SPENGERGASSE

Vergleich des Java EE Standard-Frameworks im Vergleich zu Spring zur Entwicklung von Business Applications

Paul MAUROVICH

Version 0.5 - 2021-04-05

Inhaltsverzeichnis

Chapter 1. Abstract

In diesem Paper werden die zwei (Web)-Frameworks, Java EE und Spring, auf unterschiedliche Kriterien untersucht:

- **Regelmäßige Updates und Wartbarkeit**: Wann und wie oft kommen Updates? Ist noch mit Updates zu rechnen? Wie wartbar ist der Code/das Framework?
- **Langfristiger Support**: Unter welcher Leitung gibt es noch Support? Ist mit langfristigem Framework Support zu rechnen?
- **Kosten und Programmiereleganz**: Ist das Framework Open Source, welche Lizenzkosten, welche Programmierparadigmen und welche allgemeinen Prinzipien gibt es?
- **Dokumentation**: Gibt es vorhandene übersichtliche und inhaltlich aussagekräftigen Dokumentationen und Anleitungen? Was beinhaltet die Dokumentation?
- **Funktionsumfang**: Welche Funktionen gibt es und welche Unterschiede gibt es zwischen den Frameworks?
- **Querschnittsfunktion**: Welche Metriken, Dependency Injection und welche Querschnittsfunktionen gibt es?
- **Developer Codebase und Community Support**: Wie groß ist die Community, gibt es genügend Hilfe und Informationen im Internet zu dem Framework, und wie aktuell wird das jeweilige Framework genutzt?

Die aus den Kriterien resultierenden Vor- und Nachteilen werden gegenübergestellt und bewertet.

Um die Arbeit angemessen verstehen zu können, werden fortgeschrittene Fähigkeiten und Erfahrungen in Informatik bzw. in der Programmierung vorausgesetzt. Das Paper dient als Basis einer Entscheidungsgrundlage des zu wählenden Frameworks für die Entwicklung von Business Applikationen.

Einleitung

Java Entwickler und Java Entwicklerinnen Teams stehen, sei es im Projekt für einen Kunden oder für das eigene Unternehmen, vor der Entscheidung, welches (Web) Framework sie für die Entwicklung von Business Applikationen im Projekt wählen sollten. Sei es *Hibernate*, *Spring*, *Apache Wicket*, *Grails* oder *Java EE* bzw. *Jakarta EE*, für solch ein Projekt stehen viele Frameworks zur Verfügung. Die Entscheidung fällt somit schwer, welches Framework am besten für das Projekt geeignet ist, weswegen hierfür in diesem Paper auf die zwei Frameworks Spring und Java EE, speziell auf die Entwicklung von Business Applikationen und Web Lösungen (REST), eingegangen wird.

Zu den jeweiligen Kategorien wird eine Bewertung in Form von Punkten vergeben, welchen den positiven Nutzungsgrad widerspiegeln. Hierfür werden Punkte von einer Skala von minimal 0 Punkten bis maximal 5 Punkten vergeben, wobei 0 Punkte für *"Nicht ausreichend bzw. nicht vorhanden"* und 5 Punkte für *"Sehr gut"* steht. Die Punktezahl wird am Ende jeder Kategorie evaluiert und am Ende des Papers ein finales resümierendes Fazit abgegeben.

Wichtig zu beachten ist, dass es kein "perfektes Framework" für die Entwicklung von Business Applikationen gibt, denn es kommt ganz darauf an, was die **spezielle und individuelle** Business Lösung können soll und welche Rahmenbedingungen wie Kosten, Umfang und Entwicklungsstandards es erfüllen soll.

Chapter 2. Java EE Framework

Java Enterprise Edition, kurz Java EE, früher auch J2EE genannt, ist ein Framework, ausgelegt für mittel bis große Projekte, welches es ermöglicht, umfassende und meist komplizierte (Unternehmens)-Anwendungen zu entwickeln. Vor allem für Webanwendungslösungen im Unternehmensbereich, meist in Kombination mit REST-API Schnittstellen und verteiltem Computing, kommt Java EE häufig zum Einsatz. Das Framework baut auf *Java SE* (Java Standard Edition) auf und wird meist mit einer Monolithen-Architektur entwickelt. Es wurde seit Java EE 8 auf Jakarta EE 8 umbenannt, da Oracle die Leitung dafür abgab, und es wurde Fokus auf Cloud Migration gelegt. Java EE baut auf offiziell auf der Verwendung von der objektorientierten Programmiersprache Java, inoffiziell könnte es auch mit anderen Programmiersprachen wie Kotlin funktionieren.

Benutzt wird es hauptsächlich zur Entwicklung von browserbasierten Web- und Enterprise (Business)-Applikationen und bietet zahlreiche Spezifikationen wie beispielsweise:

- **Web**: Web(Servlets) - Handhabung von Web Requests und Responses; *WebSockets*; *Java Server Faces*
- **Web Service**: *Java API* für RESTful Webservices, *JSON Processing* und *Binding* sowie *XML Webservices (SOAP)*
- **Enterprise**: Container für *Dependency Injection*; *Enterprise JavaBean APIs*; *Java Persistence- und Transaction API*; Senden und Empfangen von Enterprise Nachrichten durch *Java Message Service*
- **Weitere**: *Bean Validation API*; Lang laufende Hintergrundaufgaben werden durch *Batch Anwendungen* unterstützt; Anbindung von Java-Servern an Enterprise Information Systemen durch die *Java EE Connector Architektur*
- **JPA und JMS**: Datenbank Verbindungen und Datenbank Transaktionen

- **Bean Validation**: Deklariert eindeutige Constraints über eine Methode, ein Feld oder einer Klasse einer *JavaBeans-Komponente*
- **Application Server**: Genaue Konfiguration notwendig, jedoch vertiefende eigene Definierungen möglich

Eine grundlegende Java EE Architektur sieht wie folgt aus:

