Hochschule für
Technik und Wirtschaft
Dresden
University of Applied Sciences

Fakultät Informatik/Mathematik

Diplomarbeit

im Studiengang Allgemeine Informatik

Thema:

Webentwicklung mittels Kotlin am Beispiel eines RESTful-Schachservers

Eingereicht von:	Felix Dimmel
Eingereicht am:	19. Juli 2018
Betreuer:	Prof. Dr.-Ing. Jörg Vogt
2. Gutachter:	Prof. Dr.-Ing. Arnold Beck

Inhaltsverzeichnis

Abkürzungsverzeichnis

Bezeichnung	Beschreibung
AMD	Asynchronous Module Definition
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CBOR	Concise Binary Object Representation
CI	Continuous Integration
CSV	Comma-separated Values
DAO	Database Access Object
DSL	Domain-Specific Language
FEN	Forsyth-Edwards-Notation
GPL	GNU General Public License
GWT	Google Web Toolkit
HAL	Hypertext Application Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JRE	Java Runtime Enviroment
JsInterop	JavaScript interoperating

Bezeichnung	Beschreibung
JSNI	JavaScript Native Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
ORM	Object-relation mapping
Protobuf	Protocol Buffers
PYPL	PopularitY of Programming Language
REST	Representational State Transfer
SAN	Standard Algebraic Notation
SDK	Software Development Kit
UMD	Universal Module Definition
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

KAPITEL 1

Einleitung

Laut einer Onlinestudie der ARD/ZDF-Medienkommission von 2017 [ARD17a; ARD17b] ist die Nutzung des Internets im letzten Jahr um 6% deutschlandweit gestiegen. Dieser große Bedeutungszuwachs allein in Deutschland zeigt auch die immer mehr zunehmende Nachfrage nach Webinhalten. Ausdruck dafür könnte die große Beliebheit der Skriptsprache JavaScript sein. Denn diese ist laut der letzten jährlichen Onlineumfrage der Stack Exchange Inc. [Inc17] auf Platz 1 der beliebtesten Programmiersprachen. Dennoch kann es mitunter schwierig werden, große Applikationen mit JavaScript zu entwickeln. An dieser Stelle kommt die noch sehr junge Programmiersprache Kotlin ins Spiel.

Kotlin ist eine statisch typisierte objektorientierte Programmiersprache, welche in Java-Bytecode, in JavaScript und in Nativen Code kompiliert werden kann. Des Weiteren ist Kotlin neben Java seit Mai 2017 offizielle Programmiersprache für Android [Sha17]. Dadurch ist Kotlin sehr vielseitig und lässt sich für viele Anwendungsbereiche einsetzen.

1.1 Ziel der Arbeit

Grundsätzlich besteht die Programmierung einer Web-Applikation aus der Erstellung eines Servers sowie eines Clients. Für die serverseitige Programmierung stehen dafür eine ganze Reihe verschiedener Programmiersprachen zur Verfügung, aber für die clientseitige Programmierung ist meistens die Skriptsprache JavaScript das Mittel der Wahl. Nun ist es aber umständlich Web-Applikationen im mehreren Programmiersprachen zu entwickeln und es wäre wesentlich angenehmer nur eine einzige Sprache zu verwenden.

Diese Möglichkeit bietet Kotlin. Allerdings handelt es sich dabei um eine sehr junge Programmiersprache und es können daher gegebenenfalls derzeit noch Probleme auftreten. Daher zielt diese Arbeit auf die Analyse von Kotlin ab. Dabei soll, am Beispiel des bekannten Strategiespiels Schach, ein Server und ein Client entwickelt werden. Die daraus resultierende Implementierung soll etwaige auftretende Probleme für diesen Anwendungsbereich aufzeigen, um so ein Fazit zum möglichen Einsatz von Kotlin für Server-Client-Anwendungen ziehen zu können.

1.2 Aufbau der Arbeit

Diese Arbeit ist in acht weitere Kapitel unterteilt. Dabei werden zunächst im zweiten Kapitel grundlegende Themen behandelt, welche für das Verständnis der nachfolgenden Abschnitte von Bedeutung sind. Im Kapitel drei folgt ein Vergleich zwischen der Programmiersprache Kotlin und dem Google Web Toolkit (GWT), mit dessen Hilfe Java- in JavaScript-Code kompiliert werden kann. Es bietet somit eine mögliche Alternative zu Kotlin, um mit einer Programmiersprache eine Server-Client-Anwendung zu entwickeln.

Die konzeptionelle Ausarbeitung des schon oben erwähnten Beispiels Schach soll in den Kapiteln vier für den Server und fünf für den Client erfolgen. Dabei werden alle benötigten Anforderungen und erste Designentscheidungen für die Umsetzung definiert. Auf Grundlage dieser Entwürfe befassen sich die Kapitel sechs und sieben mit der konkreten Erläuterung der Implementierung, wobei im sechsten der Server und im siebten der Client beleuchtet wird.

Anschließend dient das achte Kapitel zur Präsentation des aus der Arbeit resultierenden Fazits, in welchen Rückschlüsse zum Einsatz von Kotlin, für die Entwicklung von Server-Client-Anwendungen, gezogen werden. Des Weiteren wird zusätzlich ein Fazit zur server- und clientseitigen Programmierung mit Kotlin gegeben. Abschließend rundet das Kapitel neun die Arbeit, mit einem Ausblick auf Ideen zur Verbesserungen der Implementierung und für weiterführende wissenschaftliche Arbeiten, ab.

KAPITEL 2

Grundlagen

In diesem Kapitel werden die Grundlagen, welche zum besseren Verständnis dieser Arbeit notwendig sind, behandelt. Dabei dient der erste Abschnitt zur Erläuterung der Application Programming Interfaces (APIs), welche den Representational State Transfer (REST) Architekturstil befolgen. Im zweiten Abschnitt wird das Build-Tool „Gradle“ näher betrachtet, welches für die Umsetzung des praktischen Teils dieser Arbeit verwendet werden soll. Eine genauere Analyse der Schachnotationen Forsyth-Edwards-Notation (FEN) und Standard Algebraic Notation (SAN) sind Inhalt des dritten Abschnittes. Da zur praktischen Veranschaulichung dieser Arbeit das Brettspiel „Schach“ dienen soll, erfolgt am Ende dieses Kapitel ein kurzer Verweis auf die Schachregeln.

2.1 REST-API

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

Laut [Spi16, S. 143] unterstützt der REST Architekturstil eine Reihe von Protokollen, mit welchen diese umgesetzt werden können. Der bekannteste bzw. am häufigsten verwendete Vertreter ist dabei das Hypertext Transfer Protocol (HTTP). Es wird dabei im Zusammenhang mit REST 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 der Arbeit REST immer im Sinne von RESTful HTTP verstanden werden.

In den nachfolgenden Abschnitten 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. Als letztes wird das Konzept HATEOAS genauer erläutert, welches laut Roy Fielding ein Muss [Fie08] für jede RESTful API ist.

Als Grundlage der nachfolgenden Unterkapitel dient das Buch „API-Design“ [Spi16, S. 7–10, 13–14, 144–148, 189] von Kai Spichale.

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, so dass die Implementierung variieren kann, ohne die Benutzer der Softwarekomponente zu beeinträchtigen“. Des Weiteren unterteilt dieser die APIs in die zwei Kategorien Programmiersprachen- und Remote-APIs ein, wobei erstere abhängig und letztere unabhängig gegenüber Sprache und Plattform sind.

2.1.2 Vorteile einer API

Stabilität durch lose Kopplung

Mithilfe von APIs sollen die Abhängigkeiten zum Benutzer minimiert werden. Dadurch wird eine schwächere Kopplung an die Implementierung erreicht. Das ermöglicht eine Veränderung der eigentlichen Implementation einer Softwarekomponente, ohne dass der Benutzer davon etwas bemerken muss.

Portabilität

Es ist möglich für unterschiedliche Plattformen eine einheitliche Implementierung einer API bereitzustellen, obwohl diese im inneren unterschiedlich implementiert sind. Ein bekanntes Beispiel ist dabei die Java Runtime Enviroment (JRE), welches diese Funktionalität für Java-Programme bereitstellt.

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. Das wiederum kann Entwicklungskosten sparen.

Softwarewiederverwendung und Integration

Neben dem Verbergen von Details zur Implementierung sollten die Funktionen einer Softwarekomponente durch die API leicht verständlich bereitgestellt werden. Dies ermöglicht den API-Nutzern

eine vereinfachte Verwendung bzw. Integration der API, weshalb diese auch dahingehend optimiert werden sollte.

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 der Programmiersprache „Go“ geschrieben wurde, nicht auf die Java-API zugreifen. Als Problemlösung stehen hierbei aber die Remote-APIs bereit. Diese arbeiten mit Protokollen wie HTTP oder Advanced Message Queuing Protocol (AMQP), welche sprach- und plattformunabhängig sind[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 oder kontrollierbar sind. In so einem Fall spricht man von veröffentlichten APIs. Als Gegenstück dazu können interne APIs betrachtet werden. Diese haben eine wesentlich beschränktere Benutzerzahl, wodurch eine Kontrolle dieser eher möglich ist.

2.1.4 Qualitätsmerkmale

Benutzbarkeit

Als zentrale Anforderungen an einem guten Design für APIs stehen die leichte Verständlichkeit, Erlernbarkeit und Benutzbarkeit für API-Benutzer. Um diese Anforderung umzusetzen, haben sich ein Reihe von allgemeinen Eigenschaften bzw. Zielen etabliert. Eine Auflistung, inklusive einer kurzen Beschreibung der jeweiligen Eigenschaften/Ziele, werden in der Tabelle 2.1.4 aufgeführt. Detaillierte Informationen zu diesen können im Buch [Spi16, S. 14–23] nachgeschlagen werden.

Tabelle 2.1: Eigenschaften/Ziele des Qualitätsmerkmals „Benutzbarkeit“ (verändert nach [Spi16, S. 14–23])

Eigenschaft/Ziel	Beschreibung
Konsistenz	Damit ist gemeint, dass Entscheidungen hinsichtlich des Entwurfs einheitlich im Quellcode angewandt werden. Das betrifft beispielsweise die Namensgebung von Variablen oder Funktionen.

weiter auf der nächsten Seite ...

1 siehe Kapitel 2.1.1

Eigenschaft/Ziel	Beschreibung
Intuitiv verständlich	Für dieses Ziel ist eine konsistente API unter Verwendung von Namenskonventionen unabdingbar. Grundlegend kann dabei gesagt werden, dass gleiche Dinge einheitliche Namen, aber auch ungleiche Dinge unterschiedliche Namen haben sollten. So können bereits durch bekannte Funktionen Rückschlüsse auf andere noch unbekannte gezogen werden. Ein konkretes Beispiel dafür sind `get`- und `set`-Methoden in der Java-Welt.
Dokumentiert	Eine ausführliche Dokumentation mit Beschreibungen zu den einzelnen Klassen, Methoden und Parametern ist für eine einfache Benutzung dringend erforderlich. Zusätzlich sollten Beispiele diese Erläuterungen unterstreichen.
Einprägsam und leicht zu lernen	Damit eine API einfach zu erlernen ist, müssen die ersten drei Punkte dieser Tabelle dringend erfüllt werden. Wichtig ist dabei eine geringe Einstiegshürde, um potenzielle Nutzer nicht abzuschrecken. Prinzipiell ist es immer sehr von Vorteil, wenn mit relativ wenig Code erste sichtbare Ergebnisse erzielt werden können.
Lesbaren Code fördern	Eine API kann großen Einfluss auf die Lesbarkeit des Client-Codes haben und diesen so signifikant verbessern. Durch eine gute Lesbarkeit können zum Beispiel besser bzw. schneller Fehler entdeckt werden. Um einen möglichst schmalen Client-Code zu fördern, sollten APIs benötigte Hilfsmethoden bereitstellen, sofern dies möglich ist. Als Prämisse kann dabei angesehen werden, dass die API alle Funktionen bereitstellen sollte, die sie dem Client abnehmen kann.
Schwer falsch zu benutzen	Unerwartetes Verhalten aus Sicht des Nutzers sollte vermieden werden, um unerwartete Fehler zu verhindern sowie die API intuitiv bedienbar zu halten.
Minimal	Prinzipiell gilt es eine API so klein wie möglich zu halten und Funktionen, welche nicht unbedingt benötigt werden, im Zweifel weg zu lassen. Denn nachträglich können solche Elemente nicht mehr mit trivialem Aufwand entfernt werden. Außerdem lässt sich sagen, dass je größer die API desto größer die Komplexität ist.
Stabil	Durch Stabilität soll sichergestellt werden, dass Änderungen keine Auswirkung auf die Benutzer haben, welche eine ältere Version benutzen. Sind auftretende negative Auswirkungen unvermeidbar, sollten diese entweder ausführlich kommuniziert oder durch eine neue Version gekapselt werden.

weiter auf der nächsten Seite ...

Eigenschaft/Ziel	Beschreibung
Einfach erweiterbar	Um den Aufwand der Erweiterung einer API möglichst einfach zu halten, muss die Anpassung von bestehenden Clients berücksichtigt werden. Der Idealfall ist dabei, wenn Clients von Änderungen unberührt bleiben. Ein solches Verhalten kann beispielsweise durch Vererbung im Falle einer objektorientierten Programmiersprache erreicht werden.

Effizienz

Unter dem Qualitätsmerkmal Effizienz kann beispielsweise der geringe Verbrauch von Akkuleistung oder an Datenvolumen bei mobilen Geräten verstanden werden. Ein weiterer Aspekt ist aber auch die Skalierbarkeit einer API, welche bei einem großen Zuwachs von Aufrufen durchaus entscheidend für die Stabilität oder die Performance sein kann.

Zuverlässigkeit

Unter der Zuverlässigkeit einer API wird die geringe Fehleranfälligkeit verstanden bzw. wie gut diese auf Fehler reagiert. Ein wichtiger Punkt, der dabei auf jeden Fall beachtet werden sollte, ist die Rückgabe von standardisierten HTTP-Statuscodes. Dies ermöglicht dem Benutzer ein ordentliches und verständliches Feedback. Dieses kann noch, wenn notwendig, durch konkretisierte Fehlermeldungen ergänzt werden.

2.1.5 Grundprinzipien von REST

Eindeutige Identifikation von Ressourcen

Für jede Ressource muss eine eindeutige Identifikation definiert werden. Im Web stehen dabei Uniform Resource Identifiers (URIs) für diesen Zweck bereit. Die Wichtigkeit dieses Prinzips liegt nahe. Wenn zum Beispiel Produkte von einem Online-Shop nicht eindeutig identifiziert werden könnten, dann wäre das für Werbezwecke mehr als unpraktisch. E-Mails mit personalisierter Werbung für Produkte wären ohne eine eindeutige Identifikation nicht bzw. nur umständlich möglich.

