16.1	Warum Architektur-Governance? .....	147
16.2	Komponentenbasierte Architektur .....	148
16.3	Komponentenbasierte Architektur mit Spring Boot .....	149
16.4	Governance mit ArchUnit .....	152
16.5	Spring Modulith .....	153
<b>Teil III</b>	<b>Referenz</b>	<b>157</b>
<b>17</b>	<b>Testing</b>	<b>159</b>
17.1	Überblick .....	159
17.2	Unit Testing .....	161
17.2.1	Unit Testing ohne Spring .....	162
17.2.2	Unit Testing Utilities im Spring Framework .....	164
17.3	Integration Testing .....	166
17.3.1	Spring TestContext Framework .....	167
17.3.2	SpringExtension und SpringRunner .....	168
17.3.3	TestContextManager und ContextConfiguration .....	171

17.4	Wichtige Testfunktionalitäten im Überblick	173
17.4.1	@SpringBootTest	175
17.4.2	Context Initializers	176
17.4.3	@TestPropertySource	177
17.4.4	@ActiveProfiles	178
17.4.5	@TestConfiguration	179
17.4.6	@TestExecutionListeners	181
17.5	Testen von einzelnen Schichten mit Test Slices	182
17.5.1	Was sind Test Slices?	183
17.5.2	@WebMvcTest	184
17.5.3	@DataJpaTest	185
17.5.4	@JsonTest	186
17.5.5	Weitere Test Slices	187
<b>18</b>	<b>Spring Reactive</b>	<b>189</b>
18.1	Warum Reactive?	189
18.2	Grundlagen	192
18.2.1	Reactive Streams und Project Reactor	192
18.2.2	Reaktive Datentypen: Mono und Flux	193
18.2.3	Reactive Chaining	195
18.3	Spring WebFlux	196
18.3.1	Annotierte Controller	198
18.3.2	Funktionale Endpoints	201
18.3.3	WebClient statt RestTemplate	204
18.4	Testen von Reactive Streams	207
18.4.1	StepVerifier	207
18.4.2	WebFluxTest	209
<b>19</b>	<b>Spring Web MVC</b>	<b>211</b>
19.1	REST	211
19.1.1	RestController	211
19.1.2	Request-Daten auslesen	213
19.1.3	Reactive RestController	214
19.2	JSON-Mapping mit Jackson	214
19.2.1	Jackson Annotations	215
19.2.2	Custom Serializer	216
19.3	Error Handling	217

---

19.4	API-Dokumentation mit Spring REST Docs .....	219
19.4.1	Setup für REST Docs .....	220
19.4.2	JUnit-Test zur Generierung .....	220
19.4.3	Nutzen von Snippets .....	221
19.5	HATEOAS .....	221
19.5.1	Wieso überhaupt HATEOAS? .....	221
19.5.2	Repräsentation von Ressourcen .....	222
19.5.3	Hypermedia Links & Relations .....	222
19.5.4	Komfortable Links mit dem WebMvcLinkBuilder .....	223
19.5.5	Affordances .....	223
19.5.6	Integration mit RestTemplate und WebClient .....	224
<b>20</b>	<b>HTTP-Clients mit Spring</b> .....	<b>225</b>
20.1	RestTemplate .....	225
20.2	RestClient .....	225
20.3	WebClient .....	227
20.4	HTTP-Interface .....	228
<b>21</b>	<b>GraphQL</b> .....	<b>231</b>
21.1	Einführung .....	231
21.2	GraphQL in Kürze .....	231
21.2.1	Das GraphQL-Schema .....	232
21.2.2	Die Query Language .....	236
21.2.3	Ein GraphQL-Schema kann noch mehr .....	238
21.3	Ablauf eines GraphQL-Requests .....	238
21.3.1	Transportmethoden .....	238
21.4	Erstellung einer Spring-Boot-Anwendung mit GraphQL-API .....	239
21.4.1	Verwenden des Spring Initializr .....	239
21.4.2	Manuelles Hinzufügen der notwendigen Abhängigkeiten .....	240
21.4.3	Interaktives Testen mit GraphiQL .....	240
21.5	Implementierung einer GraphQL-API .....	241
21.5.1	Definieren eines GraphQL-Schemas .....	243
21.5.2	GraphQL Controller in Spring Boot .....	243
21.5.3	Optimierte Abfragen mit Batches .....	245
21.5.4	Definition eigener Skalare .....	250
21.6	Fehlerbehandlung .....	252