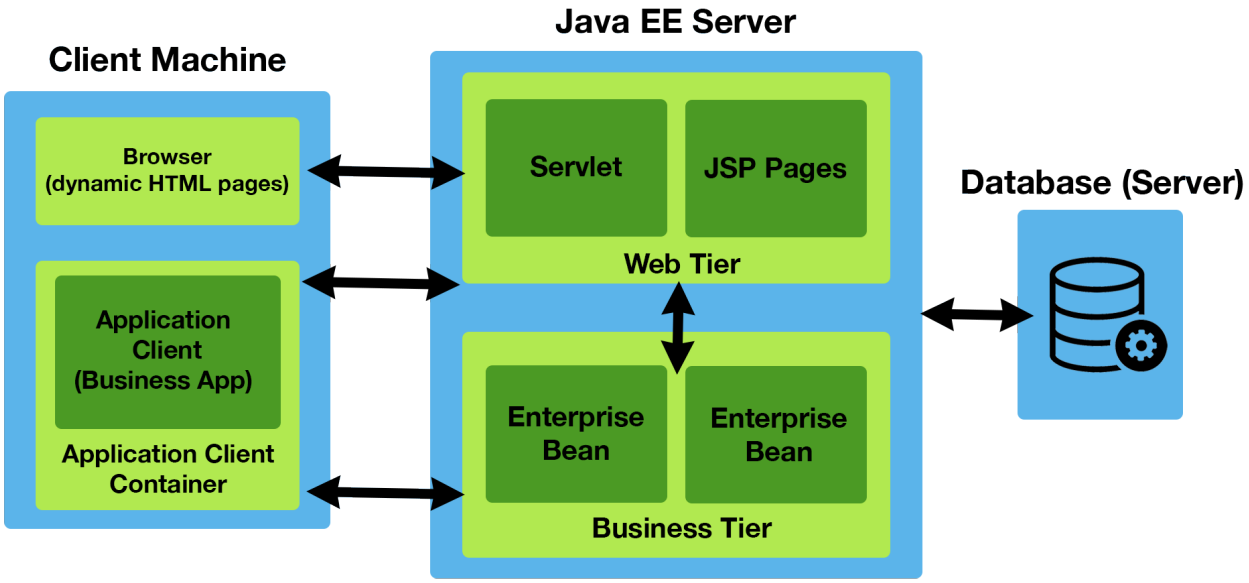

Figure 1. Java EE Architektur Überblick

Die Architektur besteht hauptsächlich aus drei Komponenten. Die *"Client Machine"* ist das Endgerät: *Browser* und/oder die *Business-Applikation*. Diese hat Verbindungen zum zweiten Teil, dem *"Java EE Server"*, welcher über ein Web Container, mit den *Web Servlets* sowie den *JSP Pages* (Jakarta Server Pages), welche es ermöglichen, Java Code in HTML- oder XML-Seiten einzubetten, und dem *Business Container*, wo sich die *Enterprise Beans* befinden. Der Server ist mit einer *Datenbank* bzw. einem *Datenbank-Server* verbunden, worüber Datenbanktransaktionen (CRUD) durchgeführt werden können. Transaktionen können mit zum Beispiel einer REST-API oder via einer SOAP-API durchgeführt werden.

Chapter 3. Spring Framework

Das Spring Framework ist eine Erweiterung von der Java EE Plattform ausgehend, für die Entwicklung von hauptsächlich Java Applikationen im Hinblick auf Web Services und REST Schnittstellen. Vor allem die zwei Komponenten von Spring, Spring Boot und Spring MVC, werden dafür häufig in Gebrauch gebracht. Hauptaugenmerk liegt auf der Entwicklung von Business Logik, ohne das viel Konfiguration von der Umgebung sowie Spezifikationen notwendig ist.

Spring (Boot) kommt mit einem automatisch konfigurierten Anwendungskontext sowie einem Web Server, wie Tomcat, um den Einstieg zu erleichtern und aufwendiges Aufsetzen des Frameworks zu vermeiden. Wohingegen Java EE offiziell nur mit der Programmiersprache Java läuft, so benötigt das Spring Framework keine spezifische Sprache, sie wird jedoch meist wird Java, Groovy oder Kotlin verwendet.

Der Schwerpunkt von Spring (Boot) liegt auf der Entwicklung von Microservices. Es ist größtenteils dafür ausgelegt und verfügt, wie auch Java EE, über einige Module. Diese sind zum Beispiel:

- **Datenbank Zugang, Daten Integrität**: JDBC, JMS, ORM, OXM, Transaktionen
- **Web**: Servlet, (Web)-Socket, Portlet
- **Core Container**: Beans, Core, Context und SpEL
- **Testing**: JUnit, Mockito etc.
- **Weitere**: AOP, Aspects, Messaging

In diesem Beispiel wird sich auf die Spring Komponente **Spring Boot** fokussiert. Die Architektur dessen sieht wie folgt aus:

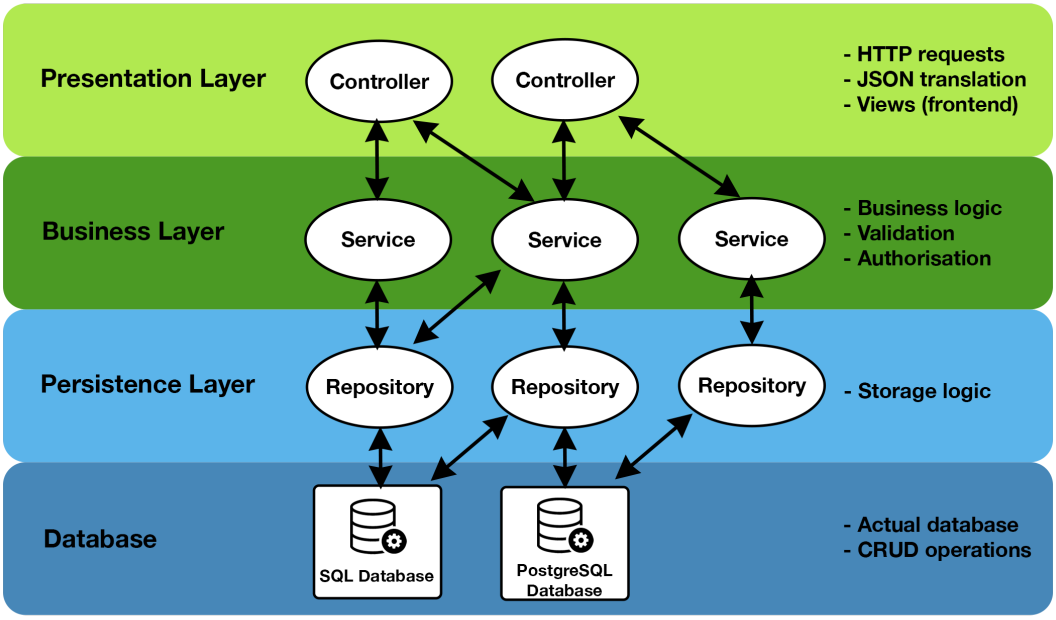

Figure 2. Spring Boot Framework 4-Schichten Architektur Überblick

Die Spring Boot Framework Architektur besteht aus vier Teilen. Die *"Presentation Layer"* Ebene, welche sich zum Beispiel um eingehende und ausgehende HTTP Requests/Responses kümmert. Sie übersetzt die Daten von und zu JSON Objekten, um den datenmässigen Umgang mit der Applikationen zu gewährleisten. Grundsätzlich ist sie mit einer Frontend HTML oder XML Seite, oder der Business Applikation verbunden.

Die *"Business Logic"* Ebene beinhaltet die Services, welche mit Business Logik ausgestattet sind. Es ist auch möglich "Fasaden" zwischen den Ebenen zu plazieren, um Datenintegrität und eine bessere Zugriffskontrolle zu gewährleisten. Die eingehenden Daten werden in diese Ebene transferiert und dort validiert bzw. autorisiert.

Um Daten abzugleichen und auf die gewählten Speicher Lösungen zuzugreifen wird die *"Persistence Layer"* Ebene benötigt. Sie verfügt über mehrere Repositorien, welche Datenbankzugriffe festlegen bzw. im Grunde eine geeignete Datenbankabfrage im Hintergrund ausführen.