Wichtig ist zu beachten, dass mit URIs nicht nur einzelne Ressourcen, sondern auch Ressourcenlisten identifiziert werden können. Um bei dem oben genannten Beispiel zu bleiben, könnte es eine URI geben, mit welcher ein einzelnes Produkt bzw. eine Liste von Produkten identifiziert werden kann. Dies ist kein Widerspruch gegen das Prinzip der eindeutigen Identifizierung der Ressourcen, denn in diesem Fall wird die Liste als selbstständige Ressource betrachtet.

Verwendung von Hypermedia

Hypermedia setzt sich aus den Begriffen Hypertext und Multimedia zusammen. Dabei kann Hypermedia als Oberbegriff von Hypertext betrachtet werden, denn Hypermedia unterstützt nicht nur Texte sondern auch andere multimediale Inhalte, wie zum Beispiel Dokumente, Bilder, Videos oder Links. Letzteres kann für das Ausführen von Funktionen oder zum Navigieren innerhalb des Browsers verwendet werden und ist ein bekanntes Element in der Hypertext Markup Language (HTML). Daher kann HTML auch als klassischer Vertreter des Hypermedia Formates betrachtet werden. Damit der Client weiß, welche Aktionen bzw. welchen Pfaden er folgen kann, können diese ihm vom Server mithilfe von Links zur Verfügung gestellt werden.

Verwendung von HTTP-Standardmethoden

Um die vom Server bereitgestellten Links ordnungsgemäß auszuführen, müssen neben den URIs auch einheitliche Schnittstellen bekannt sein. Das setzt voraus, dass alle Clients über Verwendung und Semantik der Schnittstellen Bescheid wissen. An dieser Stelle kommen die HTTP-Standardmethoden zum Einsatz. Die Schnittstellen von HTTP bestehen dabei im wesentlichen aus den Request-Methoden `GET`, `HEAD`, `POST`, `PUT` und `DELETE`, welche alle in der HTTP-Spezifikation [Fie] definiert sind. Die Methoden `PATCH` oder auch `LINK` waren dabei nicht von Anfang an in der Spezifikation enthalten, sondern wurden erst nachträglich hinzugefügt.

Ein Client kann beispielsweise mithilfe der Methode `GET`, ohne genaueres Wissen über die Ressource, eine Repräsentation dieser abfragen. Diese allgemeinen Schnittstellen werden für jede Ressource verwendet, wodurch eine mögliche Vorhersagbarkeit gewährleistet wird. Dies wiederum erfüllt ein Teil der Qualitätsmerkmale aus dem Kapitel 2.1.4. Zusätzlich ist noch anzumerken, dass durch Benutzung der Methode `GET` keine unerwünschten Effekte befürchtet werden müssen, denn diese ist idempotent. Einfach ausgedrückt, ist diese Methode ein lesender Zugriff auf eine Ressource und kann daher keine Änderung des Zustands hervorrufen.

Unterschiedliche Repräsentationen von Ressourcen

Um mit den Daten, welche eine API zurückgibt, umgehen zu können, muss der Client wissen, in welchem Format er die Daten zurückbekommen möchte. Mithilfe der Technologie „Content Negotiation“, welche durch das HTTP-Protokoll bereitgestellt wird, kann der Client angeben, in welchem Format er die Antwort erhalten möchte. Dabei besteht die Möglichkeit mehrere Formate mit unterschiedlichen Prioritäten zu übergeben. Die gängigste Methode zur Übergabe der gewünschten Formate ist diese im `Accept`-Header des HTTP-Request mitzusenden.

Statuslose Kommunikation

Das letzte Grundprinzip besagt, dass es keinen Sitzungsstatus, welcher auf dem Server über mehrere Anfragen gehalten wird, geben darf. Der Kommunikationsstatus muss demzufolge in der Ressource selber oder im Client gespeichert werden. Durch die statuslose Kommunikation wird die Kopplung zwischen Server und Client verringert. Das wiederum bringt den Vorteil, dass zum Beispiel ein Neustart des Servers den Client nicht stören oder sogar zum Absturz bringen würde. Dieser müsste zwangsläufig davon gar nichts mitbekommen. Ein weiterer nicht unerheblicher Vorteil, welcher daraus resultiert, ist die Möglichkeit der Lastverteilung auf unterschiedliche Serverinstanzen.

2.1.6 HATEOAS

„Hypermedia As The Engine Of Application State“ oder kurz HATEOAS ist ein Design-Konzept, welches von vielen APIs, die sich selber als RESTful bezeichnen, missachtet wird. [Fie08] Der Begriff HATEOAS bzw. die Aussage, die dahinter steckt, kann nach Kai Spichale mit nachfolgenden Bedeutungen beschrieben werden [Spi16, S. 156]:

- „»Hypermedia« ist eine Verallgemeinerung des Hypertexts mit multimedialen Anteilen. Beziehungen zwischen Objekten werden durch Hypermedia Controls abgebildet.“
- „Mit »Engine« ist ein Zustandsautomat gemeint. Die Zustände und Zustandsübergänge der »Engine« beschreiben das Verhalten der »Application«.“
- „Im Kontext von REST kann man »Application« mit Ressource gleichsetzen.“
- „Mit »State« ist der Zustand der Ressource gemeint, deren Zustandsübergänge durch die »Engine« definiert werden.“

Der Grundgedanke hinter diesem Konzept ist die Selbstbeschreibung einer API. Das Ziel soll dabei sein, dass Clients nicht genau über die API Bescheid wissen müssen, sondern der Server dem Client zeigt wie er durch die API navigieren kann. Als Resultat des Ganzen benötigt der Client nur das Wissen über die Uniform Resource Locator (URL) des Einstiegs, alle weiteren Informationen bekommt er vom Server in Form von Links übermittelt.

Durch dieses Konzept wird ein dynamischer Workflow erzeugt, wodurch der Server die volle Kontrolle über die API behält. Damit ist es möglich anhand des anzufragenden Clients unterschiedliche Links bzw. Navigationspunkte auszuliefern oder Links sogar komplett auszutauschen, ohne dass der Client davon etwas bemerken muss. Aber um diese Vorteile nutzen zu können, ist es natürlich notwendig, dass Clients die bereitgestellten Links auch verwenden und diese nicht fest einprogrammiert haben.

2.2 Das Build-Tool Gradle

Gradle ist ein Open-Source Build-Tool, welches in erster Linie für die Java-Welt entwickelt wurde, aber mittlerweile auch andere Sprachen wie zum Beispiel Kotlin unterstützt. Es basiert auf den Erfahrungen von anderen großen Build-Tools wie *Ant* und [Maven] und hat damit große Akzeptanz gefunden. Indiz dafür ist der Wechsel zu Gradle von einigen großen und bekannten Projekten, wie zum Beispiel Android oder das Framework Spring.

Die in diesem Kapitel behandelten Fakten und Erklärungen entstammen aus dem Buch [Var15] von Balaji Varanasi und Sudha Belida.

2.2.1 Eigenschaften von Gradle

Declarative Dependency Management

Gradle ist nicht nur ein Tool um eigene Anwendungen zu bauen, sondern dient auch dazu, die Abhängigkeitsstruktur dieser aufzulösen. Eine manuelle Auflösung dieser Strukturen mit allen zugehörigen Versionen kann je nach Tiefe sehr aufwendig werden. Denn viele große Bibliotheken bringen selber Abhängigkeiten mit, welche wiederum eine eigene Abhängigkeitsstruktur besitzen. Mit Gradle genügt es ausschließlich die projektspezifischen Abhängigkeiten mit den zugehörigen Versionen zu definieren. Eine Konfiguration, wie und welche Unterabhängigkeiten aufgelöst werden müssen, ist dabei nicht notwendig. Darum kümmert sich Gradle automatisch. Es ist demzufolge nur wichtig über das „Was“ aber nicht über das „Wie“ Bescheid zu wissen.

Declarative Builds

Damit Skripte für den Build-Prozess einfach und verständlich sind, verwendet Gradle dafür eine Domain-Specific Language (DSL) auf Basis der Programmiersprache Groovy. Dadurch werden eine Reihe von Sprach-Elementen bereitgestellt, welche einfach zusammengestellt werden können und ihre Absicht klar zum Ausdruck bringen.

Build by Convention

Um den Konfigurationsaufwand von Projekten zu minimieren, bietet Gradle eine Reihe von Standardwerten und Konventionen an. Durch die Einhaltung dieser werden Build-Skripte sehr prägnant, sind aber dennoch nicht an diese gebunden. Da die Skripte auf Groovy basieren, können die Konventionen leicht durch Schreiben von Groovy-Code umgangen werden.

Incremental Builds

In größeren Projekten kommt es oft zu sehr langsamen Build-Zeiten, weil andere Tools, im Gegensatz zu Gradle, versuchen den Code immer zu säubern und neu aufzubauen. Anhand der inkrementellen Builds, welche Gradle bereitstellt, kann dieses Problem umgangen werden. Tasks die der Build-Prozess ausführt, werden übersprungen, sofern keine Änderungen festgestellt wurden. Dafür wird überprüft, ob sich Ein- oder Ausgänge eines Tasks geändert haben.

Gradle Wrapper

Durch dieses Feature ist es möglich, projektspezifische Tasks auszuführen, obwohl keine Installation von Gradle auf dem System vorhanden ist. Der Gradle Wrapper stellt dafür eine Batch-Datei für Windows- und ein Shell-Skript für Linux- bzw. Mac-Umgebungen bereit. Ein weiterer Vorteil ist der Einsatz in Continuous Integration (CI) Servern, welche mithilfe des Gradle-Wrappers keine zusätzlichen Konfigurationen benötigen, um die Build-Prozesse auszuführen.

Plugins

Anhand von Plugins bzw. Erweiterungen kann die Funktionalität von Gradle beliebig erweitert oder angepasst werden. Dabei dienen diese als Kapselung von Build- oder Task-Logik und können bequem verteilt bzw. in anderen Projekten wiederverwendet werden. Somit ist eine Unterstützung von zusätzlichen Programmiersprachen ohne Weiteres möglich.

2.2.2 Verwaltung von Projekten und Tasks

Innerhalb von Gradle wird zwischen zwei grundlegenden Build-Bereichen unterschieden, den Projekten und den Tasks. Gradle kann dabei für ein oder mehrere Projekte verwendet werden, wobei jedes Projekt wiederum einen oder mehrere Tasks beinhalten kann.

Projekte

Standardmäßig wird innerhalb des Projektverzeichnisses nach der Datei `build.gradle` gesucht, welche ein Projekt identifiziert und in welcher die benötigten Tasks definiert werden. Es ist dabei möglich, die `build.gradle` Datei umzubenennen, wodurch jedoch der geänderte Name bei einem Aufruf eines Tasks explizit mit übergeben werden muss.

Zusätzlich zur Definition von Tasks dient die Build-Datei dazu weitere Informationen festzulegen. Darunter zählen unter anderem Abhängigkeiten und Repositories. Gradle sucht dann in allen angegebenen Repositories nach den definierten Abhängigkeiten.

Auch die Angabe des Namens, der Version und der Beschreibung des eigenen Projektes ist über diese Datei möglich. Des Weiteren kann auch ein Elternprojekt definiert werden.

Tasks

Die Tasks sind das Kernstück eines jeden Projektes, mit welchen sich der geschriebene Quellcode kompilieren, Tests starten oder fertig zusammengebaute Applikation auf einen Server veröffentlichen lassen.

Jeder Task verfügt dabei über die Funktionen `doFirst` und `doLast` um Code vor oder nach einem Task auszuführen. Mithilfe dieser Funktionen ist es möglich in den Prozess eines bestehenden Task einzugreifen, ohne dessen eigentliche Logik zu verändern. Durch die Funktion `dependsOn`, welche auf einem Task aufgerufen werden kann, ist es möglich Abhängigkeiten zwischen Tasks zu definieren. So kann die Ausführung eines bestimmten Tasks vor einem anderen erzwungen werden.

Des Weiteren besteht die Möglichkeit bestimmte Typen für Tasks anzugeben. So können diese standardmäßig als `Zip`, `Copy`, `Exec` oder `Delete` definiert werden. `Copy` erlaubt dabei zum Beispiel das Kopieren von Dateien und `Exec` das Ausführen von Kommandozeilen-Befehlen.

In dem Listing 2.1 wird ein Beispiel für einen kopierenden Task gezeigt. Dieser kopiert alle Dateien mit den Endungen `.js` und `.js.map`, aus dem Build-Verzeichnis der Source-Dateien in ein Unterverzeichnis `js` des `dist`-Verzeichnisses. Dabei ist außerdem eine Abhängigkeit zum Build-Task definiert, wodurch dieser immer zuvor ausgeführt wird.

```
1 task copyCompiledFilesIntoDist(type: Copy) {
2   from "$buildDir/classes/kotlin/main/"
3   into "${projectDir}/dist/js/"
4   include "*.js"
5   include "*.js.map"
6 }
7
8 copyCompiledFilesIntoDist.dependsOn(build)
```

Listing 2.1: Beispiel: Gradle-Task

2.3 Schachregeln

Schach ist ein strategisches Spiel zwischen zwei Parteien, bei welchem jeder versucht die Königsfigur des Gegenspielers zu schlagen. Dabei gibt es eine Reihe verschiedener Figuren, mit unterschiedlichen Fähigkeiten. Um ein reibungsloses Verständnis dieser Arbeit zu gewährleisten empfiehlt es sich die zugrunde liegenden Regeln zu kennen. Da eine detaillierte Erläuterung dieser

aber den Rahmen der Arbeit sprengen würde, können sämtliche Regeln in dem Buch [Los08] nachgeschlagen werden. In diesem Buch sind neben den allgemeinen Regeln, auch die der Bewegung einzelner Figuren zu finden. Zusätzlich sind noch eine Reihe von Tipps zum Einstieg und Testfragen zu bekannten Partien mit beigefügten Lösungen enthalten.

2.4 Schachnotationen FEN und SAN

FEN und SAN sind Notationen, welche für die elektronische Verarbeitung von Schachspielen entwickelt wurden. Beide Notationen werden als Text dargestellt und ermöglichen so eine statuslose Kommunikation. Die FEN dient dabei zur Repräsentation eines ganzen Spielstandes und die SAN zur Darstellung eines Spielzuges.

Die FEN setzt sich aus mehreren Teilen zusammen, welche aus der Figurenstellung, dem aktuellen Spieler, der Möglichkeit zur Rochade, dem Feld zum schlagen „en passant“, den Halbzügen und der aktuellen Zugnummer besteht. Die einzelnen Teile der Notation werden dabei durch Leerzeichen getrennt. Das Listing 2.2 stellt den Startspielstand eines Schachspiels in der FEN dar.