<b>22</b>	<b>Spring Boot Developer Tools</b>	<b>255</b>
22.1	Warum Developer Tools? .....	255
22.2	Developer Tools aktivieren .....	256
22.3	Restart .....	256
22.4	Live Reload .....	257
22.5	Einschränkungen der Developer Tools .....	258
<b>23</b>	<b>Events</b>	<b>259</b>
23.1	Lose Kopplung .....	259
23.2	Events versenden .....	260
23.3	Events empfangen .....	261
	23.3.1 ApplicationListener .....	262
	23.3.2 @EventListener .....	263
23.4	Synchron oder asynchron? .....	264
23.5	Spring Boot ApplicationEvents .....	266
<b>24</b>	<b>Caching</b>	<b>269</b>
24.1	Cache-Configuration .....	269
24.2	Caching-Annotationen .....	270
24.3	Cache-Implementierungen .....	272
	24.3.1 EhCache .....	273
	24.3.2 Caffeine .....	273
<b>25</b>	<b>Messaging</b>	<b>275</b>
25.1	Messaging im Überblick .....	275
25.2	JMS .....	277
	25.2.1 Mit einer JMS-API verbinden .....	277
	25.2.2 Nachrichten senden .....	278
	25.2.3 Nachrichten empfangen .....	280
	25.2.4 Message Converter .....	280
25.3	AMQP .....	281
	25.3.1 Das AMQP-Protokoll .....	281
	25.3.2 Konfiguration .....	281
	25.3.3 Nachrichten versenden .....	282
	25.3.4 Nachrichten empfangen .....	283
25.4	Kafka .....	283
	25.4.1 Konfiguration .....	283
	25.4.2 Nachrichten versenden und empfangen .....	284

---

25.5	Ausblick: Spring Cloud Stream .....	284
25.5.1	Konzepte .....	285
25.5.2	Nachrichten senden und empfangen .....	286
25.5.3	Vorteile .....	287
<b>26</b>	<b>Spring Data</b>	<b>289</b>
26.1	Überblick über Spring Data .....	289
26.2	Spring Data Repositories .....	290
26.2.1	Das Repository-Interface .....	291
26.2.2	Query-Methoden .....	294
26.3	Die DataSource Bean .....	296
26.4	JPA .....	298
26.4.1	Welchen Mehrwert bietet Spring Data JPA? .....	298
26.4.2	JPA-Repositories und -Entities .....	299
26.4.3	Datenbankinitialisierung mit JPA und Hibernate .....	302
26.5	R2DBC .....	303
26.5.1	Datenbankverbindung über ConnectionFactory .....	304
26.5.2	Datenbankzugriff .....	305
26.6	NoSQL mit Spring Data MongoDB .....	307
26.6.1	MongoDatabaseFactory und MongoTemplate .....	307
26.6.2	Spring Data Repository für MongoDB .....	309
26.6.3	Integrationstests mit @DataMongoTest .....	310
26.7	Schema-Migration mit Flyway .....	312
<b>27</b>	<b>Spring Cloud Config</b>	<b>315</b>
27.1	Warum Spring Cloud Config? .....	315
27.2	Der Spring Cloud Config-Server .....	316
27.3	Environment Repository .....	317
27.4	Spring Cloud Config-Client .....	319
<b>28</b>	<b>Spring Security</b>	<b>323</b>
28.1	Konzepte .....	323
28.1.1	Authentication .....	323
28.1.2	Authorization .....	324
28.1.3	Auto-Configuration .....	325
28.2	Absicherung von Methoden .....	327
28.3	Absicherung von HTTP-Pfaden .....	329
28.4	Benutzerverwaltung .....	331