Schlussendlich werden die Daten zum Beispiel in SQL, H2 oder PostgreSQL

Datenbanken gespeichert und dort verfügbar gemacht. Die *"Persistence Layer"* Ebene führt mit der *"Database Layer"* Ebene "CRUD-Operationen" (Create, Read, Update und Delete) durch, welche das Hinzufügen, Lesen, Aktualisieren und Löschen von Daten bereitstellen.

Chapter 4. Erste Schritte

Die sehr grundlegenden REST-API Realisierungs-Beispiele werden mittels der *IntelliJ IDEA* Entwicklungsumgebung umgesetzt, da sich diese sehr gut für die beiden Frameworks eignet.

4.1. Java EE Framework

1. Zu Beginn in *IntelliJ IDEA* ein neues Projekt erstellen und als Typ *"Java Enterprise"* auswählen. Danach bei *"Projekt Template"* den Punkt *"Rest service"* wählen und zum Beispiel *"GlassFish 5.0.0"* als *"Application server"* auswählen. Ein Application Server wie "Glassfish" muss jedoch zuvor installiert werden.
2. Anschließend die gewählten Dependencies überprüfen und geeignet Projekt- und Packagenamen festlegen.

Nach der Erstellung des Projektes wurden von *IntelliJ* Demo Klassen angelegt und die Dependencies werden in der `pom.xml` Datei verwaltet:

```
<!-- File: pom.xml -->
<dependencies>
    <dependency>
        <groupId>javax.ws.rs</groupId>
        <artifactId>javax.ws.rs-api</artifactId>
        <version>2.1.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <!-- ... -->
</dependencies>
```

Außerdem wird ein grundlegender API Controller in beispielsweise, je nach gewählten Namen, der Klasse `JavaEETest.java` angelegt:

```
// File: JavaEETest.java
@Path("/hallo-java-ee") 
public class JavaEETest {
    @GET 
    @Produces("text/plain") 
    public String hello() {
        return "Willkommen zu Java EE!"; 
    }
}
```

 Name der Basis Web Route nach der URL

 Festlegung der Operation (GET, POST, PUT, DELETE)

 Definierung des Rückgabeformats, welches in diesem Fall normaler Text ist

 Rückgabe eines String, um die Funktion zu testen

Nachdem das Projekt gestartet wurde, ist beispielsweise, je nach URL Definierung, unter `http://localhost:8080/JavaEEDemo-1.0-SNAPSHOT/hello-world` die sehr grundlegende REST-API abrufbar:

Figure 3. Java EE REST API GET-Response

4.2. Spring Framework

In diesem Beispiel wird sich auf eine Komponente von dem Spring Framework konzentriert: Spring Boot, in Kombination mit Java und dem Build Tool *Maven*.

Mithilfe des Spring Initializrs ist es möglich, ein fertiges Spring Boot Projekt Template anlegen zu lassen. Dafür werden folgende Schritte benötigt:

1. Erstens die Initialisierungswebseite besuchen und wichtige Informationen wie das Build Tool, die Programmiersprache, die Versionen und geeignete Projekt-

und Packagenamen vergeben.

2. Anschließend mit Klick auf den Knopf *"ADD DEPENDENCIES..."* die gewünschten Abhängigkeiten wie *"Spring Web"*, *"Spring Data JPA"*, *"H2 Database"* und *"Thymeleaf"* mit erneutem Klick hinzufügen.
3. Letztlich auf den Knopf *"GENERATE"* klicken und ein ZIP-Ordner mit dem vorkonfiguriertem Projekt wird automatisch heruntergeladen und ist, sofern es in einer Entwicklungsumgebung wie IntelliJ geladen wird, einsatzbereit. Eine Konfiguration des Application Servers ist bei Spring Boot nicht notwendig.

Die Dependencies werden im späteren `pom.xml`, in dem die Dependencies verwaltet werden, wie folgt angezeigt:

```
<!-- File: pom.xml -->
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
    <!-- ... -->
</dependencies>
```

Anschließend das Projekt öffnen und beispielsweise einen Controller `BootTestController.java` anlegen:

```java
//File: BootTestController.java
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("api/v1/test") 
public class BootTestController {

    @GetMapping("/hello") 
    public String hello() {
        return "Willkommen zu Spring Boot!"; 
    }
}
```