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

Listing 2.2: Startposition eines Schachspiels in der FEN

Die SAN besteht im allgemeinen aus dem Buchstaben der bewegenden Spielfigur und dem Zielfeld. Zu beachten ist dabei, dass für eine Bauernfigur kein Buchstabe angegeben wird. Das Listing 2.3 zeigt den Zug eines Bauern vom Feld a2 nach a4 und das Listing 2.4 den eines Springers vom Feld b1 nach c3.

```
a4
```

Listing 2.3: Beispiel SAN: Bauer zieht von a2 nach a4

```
Nc3
```

Listing 2.4: Beispiel SAN: Spring zieht von b1 nach c3

Die Buchstabencodes der einzelnen Figuren sind in der Tabelle 2.2 für die FEN und die SAN aufgelistet. Dabei ist zu beachten das die Codes für die Farbe Schwarz klein und für die Farbe Weiß groß geschrieben werden. Dies gilt aber nur für die FEN, denn in der SAN werden diese immer groß geschrieben. Genauere Informationen zur FEN und SAN können in der Bachelorarbeit [Kre15, S. 9–10] von Christoph Kretzschmar nachgelesen werden.

Tabelle 2.2: Figurenbedeutung in der FEN und SAN (Quelle: [Kre15, Tabelle 2.1])

	R	**N**	**B**	**Q**	**K**	**P**
Bedeutung	Rook	Knight	Bishop	Queen	King	Pawn
Figur	Turm	Springer	Läufer	Königin	König	Bauer

KAPITEL 3

Vergleich zwischen Kotlin und dem Google Web Toolkit

Ziel dieses Kapitels soll der Vergleich von Kotlin und dem GWT sein, da es im Zusammenhang mit der Programmiersprache Java auch zum Erstellen von Server-Client-Anwendungen geeignet ist. Prinzipiell stellt GWT einen Compiler bereit, mit welchem es möglich ist Java- nach JavaScript-Code zu kompilieren. Dabei ist aber zu beachten, dass Kotlin eine Programmiersprache und GWT ein Framework ist, welches in und für Java implementiert wurde. Um einen sinnvollen Vergleich zu ermöglichen, wird im Nachfolgenden GWT in Zusammenhang mit Java bzw. als Erweiterung betrachtet.

In dem ersten Unterabschnitt 3.1 werden dafür zuallererst die Vergleichskriterien ermittelt, welche für den Vergleich benötigt werden. Anschließend dient der Unterabschnitt 3.2 zur Präsentation des Vergleichsergebnisses.

Da für diese Arbeit die Benutzung der Programmiersprache Kotlin vorgegeben ist, wird auf eine Benotung der Vergleichsteilnehmer verzichtet.

3.1 Vergleichskriterien

Für die Ermittlung der Vergleichskriterien wurde das Paper [Jad11] von Jadhav und Sonar verwendet. In diesem werden zwar Bewertungsmethodiken für den Entscheidungsprozess von Software-Produkten ermittelt und näher erläutert, aber viele dieser Kriterien lassen sich auch auf Programmiersprachen anwenden.

Jadhav und Sonar haben dafür, neben der Vorgehensweise eines Vergleichs, die sieben Hauptkriterien aus der Abbildung 3.1 entwickelt und anschließend diesen granularere Unterkriterien zugeordnet. Um eine Bewertung der einzelnen Kriterien zu vereinfachen, wurden diesen Leitfragen zugewiesen. Mit der Beantwortung dieser Fragen soll die Benotung vereinfacht werden.

Wie schon Anfangs erwähnt, lassen sich nicht alle Kriterien auf Programmiersprachen anwenden, weshalb für diesen konkreten Fall die Hauptkriterien „Technical“ und „Output“ entfallen. Da

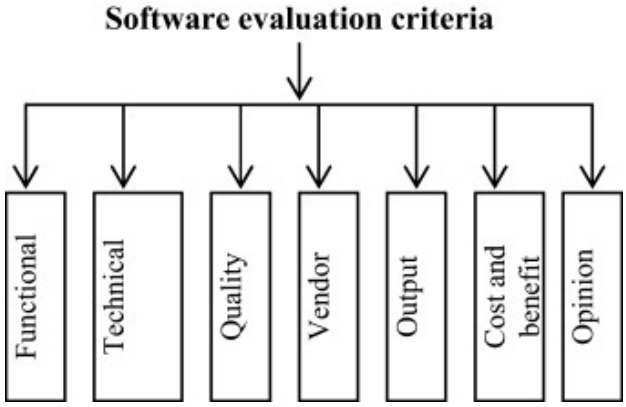

Abbildung 3.1: Hauptkriterien des Softwarevergleichs von Jadhav und Sonar [Jad11]

mit „Technical“ Hardware- und mit „Output“ Ausgabe-Kriterien gemeint sind, finden diese keine Verwendung.

In den nachfolgenden Unterabschnitten 3.1.1 bis 3.1.5 werden dafür den Hauptkriterien einzelne Unterkriterien zugeordnet und mit Leitfragen detaillierter beschrieben. Dabei beziehen sich diese auf den konkreten Anwendungsfall des Vergleichs, wobei die Kompilierung nach JavaScript im Vordergrund stehen soll.

3.1.1 Funktionale Kriterien (Functional)

JavaScript-Kompilierung

- In welche JavaScript-Versionen kann kompiliert werden?
- Welche Java-Script Modul-Systeme werden unterstützt?

Build-Tools

- Welche Build-Tools werden offiziell unterstützt?

Bibliotheken

- Können schon bestehende JavaScript-Bibliotheken eingebunden werden?
- Wie komfortabel ist die Einbindung bestehender Bibliotheken?
- Können Java-Bibliotheken eingebunden werden?

Programmierung

- Welcher Programmierstil wird verwendet?
- Welcher Grad der Typisierung wird verwendet?

- Werden Aspekte der funktionalen Programmierung unterstützt?

3.1.2 Qualitative Kriterien (Quality)

Kommunikationsstandards

- Welche gängigen Formate können standardmäßig verarbeitet werden?
- Können diese direkt in Objekte geparst werden?

Browser-Support

- Welche Browser werden unterstützt?

Lernkurve

- Wie leicht lässt sich die Programmiersprache erlernen bzw. bedienen?
- Gibt es Online-Plattformen, um die Programmiersprache auszuprobieren?

3.1.3 Anwenderkriterien (Vendor)

Dokumentation

- Gibt es eine ausführliche Dokumentation?
- Gibt es offizielle Tutorials?

Support

- Wird Support von Entwicklern angeboten?
- Gibt es ein offizielles Forum oder ähnliches?
- Auf welchen weiteren Community-Plattformen ist die Programmiersprache vertreten?

Community

- Wie viele Entwickler sind an der Programmierung der Sprache beteiligt?
- Gibt es Unternehmen, welche die Entwicklung unterstützen?

3.1.4 Kosten- und Nutzenorientierte Kriterien (Cost and benefit)

Lizenz

- Wurde die Programmiersprache unter einer freien Lizenz veröffentlicht?

- Entstehen durch die Benutzung Lizenzkosten?

Benutzungskosten

- Sind kostenpflichtige Programme für die Benutzung notwendig?

3.1.5 Kriterien an die öffentliche Meinung (Opinion)

Beliebtheit

- Wie viele Repositories gibt es auf Github, welche mit der Programmiersprache entwickelt wurden?
- Wie viele Fragen wurden auf der Plattform Stack Overflow gestellt?

Veröffentlichung

- Wann wurde die erste Version veröffentlicht?

Indizes

- Welchen Platz hat die Programmiersprache beim Triobe Index belegt?
- Welchen Platz hat die Programmiersprache beim RedMonk Index belegt?
- Welchen Platz hat die Programmiersprache beim PopularitY of Programming Language (PYPL) Index belegt?

3.2 Vergleichsergebnis

3.2.1 Funktionale Kriterien (Functional)

JavaScript-Kompilierung

Kotlin Als Ziel der Kompilierung dient der JavaScript-Standard ECMAScript 5. Es existieren aber bereits Pläne für die Umsetzung des ECMAScript 2015 Standards. Als Modul-System kann zwischen Asynchronous Module Definition (AMD), CommonJS und Universal Module Definition (UMD) gewählt werden. [sroe; srof]

GWT(Java) Der Standard ECMAScript 5 dient auch für GWT als Kompilierungsziel. Aber auch hier ist bereits eine Planung für den ECMAScript 2015 Standard bekannt, welcher mit der Version 3.0 veröffentlicht werden soll. Eine Verwendung eines JavaScript-Modul-Systems ist derzeit mit GWT nicht möglich. [Dew17]

Build-Tools

Kotlin Es werden gleich eine ganze Reihe an Build-Tools unterstützt. Vertreten sind dabei Gradle, Maven und Ant. Dabei wird aber die Nutzung von Gradle für die Kompilierung nach JavaScript offiziell empfohlen. [srog; srom]

GWT(Java) Nur Ant wird unterstützt bzw. gibt es nur zu diesem eine offizielle Dokumentation. Aber dennoch können auch andere Build-Tools wie zum Beispiel Maven oder Gradle über Drittanbieter-Software verwendet werden. [Incd]

Bibliotheken

Kotlin Es besteht die Möglichkeit bereits erstellte JavaScript-Bibliotheken, mittels JavaScript interoperating (JsInterop) zu integrieren. Dafür muss die API der Bibliothek nachgebaut werden. Da das für große Projekte sehr mühselig sein kann, bietet Kotlin dafür das Tool „ts2kt“[1] an, welches diese Aufgabe übernimmt. Voraussetzung dafür ist nur die Existenz der zugehörigen TypeScript Definitionsdateien. Eine Integration von Java-Bibliotheken ist nicht möglich. [Jem; sroa]

GWT(Java) Ebenso wie in Kotlin besteht die Möglichkeit der Einbindung bereits bestehender JavaScript-Bibliotheken mithilfe von JsInterop oder den JavaScript Native Interface (JSNI). Es wird jedoch nicht mehr empfohlen JSNI zu benutzen, weil diese Technologie veraltet ist und außerdem mit der dritten Version entfernt wird. Im Gegensatz zu Kotlin können in GWT auch Java-Bibliotheken eingebunden werden. Was daher kommt, dass GWT generell Java- in JavaScript-Code kompiliert. [Ince; Incf]

Programmierung

Kotlin Kotlin ist eine statisch typisierte und objekt-orientierte Programmiersprache, welche aber ebenso Aspekte der Funktionalen Programmierung enthält. [sroc]

GWT(Java) Auch wie Kotlin ist Java eine statisch typisierte und objekt-orientierte Programmiersprache. Funktionale Aspekte wie Lambda-Ausdrücke sind seit der Version acht verfügbar. [Gos15]

1 siehe `https://github.com/Kotlin/ts2kt`

3.2.2 Qualitative Kriterien (Quality)

Kommunikationsstandards

Kotlin Mithilfe der Bibliothek „kotlinx.serialization“ können die Formate JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR) und Protocol Buffers (Protobuf) verarbeitet und auch direkt in Objekte geparst werden. Dabei besteht auch die Möglichkeit, die Bibliothek für andere Formate zu erweitern. Das Format Extensible Markup Language (XML) kann auch mit Standard JavaScript Methoden verarbeitet, aber nicht direkt in Objekte geparst werden. [Staa]

GWT(Java) Auch GWT bietet die Möglichkeit JSON direkt in Objekte zu parsen, aber genau wie bei Kotlin gilt das nicht für das Format XML. Hierfür werden zwar benötigte Funktionen und Methoden durch GWT bereitgestellt, aber ein Objekt-Parsing ist dennoch nicht möglich. Andere Formate werden nicht offiziell unterstützt. [Incg; Inci]

Browser-Support

Kotlin Offiziell gibt es keine Information, welche Browser unterstützt werden.

GWT(Java) Die Browser Internet Explorer in den Versionen acht, neun, zehn und elf, Safari in den Versionen fünf und sechs, Firefox, Chrome und Opera werden offiziell unterstützt. [Incd]

Lernkurve

Kotlin Es wird eine kompakte Einführung bereitgestellt, welche auf ein schnelles erfolgreiches Ergebnis abzielt. Des Weiteren gibt es eine ganze Reihe an Tutorials, welche sehr ausführlich erläutert und immer mit reichlich Beispielen bestückt sind. Zu dem wird noch eine Online-Plattform[1] angeboten, wo weitere Beispiele direkt ausgeführt und auch eigener Code getestet werden kann. [sroj]

GWT(Java) Auch für die Bibliothek GWT wird eine kompakte Einführung mit schnellem Ergebnis bereitgestellt. Ergänzend dazu gibt es eine ganze Reihe an Tutorials, wo ebenfalls etwas größere und komplexere Beispiele vertreten sind. Leider gibt es für GWT keine Online-Plattform, um ohne eine Installation diese zu testen. Für reine Java-Projekte[2] hingegen gibt es solche.

1 siehe https://try.kotlinlang.org/
2 siehe https://www.jdoodle.com/online-java-compiler

3.2.3 Anwenderkriterien (Vendor)

Dokumentation

Kotlin Die Dokumentation von Kotlin ist sehr ausführlich und verständlich erläutert. Dabei wird diese, wie schon zuvor erwähnt, durch eine ganze Reihe an Tutorials unterstützt. [sroj]

GWT(Java) Für GWT und auch für Java gibt es eine sehr ausführliche, mit Tutorials bestickte Dokumentation. [Corc; Incc]

Support

Kotlin Durch die Firma JetBrains, welche für die Entwicklung von Kotlin verantwortlich ist, wird kein offizieller Support angeboten. Fragen können im Forum[1] gestellt werden, wo jedoch jede Menge JetBrains-Entwickler vertreten sind und auch auf Fragen eingehen. Zusätzlich können Informationen oder auch Fragen auf den Community-Plattformen Reddit, Slack, Stack Overflow, Twitter oder Google+ erhalten bzw. gestellt werden. [srok]

GWT(Java) Auch hier wird kein offizieller Support durch die Entwickler angeboten. Aber genau wie Kotlin bietet GWT eine ganze Reihe von Community-Plattformen an. Darunter Reddit, Stack Overflow, Twitter und Google+. Als Foren dienen verschiedene je nach Anwendungsfall spezialisierte Google Gruppen. [Inca]

Community

Kotlin Laut dem Repository auf Github haben sich bisher 233 Leute (Stand 13.06.2018) an dem Projekt beteiligt. Davon sind laut eigenen Aussagen von JetBrains über 40 Teilnehmer von ihnen gestellt. [srol]

GWT(Java) Für Java existiert keine konkrete Zahl an beteiligten Entwicklern, bekannt ist nur, dass die Community von den großen Firmen Oracle und IBM unterstützt wird. GWT hat laut dem Github Repository 87 Beteiligte (Stand 13.06.2018) und wird von der Firma Google entwickelt bzw. unterstützt. [Inch; Led]

3.2.4 Kosten- und Nutzenorientierte Kriterien (Cost and benefit)

Lizenz

Kotlin Dieses Projekt wurde unter der freien Apache-Lizenz in der zweiten Version veröffentlicht. Daher entstehen bei der Benutzung der Programmiersprache keinerlei Kosten. [srod]

1 siehe `https://discuss.kotlinlang.org/`

GWT(Java) Auch Java und GWT lassen sich kostenlos nutzen, denn diese werden ebenfalls unter freien Lizenzen veröffentlicht. Im Falle von Java ist das die GNU General Public License (GPL) mit Classpath Exception und bei GWT ist das, genau wie bei Kotlin, die Apache-Lizenz in der zweiten Version. [Corb; Inch]

Benutzungskosten

Kotlin Um mit Kotlin ein Projekt zu starten ist keinerlei kostenpflichtiges Programm notwendig. Zum Entwickeln können die Plugins für die kostenlosen Integrated Development Environments (IDEs) IntelliJ IDEA, Android Studio oder Eclipse benutzt werden. Alternativ steht auch ein „standalone“ Compiler zur Verfügung.