28.5	Security Testing	332
28.6	OAuth	334
28.6.1	Was ist OAuth?	334
28.6.2	OAuth Resource-Server	335
28.6.3	OAuth-Client	336
<b>29</b>	<b>Observability</b>	<b>339</b>
29.1	Warum Observability?	339
29.2	Admin-Endpoints mit Actuator	339
29.2.1	Anwendungs-Metadaten ausgeben	342
29.2.2	Einen eigenen Actuator-Endpoint entwickeln	347
29.3	Logging	349
29.3.1	SLF4J	349
29.3.2	Logging via application.yml konfigurieren	350
29.3.3	Logback und Log4J direkt konfigurieren	351
29.3.4	Kombination der Logging-Konfigurationen	352
29.4	Metriken	353
29.4.1	Standardmetriken exportieren	353
29.4.2	Metriken an ein Observability-Produkt exportieren	355
29.4.3	Eigene Metriken exportieren	356
29.4.4	Tags	359
29.4.5	Metriken programmatisch anpassen	360
29.4.6	Histogramme und Perzentile	363
29.5	Tracing	366
29.5.1	Traces programmatisch auswerten	368
29.5.2	Traces in Log-Events ausgeben	369
29.5.3	Traces exportieren	370
29.5.4	Clients instrumentieren	372
29.5.5	Baggage	374
<b>30</b>	<b>Docker-Images mit Spring Boot</b>	<b>377</b>
30.1	Warum Docker?	377
30.2	Einfaches Docker-Image mit Spring Boot	377
30.3	Optimierte Docker-Images	379
30.4	Optimiertes Docker-Image mit Spring Boot	381
30.5	Docker abstrahieren mit Buildpacks	383



---

<b>31</b>	<b>Native Images mit Spring Boot</b>	<b>385</b>
31.1	Warum Native? . . . . .	385
31.2	Was ist ein Native Image? . . . . .	386
31.3	Anwendungsfälle für Native Images . . . . .	387
31.4	Ahead-of-Time-Optimierung mit Spring Boot . . . . .	388
31.5	Ein natives Image erstellen . . . . .	389
31.6	Ein natives Image testen . . . . .	390
31.7	Reachability-Metadaten erstellen . . . . .	390
<b>32</b>	<b>Spring Boot erweitern</b>	<b>393</b>
32.1	Cross-Cutting Concerns . . . . .	393
32.2	@Configuration und @Import . . . . .	394
32.3	@Enable...-Annotationen . . . . .	394
32.4	@AutoConfiguration . . . . .	395
32.5	Bedingte @Configuration . . . . .	396
32.6	Testen von @AutoConfigurations . . . . .	399
32.7	Starter . . . . .	400
32.8	Fortgeschrittene Erweiterungspunkte . . . . .	401
32.8.1	FactoryBean . . . . .	401
32.8.2	BeanPostProcessor . . . . .	402
32.8.3	BeanDefinitionRegistryPostProcessor . . . . .	403
32.8.4	EnvironmentPostProcessor . . . . .	405
<b>33</b>	<b>Coordinated Restore at Checkpoint (CRaC)</b>	<b>407</b>
33.1	Warum CRaC? . . . . .	407
33.2	Checkpoint und Restore . . . . .	408
33.3	Checkpoint und Restore mit Spring Boot . . . . .	409
33.4	Automatische Checkpoints . . . . .	410
33.5	Checkpoints in Docker . . . . .	410
33.6	CRaC vs. GraalVM . . . . .	411
<b>34</b>	<b>Migration von Spring Boot 2 zu Spring Boot 3</b>	<b>413</b>
34.1	Überblick . . . . .	413
34.2	Schritt 1: Bibliotheken analysieren und aktualisieren . . . . .	414
34.3	Schritt 2: Auf Java 17 aktualisieren . . . . .	415

34.4	Schritt 3: Das Spring-Boot-Upgrade vorbereiten .....	416
34.4.1	WebSecurityConfigurerAdapter durch WebSecurityFilterChain ersetzen .....	417
34.4.2	@AutoConfiguration .....	418
34.4.3	@LocalServerPort .....	419
34.4.4	@EnableWebFluxSecurity .....	419
34.5	Schritt 4: Spring Boot aktualisieren .....	420
34.5.1	Spring Boot auf 3.x aktualisieren .....	420
34.5.2	Bibliotheken aktualisieren .....	421
34.5.3	javax durch jakarta ersetzen .....	421
34.5.4	Spring Cloud Sleuth durch Micrometer ersetzen .....	422
34.5.5	@ConstructorBinding .....	423
34.5.6	HttpStatusCode .....	424
34.5.7	Konfigurationsparameter .....	424
<b>35</b>	<b>Ausblick</b>	<b>425</b>
	<b>Index</b>	<b>427</b>