 Name der Basis Web Route nach der URL

 Name der spezifischen Route, welcher der Basis Route ergänzt wird, der Methode

 Rückgabe eines einfachen Strings, um die Funktion zu testen

Darüber hinaus hat der *"Spring Initializr"* auch eine Main Klasse erzeugt, welche nach nötigen Gebrauch mit zum Beispiel *@Bean Annotations* ergänzt werden kann.

```java
//File: DemoApplication.java
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

Die Applikation kann nun entweder via spezifischen Startknopf der gewählten IDE oder mit dem Befehl, je nach gewählten Build Tool, `./mvnw spring-boot:run` mit *Maven* gestartet werden.

Für den Gebrauch von *Maven* Kommandos muss *Maven* zuerst auf der *CLI* installiert werden oder kann in der IntelliJ IDEA mit Klick auf den Knopf *"Maven"* in der rechten Leiste benutzt werden.

Nun ist unter `http://localhost:8080/api/v1/test/hello` die sehr grundlegende REST-API abrufbar:

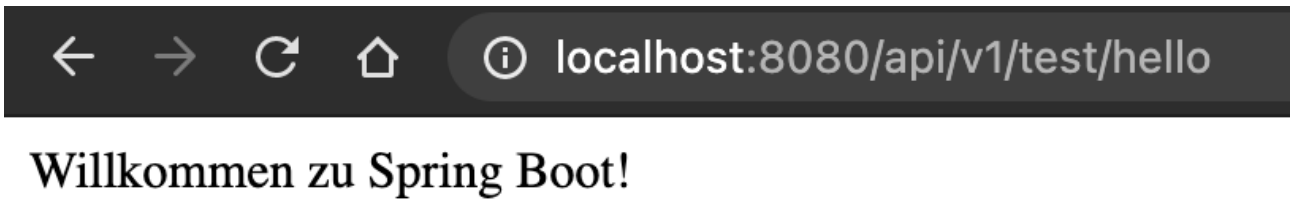

Figure 4. Spring Boot REST API GET-Response

Chapter 5. Bewertungskriterien

5.1. Regelmäßige Updates und Wartbarkeit

Java EE, seit Java EE 8 umbenannt auf Jakarta EE, erhält regelmäßig neue Versionsupdates. Die Frequentierung der Major Updates ist durchschnittlich alle zwei bis vier Jahre und enthält meist neue Features und Verbesserungen. Das letzte Hauptupdate (Jakarta EE 9) fand im Jahr 2020 statt. Das Spring Framework wird kontinuierlich aktualisiert und dessen letztes großes Update war im Jahr 2017 mit der Version Spring 5.0 und als letztmaliger Stable Release gilt die Version *Spring 5.3.4*, welche im Februar 2021 veröffentlicht wurde. Zwar erhalten beide Frameworks regelmäßig Updates, wohingegen das Spring Framework öfters Updates bekommt als Java EE. Dies ist wohl auch der stetig expandierenden Nutzerbasis von Spring und der Leitungsabgabe von Java EE durch Oracle geschuldet. Auch Dependencies werden seitens beider Frameworks im Laufe von Updates mit aktualisiert.

Durch Dependency Injection und Cloud Migration von den zwei Frameworks, bei Spring ist dies beispielsweise *Spring Boot*, ist eine gute Wartbarkeit und Aktualität gegeben. Spring Boot verfügt über *"Plain Old Java Objects (POJO)"*, welche sich durch kleine und "leichtgewichtige Klassen" auszeichnen. Sie ermöglichen eine präzise Wartbarkeit, da jegliche Logik in kleinen Klassen leicht erreichbar und nicht zu umfassend verschachtelt, platziert ist. Auch Java EE verfügt über Dependencies, welche einfach aktualisiert werden können. Meist reicht das bloße Ändern der Version und eine Aktualisierung der Abhängigkeiten. Bei der Wartbarkeit beider Frameworks kommt es vor allem darauf an, ob eine *Monolithen- oder Microservice Architektur* gewählt wurde. Letzteres bietet einen weit ausgehend mehr wartbaren Code, da der Code je nach Spezifikationen in verschiedene Module aufgesplittet ist. Hunderte Klassen in einem Package, unübersichtliche Klassennamen und hunderte Codezeilen in Klasse sind bei beiden Frameworks

hauptverantwortlich für schlechte Wartbarkeit.

Durch immer neue Updates und deren neuen Funktionen und Verbesserungen wird die Wartbarkeit immer besser, sei es mit der Cloud Migration von Java EE 8 oder Spring Boot, beide wurden auf den heutigen Stand der Technik gehoben und erreichen somit die volle Punktezahl von 5 Punkten.

Fazit:

	Java EE	**Spring**
Punkte	5	5

Table 1. Punkte Resümee "Regelmäßige Updates"

5.2. Langfristiger Support

Wie bereits im vorhergehenden Kapitel erwähnt, erhalten beide Frameworks stetig neue Versionsupdates. Jedoch wirkt seit 2017 Oracle, der damalige Leiter von der Java Enterprise Plattform, nicht mehr primär an der Entwicklung von Java EE mit, da sie die Leitung dafür aus mangelnder Interesse einer Weiterentwicklung abgegeben haben. Dies zeigt auf, dass Java EE immer weniger an Relevanz in der heutigen Software Gemeinschaft hat und einen langfristigen Support fragwürdig macht. Zwar wird mit einigen Updates in der Zukunft gerechnet, vor allem notwendige Sicherheitsupdates, doch bahnbrechende Weiterentwicklungsupdates werden wohl auf sich warten lassen.

Die Website *"JRebel"* beispielsweise, hat einige Entwickler befragt, ob sie von Java EE zu Spring migriert hätten bzw. dies tun möchten. Die Befragung ergab, dass lediglich 14 Prozent von Spring zu Java EE und im Kontrast dazu 36 Prozent von Java EE zu Spring migriert haben bzw. den Wechsel durchführen wollen. Deswegen enthält Java EE resümierendes für diese Kategorie 3 Punkte.

Viele Portale sprechen von dem *"Tod von Java EE"*, nachdem Oracle die Leitung