GWT(Java) Auch für Java- bzw. GWT-Projekte fallen keine Kosten an. Die Auswahl an IDEs ist noch etwas größer als bei Kotlin. Unter anderem sind dabei IntelliJ IDEA, Eclipse und NetBeans IDE vertreten. Bei der Installation des Software Development Kit (SDK) von Java wird zwar ein Compiler mitgeliefert, aber um ein GWT-Projekt zu kompilieren, muss das Build Tool Ant benutzt werden, welches aber ebenfalls kostenlos zur Verfügung steht.

3.2.5 Kriterien an die öffentliche Meinung (Opinion)

Beliebtheit

Kotlin Laut Github existieren derzeit über 72 590[1] Repositories, welche mit Kotlin entwickelt wurden bzw. noch werden. Die Zahl der auf Stack Overflow gestellten Fragen beläuft sich auf 11 398[2]. (Stand: 14.06.2018)

GWT(Java) Die Anzahl der Github-Repositories beträgt derzeit 4 810 228[3] für Java-Projekte. Da GWT ein Framework ist, lässt sich über Github keine verwertbare Zahl ermitteln, wie viele Repositories dieses benutzen. Bei den gestellten Fragen auf Stack Overflow sieht das anders aus, denn derzeit sind 1 426 726[4] mit Java und 20 443[5] mit GWT getaggt. (Stand: 14.06.2018)

Veröffentlichung

Kotlin Am 22. Juli 2011 wurde Kotlin erstmals als neue Sprache für die Java Virtual Machine (JVM) vorgestellt und am 15. Februar 2016 wurde die erste Version released. [Bre16; Kri11]

1 siehe https://api.github.com/search/repositories?q=language:Kotlin
2 siehe https://stackoverflow.com/questions/tagged/kotlin
3 siehe https://api.github.com/search/repositories?q=language:Java
4 siehe https://stackoverflow.com/questions/tagged/java
5 siehe https://stackoverflow.com/questions/tagged/gwt

GWT(Java) Die erste Version von Java wurde 23. Mai 1995 und die von GWT am 16. Mai 2006 veröffentlicht. [Byo03; Inc06]

Indizes

Kotlin Laut Tiobe Index landet Kotlin im Ranking der beliebtesten Programmiersprache auf Platz 49, laut RedMonk Index auf Platz 27 und laut PYPL Index auf Platz 16. [BV; Car18; OGr18]

GWT(Java) Da GWT keine Programmiersprache ist, beziehen sich die Platzierungen nur auf Java, welche laut Tiobe Index den Platz eins und laut RedMonk und PYPL Index den Platz zwei belegt. [BV; Car18; OGr18]

KAPITEL 4

Konzept des Servers

Inhalt dieses Kapitels soll die Planung sein, welche für die Umsetzung des RESTful Schachservers benötigt wird. Dabei dient der erste Abschnitt zur Erläuterung der Anforderungen, welche der Server mitbringen bzw. erfüllen soll. Enthalten ist dabei auch eine Erläuterung der benötigten Ressourcen. Der zweite Abschnitt dieses Kapitels befasst sich anschließend damit, wie der Zugriff auf einzelne Ressourcen des REST-Servers erfolgen soll. Dabei werden alle möglichen Request-Methoden für die jeweiligen Ressourcen näher beleuchtet. Abschließend wird erläutert, wie das Designkonzept HATEOAS in dieser Anwendung erreicht werden soll.

4.1 Anforderungen

Die Grundanforderungen an den RESTful Schachserver sollen in erster Linie die Bereitstellung aller benötigten Ressourcen sein. Dabei sollen Elemente erstellt, ggf. bearbeitet und gelöscht werden können. Zusätzlich soll die Möglichkeit bestehen, einzelne oder alle gespeicherten Elemente einer Ressource abzufragen. Beim Erstellen eines neuen Ressourcenelements soll dieses in einer SQLite Datenbank gespeichert und die ID automatisch durch SQLite generiert werden.

Um ein Schachspiel abzubilden, bedarf es dabei der Ressourcen Player (Spieler), Match (Partie) und Draw (Zug), welche in den nachfolgenden Unterabschnitten 4.1.1 bis 4.1.3 näher betrachtet werden.

Als abschließende Anforderung ist noch die Fehlerresistenz zu erwähnen. Denn die im Rahmen dieser Arbeit entstandene Praktikumsaufgabe[1] soll durch zukünftige Studenten bearbeitet werden, wobei der Server als Grundlage dienen soll.

Zur Unterstützung der Erläuterungen in den Kapiteln 4.1.1 bis 4.1.3 bietet die Abbildung 4.1, in Form eines Unified Modeling Language (UML) Klassendiagramms, eine Veranschaulichung.

1 siehe Anhang A

4.1.1 Ressource: Player (Spieler)

Neben der ID, welche schon im Abschnitt 4.1 erwähnt und durch die SQLite Datenbank generiert werden soll, besitzt der Player noch Informationen über seinen Namen und sein Passwort.

Nach dem Anlegen eines neuen Players soll eine Änderung des Passwortes möglich, aber die des Namens nicht möglich sein.

4.1.2 Ressource: Match (Partie)

Neben der ID besitzt ein Match Informationen über die beiden Spielteilnehmer und deren Figurenstellung auf dem Schachbrett. Zusätzlich wird registriert, welcher der beiden Player als nächstes seinen Zug tätigen muss, welche Möglichkeiten zum rochieren bestehen, ob ein Schlag „en passant“ möglich ist und wenn ja auf welches Feld gezogen werden muss und wie viele Halbzüge gespielt wurden. Der Wert der Halbzüge wird dabei zurückgesetzt, sobald eine Bauernfigur gezogen oder eine beliebige Figur geschmissen wurde. Zusätzlich kann über ein Match ermittelt werden, ob ein Spieler im Schach steht oder ob das Spiel schon bis zum Schachmatt gespielt wurde. All diese Informationen werden zusätzlich noch als String in der FEN[1] gespeichert.

4.1.3 Ressource: Draw (Zug)

Die Ressource Draw speichert zusätzlich zur ID die Farbe des Spielers, die Art der Spielfigur, Start- und Endfeld des Zuges, ob eine Figur geschlagen wurde, wenn ja ob durch „en passant“ und ob seitens der Dame oder des König rochiert wurde. Diese Informationen werden ähnlich zum Match als String aber in diesem Fall in der SAN[1] gespeichert.

1 siehe Kapitel 2.4

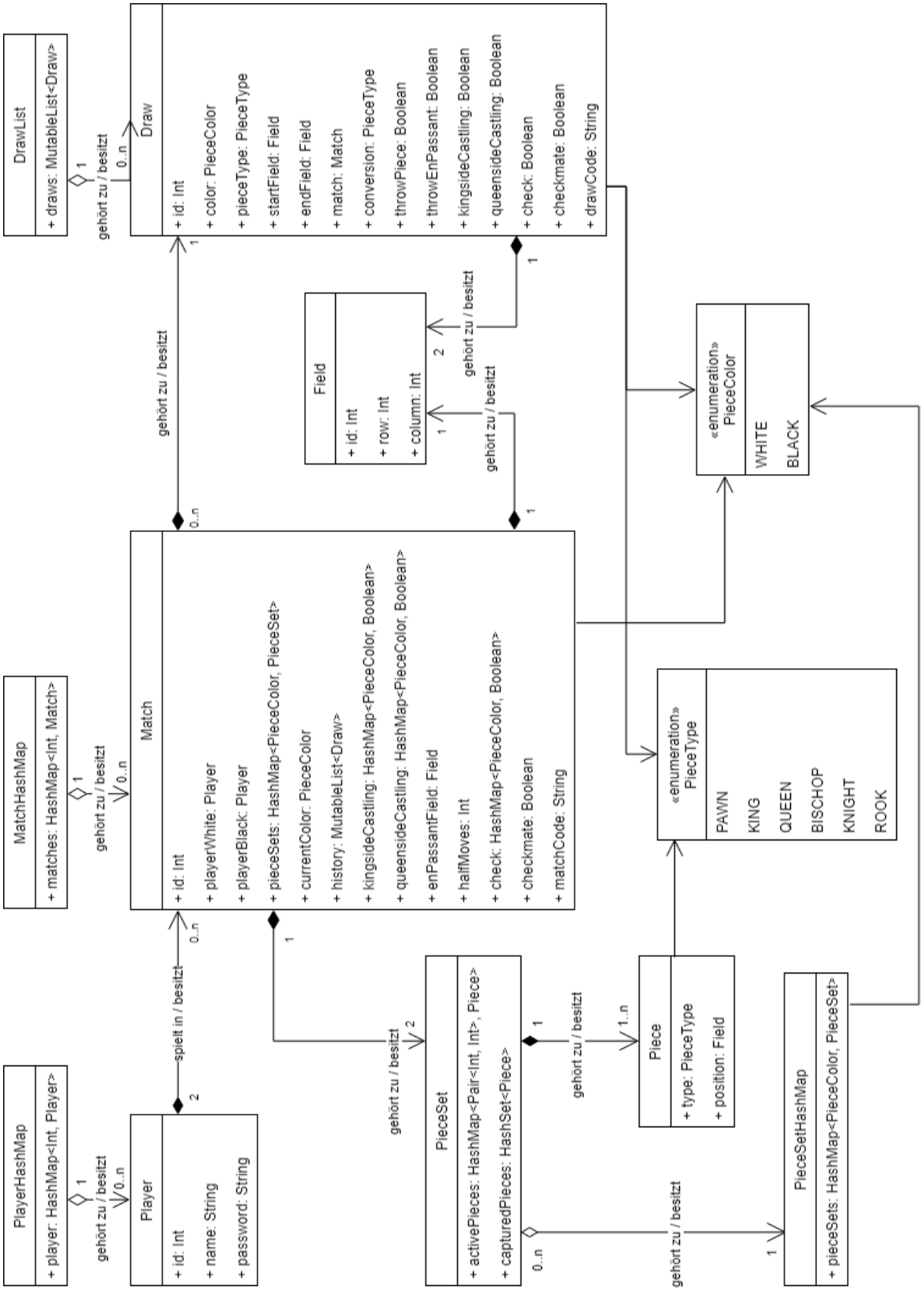

Abbildung 4.1: Klassendiagramm: Modelle des Servers

4.2 Ressourcenzugriffe mithilfe von Controllern

Die einzelnen Zugriffe auf die Ressourcen werden in den Kapiteln 4.2.1 bis 4.2.5 nach ihrer Art bzw. deren Aufgaben in einzelne Controller unterteilt, um eine gute Übersicht zu wahren. Für alle Einstiegspunkte der REST-API soll die Request-Methode `OPTIONS` bereitstehen, über den ermittelt werden kann, welche Methoden für den jeweiligen Einstiegspunkt zur Verfügung stehen.

Etwaige Requestparameter sollen in den Formaten JSON oder x-www-form-urlencode an den Request übergeben werden können. Gesendeten Anfragen sollen anschließend ihr Feedback je nach Wunsch des Clients, via Content Negotiation, entweder in der JSON oder als XML zurückgeben. Dafür sind drei Strategien zur Umsetzung bereitzustellen, entweder mittels Suffix, einem URL-Parameter oder dem HTTP Accept-Header.

4.2.1 Root Controller

Mit dem Root Controller soll der Einstiegspunkt an der URI `/api` bereitgestellt werden. Ein GET-Request an diesem Einstiegspunkt soll aber ausschließlich Links zu den einzelnen Startpunkten der Ressourcen Player, Match und Draw bereitstellen. Dies wiederum dient der Umsetzung des Konzeptes HATEOAS, welches im Kapitel 4.3 genauer definiert wird und ausführlichere Erläuterungen enthält, wie das Ziel dieses Konzeptes erreicht werden soll.

Die nachfolgende Abbildung 4.2 dient dabei als Veranschaulichung des Einstiegspunktes.

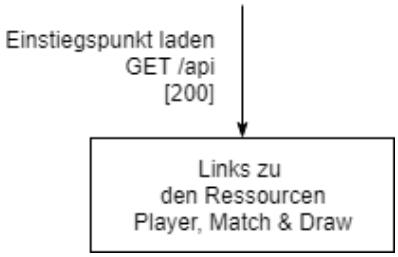

Abbildung 4.2: Root Controller - Übersicht der Einstiegspunkte

4.2.2 Player Controller

Der Player Controller soll zwei Einstiegspunkte an den URIs `/api/players` und `/api/players/{id}` zur Verfügung stellen. Der Parameter `{id}` dient dabei als Platzhalter für die ID eines Players.

Am ersten Einstiegspunkt ist eine Liste aller Spieler über einen GET-Request bereitzustellen. Des Weiteren soll an diesem die Möglichkeit bestehen, einen neuen Player mithilfe eines POST-

Requests zu erzeugen. Dabei muss als Parameter der Name und das Passwort des Players mitgegeben werden. Die ID ist durch die SQlite-Datenbank mittels Autoincrement zu erzeugen. Bei erfolgreicher Erstellung des Players ist dieser als HTTP-Response zurück zugegeben. Tritt bei diesem Prozess jedoch ein Fehler auf so muss, anstatt des Players, dieser zurück gegeben werden.

Am zweiten Einstiegspunkt ist ein einzelner existierender Player über einen GET-Request bereitzustellen, über einen DELETE-Request gelöscht und über einen PATCH-Request aktualisiert werden können. Dabei darf bei einer Aktualisierung eines Players nur das Passwort, laut Anforderungen[1], geändert werden.

Für ein besseres Verständnis der einzelnen Einstiegspunkte, mit den dazugehörigen Request-Methoden, bietet die Abbildung 4.3 eine visuelle Verdeutlichung des Player Controllers.

