# Was ist Spring?

Spring ist ein weitverbreitetes Framework, dessen Entwicklung momentan von Pivotal Software vorangetrieben wird. Es ist für Java-Anwendungen konzipiert und ist das größte und mächtigste Framework für die Programmiersprache Java, das man zurzeit auf dem Markt finden kann. Im Jahr 2002 wurde die erste Version des Spring Frameworks als Open Source Projekt veröffentlicht.

Die Intention hinter Spring ist zum einen die Entwicklung mit Java zu vereinfachen. Oft steht man am Anfang eines Projekts und hat noch nichts auf das man aufbauen oder anknüpfen kann. Dabei haben die meisten Projekte in ihren Grundzügen ähnliche Aufgaben. Für diese gleichbleibenden Aufgaben bietet sich der Einsatz eines Frameworks, diese bildet ein Rahmengerüst für den Programmaufbau an, der genau jene sich wiederholenden Aufgaben bereits implementiert.

Die andere große Intention hinter Spring ist es saubere und gute Programmierpraktiken zu unterstützen und zu fördern. In großen Programmen geht schnell die Übersichtlichkeit verloren. Durch einen Einheitlichen Aufbau mit Spring ist es oft möglich auch bei großen und komplexen Programmen noch den Durchblick zu behalten.

Spring-Projekte nutzen zur Projektverwaltung Apache Maven oder Gradle. In der Fallstudie wird mit Apache Maven gearbeitet. Maven ist ein Build-Management-Tool, das in Java geschrieben wurde, um Java-Projekte zu verwalten. Es wurde von der Apache Software Foundation entwickelt und ist ein Open-Source-Projekt. Maven erleichtert das Bauen, Verpacken und Testen von Anwendungen. Zudem bietet Maven eine Verwaltung der Abhängigkeiten zu anderen Projekten und Bibliotheken in einer zentralen Datei, der sogenannten pom.xml. In ihr werden auch wichtige Eigenschaften des Projekts definiert. In der POM werden neben der Artefakt-ID und der Group-ID, die zur eindeutigen Identifizierung dienen, auch Parameter definier, wie das Dateiformat, in das die Anwendung verpackt werden soll.

Das Grundkonzept von Spring beruht auf der Inversion of Control und der Dependency Injection. Die Inversion of Control beschreibt ein Umsetzungsparadigma, das häufig in der objektorientierten Programmierung Anwendung findet. Anstatt das ein Programm alle seine Dienste steuert und so nur die Standardfunktionen benutzt, wird die Steuerung der Ausführung Unterprogrammen des Frameworks überlassen. Dieses aufgerufenen Unterprogramm muss jedoch zuvor sich und seine Funktion bei der Standardbibliothek registrieren, um aufgerufen werden zu können.

Das Prinzip der Dependecy Injection ist eine Spezifizierung der Inversion of Control. In einem klassisch aufgebauten objektorientierten Programm kümmert sich jedes Objekt selbstständig um seine Abhängigkeiten. Es ist selbst verantwortlich für den Aufbau des Abhängigkeitsnetzes zu den Objekten und Ressourcen, die es benötigt. Dies bedeutet in der Praxis, das Objekt muss seine Umgebung kennen und andere Objekte instanziieren. Diese Art der Arbeit gehört aber normalerweise nicht in den Aufgabenbereich des Objekts, dessen Aufgaben haben oft nichts damit zu tun. Aus diesem Grund teilt Spring die einzelnen funktional abgegrenzten Bereiche in unterschiedliche Komponenten, die Beans. Diese stellen die Unterprogramme dar, die aufgerufen werden können. Damit die Lebenszyklen der einzelnen Komponenten so weit wie möglich auseinandergehalten werden können, werden die Abhängigkeiten im Spring-Anwendungskontext registriert und verwaltet. Wenn zur Laufzeit eine Bean nun eine Abhängigkeit fordert, wendet diese sich an den Spring-Anwendungskontext. Dieser erzeugt daraufhin eine separate Entität, die dann in die richtige Bean injiziert wird. So bleiben die einzelnen Komponenten relativ unabhängig voneinander, aber es wird trotzdem die gesamte Anwendungsfunktionalität gewährleistet.