dafür abgegeben hat und allgemein bessere Frameworks auf dem Markt gebracht worden sind. ***"Negotiations Failed: How Oracle killed Java EE"***, so schreibt es beispielsweise der Autor *Markus Krag* in seinem Blog. In dem Bericht geht hervor, dass es einen Markenstreit zwischen Oracle und der Eclipse Foundation, der neuen Leitung von Jakarta EE, gab, welcher in keiner Einigung resultierte und Java EE dadurch einiges an Relevanz und Ansehen in der Software Gemeinschaft kostete.

Seitens des Spring Frameworks, vor allem bei den zwei Komponenten Spring Boot und Spring MVC, gibt es keine Anzeichen eines nahestehenden Endes des Supports. Unter der Leitung der Apache Foundation gewinnt das Framework immer mehr und mehr an Interesse und Nutzung unter der Entwicklergemeinschaft. Nicht nur sind große skalierbare Projekt mit dem Framework möglich, auch regelmäßige stabile /Major) Versionen kommen auf den Markt. Diesen Fakten geschuldet, erhält das Spring Framework in dieser Kategorie die volle Punktezahl von 5 Punkten.

Fazit:

	Java EE	**Spring**
Punkte	3	5

Table 2. Punkte Resümee "Langfristiger Support"

5.3. Kosten und Programmiereleganz

Seit der Übernahme durch die *Eclipse Foundation* ist Jakarta EE komplett Open Source. Oracle verfügt über die Markenrechte von "Java EE", weswegen die neue Leitung es auf "Jakarta EE" mit zusätzlich neuen Packagename umbenannt hat. Dadurch ist Jakarta EE größtenteils kostenlos zu nutzen, jedoch gibt es neben den frei zugänglichen Java EE Servern wie "Tomcat" oder "Glassfish", auch kostenpflichtige Server.

Java EE bietet folgende Paradigmen:

- **Cloud und PaaS**: Cloud Migration (Web), durch Java EE 8, und PaaS (Platform as a service)
- **Aspect oriented programming (AOP)**
- **Java Programmierparadigmen**: Die Standardprogrammierparadigmen von Java
- **Design Paradigmen POJO**: Unterstützung von POJO (Plain Old Java Object)
- Java EE unterstützt die **Reactive Programmierung**: Daten werden mit statischen oder dynamischen Datenflüssen verarbeitet

Jedoch verfügt es nicht über so viele Prinzipien wie Spring. Das Spring Framework verfolgt zudem neuartige Paradigmen, wo nach Java EE dabei zurückliegt.

Das Spring Framework unterliegt der Apache-Lizenz, welche eine Free-Software-Lizenz ist. Es ist somit unentgeltlich und auch Open-Source. Auch Spring hat sowohl kostenlose als auch kostenpflichtige Module und Server, welche aber grundsätzlich nicht notwendig sind. Das Framework verfügt unter anderem über folgende Prinzipien:

- **Lightweight**: Spring ist einfach aufgebaut und benötigt nicht viel Speicherplatz, beispielsweise ist die Basis Version nur ein Megabyte groß.
- **Inversion of control (IOC)**: Entwickler müssen Komponenten wie Bibliotheken nicht selbst erstellen/anlegen, sondern diese lediglich durch Dependency Injection in einer Konfigurationsdatei, zum Beispiel in der POM Datei, bestimmen. Spring IOC hat die Aufgabe, alle Dependencies lauffähig zu vereinen.
- **Aspect oriented programming (AOP)**: Spring unterstützt auch die aspektorientierte Programmierung. Wartbarkeit und Modularität wird durch die Trennung von logischen Aspekten und der Business Logik gewährleistet. AOP trennt diese zwei Komponenten, was bei der einfachen objektorientierten Programmierung schwer möglich ist.

- **Container**: Spring unterteilt Code in Container und handhabt Lebenszyklen und Anwendungskonfigurationen.
- Spring unterstützt auch die **Reactive Programmierung** mit der Dependency *"Reactor"*, vor allem in Kombination mit einer Microservice Architektur.

Fazit:

	Java EE	**Spring**
Punkte	3	5

Table 3. Punkte Resümee "Kosten, Eleganz beim Programmieren"

5.4. Dokumentation

Java EE bietet eine Dokumentation von Oracle, welche bei Weitem nicht so umfangreich, leicht zugänglich und übersichtlich wie die von dem Spring Framework ist. Die Oracle Dokumentation ist verschachtelt durch einige Links erreichbar und listet alle nennenswerte Packages chronologisch auf, welches an die Standard Java Dokumentation erinnert. Die Documentation hat sich durch Jakarta jedoch verbessert.

Die Spring Framework Dokumentation bietet zwar auch die einzelnen Package Dokumentation, aber übersichtlicher und leichter verständlich dargestellt. Außerdem gibt es zahlreiche Guides und Anleitungen, wie eine bestimmte Sache von Spring genau funktioniert, wie zum Beispiel wie man die ersten Schritte mit dem Framework durchführt oder wie eine REST-API mit Spring Boot aufgesetzt wird. Auch allgemein bietet Spring mehr Anleitung als das Java EE Framework. Beide teilen jedoch einige Portale, so wie zum Beispiel die Webseite *"Baeldung"*, welche sich auf Java und dem Spring Framework spezialisiert hat und hilfreiche Tutorials liefert.

