Inhaltsverzeichnis

KAPITEL 2

Grundlagen

2.1 REST-API

Representational State Transfer (REST) ist ein von Roy Fielding entwickelter Architekturstil, welchen er in seiner Dissertation [Fie00] beschrieb. Dabei geht er ebenfalls auf eine Reihe von Leitsätzen und Praktiken ein, welche sich in System auf Basis von Netzwerken bewährt haben.

Laut [Spi16, S. 143] unterstützt der REST Architekturstil eine Reihe von Protokollen, mit welchen solcher umgesetzt werden kann. Der bekannteste bzw. am häufigsten verwendete Vertreter ist dabei das Protokoll Hypertext Transfer Protocol (HTTP). Es wird dabei REST im Zusammenhang mit HTTP als RESTful HTTP bezeichnet.

Da in dieser Arbeit die Webentwicklung im Vordergrund steht und HTTP einer der wichtigsten Standards im Web ist, soll im nachfolgenden Verlauf REST immer im Sinne von RESTful HTTP verstanden werden.

In den nachfolgenden Abschnitte soll die allgemeine Definition, die Vor- und Nachteile und die Qualitätsmerkmale einer API näher beleuchtet werden. Anschließend wird auf die Grundprinzipien von REST detaillierter eingegangen.

2.1.1 Allgemeine Definition einer API

Laut [Spi16, S. 7] definiert Kai Spichale eine API mit den Worten von Joshua Bloch wie folgt: „Eine API spezifiziert die Operationen sowie die Ein- und Ausgaben einer Softwarekomponente. Ihr Hauptzweck besteht darin, eine Menge an Funktionen unabhängig von ihrer Implementierung zu definieren, sodass die Implementierung variieren kann, ohne die Benutzer der Softwarekomponente zu beeinträchtigen“. Des weiteren unterteilt dieser die APIs in zwei Kategorien ein, in Programmiersprachen- und Remote-APIs. Des weiteren definiert er die APIs der Programmiersprachen als abhängig und die Remote als unabhängig gegenüber der Sprache und der Plattform. [Spi16, S. 7–8]

2.1.2 Vorteile einer API

Stabilität durch lose Kopplung

Mithilfe von APIs soll die Abhängigkeit zum Benutzer minimiert werden, der dadurch nicht mehr stark an die Implementierung gekoppelt ist. Das ermöglicht eine Veränderung der eigentlichen Implementation einer Softwarekomponente, ohne das der Benutzer davon etwas bemerkt. (vgl. [Spi16, S. 9])

Portabilität

Es ist möglich für unterschiedliche Plattformen eine einheitliche Implementierung einer API bereitzustellen, obwohl diese im inneren verschieden implementiert sind. Ein bekanntest Beispiel ist dabei die Java Runtime Enviroment (JRE), welches diese Funktionalität für Java-Programme bereitstellt. (vgl. [Spi16, S. 9])

Komplexitätsreduktion durch Modularisierung

Der API-Benutzer besitzt in erster Linie keine genauen Informationen über die Komplexität der Implementierung. Diese Tatsache folgt dem Geheimprinzip und soll der Beherrschung großer Projekte in Hinsicht ihrer Komplexität dienen. Zusätzlich bringt dieser Aspekt auch einen Wirtschaftlichen Vorteil, denn durch die Modularisierung ist eine bessere Arbeitsteilung möglich, was wiederum Entwicklungskosten sparen kann. (vgl. [Spi16, S. 10])

Softwarewiederverwendung und Integration

Neben dem verbergen von Details zur Implementierung sollten APIs Funktionen einer Komponente leicht verständlich bereitstellen, um API-Nutzern eine einfache Integration bzw. Verwendung zu ermöglichen. Aus diesen sollten APIs auch dahingehend optimiert werden. (vgl. [Spi16, S. 10])

2.1.3 Nachteile einer API

Interoperabilität

Ein Nachteil, der allerdings nur Programmiersprachen-APIs betrifft, ist die Interoperabilität zu anderen Programmiersprachen. Beispielsweise kann ein Programm, welches in Go geschrieben wurde, nicht auf die Java-API zugreifen. Als Problemlösung stehen hierbei aber die Remote-API bereit. Diese arbeiten mit Protokollen wie HTTP oder Advanced Message Queuing Protocol (AMQP), welche sprach- und plattformunabhängig sind[1].[Spi16, S. 10]

1 siehe Kapitel 2.1.1

Änderbarkeit

Dadurch das geschlossene API-Verträge mit den Benutzern nicht gebrochen werden sollten, kann es hinsichtlich der Änderbarkeit zu Problemen kommen. Das ist aber nur der Fall sofern die Benutzer nicht bekannt sind oder nicht kontrolliert werden können. In so einem Fall spricht man von veröffentlichten APIs. Als Gegenstück dazu können interne APIs betrachtet werden, denn bei diesen wäre eine Kontrolle der Benutzer möglich.

2.1.4 Qualitätsmerkmale

Benutzbarkeit

Effizienz

Unter dem Qualitätsmerkmal Effizienz kann zum Beispiel der geringe Verbrauch von Akku oder dem Datenvolumen bei Mobilen Geräten verstanden werden. Oder aber die Skalierbarkeit einer API, welches bei einem großen Zuwachs von Aufrufen durchaus entscheidend sein kann. [Spi16, S. 13]

Zuverlässigkeit

Unter der Zuverlässigkeit einer API kann die Fehleranfälligkeit verstanden werden bzw. wie gut diese auf Fehler reagieren kann. Ein wichtiger Aspekt der dabei auf jeden Fall beachtet werden sollte ist die Rückgabe von standardisierten HTTP-Statuscodes. Dies ermöglicht dem Benutzer ein ordentliches Feedback, welches noch durch konkretisierte Fehlermeldungen verdeutlicht werden sollte. [Spi16, S. 13, 189]

2.1.5 Grundprinzipien von REST

2.2 Das Build-Tool Gradle

2.3 Schachnotationen FEN und SAN

2.4 Schachregeln

Literatur

[ANO] ANONYM: „Why AMD?“ (), Bd. URL: http://requirejs.org/docs/whyamd.html (besucht am 13. 04. 2018) (siehe S. 27).

[Fie00] FIELDING, ROY THOMAS: „Architectural Styles and the Design of Network-based Software Architectures“. phd. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (besucht am 07. 05. 2018) (siehe S. 3).

[Mas] MASHKOV, SERGEY: „DOM trees“. (), Bd. URL: https://github.com/Kotlin/kotlinx.html/wiki/DOM-trees (besucht am 13. 04. 2018) (siehe S. 27).

[Spi16] SPICHALE, KAI: *API-Design. Praxishandbuch für Java- und Webservice-Entwickler*. German. 1st. dpunkt.verlag GmbH, Dez. 2016 (siehe S. 3–5).

[Wat] WATSON, GRAY: „OrmLite - Lightweight Object Relational Mapping (ORM) Java Package“. (), Bd. URL: http://ormlite.com/ (besucht am 10. 04. 2018) (siehe S. 27).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Quelle: [Wat]
1 verändert nach [Wat]
1 Quelle: [ANO]
2 Quelle: [Mas]

Acknowledgments

I thank ?? and ?? for giving me the opportunity to write this bachelor/master/phd thesis at ??, and for their professional advise.

I thank in particular the ?? team who readily/willingly provided information at any time and ??.

I would also like to than all people who supported me in writing this thesis.

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Dresden, den 7. Mai 2018 Felix Dimmel

A First chapter of appendix

A.1 Parameters