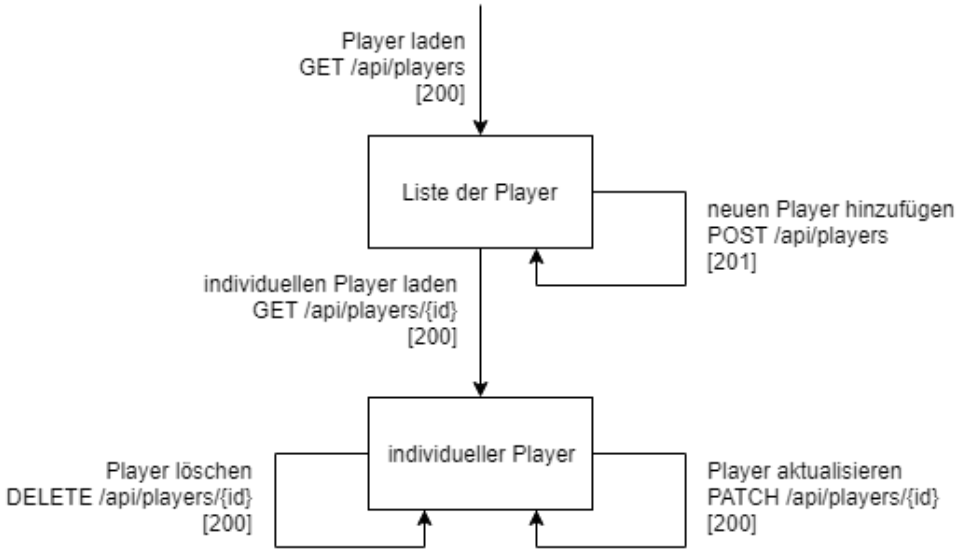

Abbildung 4.3: Player Controller - Übersicht der Einstiegspunkte

4.2.3 Match Controller

Die URIs `/api/matches`, `/api/matches/{id}`, `/api/matches/{id}/draws` und `/api/matches/{id}/pieceSets` sollen durch den Match Controller bereitgestellt werden.

Dabei dient die erste URI ebenso wie beim Player Controller der Zurverfügungstellung einer Liste aller registrierten Matches und dem Anlegen neuer. Das Bereitstellen der Liste hat mittels GET- und das Anlegen mittels POST-Request zu erfolgen. Der GET-Request soll zwei optionale

1 siehe Kapitel 4.1

boolesche Parameter bereitstellen, womit das Senden der Historie von Draws bzw. der Figurenstellung bestimmt wird. Standardmäßig sollen die beiden Parameter `true` sein. Nachfolgend wird genauer erläutert wie diese Informationen separat angefordert werden können. Um ein neues Match zu registrieren, müssen dabei die ID's der beiden Spielteilnehmer als Parameter mitgesendet werden. Anhand des Parameternamens wird festgelegt, welcher Spieler Weiß und welcher Schwarz spielt.

Der zweite Einstiegspunkt in diesem Controller soll ausschließlich dazu verwendet werden, um einzelne Matches mithilfe eines GET-Request anzufordern oder mit einem DELETE-Request zu löschen. Für das Anfordern eines einzelnen Matches stehen dabei, ebenso wie beim Request einer Match-Liste, zwei boolesche Parameter bereit. Sollte ein Nutzer ein Match löschen, so sind die zugehörigen Draws automatisch mit zu löschen. Um eine unrechtmäßige Manipulation der Match-Daten durch einen Nutzer zu verhindern, soll keine Möglichkeit bereitstehen, ein Match direkt zu aktualisieren. Änderungen der Match-Daten dürfen ausschließlich über das Hinzufügen von neuen Draws[1] erfolgen.

Die letzten beiden Einstiegspunkte sollen dazu dienen, große Match bezogene Daten separat zu ermitteln. Dabei ist mittels GET-Request an der URI `/api/matches/{id}/draws` eine Liste aller Draws und über die URI `/api/matches/{id}/pieceSets` eine Map mit allen Figuren der beiden Spielteilnehmer bereitzustellen. Neben den aktuell auf dem Spielfeld stehenden Figuren sollen dabei auch die Daten der bereits Geschmissenen mitgeschickt werden.

Die Abbildung 4.4 bietet für die zuvor definierten Einstiegspunkte eine grafische Veranschaulichung.

4.2.4 Draw Controller

Der Draw Controller stellt drei Einstiegspunkte an den URIs `/api/draws`, `/api/draws/{id}` und `/api/draws/ai` bereit.

Am ersten Einstiegspunkt soll dabei wieder eine Liste mittels GET- und das Hinzufügen mittels POST-Request von Draws zur Verfügung gestellt werden. Für das Hinzufügen eines neuen Draws ist die ID des Matches und der Draw-Code in der SAN als Parameter notwendig. Zusätzlich soll die Möglichkeit bestehen, die Zeilen- und Spaltennummer der Startposition mitzugeben. Wenn diese Informationen nicht mitgegeben werden, soll der Controller die Startposition kalkulieren.

1 siehe Kapitel 4.2.4

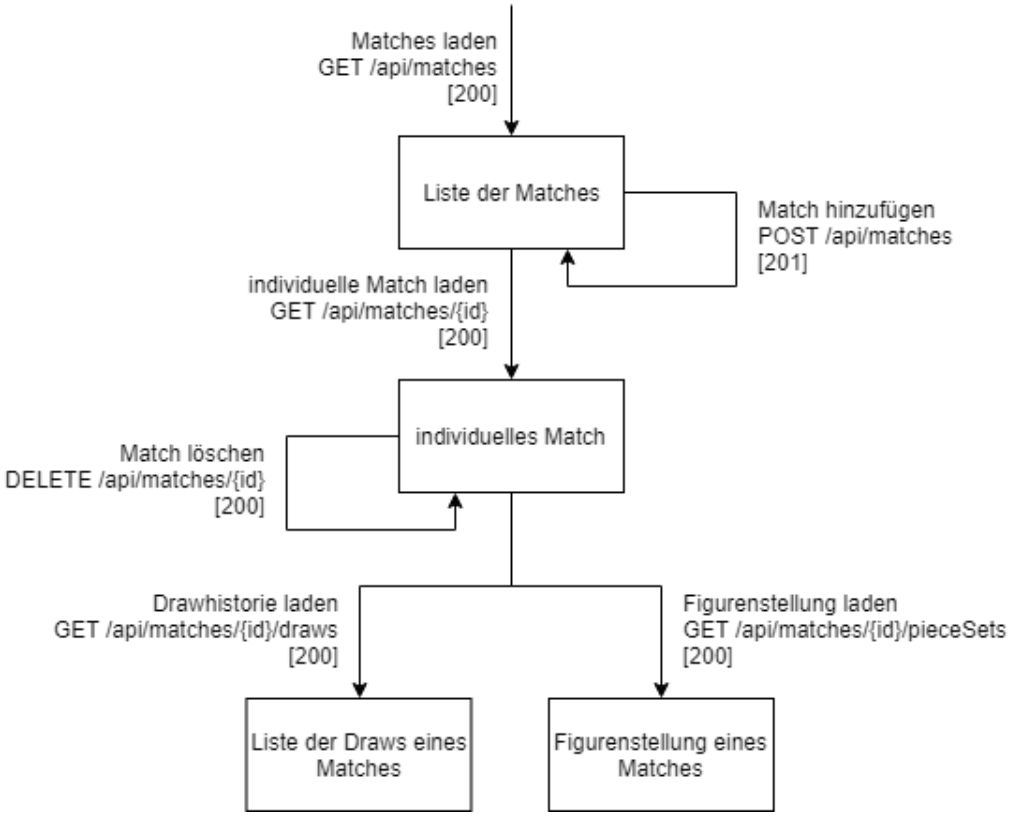

Abbildung 4.4: Match Controller - Übersicht der Einstiegspunkte

Spalten sind dabei als Nummern und nicht als Buchstaben zu übergeben[1]. Nach erfolgreicher Validierung des Draw-Codes soll der Draw dem zugehörigen Match hinzugefügt und die Match-Daten aktualisiert werden.

Über den zweiten Einstiegspunkt soll die Abfrage nach einem einzelnen Draw möglich sein, aber eine Löschung oder Aktualisierung nicht. Sonst wäre eine Manipulation der Match-Daten seitens des Nutzers möglich. Die Entfernung von Draws ist ausschließlich über die Löschung des dazugehörigen Matches[2] zu erfolgen.

Der dritte Einstiegspunkt soll mittels POST-Request die Möglichkeit bereitstellen, einen Draw, mithilfe einer künstlichen Intelligenz, zu einem Match hinzuzufügen. Dafür muss ausschließlich die ID des benötigten Matches als Request-Parameter mitgegeben werden.

Wie in den vorherigen Kapiteln bietet die Abbildung 4.5 ein Veranschaulichung der definierten Einstiegspunkte.

1 A → 1; B → 2; C → 3; D → 4; E → 5; F → 6; G → 7; H → 8
2 siehe Kapitel 4.2.3

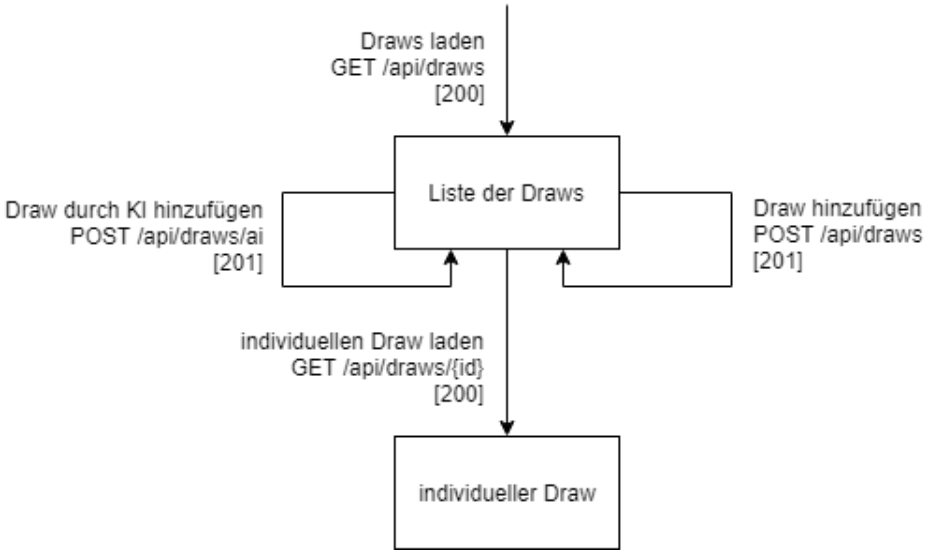

Abbildung 4.5: Draw Controller - Übersicht der Einstiegspunkte

4.2.5 Error Controller

Mithilfe des Error Controllers muss gewährleistet werden, dass alle nicht in der REST-API definierten Einstiegspunkte bzw. nicht definierte Request-Methoden an den Einstiegspunkten abgefangen werden. Tritt ein solcher Fall auf, so ist ein Fehler vom Server zurück zugeben. Somit wird verhindert, dass ein Nutzer der API einen Serverfehler, mit dem HTTP-Statuscode `500`[1], zurückerhält.

4.3 Erreichung des Designkonzepts HATEOAS

Um das Ziel der Selbstbeschreibung, welches das Konzept HATEOAS verfolgt, zu erreichen, gibt es mehrere Lösungswege.

Zum einen besteht die Möglichkeit der Erweiterung einer Ressource, wodurch diese als Hypermedia Format angesehen werden kann. Ein Beispiel dafür ist die Hypertext Application Language (HAL), welche in der Spezifikation [Kel] definiert wurde. Diese stellt mittels eines Dialektes der JSON die Multipurpose Internet Mail Extensions (MIME) Typen `application/hal+json` und `application/hal+xml` bereit, wodurch Ressourcen mit Relationen, in Form von Hyperlinks, ergänzt werden können.

Eine weitere Lösungsvariante kann durch den Link-Header des HTTP-Response realisiert werden. Diesem kann einfach eine Menge von Links in Form eines Strings übergeben werden, welchen der

1 siehe [Kre15, A.2.5]

Client anschließend auswerten kann.

Welche Variante die bessere ist, liegt im Auge des Betrachters. Für die Implementierung des Schach-Servers wurde Variante zwei gewählt, weil dadurch die Trennung der Links und der eigentlichen Ressource strikter wahrgenommen werden kann. Auch Kai Spichale bevorzugt diese Variante, denn dieser betrachtet die Links in seinem Buch [Spi16, S. 158] als Metainformationen und diese gehören seiner Meinung nach nicht in die Ressource.

KAPITEL 5

Konzept des Clients

In diesem Kapitel wird die Planung des Schachclients näher erläutert, welches als Grundbaustein für die Implementierung dient. Dabei werden im ersten Abschnitt die benötigten Anforderungen beschrieben, die der Client erfüllen muss. Im zweiten Abschnitt erfolgt die Visualisierung und Beschreibung der einzelnen Ansichten, welche für eine bequeme Nutzerinteraktion benötigt werden.

5.1 Anforderungen

Grundlegend soll der Client als Visualisierung des Servers dienen. Dafür ist eine Verwaltung von Playern und Matches bereitzustellen, inklusive der Möglichkeit zum Anlegen neuer bzw. Bearbeiten schon angelegter Einträge. Um anschließend auch Schach spielen zu können, muss der Client dafür eine komfortable Möglichkeit in Form eines virtuellen Schachbrettes bieten.

Als Grundanforderung dafür muss der Client natürlich mit dem Server kommunizieren können. Dafür muss dieser Requests senden und die empfangenen Response-Nachrichten verarbeiten können. Da der Server für manche Request spezielle Parameter benötigt, wie zum Beispiel einen String in der SAN, müssen diese gegebenenfalls ermittelt werden können.

Für ein bequemes Spielerlebnis soll der Client ein gestartetes Match automatisch aktualisieren, sobald sich die Daten auf dem Server verändert haben. Um dies zu realisieren, ist ein einfaches Polling-Verfahren zu implementieren.

Die letzte Grundanforderung soll eine innovative und benutzerfreundliche Bedienung der Anwendung sein, so dass der Nutzer keinerlei Kenntnisse, bis auf die Schachregeln, besitzen muss.

5.2 Mockup-Entwicklung der benötigten Client-Ansichten

In diesem Abschnitt werden die im Kapitel 5.1 zuvor definierten Anforderungen konkretisiert und visuell aufbereitet. Ziel ist dabei die Erstellung von Mockups der einzelnen Ansichten, um die

Implementierung zu vereinfachen bzw. zu beschleunigen.

5.2.1 Startansicht

Die Startansicht ist der Ausgangspunkt für die Nutzer, welche beim Aufruf der Root-URL zurück zugeben ist. Mithilfe der Startansicht soll dem Nutzer die Möglichkeit bereitgestellt werden zur Player-Ansicht bzw. zur Match-Ansicht zu wechseln. Dafür ist ihm jeweils ein Button zum Auslösen dieses Wechsels zur Verfügung zustellen.

Die Abbildung 5.1 visualisiert dabei die zuvor definierten Anforderungen an die Startansicht.

Abbildung 5.1: Mockup: Startansicht des Clients

5.2.2 Player-Ansicht

Mit dieser Ansicht ist dem Nutzer die Möglichkeit bereitzustellen, alle angelegten Player zu verwalten. Dafür muss ihm eine Tabelle für die Übersicht und ein Formular zum Anlegen neuer oder Bearbeiten bereits angelegter Player bereitgestellt werden. Über eine Spalte innerhalb der Tabelle sollen Buttons zur Verfügung stehen, über welche ein Player bearbeitet oder gelöscht werden kann.

Aus diesen gegebenen Anforderungen wurde das Mockup aus der Abbildung 5.2 entworfen, um diese visuell hervorzuheben.