---

# 1 Einleitung

## 1.1 Warum Spring Boot?

Das Spring Framework und Spring Boot sind aus dem Java-Ökosystem nicht mehr wegzudenken. Ohne Zweifel führt Spring Boot das Feld der Java-basierten Application-Frameworks an. In einer Umfrage von JetBrains aus dem Jahr 2022 gaben 67% der befragten Java-Entwickler an, Spring Boot zu nutzen, und 41% nutzten Spring MVC (beide Technologien werden in diesem Buch behandelt)<sup>1</sup>. Das Framework auf Platz 3 kommt nur auf einen einstelligen Prozentwert, und 23% der Befragten gaben an, gar kein Web-Framework zu nutzen, was die Prominenz von Spring nur verdeutlicht.

Softwareunternehmen wie Netflix und Atlassian setzen auf Spring Boot, um die Softwareentwicklung über Hunderte von Teams hinweg zu standardisieren!

Die führende Rolle von Spring Boot kommt nicht von ungefähr. Spring Boot basiert auf dem Spring Framework, das seit Anfang der 2000er Jahre kontinuierlich und sorgsam weiterentwickelt wird. Die Idee, ein auf Spring basierendes »Boot«-Framework zu entwickeln, das es uns vereinfacht, eine Spring-Anwendung zu konfigurieren, zu integrieren und zu betreiben, war nur eine Frage der Zeit.

Spring Boot macht uns als Entwickler produktiver, weil es uns viel Arbeit abnimmt. Es setzt dabei stark auf Konventionen, sodass wir nicht jede Kleinigkeit selbst konfigurieren müssen und sofort loslegen können. Diese große Stärke von Spring Boot wird aber teilweise auch als Hindernis empfunden, wenn man mit diesen Konventionen nicht vertraut ist oder nicht weiß, wo man sie nachschlagen kann.

Genau dieses Hindernis möchten wir mit diesem Buch aus dem Weg räumen.

---

1. <https://www.jetbrains.com/lp/devecosystem-2022/java/#what-web-frameworks-do-you-use-if-any->

## 1.2 Für wen ist dieses Buch?

Dieses Buch richtet sich sowohl an Java-Entwickler, die noch keine Erfahrung mit Spring oder Spring Boot haben, als auch an solche, die bereits Spring Boot 2 gearbeitet haben und einige Neuerungen in Spring Boot 3 kennenlernen oder ihr allgemeines Spring-Boot-Wissen auffrischen möchten.

Es bietet sich an, grundlegende Java-Kenntnisse (oder allgemeine Kenntnisse in objektorientierter Programmierung) mitzubringen, andernfalls sind die Konzepte und Codebeispiele in diesem Buch schwer nachzuvollziehen.

Da Spring Boot eine Fülle von Integrationen mit anderen Systemen anbietet, von denen einige in diesem Buch behandelt werden, eignet sich das Buch ebenfalls als Nachschlagewerk für Architekten, die Spring und Spring Boot für den Einsatz in einem Projekt evaluieren möchten.

Auch wenn dieses Buch die wichtigsten Features und Konzepte erläutert, erhebt es keinen Anspruch auf Vollständigkeit. Das Spring-Ökosystem ist mittlerweile so umfangreich, dass ein Buch mit einer vollständigen Abdeckung unpraktikabel wäre und vermutlich nur als Monitorstütze verwendet werden würde. Daher konzentrieren wir uns in diesem Buch auf die wichtigen Aspekte und verweisen an vielen Stellen für bestimmte Details auf die sehr gute Referenzdokumentation der jeweiligen Spring-Projekte. Es geht nicht darum, jeden Konfigurationsparameter auswendig zu lernen, sondern darum, die Konzepte zu verstehen, um sie dann bei Bedarf anwenden (und nachschlagen) zu können.

## 1.3 Aufbau des Buchs

Dieses Buch ist in drei Teile aufgeteilt.

In Teil I werden die **Grundlagen** von Spring und Spring Boot beschrieben. Dieser Teil ist insbesondere für Leser geeignet, die sich noch nicht viel mit Spring und Spring Boot beschäftigt haben. Leser, die bereits Erfahrung mit beiden Frameworks gesammelt haben, mögen Teil I gegebenenfalls ganz oder teilweise überspringen. Die Konzepte aus Teil I sind für das Verständnis vieler Kapitel aus den Teilen II und III erforderlich.