# Warum Spring?

Aus bereits genannten Gründen bietet Spring einige Vorteile mit sich. Die Struktur wird einfach und übersichtlich gehalten. Wenn man eine standalone Anwendung entwickeln möchte, muss man zusätzlich weitere Aspekte beachten. Hier bietet Spring umfangreiche Unterstützung. Zum Erstellen einer eigenständig lauffähigen Spring-Applikation bietet Spring die Erweiterung Spring Boot an. Im Wesentlichen stellt Spring Boot lediglich eine Vereinfachung der Konfiguration einer Spring-Anwendung dar. Mit Hilfe des Online-Tools Spring Initializr kann man recht einfach ein Projekt erstellen. Dort kann man zunächst die grundlegenden Eigenschaften festlegen. Zu diesen zählen die Programmiersprache, mittlerweile unterstützt Spring neben Java auch Kotlin und Groovy, die Projektverwaltung, die Version von Spring, so wie Projektmetadaten. Zur Projektverwaltung unterstützt Spring Maven und Gradle. Wie oben bereits erwähnt wurde, wird in der Fallstudie das Build-Management-Tool Maven verwendet. Zu den Projektmetadaten gehören wichtige Informationen wie die Group-Id, die Artefakt-Id und der Name des Projekts. Es kann auch eine Beschreibung des Projekts angelegt werden. Wahlweise kann auch der Package Name geändert werden, was im Allgemeinen wenig Sinn macht, da dieser sich meist aus der Group- und der Artefact-Id zusammensetzt. Eine weitere wichtige Entscheidung stellt das Packaging Format dar. Je nach Anwendungszweck kann das Java-Projekt als War oder Jar verpackt werden. All diese Informationen können bei Bedarf auch noch zu einem späteren Zeitpunkt in der pom.xml geändert werden. Im Inizializr wird auch die Version von Java festgelegt, mit der gearbeitet werden soll. Momentan stehen die Versionen 8, 11 und 15 zur Auswahl. Da mit Java 9 einige Features dazu gekommen sind, auf die wir in der Entwicklung nicht verzichten wollte war klar, dass die Version eine neuere als Java 8 sein sollte. Da Java 15 noch frisch und neu ist und noch kaum jemand Erfahrungen mit den neusten Änderungen hat viel die Entscheidung auf Java 11. Hierzu gibt es im Team der Fallstudie ausreichende Erfahrungen und auch umfangreiche Dokumentation mit Beispielen und Erklärungen im Internet. Zum Konfigurationszeitpunkt können auch schon Abhängigkeiten hinzugefügt werden. Dies ist aber nicht zwingend erforderlich. Auch diese können im Laufe der Entwicklung in der pom.xml verändert und gepflegt werden.

Hat man schlussendlich das Projekt nach den eigenen Wünschen konfiguriert kann man das Projekt erzeugen und anschließend als zip-komprimierten Ordern herunterladen. Nach dem Entpacken kann man das Projekt direkt in der Enwicklungsumgebung seiner Wahl importieren und ist startklar zum Entwickeln.

Spring Boot greift einem nicht nur bei der Initialisierung unter die Arme. Auch viele andere Konfigurationen werden größtenteils automatisch vorgenommen. Da wir uns im Projekt für eine HTML-Oberfläche entschieden haben bietet es sich an mit Spring MVC zu arbeite. Hierzu wird lediglich die Abhängigkeit zum „spring-boot-starter-web“ benötigt. Diese bringt nun eine Konfiguration mit, mit der der integrierte Tomcat-Server automatisch beim Start betrieben wird. Auch für den Datenbankzugriff kann Spring Boot Konfigurationen übernehmen. Erkennt Spring beim Scannen der pom.xml, dass eine Abhängigkeit zu einem Datenbanktreiber für JDBC existiert, übernimmt Spring im Folgenden die nötigen Einstellungen, damit eine Verbindung zu einer Datenbankaufgebaut werden kann.

# Erweiterungen von Spring

Damit Spring auch im Projekt der Fallstudie seine Konfigurationen richtig erstellen kann benötigen wir einige zusätzlichen Abhängigkeiten. Folgende Spring Boot Abhängigkeiten wurden von uns eingebunden:

* spring-boot-starter-web
* spring-boot-startet-data-jpa
* spring-boot-starter-thymeleaf
* spring-starter-starter-security

Im Folgenden werden kurz die Aufgabengebiete der einzelnen Abhängigkeiten erläutert. Eine genaue Beschreibung der Anwendung erfolgt im Teil, in dem der Aufbau des Backends beschreiben wird.