Grundsätzlich bietet die Dokumentation beide Frameworks:

- **Klassen Beschreibungen und deren Nutzung**: Übersicht, Exceptions, Interfaces etc.
- **Genaue Package Beschreibungen**: Übersicht, Spezifikationen, Interfaces etc.
- **Genaue Methoden Dokumentation**: Beispiele, Exceptions, Konstruktoren, Parameter und Datentypen
- **Installations und Get Started Anleitungen**
- **Modulerklärungen**: Wie beispielsweise Anleitung für Integrationen, Web Servlets, Daten Transaktionen/Zugriffe etc.
- Ansicht von **veralteten** "deprecated" **Klassen-, Package- und Methoden**
- **Allgemeine Dokumentation der Programmiersprache**: Java EE hat die Java Dokumentation und Spring die Java-, Kotlin- oder Groovy-Dokumentation

Den Fakten geschuldet, dass Spring eine bessere Dokumentation im Hinblick auf Übersichtlichkeit, Inhalt und Erreichbarkeit liefert, erhält Spring die volle Punktezahl und Java EE, da man allgemein wenig(er) zu Java EE bzw. Jakarta EE im Internet findet, nur 3 Punkte. Auch allgemein die Art der Aufbereitung der Dokumentation ist uneinsichtig bei Java EE, da es seit der Übernahme durch die Eclipse Foundation unterschiedliche Dokumentation gibt. Jedoch bieten beide eine fundierte und vertrauenswürdige Dokumentation und liefern die gewünschten Informationen.

Fazit:

	Java EE	**Spring**
Punkte	3	5

Table 4. Punkte Resümee "Dokumentation"

5.5. Funktionsumfang

Das Spring Framework bietet eine breite Palette an Komponenten, wie Spring Boot,

Spring MVC, Spring Batch, Spring Data oder Spring Security.

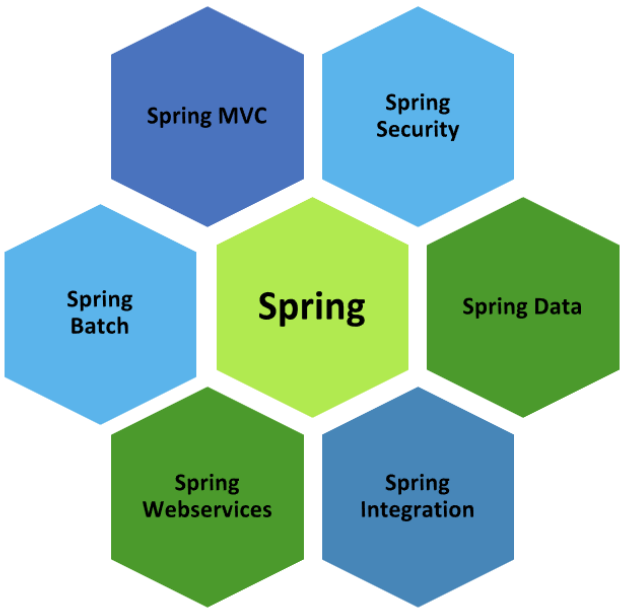

Figure 5. Spring Framework Komponenten

Java EE hingegen verfügt nur über sich selbst. Allgemein teilen beide Frameworks ähnliche Features:

- **Dependency Injection**
- **Web Anwendungen**
- **Application Server**: Wobei bei Java EE die Konfiguration von diesem notwendig ist und bei Spring (Boot) "out of box" kommt.
- **Datenbanken Verfügbarkeit**: JPA etc.

Nennenswerte Unterschiede:

Faktor	**Java EE**	**Spring (Boot)**
Einarbeitung	Aufwendig und Webserver Konfiguration notwendig	Einfach, viele Features kommen "out of box"
Programmiersprache	Java	Keine spezifische Sprache
UI	JSF2	Spring MVC

Faktor	Java EE	Spring (Boot)
Testing	Arquillian (AppServer nötig)	Spring Testing (Mockito, ...), JUnit
Transaktionen	JTA	JTA/Spring Data
AOP	Interceptor	Spring AOT
XML-lastig	Wenig	Viel
Geschwindigkeit	Schneller als Spring	Langsamer als Java EE

Table 5. Java EE vs Spring

Beide Frameworks bieten viele Features, Spring hingegen hat einige mehr. Dies liegt auch daran, dass es mehr Frameworks unterstützt und selbst beinhaltet. Java EE ist hingegen, laut Selbsttests, bei Applikationsstartzeiten um rund zehn Prozent schneller als Spring. Im Hinblick auf die Entwicklungen von Business Applikationen reichen die Features beider Framework grundlegend aus, Spring aber erleichtert durch mehr Funktionen und Frameworks die Implementierung der Applikationen, weswegen Spring insgesmat 4 Punkte, durch unter anderem die niedrigere Geschwindigkeit, und Java EE, durch den höheren Konfigurationsaufwand als Spring, 3 Punkte erhält.

	Java EE	Spring
Punkte	3	4

Table 6. Punkte Resümee "Funktionsumfang"

5.6. Querschnittsfunktion

Java EE sowie auch Spring unterstützen die Programmiersprache Java sowie Dependency Injection, weswegen sie einige Querschnittsfunktionen teilen:

- **Logging und Tracing**: Mit *log4j*, *Zipkin*, *Sleuth* und dem *ELK-Stack*
- **Caching**: Mechanismus, welcher es erlaubt, oft zu gegriffene Objekte und

Informationen, temporär zwischenzuspeichern und so ein erneutes Laden zu unterbinden. Daten werden im Cache der Applikation gespeichert.

- **Security**: Sicherung von Daten und Zugriffskontrolle durch hohe Authentifizierungsstandards. Zudem gibt es *"Spring Security"*, welches in einer Art auf beide Frameworks anwendbar ist.

Spring hat automatisierte Sicherheitsfunktionen in die Security Architektur implementiert, Java EE hingegen ist nicht so ausgebaut und hat keine speziellen Funktionen, wie "lightweight" Funktionen, *LDAP* (Lightweight Directory Access Protocol), *Web Form Authentifizierung* sowie *HTTP Authentisierung (Web Requests)*.

- **Health Endpunkt Metriken**: Vor allem in Kombination mit einer Microservice Architektur bieten *"Health Endpoint"* Metriken an, den Gesundheitsstand der Applikation, wie Uptime oder Latenz, zu überprüfen. Das Spring Framework bietet dafür zudem den **Actuator** an.

	Java EE	**Spring**
Punkte	3	5

Table 7. Punkte Resümee "Querschnittsfunktion"