Teil II ist gedacht als Nachschlagewerk für bestimmte **Anwendungsfälle**. Jedes Kapitel in Teil II betrachtet einen konkreten Anwendungsfall (zum Beispiel die Entwicklung einer REST-API), mit dem man jederzeit in einem Softwareprojekt konfrontiert sein kann. Die Kapitel geben jeweils einen Überblick darüber, wie man den entsprechenden Anwendungsfall mit Spring Boot lösen kann.

Die Kapitel in Teil III sind **Referenzkapitel**, die im Detail auf Spring-Boot-Features eingehen. Hier lernt der Leser alles, was notwendig ist, um diese Features in einer Anwendung zu nutzen.

Mit Ausnahme von Teil I bauen die Kapitel nicht aufeinander auf, sodass die Kapitel größtenteils unabhängig voneinander gelesen werden können. Wir laden dazu ein, der Neugier zu folgen und diejenigen Kapitel zuerst zu lesen, die gerade das Interesse wecken oder im aktuellen Projekt von Bedeutung sind. Sollte ein Kapitel doch mal ein Konzept aus einem anderen Kapitel erfordern, findet sich ein Verweis im Text.

## 1.4 Codebeispiele

Ein Codebeispiel sagt mehr als tausend Worte. Das haben wir uns mit diesem Buch zu Herzen genommen. Die Konzepte von Spring und Spring Boot werden mit weit über 300 Codebeispielen veranschaulicht, um sie in Aktion zu sehen.

Noch viel mehr als ein Codebeispiel sagt jedoch eine lauffähige Anwendung. Deshalb haben wir für viele der Kapitel jeweils eine Spring-Boot-Anwendung entwickelt und in einem GitHub-Repository zur Verfügung gestellt. Wir empfehlen, dieses Repository zu klonen und mit den darin enthaltenen Spring-Boot-Anwendungen beim Lesen der Kapitel herumzuspielen. Am besten lernt man doch beim Spielen mit Code!

Das Repository ist unter der URL <https://github.com/dxfrontiers/spring-boot-3-buch> frei verfügbar.

Damit alle Codebeispiele aus diesem Buch und dem Repository lauffähig sind, empfehlen wir die Installation von JDK 21 und die Nutzung der aktuellen Versionen von Spring Boot 3.x und Spring 6.x. Die meisten Beispiele werden auch mit etwas älteren Versionen funktionieren, aber einige Features sind erst mit späteren Versionen verfügbar.



Teil I

---

# Grundlagen





---

## 2 Hallo, Spring Boot

*Im ersten Kapitel bauen wir direkt eine lauffähige »Hello World«-Anwendung, um mit einem Erfolgserlebnis zu starten.*

### 2.1 Einleitung

Dieses Buch ist ein Praxisbuch. In fast allen Kapiteln werden wir anhand von Codebeispielen die verschiedenen Aspekte einer Spring-Boot-Anwendung diskutieren. In den meisten Kapiteln sind diese Codebeispiele einer Beispielanwendung entnommen, deren Code auf GitHub öffentlich verfügbar ist. Wir laden dazu ein, den Beispielcode beim Lesen des Buches in der lokalen IDE durchzustöbern.

In diesem Sinne beginnen wir dieses Buch auch mit einem praktischen »Hello World«-Beispiel. Leser, die bereits Erfahrung mit Spring Boot haben, können dieses Kapitel guten Gewissens überspringen. Wenn nicht, laden wir dazu ein, den Beispielen in diesem Kapitel zu folgen und eine erste lauffähige Spring-Boot-Anwendung zu bauen.

In diesem Kapitel werden wir nicht jedes kleine Detail erläutern. Aber keine Angst, wir verweisen stets auf spätere Kapitel, in denen die Details vertieft werden.

### 2.2 JDK installieren

Um eine Spring-Boot-Anwendung zu entwickeln, müssen wir ein JDK (Java Development Kit) auf unserem Rechner installieren. Spring Boot 3 benötigt ein JDK der Version 17 oder höher. Wenn Sie bereits ein solches JDK installiert haben, können Sie mit dem nächsten Abschnitt fortfahren.