Die starter-web-Abhängigkeit bringt alle grundlegenden Funktionen für unsere Anwendung mit. Dazu gehören neben Spring Core, Spring MVC mit dem eingebundenen Servlet Container Tomcat und Logging-Funktionen mit logback oder slf4j. Spring Core stellt die bereits beschriebenen Grundfunktionen von Spring zur Verfügung. Das Logging wird in unserer Anwendung nicht verwendet. Spring MVC unterstützt uns beim Bereitstellen eines Webservices. MVC konfiguriert nicht nur den Servlet Container Tomcat. Des Weiteren erzeugt und verwaltet Spring MVC auch das Dispatcher Servlet. Dadurch handelt Spring eingehende Anfragen automatisch und gleicht diese mit dem manuell erstellten Mapping ab. Bei Fehlern bietet Spring MVC auch eine automatische Fehlerseite, die gemäß dem http-Standard den Fehlercode ausgibt. Diese Funktion wir bei einer komplexeren Anwendung oder bei einer Anwendung die Produktiv gesetzt wird höchstwahrscheinlich überschreiben.

Die startet-data-jpa-Abhängigkeit ist ein wichtiger Bestandteil der Anwendung, da durch sie ein sehr einfacher und übersichtlicher Datenbankzugriff ermöglicht wird. Spring Data JPA baut auf die bestehenden Funktionen der Java Persistence API (JPA) auf. Dadurch, dass Spring einen hohen Teil an automatischen Konfigurationen durchführt gibt es in einem Spring Data JPA-Repository keinen unnötigen Code. Im Idealfall benötigt man nicht einmal SQL-Code. Spring kann aus gewissen Schlagworten automatisch SQL-Abfragen generieren.

|  |
| --- |
| public interface UserRepository extends JpaRepository<User, String> |

Ein Repository wird als Interface implementiert, das von einem JPA-Repository erbt. Das JPA-Repository ist generisch und wird immer von dem Typ angelegt, den es mit der Abfrage befüllen soll. Der zweite Typ ist derselbe wie der Key der Tabelle, auf der Abgefragt werden soll. Der erste Typ, in diesem Beispiel der User, stellt mit seinen Attributen eine Zeile auf der Tabelle dar. Aus diesem Grund trägt die Klasse User die Annotation @Entity. Das Schlüsselattribut muss zusätzlich mit @Id annotiert werden, damit Spring die Klasse auf die Datenbank mappen kann. Ein Repository verfügt standardmäßig viele Funktionen, wie findAll(). Mit dieser Methode wird automatisch eine SQL-Abfrage generiert:

SELECT \* FROM User

Als Ergebnis liefert diese Methode eine Liste des Typs User. Möchte man Querys personalisieren kann man einfache Abfragen als Methoden vorgeben. Wenn man einen User bei seiner ID finedn möchte, erstellt man eine Methode: findByID(String id). Daraus generiert Spring die Abfrage:

SELECT \* FROM User WHERE ID=id

id steht hier als Platzhalter für den übergebenen Wert. Mit dem in der pom.xml hinterlegten Datenbanktreiber kann in Spring so eine Verbindung zu einer Datenbank aufbauen und eine Abfrage auf dieser absetzen, ohne dabei eine Zeile SQL zu schreiben.

Die Thymeleaf Starter-Abhängigkeit bietet eine Einbindung der Tempalting Engine Thymeleaf. Diese wird in ihrem eigenen Abschnitt genauer beschreiben.

Auch für die einfache Implementierung einer Authentifizierung bietet Spring eine Erweiterung. Mit dem Spring Security Starter fordert Spring automatisch eine Authentifizierung für alle Pfade, die vom Tomcat-Servlet bedient werden. Benutzer und Passwort können statisch in der application.properties hinterlegt werden. In diesem Fall bietet Spring ein einfaches Login-Interface. Man kann das Login auch abändern und die Verifizierung selbst übernehmen. In diesem Fall müssen aber mehrere Beans überschrieben werden. Dies wird ausführlicher in der Umsetzung beschrieben.

Das Spring Framework bietet ein breites Spektrum an Funktionalitäten, mit denen für beinahe jeden Anwendungsfall eine passende Unterstützung bereitgestellt. Folglich ist Spring auch für die Anwendung der Fallstudie das geeignete Werkzeug.