5.7. Developer Codebase und Community Support

Seitens beider Frameworks gibt es eine mittel bis große Developer-Base. Java EE bzw. Jakarta EE wird laut der Webseite stackshare.io, welche unterschiedliche Frameworks. Programmiersprachen etc. bewertet und aufzeigt, welche Technologien heutzutage verwendet werden, von nur 29 Unternehmen genutzt. Darunter "TripAdvisior", "Biting Bit" und "IWB". Außerdem hat Java EE auch Integrationen in "Eclipse", "NetBeans IDE" sowie "Apache Wicket".

Auch Spring ist auf derselben Webseite vertreten. Angaben zufolge benutzen 501 Unternehmen Spring in ihrem Stack, darunter "Accenture", "Zalando" und auch "deleokorea".

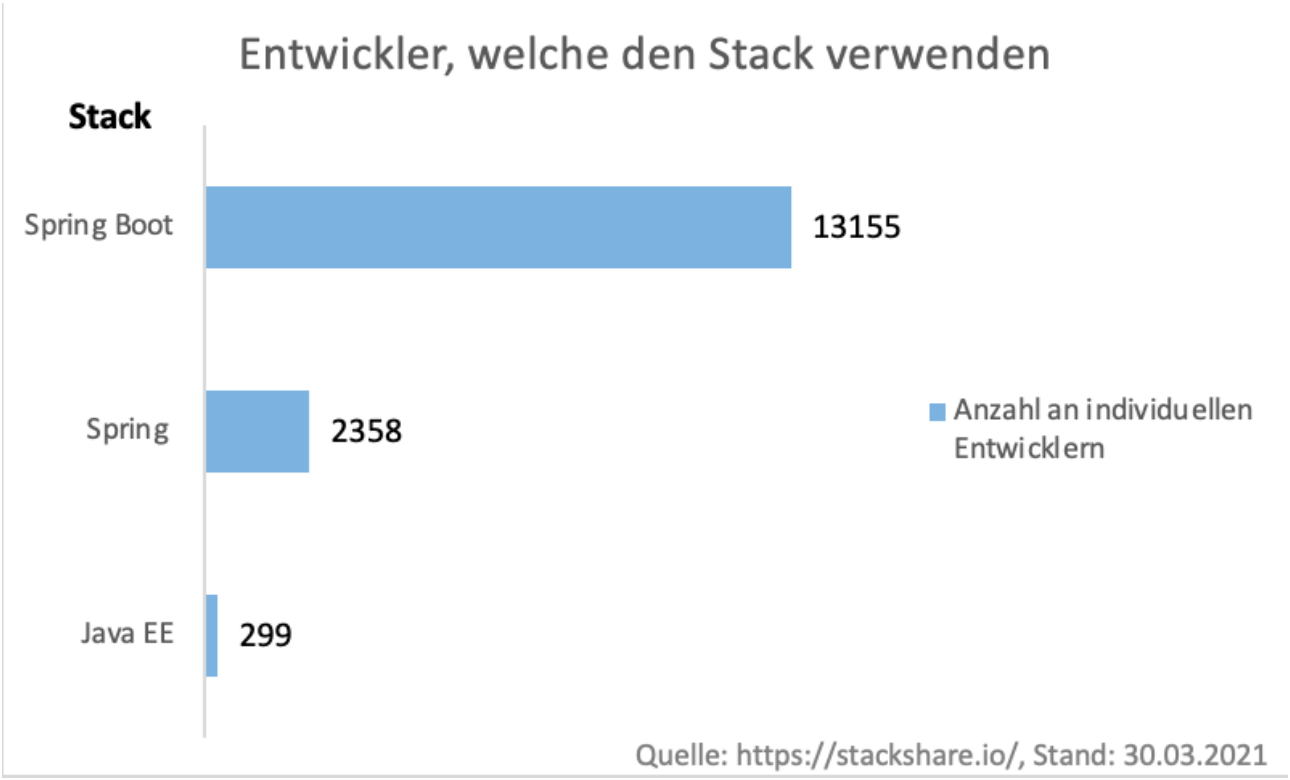

Figure 6. Entwickler, welche den Stack verwenden

Rund 13155 Entwickler haben bekannt gegeben, dass sie die Spring Komponente Spring Boot in ihrem Stack benutzten, bei Spring sind es rund 2358 Entwickler und Java EE mit nur wenigen 299 Entwicklern.

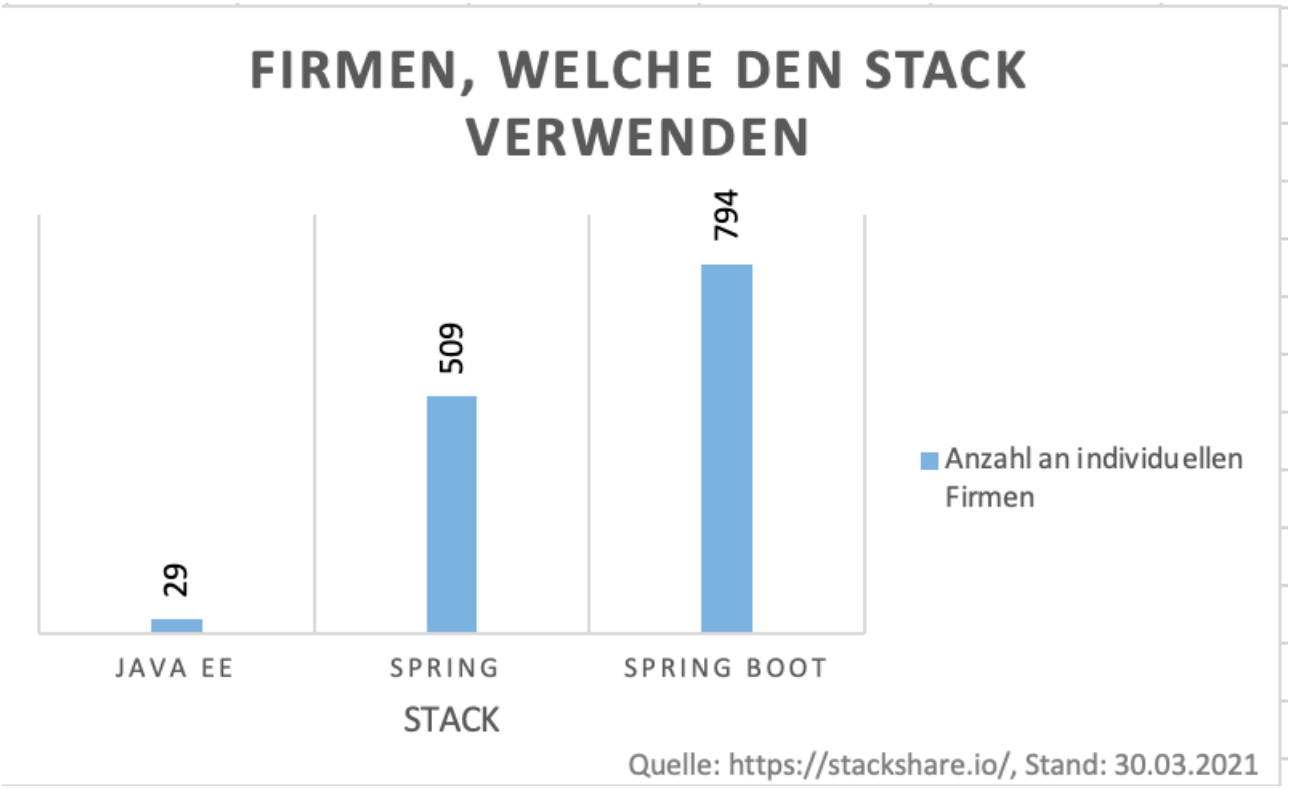

Figure 7. Firmen, welche den Stack verwenden

Auch zeigt der Trend, dass viele Firmen auf neue Stacks wie Spring und folglich Spring Boot setzen und nur mehr wenige Unternehmen Java EE als Stack angeben, welchen sie verwenden. Dies zeigt den heutigen Einsatz der zwei Frameworks ziemlich eindeutig, denn Spring (Boot) hat hierbei klar die Führung.

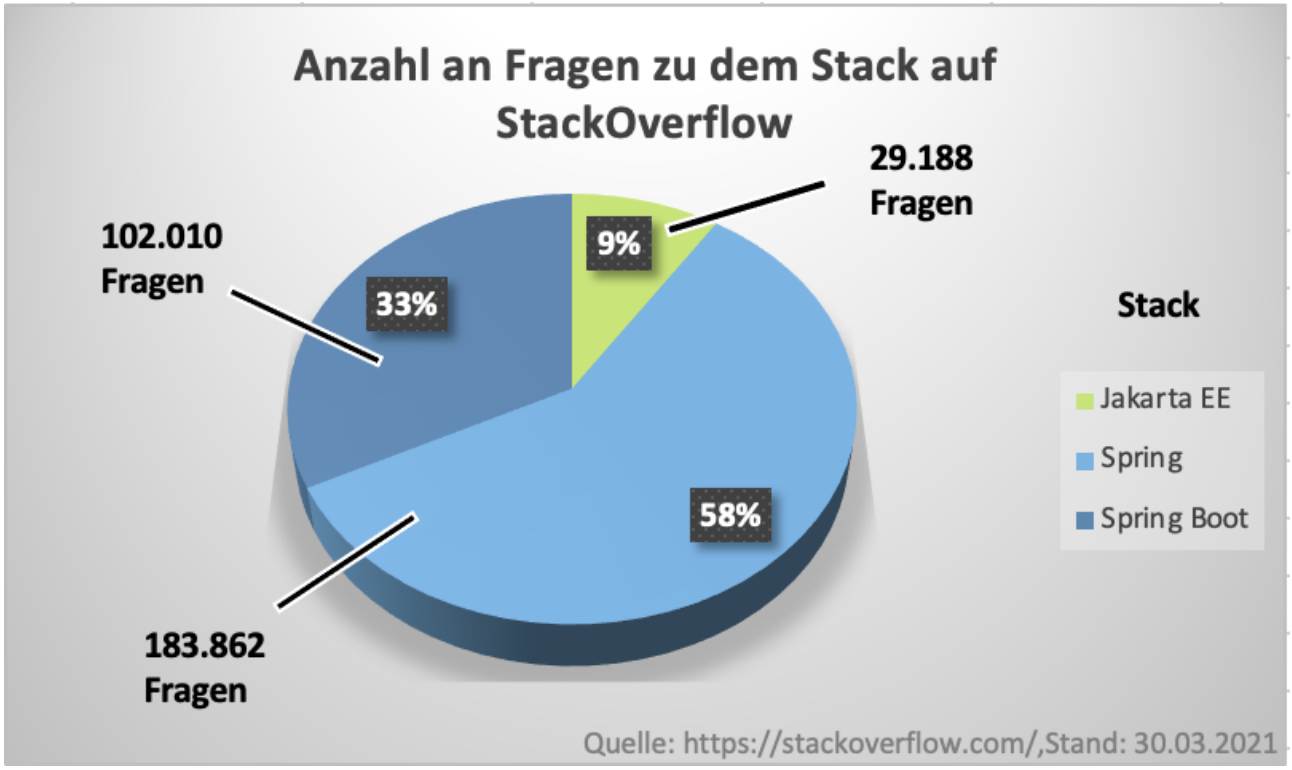

Figure 8. Anzahl an Fragen auf StackOverflow zu dem Stack

Am wohl bekanntesten Coding Portal "StackOverflow", wo täglich tausende Coding spezifische Fragen gestellt werden, ist verzeichnet, dass es bei Spring insgesamt über 100000 Fragen gibt, bei Spring Boot sogar mehr als 180000 Fragen. Java EE bzw. Jakarta EE hat demnach nur mehr als 29.000 Fragen. Dies zeigt, dass eine größere Community hinter Spring (Boot) steht und es diesbezüglich sehr viele Fragen bzw. Informationsquellen gibt, wohin gegen Java EE nur etwa ein Drittel der Fragen von Spring hat, somit weniger relevant ist und Entwickler weniger Fragen bzw. hilfreiche Informationen auf "StackOverflow" diesbezüglich zur Verfügung stehen.

Durch die wenige Benutzung (von Firmen) und Information auf StackOverflow, erhält Java EE eine Punktezahl von 3 Punkten, wohingegen Spring (Boot) mit

weitaus gehend mehr Entwicklern, Firmen und Informationen die volle Punktezahl von 5 Punkten erhält.

	Java EE	**Spring**
Punkte	3	5

Table 8. Punkte Resümee "Developer Codebase und Community Größe"

Chapter 6. Entscheidungsresümee

Die einzelnen Bewertungen haben folgendes Ergebnis ergeben:

Kriterium	Java EE Framework	Spring Framework
Regelmäßige Updates	5	5
Langfristiger Support	3	5
Kosten, Eleganz beim Programmieren	3	5
Dokumentation	3	5
Funktionsumfang	3	4
Developer Codebase und Community Größe	3	5
Querschnittsfunktion	3	5
Ergebnis	**23**	**34**

Table 9. Entscheidungsresümee Java EE vs. Spring

Das Spring Framework gewinnt mit deutlichem Abstand die Auswertung, was nicht bedeutet, dass Java EE schlecht(er) ist, denn es kommt immer darauf an, welches Ziel die Applikation haben soll. Beide sind ähnlich aufgebaut, haben Dependency Injection, sind modular aufgebaut, "stable" und für Performance und hohe Verfügbarkeit ausgelegt. Doch folgende Punkten sind zu beachten:

- Java EE eignet sich für leichte skalierbare monolithische Anwendungen
- Spring (Boot) ist für Anwendungen mit GUI im Frontend, für Microservice Architektur gut
- Beide bieten Enterprise Support
- Spring hat ein großes Ökosystem, was einen Wechsel von Spring zu anderen Frameworks erschwert

- Spring hat längere Build/Start Zeiten als Java EE
- Beide sind für kleine aber auch große Projekte als Business Applikation in Unternehmen einsetzbar
- Beide sind im Markt ausreichend etabliert, haben Community Support und sind anerkante nützliche Frameworks
- Java EE bzw. Jakarta EE ist nicht mehr so modern und innovativ, wie Spring es ist
- Lernkurve ist bei beiden Frameworks mittelmäßig, bei Spring jedoch ein wenig steiler, da es mehr Module und Zusätze gibt

Die beiden Frameworks werden oftmals als Konkurrenten gesehen, wo sie doch so ähnlich sind, da Spring auf Java EE aufbaut und somit eine Art Erweiterung dessen ist. Resümmierend gesehen überwiegt jedoch das Spring Framework mit seinen Funktionen, Community Support, Wartbarkeit, Update Regelmäßigkeit und es eignet sich besser für die Entwicklung von Business Applikationen.

Chapter 7. Verwendung von Spring Boot im Diplomprojekt

Im Diplomprojekt wurde als Framework auf die Verwendung von Spring, genauer *Spring Boot*, gesetzt.