Wir empfehlen zur Verwaltung des JDK einen Runtime Manager wie `asdf`<sup>1</sup>, `jEnv`<sup>2</sup> oder `SDKMan`<sup>3</sup>. Diese Tools machen es einfach, zwischen verschiedenen Versionen eines JDK hin- und herzuwechseln. `SDKMan` und `asdf` automatisieren darüber hinaus auch das Herunterladen von verschiedenen JDK-Versionen. Wir können mit einem einzigen Kommandozeilenbefehl ein bestimmtes JDK herunterladen oder bestimmen, welches der bereits heruntergeladenen JDKs wir gerade benutzen möchten.

Zur Installation der Runtime Manager verweisen wir auf die Websites der Tools. Im Folgenden zeigen wir, wie wir mit `asdf` ein JDK installieren, das wir für alle Beispiele in diesem Buch benutzen können.

Vorausgesetzt, `asdf` ist erfolgreich installiert, müssen wir zunächst das Java-Plugin installieren (`asdf` unterstützt nämlich auch Runtimes anderer Programmiersprachen):

```
> asdf plugin add java
```

Dann können wir zum Beispiel mit dem folgenden Befehl ein JDK installieren:

```
> asdf install java latest:corretto
```

Hier installieren wir die aktuelle Version des Amazon-Corretto-JDK. Wir können uns auch eine Liste aller anderen JDKs anzeigen lassen und dann eines von denen installieren:

```
> asdf list all java | grep \\-21\\.
```

Diese Liste ist sehr lang, sodass wir sie hier mithilfe von `grep` auf die JDKs der Version 21 filtern.

Nun, da wir ein JDK installiert haben (oder mehrere), müssen wir unsere Umgebung noch so konfigurieren, dass dieses JDK auch überall genutzt wird:

```
> asdf local java latest:corretto
```

Der Befehl `java --version` sollte nun die erwartete Version des JDK ausgeben.

## 2.3 Kickstart mit dem Spring Initializr

Nun können wir unser Beispielprojekt erstellen. Das geht am einfachsten mit dem Spring Initializr, der unter <https://start.spring.io/> verfügbar ist. Hier können wir einige Parameter, wie zum Beispiel die Java-

---

1. <https://github.com/asdf-vm/asdf>
2. <https://www.jenv.be/>
3. <https://sdkman.io/>

Version, auswählen und uns eine ZIP-Datei mit einem Projektskelett erstellen lassen. Wir werden den Spring Initializr im Rest dieses Buches noch häufig verwenden, um Beispielprojekte für unterschiedliche Anwendungsfälle zu generieren.

Für unser »Hello World«-Beispielprojekt wählen wir die folgenden Parameter:

- **Project:** Gradle-Kotlin
- **Language:** Java
- **Spring Boot:** aktuelle 3.x-Version (Standardeinstellung)
- **Dependencies:** Spring Web

Alle anderen Einstellungen belassen wir mit ihren Standardwerten. Über den Button »Add Dependencies« können wir eine Auswahl bestimmter Integrationen und Features auswählen, die unserem Projekt dann als Dependencies hinzugefügt werden. Wir wählen zu Beginn nur die »Spring Web«-Dependency, um eine einfache Webanwendung zu entwickeln.

Mit einem Klick auf den »Generate«-Button lassen wir uns das Projekt erstellen und laden die ZIP-Datei herunter, die wir auf unserer Festplatte entpacken. Dann laden wir den Ordner als Projekt in unsere IDE und schauen uns im Projekt um.

## 2.4 Projektstruktur

Die zentrale Datei im Projektordner ist die Datei *build.gradle.kts*, unser Gradle-Build-Skript. Es ist in der Kotlin-DSL geschrieben, weil wir bei Erstellung des Projekts »Gradle-Kotlin« ausgewählt haben. Wir können auch »Gradle-Groovy« wählen, um das Skript in Groovy zu erstellen, oder »Maven«, um Maven als Build-Tool zu nutzen und eine *pom.xml*-Datei zu erzeugen.