5.2.3 Match-Ansicht

Mithilfe der Match-Ansicht ist dem Nutzer eine Ansicht zur Match-Verwaltung zur Verfügung zustellen. Um dabei die Konsistenz zu wahren, ist auch diese genau so aufgebaut wie die Player-Ansicht, mit dem Unterschied, dass ein Match nicht gelöscht aber dafür gestartet werden kann. Daher ändern sich geringfügig die Buttons in der Tabelle.

Dabei werden die Anforderungen durch die Abbildung 5.3 grafisch dargestellt.

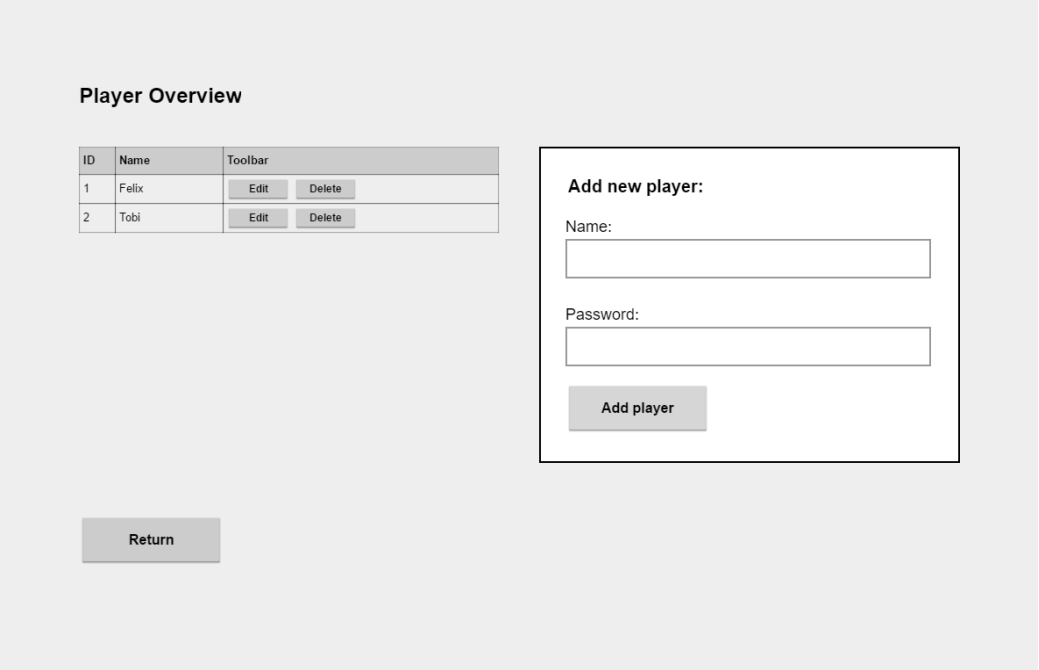

Abbildung 5.2: Mockup: Player-Ansicht des Clients

5.2.4 Ansicht eines gestarteten Matches

Mithilfe dieser Ansicht soll dem Nutzer ermöglicht werden ein gestartetes Match zu spielen. Dafür wird in erster Linie ein Schachbrett benötigt, auf welchem die Figuren dargestellt werden. Mittels „Drag & Drop“ soll der Nutzer anschließend Spielfiguren bewegen können. Für eine einfachere

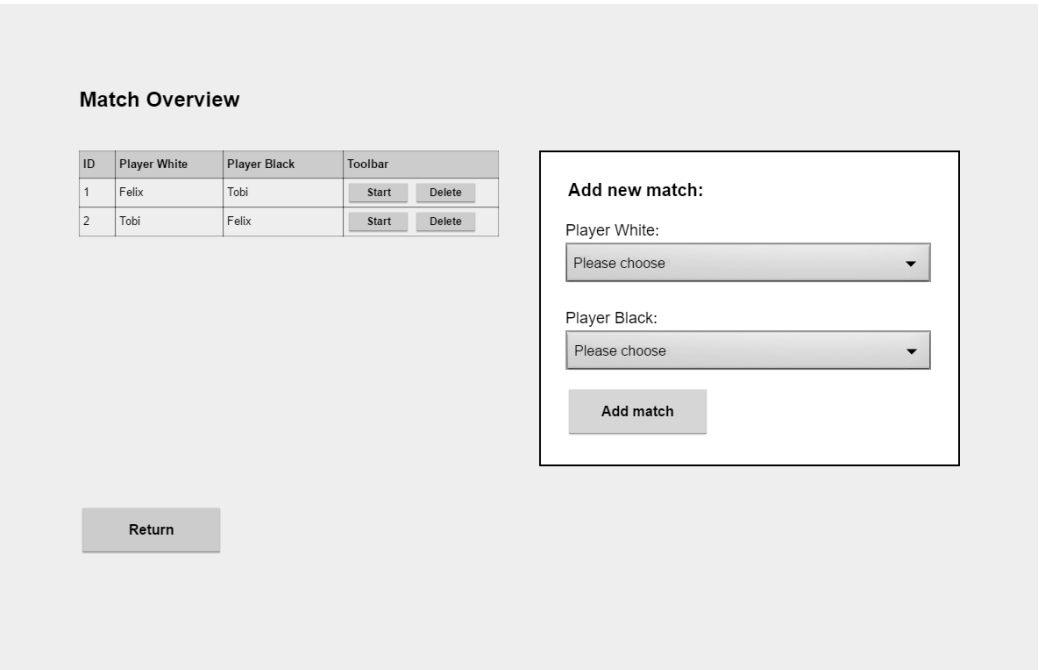

Abbildung 5.3: Mockup: Match-Ansicht des Clients

Bedienung und zur Unterstützung des Verständnisses der Schachregeln sollen alle möglichen Züge einer Figur hervorgehoben werden, sobald über diese mit der Maus gefahren wird. Da Informationen über bereits geschmissene Figuren oder welche Züge bisher getätigt wurden sehr hilfreich sein können, sollen diese neben dem Schachbrett dargestellt werden.

Anhand dieser Kriterien an die Ansicht eines gestarteten Matches wurde das Mockup aus der Abbildung 5.4 entwickelt.

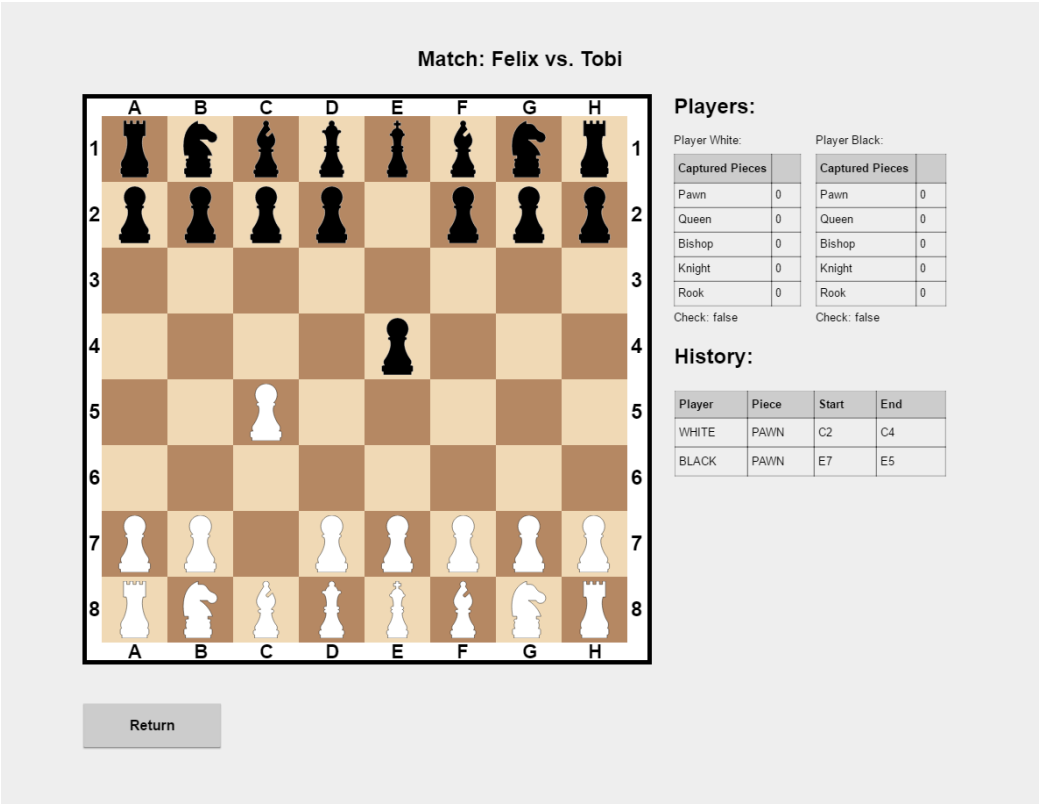

Abbildung 5.4: Mockup: Ansicht eines gestarteten Matches

KAPITEL 6

Implementation des Servers

Inhalt dieses Kapitels ist die Umsetzung des Konzeptes aus Kapitel 4. Dafür wird im ersten Abschnitt auf die Bibliotheken bzw. Frameworks eingegangen, welche für die Umsetzung verwendet wurden. Der zweite Abschnitt behandelt dabei die Anbindung an die SQLite Datenbank. Anschließend folgt eine Erläuterung zur Konfiguration einer Spring-Applikation, welche Content Negotiation unterstützt. Daraufhin folgt eine Erklärung zur Umsetzung des Designkonzeptes HATEOAS am Beispiel eines Requestes. Im Abschnitt fünf wird die Fehlerbehandlung bzw. das Exceptionhandling näher beschrieben. Das letzte Unterkapitel befasst sich mit der Ermittlung bzw. der Analyse von Strings, welche eine der Schachnotationen FEN oder SAN enthalten.

6.1 Verwendete Bibliotheken/Frameworks

Die verwendeten Bibliotheken bzw. Frameworks sollen die Umsetzung der Anforderungen aus dem Kapitel 4.1 vereinfachen und beschleunigen. Dabei werden diese in den nachfolgenden Unterabschnitten 6.1.1 bis 6.1.4 in den Punkten Zweck, Einrichtung und Verwendung näher betrachtet.

6.1.1 Spring Boot

Spring Boot wird als ein leichtgewichtiges Framework für die Umsetzung von Java Applikationen beschrieben. Dabei bezieht sich leichtgewichtig nicht auf die Größe oder Anzahl der enthaltenen Klassen. Es ist eher als geringer Aufwand an Änderungen am eigenen Programmcode zu verstehen, um die Vorteile des Frameworks nutzen zu können. [Cos17] Spring bietet eine Vielzahl an Einsatzmöglichkeiten, aber für die Umsetzung des Schachservers soll es für die Erzeugung der REST-API dienen.

Um Spring Boot in einem Projekt einzubinden, müssen die Zeilen aus dem Listing 6.1 in der Build-Datei `build.gradle` eingefügt werden. Zu beachten ist dabei, dass Gradle nur eine Lösung für die Einbindung ist. Da aber für die Umsetzung ebenfalls Gradle verwendet wird, werden alle anderen Lösungen an dieser Stelle vernachlässigt.

```
buildscript {
  repositories {
    mavenCentral()
  }
  dependencies {
    classpath "org.jetbrains.kotlin:kotlin-allopen:1.2.30"
    classpath "org.springframework.boot:spring-boot-gradle-plugin:1.5.4.RELEASE"
  }
}
apply plugin: "kotlin-spring"
apply plugin: "org.springframework.boot"

repositories {
  mavenCentral()
}

dependencies {
  compile "org.springframework.boot:spring-boot-starter-web:1.5.4.RELEASE"
}
```

Listing 6.1: Einbindung des Spring Framework mittels Gradle

Für die Erstellung einer REST-API muss zunächst ein Controller für eine Ressource angelegt werden. Dabei ist es sinnvoll für jede Ressource einen eigenen Controller zu definieren. Innerhalb dieser werden anschließend alle Einstiegspunkte erzeugt. Als Beispiel soll das Listing 6.2 in Form eines klassischen „Hello World“ dienen. Dafür wird eine Klasse `GreetingController` mit einer Funktion `getGreeting` definiert. Als Parameter bekommt diese Funktion einen Namen übergeben, welcher standardmäßig den String „World“ beinhaltet. An dieser Stelle kommt das Spring Boot Framework ins Spiel. Dieses stellt eine Reihe von Annotationen bereit, über welche eine REST-API konfiguriert werden kann. Dabei dient die Annotation `@RestController` dazu, die Klasse `GrettingController` in einen „RestController“ zu portieren, mit welchen anschließend Einstiegspunkte definiert werden können. Innerhalb dieses Controllers besteht daraufhin die Möglichkeit eine Funktion, mit der Annotation `@GetMapping`, als Einstiegspunkt für einen `GET`-Request an einer Ressource zu definieren. Um nun dem Nutzer der API die Möglichkeit zu bieten Parameter im Request mitzugeben, müssen diese mit der Annotation `@RequestParam` erweitert werden, wie im Listing 6.2 am Parameter `name` zu sehen.

```
import org.springframework.web.bind.annotation.*

@RestController
class GreetingController {
  @GetMapping("/greeting")
  fun getGreeting(@RequestParam name: String = "World"): String {
    return "Hello $name!"
  }
}
```

Listing 6.2: Beispiel: Spring Controller

Abschließend muss nur noch der Startpunkt für die Spring-Applikation definiert werden. Dafür

wird mithilfe der Annotation `@SpringBootApplication` ein Klasse erweitert, welche aber keinerlei Informationen benötigt. Anschließend muss diese in der Main-Funktion, wie im Listing 6.3 zu sehen, aufgerufen werden.

```
@SpringBootApplication
class Application

fun main(args: Array<String>) {
  SpringApplication.run(Application::class.java, *args)
}
```

Listing 6.3: Beispiel: Spring Application Class

Für eine detaillierte Beschreibung dieses Beispiels sind auf den Internetseiten [Har] und [Incj] weitere Informationen zu finden.

6.1.2 SQLite

SQLite ist eine OpenSource Bibliothek, welche ein dateibasiertes relationales Datenbanksystem bereitstellt. Der größte Unterschied zu anderen relationalen SQL-Datenbanken besteht darin, das SQLite keinen separaten Serverprozess besitzt und somit als eingebettete Datenbank-Engine betrachtet werden kann. Alle Tabellen, Indizes, Trigger und Sichten einer Datenbank werden dabei in einem plattformunabhängigen Format in einer einzigen Datei gespeichert. Das bedeutet, dass Datenbankdateien bequem zwischen 32-Bit und 64-Bit-Systemen oder Little-Endian- und Big-Endian-Architekturen getauscht werden können. [Hip]

Für die Einbindung von SQLite in ein Projekt sind, mittels der Datenbankschnittstelle Java Database Connectivity (JDBC), folgende Zeilen in der Build-Datei von Gradle zu ergänzen:

```
repositories {
  mavenCentral()
}

dependencies {
  compile "org.xerial:sqlite-jdbc:3.21.0.1"
}
```

Listing 6.4: Einbindung der Bibliothek SQLite mittels Gradle

Da für die Implementierung eine Object-relation mapping (ORM) Bibliothek[1] Verwendung findet, wird auf eine nähere Erläuterung von SQLite in Kombination mit dem JDBC Treiber an dieser Stelle verzichtet. Sollte dieses Szenario dennoch von Interesse sein, ist auf der Internetseite [Lim] ein Beispiel zur Datenbankanbindung, Erstellung von Tabellen und zum Absenden von SQL-Anfragen

1 siehe Kapitel 6.1.3

zu finden.