Grund dafür war, dass bereits viel Erfahrung und praktische Programmierung Fähigkeiten in der Informatik Ausbildung und im Spring Boot Framework erlernt wurden und so eine Programmierung mit dem Framework am leichtesten fiel. In Kombination mit der Programmiersprache *Kotlin* und dem Build Tool *Gradle* wurde eine REST-API Lösung, welche auf einem Server deployed wurde, für das Projekt umgesetzt. Hauptaugenmerk lag auf der Verwendung einer Microservice Architektur statt einer Monolithen-Architektur, um einzelne Module unabhängiger und einzeln startfähig zu machen und neues Know-How zu erlangen.

Ausschlaggebend war außerdem, die sehr gute Dokumentation des Frameworks, die herausstechenden Funktionen wie ein vorkonfigurierter eingebetteter Application Server, automatisierte Build Abläufe, die zahlreichen Frameworks und produktionsfähige Metriken wie Health Endpoints sowie allgemein die Arbeit, welche Spring dem Backend Team durch vorgefertigte Templates, Projekte und Module abgenommen hat. Spring Boot hat sich als ein sehr gutes Framework für die Entwicklung der REST-APIs herausgestellt und ein Server Deployment ist leicht gefallen.

Glossar

Build Tool

Automatisiert den Prozess der Bildung ausführbarer Dateien. Software wird erstellt und beispielsweise werden nötige Dependencies heruntergeladen und verwaltet.

CLI

Ist das Command Line Interface, welches es ermöglicht, Kommandos auszuführen.

ELK-Stack

Steht für Elasticsearch, Logstash und Kibana. Es ermöglicht das Tracing, die Verarbeitung und die visuelle Aufbereitung von zum Beispiel Metrik Daten, wie Uptime, einer Applikation.

Framework

Programmiergerüst, bei dem vorgefertigte Rahmen, wie Funktionen und Elemente, bereitgestellt und somit der Einstieg in die jeweilige Technologie erleichtert wird.

JSP Pages

"Jakarta Server Pages" sind Seiten gebaut durch "JHTML" und erlaubt die Integration von Java Code in HTML und XML Dateien.

Microservice Architektur

Anwendungen werden in kleine Module aufgeteilt und werden besser separat steuerbar und unabhängiger. Zusammen bilden alle Module die Anwendung.

Monolithen Architektur

Alle Software Komponenten befinden sich in einem großen Anwendungssystem, sie sind zentral, einzelne Softwareteile untrennbar und

kaum unabhängig steuerbar.

REST-API

Programmierschnittstelle, welche über HTTP-Anfragen mittels CRUD Operationen agiert.

SOAP-API

Mit diesem Netzwerkprotokoll können Daten in Form von Envelopes zwischen Systemen ausgetauscht werden.

Quellen

Beschreibung	Quelle	Letzter Zugriff
Java EE Architektur: Grafik nachmodelliert	http://pawlan.com/monica/articles/j2eearch/art/container1.jpg	29.03.2021
Spring Boot Architektur: Grafik nachmodelliert	https://www.javatpoint.com/spring-boot-architecture	29.03.2021
Java EE Spezifikationen	https://www.javatpoint.com/java-ee	29.03.2021
Spring Framework Überblick	https://spring.io/projects/spring-framework	29.03.2021
Spring Funktionen	https://spring.io/why-spring	29.03.2021
Jakarta (Java) EE Wikipedia	https://en.wikipedia.org/wiki/Jakarta_EE	29.03.2021
Java EE Versionen, Funktionen	https://www.oreilly.com/library/view/java-ee-6/9781449338329/ch01.html	29.03.2021
Spring Framework Dokumentation Überblick	https://docs.spring.io/spring-framework/docs/4.3.20.RELEASE/spring-framework-reference/html/overview.html	29.03.2021
How Oracle killed Java EE	https://headcrashing.wordpress.com/2019/05/03/negotiations-failed-how-oracle-killed-java-ee/	29.03.2021
Java EE vs. Spring Statistiken	https://www.jrebel.com/blog/java-ee-vs-spring	29.03.2021

Beschreibung	Quelle	Letzter Zugriff
Spring Boot Erste Schritte	https://spring.io/guides/gs/spring-boot/	30.03.2021
Framework Community Statistiken	https://stackshare.io/	30.03.2021
Framework Fragen Statistiken	https://stackoverflow.com/	30.03.2021
Java EE REST Service Erste Schritte	https://www.jetbrains.com/help/idea/creating-and-running-your-first-restful-web-service.html#run_config	01.04.2021
Java EE Open Source	https://www.zdnet.com/article/java-finally-goes-all-in-on-open-source-with-the-release-of-jakarta-ee-8/	01.04.2021
Spring Paradigmen	https://java2blog.com/introduction-to-spring-framework/	01.04.2021
Spring Reactive	https://spring.io/reactive	01.04.2021
Querschnittsfunktionen	https://jaxenter.de/angular2-typescript-aop-45097	02.04.2021
Spring Security	https://spring.io/guides/topicals/spring-security-architecture	02.04.2021
Spring Security	https://data-flair.training/blogs/spring-security-tutorial/	02.04.2021
Java EE vs. Spring	https://blog.doubleslash.de/jee-vs-spring-gemeinsamkeiten-unterschiede-und-entscheidungskriterien/	03.04.2021

Table 10. Quellen