Im Build-Skript finden wir eine Dependency zum Modul *spring-boot-starter-web*, das uns alle Features mitbringt, um eine Webanwendung zu entwickeln. Wir finden außerdem eine Dependency zum Modul *spring-boot-starter-test*, das uns einige Testwerkzeuge zur Verfügung stellt, die wir später im Testing-Kapitel noch genauer betrachten.

Das Build-Skript konfiguriert darüber hinaus die Plugins *org.springframework.boot* und *io.spring.dependency-management*, über die wir in Kapitel 6 noch mehr lernen.

Wir finden des Weiteren die Dateien *gradlew* und *gradlew.bat* und den Ordner *gradle*, die alle zum Gradle Wrapper<sup>4</sup> gehören. Die *gradlew*-

---

4. [https://docs.gradle.org/current/userguide/gradle\\_wrapper.html](https://docs.gradle.org/current/userguide/gradle_wrapper.html)

Skripte können wir nutzen, um Gradle aufzurufen, ohne Gradle lokal installieren zu müssen. Sie prüfen, ob eine Gradle-Installation verfügbar ist, und laden sie, falls erforderlich, herunter. So können wir unser Projekt auf unterschiedlichen Rechnern laufen lassen, ohne auf jedem Rechner Gradle installieren zu müssen. Wir werden unser Projekt später mit `./gradlew build` bauen.

Im `src`-Ordner finden wir genau drei Dateien:

- ***DemoApplication.java***: Diese Klasse ist der Einstiegspunkt zu unserer Spring-Boot-Anwendung. Sie ist mit der Annotation `@SpringBootApplication` versehen, die wir in Kapitel 4 genauer untersuchen. Wird die `main()`-Methode dieser Klasse aufgerufen, startet unsere Anwendung.
- ***application.properties***: Dies ist die zentrale Konfigurationsdatei für unsere Anwendung. Diese Datei ist leer, weil unsere Anwendung aktuell mit der Standardkonfiguration von Spring Boot zufrieden ist. Wir schauen uns die Konfiguration einer Spring-Boot-Anwendung im Detail in Kapitel 5 an.
- ***DemoApplicationTests.java***: Diese Klasse ist mit der Annotation `@SpringBootTest` versehen. Diese sorgt dafür, dass unsere Anwendung für jede `@Test`-Methode gestartet wird. Die leere Test-Methode `contextLoads()` hat augenscheinlich keinen Effekt, da unsere Anwendung aber für diesen Test gestartet wird, prüft dieser leere Test implizit, dass unsere Anwendung fehlerfrei starten kann. Schleicht sich zum Beispiel ein Konfigurationsfehler ein, schlägt dieser Test fehl. Er gibt uns also eine gewisse Sicherheit, obwohl die Test-Methode leer ist. Wir beschäftigen uns mit den Details des Testing mit Spring Boot in Kapitel 7.

Das sind auch schon alle Dateien unserer Anwendung. Mehr braucht unsere Anwendung nicht, um zu starten.

## 2.5 Die Anwendung bauen

Was macht man als Erstes, wenn man ein neues Projekt lokal untersucht? Richtig, man führt den Build-Prozess aus, um Vertrauen in das Projekt zu gewinnen. Erst wenn der Build-Prozess problemlos läuft, sollten wir damit anfangen, Änderungen am Projekt vorzunehmen.

In unserem Fall nutzen wir Gradle als Build-Werkzeug, und der Build-Befehl lautet wie folgt:

```
> ./gradlew build
```

Mit diesem Befehl starten wir den Gradle Wrapper, der bei Bedarf Gradle herunterlädt, den Code kompiliert, die Tests ausführt und ein Artefakt erzeugt, das unsere Anwendung beinhaltet. Der Build-Prozess sollte mit einem befriedigenden `BUILD SUCCESSFUL` quittiert werden, wenn alles wie erwartet funktioniert.

Gradle legt während des Builds den Ordner *build* an, der alle Ergebnisse des Build-Prozesses beinhaltet. Im Ordner *build/libs* finden wir eine Datei *demo-0.0.1-SNAPSHOT.jar* (der Name kann variieren, je nachdem, was man bei der Initialisierung des Projekts auf `start.spring.io` eingegeben hat). Diese Datei ist ein sogenanntes »Fat JAR«, also eine JAR-Datei, die alles beinhaltet, was unsere Anwendung braucht. Wir diskutieren die Build-Prozesse und Fat JARs noch im Detail in Kapitel 6.