6.1.3 ORMLite

ORMLite ist ein OpenSource ORM Projekt von Gray Watson, welches für Java entwickelt wurde, aber in Kotlin durch die Möglichkeit der Interoperabilität mit Java ebenfalls verwendet werden kann. Die Bibliothek unterstützt dabei eine Reihe von Datenbank-Systemen, darunter vertreten sind beispielsweise MySQL, Postgres und SQLite.

Um ORMLite in ein Gradle Projekt einzubinden, müssen die Zeilen aus dem Listing 6.5 in die Build-Datei übertragen werden. Dabei muss neben der Core-Bibliothek die JDBC-Bibliothek von ORMLite eingebunden werden, welches für die Verbindung zur Datenbank zuständig ist. Da aber JDBC mit mehreren Datenbank-Systemen kommunizieren kann, muss noch der Datenbank-Treiber für das verwendete Datenbank-System eingebunden[1] werden.

```
repositories {
  mavenCentral()
}

dependencies {
  compile "com.j256.ormlite:ormlite-core:5.1"
  compile "com.j256.ormlite:ormlite-jdbc:5.1"
}
```

Listing 6.5: Einbindung der Bibliothek ORMLite mittels Gradle

Für die Persistierung zeigt das Listing 6.6 wie einzelne Klassen, durch die von ORMLite bereitgestellten Annotationen, vorbereitet werden müssen.

```
@DatabaseTable(tableName = "accounts")
public class Account {
  @DatabaseField(id = true)
  private String name;

  @DatabaseField(canBeNull = false)
  private String password;
  ...
  Account() {
    // all persisted classes must define a no-arg constructor with at least package visibility
  }
  ...
}
```

Listing 6.6: Beispiel: Persistierung einer Klasse mittels ORMLite [Wat]

Der Zugriff auf die Datenbank erfolgt mittels Database Access Object (DAO) Klassen, welche für

1 siehe Kapitel 6.1.2

jede Tabelle erzeugt werden müssen. Mit diesen DAOs können anschließend Datensätze erstellt, bearbeitet und gelöscht werden. Zu dem stellen diese eine Reihe von Funktionen bereit um Datensätze abzufragen, wie zum Beispiel die Abfrage nach allen Datensätzen in der Datenbank oder nach einem bestimmten Objekt anhand seiner ID. Wenn diese Standard-Funktionen nicht ausreichen sollten, besteht weiterhin die Möglichkeit komplexe Abfragen zu generieren. Für diese Generierung kommen sogenannte „Query-Builder“ zum Einsatz. Zur Veranschaulichung der Verwendung von ORMLite zeigt das Listing 6.7 ein Minimalbeispiel.

```
String databaseUrl = "jdbc:sqlite:path/to/account.db";
ConnectionSource connectionSource = new JdbcConnectionSource(databaseUrl);

Dao<Account,String> accountDao = DaoManager.createDao(connectionSource, Account.class);

TableUtils.createTable(connectionSource, Account.class);

String name = "Jim Smith";
Account account = new Account(name, "_secret");
accountDao.create(account);

Account account2 = accountDao.queryForId(name);
System.out.println("Account: " + account2.getPassword());

connectionSource.close();
```

Listing 6.7: Beispiel: Verwendung von ORMLite (verändert nach [Wat])

6.1.4 Fasterxml

Mithilfe von Fasterxml werden inkrementelle Parser- und Generator-Abstraktionen bereitgestellt, welche in der Standardimplementierung JSON verarbeiten können. Dabei bietet das OpenSource-Projekt noch weitere Unterstützung für andere Datenformate an, wie beispielsweise XML oder Comma-separated Values (CSV).[ANO18]

Für die Einbindung in ein Gradle-Projekt müssen die Zeilen aus dem Listing 6.8 zur Build-Datei hinzugefügt werden.

```
repositories {
  mavenCentral()
}

dependencies {
  compile "com.fasterxml.jackson.core:jackson-core:2.9.5"
  compile "com.fasterxml.jackson.core:jackson-annotations:2.9.5"
  compile "com.fasterxml.jackson.core:jackson-databind:2.9.5"
}
```

Listing 6.8: Einbindung der Bibliothek Fasterxml mittels Gradle

Um anschließend den XML-Support für ein Spring-Projekt einzurichten, müssen die Klassenmo-

delle mit Annotationen, erweitert werden. Das Listing 6.9 zeigt dabei, wie diese zu konfigurieren sind.

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
data class Player(
  @XmlElement
  val id: Int = 0,
  @XmlElement
  var name: String = "",
  @XmlElement
  var password: String = ""
)
```

Listing 6.9: Beispiel: Verwendung von Fasterxml

6.2 Anbindung an die Datenbank

Für die Anbindung an die Datenbank ist zuallererst die Vorbereitung der Klassenmodelle, wie im Kapitel 6.1.3 erläutert, notwendig. Als nächstes muss vor einer Interaktion eine Verbindung zur Datenbank aufgebaut sowie alle benötigten Tabellen und DAOs angelegt werden. Das Listing 6.10 zeigt dabei die Umsetzung dieser Notwendigkeiten.

```
class DatabaseUtility {
  companion object {
    private var connection: JdbcConnectionSource? = null
    var playerDao: Dao<Player, Int>? = null
      get() {
        if (field == null) connect()
        return field
      }
    ...
    private fun connect() {
      if (connection != null) return
      connection = JdbcConnectionSource("jdbc:sqlite:chessgame.db")
      createTables()
      createDaos()
    }

    private fun createTables() {
      TableUtils.createTableIfNotExists(connection, Player::class.java)
      ...
    }

    private fun createDaos() {
      playerDao = DaoManager.createDao<Dao<Player, Int>, Player>(connection, Player::class.java)
      ...
    }
  }
}
```

Listing 6.10: Verbindungsaufbau & Initialisierung der SQLite Datenbank

Dabei wird erst eine Verbindung aufgebaut sobald diese benötigt wird. Da alle Interaktionen mit der Datenbank über DAOs erfolgen, wird erst dann eine Verbindung aufgebaut sobald ein DAO

mittels `get`-Funktion angefordert wird. Es ist zu beachten, dass bei einer erneuten Anforderung eines DAOs die Verbindung nicht erneut aufgebaut wird. Dabei ist es egal, ob das selbe oder ein anderes angefordert wird. Ähnlich sieht das beim Anlegen der Tabellen aus, denn diese werden ausschließlich nur dann angelegt, sofern diese noch nicht in der Datenbank-Datei existieren. Dies tritt beispielsweise auf, sollte der Server neu gestartet werden.

6.3 Spring-Konfiguration für Content Negotiation

Content-Negotiation ist laut Kapitel 2.1.5 ein wichtiges Qualitätsmerkmal für eine REST-API. Das Framework Spring Boot[1] bietet auch hierfür Lösungen an, welche aber nicht standardmäßig aktiviert stehen. Die JSON wird dabei ohne weitere Konfiguration unterstützt, nur für XML müssen die Klassenmodelle mithilfe der Bibliothek Fasterxml[2] angepasst werden. Anschließend ist eine Erweiterung der Spring-Konfiguration, wie in Listing 6.11 zu sehen, notwendig. Die Implementierung zeigt dabei die Umsetzung der drei Strategien, welche im Kapitel 4.1 gefordert wurden.

```
@Configuration
class WebConfig : WebMvcConfigurerAdapter() {
  override fun configureContentNegotiation(configurer: ContentNegotiationConfigurer?) {
    configurer!!
      .favorPathExtension(true)
      .favorParameter(true)
      .parameterName("mediaType")
      .ignoreAcceptHeader(false)
      .useJaf(false)
      .defaultContentType(MediaType.APPLICATION_JSON)
      .mediaType("xml", MediaType.APPLICATION_XML)
      .mediaType("json", MediaType.APPLICATION_JSON)
  }
}
```

Listing 6.11: Spring-Konfiguration der drei Strategien für Content Negotiation

Die Internetseite [SRL16] bietet für dieses Thema eine ausführlichere Erläuterung der einzelnen Strategien und wie diese angewendet bzw. eingerichtet werden.

6.4 Implementation des HATEOAS-Konzeptes

Für die Implementation des Designkonzeptes aus dem Kapitel 4.3 kommt das Projekt „Spring HATEOAS“ zum Einsatz. Mithilfe des Objektes `ControllerLinkBuilder`, welches durch die Bibliothek bereitgestellt wird, können anhand der Controller und deren Funktionen Links generiert werden. Durch eine starke Kopplung an die Implementierung wird dabei eine redundante Pflege

1 siehe Kapitel 6.1.1
2 siehe Kapitel 6.1.4

der Links verhindert. Das Listing 6.12 zeigt dabei am Beispiel des `PlayerController` und der Funktion `getPlayerList` wie solche Links generiert werden können.

```
@GetMapping(produces = [APPLICATION_JSON_VALUE, APPLICATION_XML_VALUE])
fun getPlayerList(response: HttpServletResponse): ResponseEntity<PlayerHashMap> {
  val playerList = HashMap<Int, Player>()
  playerDao!!.queryForAll().forEach { playerList[it.id] = it }

  val links = HashSet<Pair<Link, String>>()

  val selfLink = linkTo(methodOn(PlayerController::class.java).getPlayerList(response))
      .withSelfRel()
  links.add(Pair(selfLink, "GET"))
  val optionsLink = linkTo(methodOn(PlayerController::class.java).playerOptions(response))
      .withRel("options")
  links.add(Pair(optionsLink, "OPTIONS"))
  val newLink = linkTo(methodOn(PlayerController::class.java)
      .addPlayer("valueOfName", "valueOfPassword", response))
      .withRel("new")
  links.add(Pair(newLink, "POST"))

  playerList.forEach { (playerId, _) ->
    val playerLink = linkTo(methodOn(PlayerController::class.java).getPlayerById(playerId, response))
        .withRel("next")
    links.add(Pair(playerLink, "GET"))
  }

  return ResponseEntity(PlayerHashMap(playerList), HateoasUtility.createLinkHeader(links), OK)
}
```

Listing 6.12: Linkaufbau mithilfe des Projektes „Spring HATEOAS“

Die ersten beiden Zeilen der Funktion dienen dazu alle Player aus der Datenbank zu holen und alle nachfolgenden zum Linkaufbau. Alle erzeugten Links werden in ein `HashSet` gespeichert, anschließend mittels der statischen Methode `createLinkHeader` aus dem Objekt `HateoasUtility` in einem String geparst. Der erstellte String wird anschließend dem Link-Header des HTTP-Response zugewiesen. Das Listing 6.13 zeigt dabei die Implementierung der statischen Methode `createLinkHeader`.

```
fun createLinkHeader(linkList: HashSet<Pair<Link, String>>): HttpHeaders {
  val headers = HttpHeaders()
  val sb = StringBuilder()

  linkList.forEach { (link, verb) ->
    sb.append("<${link.href}>;rel=${link.rel};verb='$verb'")
  }

  headers.set("Link", sb.toString())
  return headers
}
```

Listing 6.13: Implementierung: „Links zu String“ Parse-Funktion

6.5 Exceptionhandling

Laut den Anforderungen aus dem Kapitel 4.1 soll der Server so wenig wie möglich anfällig für Fehler sein. Dafür ist der richtige Umgang und auch eine verständliche und für den Endnutzer lesbare Fehlermeldung unerlässlich. Die auftretenden Fehler können dabei in einzelne Fehlerarten unterteilt werden.

Wenn ein Nutzer eine Ressource an der URI `/player/15` anfordert, aber kein Player mit der ID 15 existiert, dann wird ein Fehler mit dem HTTP-Statuscode `404` zurückgegeben. Schon der Statuscode alleine signalisiert dem Nutzer, dass die angeforderte Ressource nicht gefunden wurde. Sollte ein Benutzer der API einen Draw hinzufügen wollen aber der mitgeschickte Draw-Code nicht valide sein, so wird ein Fehler mit dem HTTP-Statuscode `409` zurückgegeben. Dieser zeigt dem Nutzer an, dass ein durch ihn verursachter Konflikt aufgetreten ist. Wenn ein Nutzer den HTTP-Statuscode `400` vom Server zurückerhält, so hat dieser eine URI angefragt, welcher durch die REST-API nicht definiert wurde. Für alle weiteren Fehler, welche nicht durch den Nutzer verursacht wurden, wird der HTTP-Statuscode `500` zurückgeben. Da so ein Fehler nur zurückgegeben wird sofern ein unerwarteter Server-Fehler aufgetreten ist, sollte dieser gar nicht bzw. nur selten auftreten. Ein Beispiel für so einen Fehler könnte ein Verbindungsabbruch zur Datenbank sein, worauf der Nutzer keinerlei Einfluss hat.

Das Listing 6.14 zeigt die notwendige Konfiguration der Spring-Applikation, um mit auftretenden Fehlern umgehen zu können.

Damit der API-Benutzer nicht nur einen Statuscode zum aufgetretenen Fehler erhält, sondern auch eine verständliche Fehlermeldung, wird zusätzlich im HTTP-Response ein Fehlerobjekt mitgeschickt. Dieses Fehlerobjekt ist eine Instanz des `ErrorResponseObject` und hält einen Zeitstempel, den Statuscode, die Bezeichnung des Statuscodes, welche Exception den Fehler verursacht hat, eine detaillierte Fehlermeldung und den Pfad an welchem der Fehler aufgetreten ist. Das Fehlerobjekt wird dabei, je nach angeforderten Format, als JSON oder XML zurückgegeben[1].

1 siehe Kapitel 6.3

```
@RestControllerAdvice
class ExceptionHandler {
  @ExceptionHandler(value = [BadRequestException::class])
  @ResponseStatus(BAD_REQUEST)
  fun handleBadRequestException(ex: Exception, request: WebRequest): ErrorResponseObject {
    return generateErrorResponseObject(ex, request, BAD_REQUEST)
  }

  @ExceptionHandler(value = [IllegalArgumentException::class])
  @ResponseStatus(NOT_FOUND)
  fun handleIllegalArgumentException(ex: Exception, request: WebRequest): ErrorResponseObject {
    return generateErrorResponseObject(ex, request, NOT_FOUND)
  }

  @ExceptionHandler(value = [RuntimeException::class])
  @ResponseStatus(CONFLICT)
  fun handleRuntimeException(ex: Exception, request: WebRequest): ErrorResponseObject {
    return generateErrorResponseObject(ex, request, CONFLICT)
  }

  @ExceptionHandler(value = [Exception::class])
  @ResponseStatus(INTERNAL_SERVER_ERROR)
  fun handleUnknownException(ex: Exception, request: WebRequest): ErrorResponseObject {
    return generateErrorResponseObject(ex, request, INTERNAL_SERVER_ERROR)
  }

  private fun generateErrorResponseObject(ex: Exception, request: WebRequest, statusCode: HttpStatus):
ErrorResponseObject {
    return ErrorResponseObject(...)
  }
}
```

Listing 6.14: Spring-Konfiguration des Exceptionhandling