Nachdem wir sichergestellt haben, dass unsere Anwendung kompiliert, können wir nun etwas mit ihr herumspielen.

## 2.6 Die Anwendung starten

Wir können unsere Anwendung auf verschiedene Wege starten, um sie auszuprobieren.

Nachdem wir die Anwendung mit `./gradlew build` gebaut haben, können wir die erzeugte JAR-Datei direkt mit dem Befehl `java -jar <DATEINAME>` starten. Diese Art, die Anwendung zu starten, ist einem echten Produktivszenario am ähnlichsten. Die JAR-Datei würde dabei auf eine Zielmaschine kopiert (am besten verpackt in einem Docker-Image, mehr dazu in Kapitel 30) und dort dann einfach gestartet.

In der täglichen Arbeit an unserer Anwendung wäre das allerdings unpraktisch, da wir zuvor immer den Build ausführen und auf deren Fertigstellung warten müssten.

Wir können unsere Anwendung auch direkt aus unserer IDE heraus starten. Wenn wir zum Beispiel in IntelliJ zu unserer Application-Klasse navigieren, sehen wir dort einen grünen »Play«-Button. Dieser startet die Anwendung, und wir können die Ausgabe direkt in der IDE-Konsole verfolgen.

Ein dritter Weg, die Anwendung zu starten, ist es, das Spring-Boot-Plugin von Gradle zu nutzen. Wir können einfach den Kommandozeilenbefehl `./gradlew bootRun` ausführen, und die Anwendung wird gestartet. Der Vorteil dieses Vorgehens ist es, dass wir etwaige Konfigurationsparameter unserer Anwendung in der *build.gradle.kts*-Datei hinterlegen können, sodass alle Entwickler die Anwendung in derselben Konfiguration starten können, ohne immer manuell die Parameter eingeben zu müssen. Aktuell hat unsere Anwendung aber noch keinerlei Konfigurationspara-

meter, deshalb ist es egal, wie wir sie starten. Das Kapitel 5 befasst sich noch im Detail mit der Konfiguration einer Spring-Boot-Anwendung.

Egal, wie wir die Anwendung starten, das Ergebnis ist aktuell noch nicht sehr beeindruckend. Das Einzige, was wir sehen, sind einige Log-Ausgaben in der Konsole, die hoffentlich mit einer Meldung wie dieser hier enden:

```
Started DemoApplication in 1.352 seconds
```

Die Anwendung läuft also. Sie tut aber noch nichts. Das möchten wir nun ändern.

## 2.7 Einen REST-Controller bauen

Das einfachste Element, das wir unserer Anwendung hinzufügen können, um ihr etwas Funktionalität zu geben, ist ein REST-Controller. Da wir das Modul *spring-boot-starter-web* unserem Projekt schon als *Dependency* hinzugefügt haben, haben wir bereits das Spring Web MVC Framework im Classpath, das alles mitbringt, was wir dafür brauchen.

Das MVC in »Spring Web MVC« steht für »Model, View, Controller«, ein beliebtes Pattern für Anwendungen, die eine Benutzeroberfläche anbieten. Der Controller nimmt Anfragen entgegen, erzeugt ein Model (oder lädt es aus einer Datenbank) und erzeugt eine View, die das Model für den Benutzer aufbereitet und zur Anzeige bringt. Diese View wird dem Benutzer dann angezeigt, in unserem Fall im Browser.

Ein einfacher »Hello World«-Controller sieht so aus:

**Listing 2-1**  
Ein einfacher  
Web-Controller

```
@RestController
public class HelloController {

    @GetMapping("/hello/{name}")
    public String hello(
        @PathVariable("name") String name
    ) {
        return String.format("Hello %s", name);
    }
}
```

Wir können diese Klasse einfach neben der *DemoApplication*-Klasse ablegen und Spring Boot greift sie automatisch auf.

Mit der Annotation *@RestController* markieren wir die Klasse als REST-Controller. Der Controller bietet also eine REST-Schnittstelle an, die Webanfragen entgegennimmt und ein Textformat als Antwort zurückgibt (üblicherweise JSON). Die »View« aus MVC besteht in diesem Fall also einfach aus Text und nicht aus einer Benutzeroberfläche.