```
class ErrorResponseObject(
  val timestamp: Date = Date(),
  val statusCode: Int = 500,
  val error: HttpStatus = HttpStatus.INTERNAL_SERVER_ERROR,
  val exception: String = "",
  val message: String = "No message available",
  val path: String = ""
) {
override fun toString(): String {
  return "ErrorResponseObject{" +
      "timestamp=" + timestamp +
      ", status=" + statusCode +
      ", error=" + error +
      ", exception=" + exception +
      ", message=" + message +
      ", path=" + path +
      "}".toString()
  }
}
```

Listing 6.15: Das Fehlerobjekt `ErrorResponseObject`

6.6 Analyse/Ermittlung der FEN bzw. SAN

Laut den Anforderungen aus dem Kapitel 4.1 wurde definiert, dass Änderungen an einem Match ausschließlich über das Hinzufügen eines Draws erfolgen dürfen. Trotzdem muss eine Validierung des übermittelten Draw-Codes durchgeführt werden, um zu überprüfen, dass dieser auch möglich ist. Dafür wurde der reguläre Ausdruck aus der Abbildung 6.1 entwickelt, welcher in acht Teile bzw. Gruppen unterteilt werden kann.

$$\underbrace{([KQBNR])?}_{1}\underbrace{([a-h]|[1-8])?}_{2}\underbrace{(x)?}_{3}\underbrace{([a-h])}_{4}\underbrace{([1-8])}_{5}\underbrace{([QBRN])?}_{6}\underbrace{(e\backslash.p\backslash.)?}_{7}\underbrace{(\backslash+\{1,2\}|\#)?}_{8}$$

Abbildung 6.1: Regulärer Ausdruck zur Validierung eines Strings in der SAN

Der erste Part ermittelt dabei die Art der Spielfigur. Anhand des Fragezeichen-Symbols ist zu sehen, dass dieser Teil optional ist, was daran liegt, dass kein Buchstabe angegeben wird, wenn eine Bauernfigur gezogen wurde. Der zweite Teil zeigt an, von welcher Spalte oder Reihe die Figur gestartet ist. Diese Information is notwendig, wenn zwei Figuren der selben Art auf das Zielfeld gelangen können. Part drei gibt an, ob eine Figur geschmissen wurde. In den Abschnitten vier und fünf wird das Zielfeld ermittelt, wobei der vierte Part die Spalte und der fünfte die Reihe darstellt. Teil sechs gibt die Art der Spielfigur an, in welche sich ein Bauer umwandelt, wenn er die hinterste Reihe erreicht hat. Der vorletzte Part zeigt an, ob ein „Schlagen im Vorbeigehen“ bzw. ein „Schlagen en passant“ durchgeführt wurde. Im letzten Abschnitt wird ermittelt, ob der Draw zu einem Schach oder einem Schachmatt führt. Dabei werden zwei Schreibweisen für ein Schachmatt akzeptiert, einmal „++“ und zum anderen „#“. Zu beachten ist, dass mithilfe des regulären Ausdrucks nicht ermittelt werden kann, ob eine Rochade durchgeführt wurde. Dies muss zuvor separat geprüft werden.

Mittels des entwickelten regulären Ausdrucks kann anschließend auf Grundlage der Figurenstellung, welche im Match gespeichert ist, überprüft werden, ob der Draw valide ist. Dafür wird als erstes das Startfeld ermittelt, sofern dieses nicht als Request-Parameter mit übergeben wurde. Wichtig hierbei ist, dass immer nur ein einziges Startfeld ermittelt werden darf, hierfür müssen ggf. die Informationen aus dem zweiten Teil des regulären Ausdrucks heran gezogen werden.

Mit dem ermittelten Startfeld kann nun die zu bewegende Figur ermittelt und anhand dieser alle möglichen Zielfelder berechnet werden. Sollte dabei das mit dem Request übermittelte Zielfeld nicht enthalten sein, so ist der Draw invalide. Wenn hingegen das Zielfeld enthalten ist, so wird überprüft, ob eine Figur des Gegners geschlagen wurde und ob der Draw zu einem Schach oder Schachmatt führen würde. Sollte dabei einer dieser Werte nicht oder falsch angegeben sein, dann ist der Draw ebenfalls invalide.

Ist der Draw nach allen Überprüfungen valide, wird dieser schließlich in der Datenbank gespeichert und das erzeugte Objekt wird, nach der Aktualisierung des dazugehörigen Matches, als HTTP-Response zurückgegeben. Für die Aktualisierung wird dafür die Position der bewegten Figur geändert, gegebenenfalls die geschlagene Figur vom Spielfeld entfernt, die Möglichkeit zur Rochade und der Wert der Halbzüge angepasst. Anhand dieser neuen Werte wird der Match-Code neu kalkuliert. Anschließend wird das geänderte Match in der Datenbank aktualisiert.

KAPITEL 7

Implementation des Clients

Dieses Kapitel beschäftigt sich mit der Umsetzung des Konzeptes aus Kapitel 5. Im ersten Abschnitt werden dabei die Bibliotheken bzw. Frameworks, welche bei der Umsetzung zum Einsatz kamen, näher beleuchtet. Der nachfolgende Abschnitt dient als Beschreibung der Implementation der Kommunikation zwischen Client und Server in Form von Requests. Im letzten Abschnitt wird zuletzt die Implementierung der verwendeten Polling-Funktionalität genauer betrachtet.

7.1 Verwendete Bibliotheken/Frameworks

Die in diesem Kapitel vorgestellten Bibliotheken bzw. Frameworks sollen die Umsetzung der Anforderungen aus dem Kapitel 5.1 unterstützen. Dabei werden diese in den nachfolgenden Unterabschnitten 7.1.1 bis 7.1.4 in den Punkten Zweck, Einrichtung und Verwendung näher betrachtet.

7.1.1 RequireJS

RequireJS [ANOa] ist eine für JavaScript entwickelte Bibliothek zum Laden von Modulen. Dabei ist es für die Nutzung innerhalb des Browsers optimiert, kann aber auch für andere Umgebungen genutzt werden. Ziel von solchen Bibliotheken wie RequireJs soll sein, den eigenen Code zu beschleunigen und die Qualität zu steigern.

Das Listing 7.1 zeigt wie RequireJs in einem Gradle Projekt eingebunden werden kann. Dafür müssen diese Zeilen in die `build.gradle` eingefügt werden.

```
1 repositories {
2   mavenCentral()
3 }
4
5 dependencies {
6   compile "org.webjars:requirejs:2.3.5"
7 }
```

Listing 7.1: Einbindung der Bibliothek RequireJs mittels Gradle

Für die Modulbeschreibung innerhalb von JavaScript stehen mehrere Formate zur Verfügung. RequireJS setzt dabei auf das Format AMD. Das Listing 7.2 stellt ein einfaches Beispiel für die Definition eines AMD-Moduls bereit. In dem Zeitschriftartikel [Bra17] können zu den einzelnen Modul-Formaten und deren Einsatzmöglichkeiten nähere Informationen nachgelesen werden.

```
define(['jquery'] , function ($) {
  return function () {};
});
```

Listing 7.2: Beispiel: Moduldefinition mittels AMD [ANOb]

7.1.2 kotlinx.html

Die Bibliothek kotlinx.html [Masb] wird offiziell durch JetBrains entwickelt, welche auch für die Entwicklung von Kotlin verantwortlich sind. Mithilfe dieser wird eine DSL bereitgestellt, mit welcher HTML-Code erzeugt und ergänzt werden kann. Dabei steht sie für die Plattformen JVM und JavaScript bereit.

Um die Bibliothek in einem Projekt mit Gradle als Build Tool zu verwenden, müssen die Zeilen aus Listing 7.3 in der Datei `build.gradle` ergänzt werden.

```
repositories {
  jcenter()
}

dependencies {
  compile "org.jetbrains.kotlinx:kotlinx-html-js:0.6.8"
}
```

Listing 7.3: Einbindung der Bibliothek kotlinx.html mittels Gradle

Mit dieser Bibliothek kann, wie schon kurz zuvor erwähnt, sämtlicher HTML-Code in Kotlin-Code ausgelagert werden. Das bringt einen großen Vorteil mit sich, denn das Erstellen des Codes wird so durch eine statische Typisierung abgesichert. Damit können mögliche Fehler im HTML-Code bereits während der Übersetzung des Quellcodes erkannt werden. Vergessene oder gar falsch verschachtelte HTML-Tags können somit leichter vermieden werden.

Ein Beispiel für die Verwendung der Bibliothek stellt das Listing 7.4 dar, welches den HTML-Code aus Listing 7.5 generiert.

```
import kotlinx.html.*
import kotlinx.html.dom.*

val myDiv = document.create.div {
  p { +"text inside" }
}
```

Listing 7.4: Beispiel: Verwendung der Bibliothek kotlinx.html [Masa]

```
<div>
  <p>
    text inside
  </p>
</div>
```

Listing 7.5: Beispiel: Verwendung der Bibliothek kotlinx.html (Ergebnis)

7.1.3 kotlinx.serialization

Ebenso wie kotlinx.html ist kotlinx.serialization [Staa] eine offiziell unterstützte Bibliothek, welche in erster Linie für das serialisieren und deserialisieren von Objekten entwickelt wurde. Dabei unterstützt die Bibliothek die Formate JSON, CBOR und Protobuf „out of the box“.

Für die Verwendung in einem Gradle-Projekt müssen die Zeilen aus dem Listing 7.6 in der Build-Datei von Gradle eingefügt werden.

```
buildscript {
  repositories {
    maven { url "https://kotlin.bintray.com/kotlinx" }
  }
  dependencies {
    compile "org.jetbrains.kotlinx:kotlinx-gradle-serialization-plugin:0.5.0"
  }
}

apply plugin: 'kotlinx-serialization'

repositories {
  maven { url "https://kotlin.bintray.com/kotlinx" }
}

dependencies {
  compile "org.jetbrains.kotlinx:kotlinx-serialization-runtime-js:0.5.0"
}
```

Listing 7.6: Einbindung der Bibliothek kotlinx.serialization mittels Gradle

Um Daten im JSON-Format zu serialisieren bzw. deserialisieren müssen die Zielobjekte, wie im Listing 7.7 zu sehen, mit der Annotation `Serializable` erweitert werden. Dabei besteht die Möglichkeit Eigenschaften einer Klasse mithilfe der Annotation `Optional` als optional zu definieren, wobei diese Eigenschaften nicht unbedingt existieren müssen.

```
import kotlinx.serialization.*

@Serializable
data class Data(val a: Int, @Optional val b: String = "42")
```

Listing 7.7: Beispiel: Model-Erweiterung für Unterstützung der Kotlinx.serialization Bibliothek (verändert nach [Stab])

Um anschließend die Beispielklasse `Data` aus dem Listing 7.7 zu serialisieren muss die erste Zeile und um sie zu deserialisieren die zweite Zeile aus dem Listing 7.8 verwendet werden.

```
JSON.stringify(Data(42))          // {"a": 42, "b": "42"}
JSON.parse<Data>("""{"a":42}""") // Data(a=42, b="42")
```

Listing 7.8: Beispiel: Serializierung und Deserialisierung der `Data` Klasse (verändert nach [Stab])

7.1.4 kotlinx.coroutines

Die Bibliothek kotlinx.coroutines [Elia] ist eine weiteres Projekt aus der kotlinx Serie von JetBrains, welche eine Reihe von high-level Coroutinen bereitstellt. Im Falle des Clients soll die Bibliothek dazu genutzt werden, um auf den Abschluss von Funktionen warten zu können. Deshalb beschränkt sich die Erläuterung dieser Bibliothek auf diese Funktionalität. Für weitere Informationen bzw. für andere Anwendungsfälle können diese in der Dokumentation unter [Elib] nachgelesen werden.

Um die Bibliothek einzubinden, müssen die Zeilen aus dem Listing 7.9 in die Build-Datei eingefügt werden. Zu beachten ist dabei, dass in der verwendeten Version „0.22.5“ zusätzlich die letzten vier Zeilen notwendig sind. Was daran liegt, dass sich die Bibliothek derzeit noch im experimentellen Stadium befindet und ansonsten nicht verwendet werden kann. Sobald dieses Stadium abgeschlossen ist, können diese Zeilen weggelassen werden.

```
repositories {
  jcenter()
}

dependencies {
  compile "org.jetbrains.kotlinx:kotlinx-coroutines-core-js:0.22.5"
}

kotlin {
  experimental {
    coroutines 'enable'
  }
}
```

Listing 7.9: Einbindung der Bibliothek kotlinx.coroutines mittels Gradle

Mithilfe der Funktion `launch`, welche als letzten Parameter einen Lambda-Ausdruck entgegen-

nimmt, kann innerhalb dieses Ausdrucks beispielsweise auf ein `Promise` Objekt gewartet werden. Ein Anwendungsbeispiel für diesen Fall wird im Kapitel 7.2 genauer betrachtet und deshalb an dieser Stelle nicht näher erläutert.

Im Kapitel 7.3 wird mithilfe der Funktion `launch` ein weiterer Anwendungsfall untersucht. Denn mit Coroutinen ist es außerdem möglich, einen Prozess für eine gewisse Zeit zu unterbrechen, ohne das dieser seinen Zustand verliert oder die Anwendung blockiert.

7.2 Implementierung der Request-Funktionalität

Im Kapitel 5.1 wurde definiert, dass der Client mit dem Server kommunizieren können muss. Um das zu gewährleisten, bedarf es einer Reihe von Funktionen um einen HTTP-Request an den Server zu senden. Dabei wird für jeden Request-Typ eine eigene statische Methode definiert. Anschließend, um diese Funktionen nutzen zu können, müssen diese gegebenenfalls Parameter und eine Antwort vom Server verarbeiten können. Als Rückgabewert der Methoden dient das JavaScript Objekt `Promise`. Mit dessen Hilfe und der Bibliothek „kotlinx.coroutines“[1] besteht die Möglichkeit auf den Abschluss eines Requestes zu warten. Dies ist teilweise notwendig, da JavaScript grundlegend asynchron arbeitet und dieses Verhalten bei Requests manchmal unerwünscht sein kann.

Das Listing 7.10 zeigt die Implementierung der statischen Methoden für die Request-Typen `GET` und `POST`. Alle weiteren Request-Typen sind ebenfalls definiert, werden aber in dieser Veranschaulichung vernachlässigt.

Die dargestellte Funktion `parseParams` verarbeitet dabei eine übergebene Parameterliste, welche aus `Key-Value`-Paaren bestehen muss. Der `Key` dient dafür als Parameterindikator in Form eines Strings und der `Value` als Wert des Parameters. Der Funktionsparameter `callback` kann, in Form eines Lambda-Ausdrucks, beim Funktionsaufruf definiert werden und wird anschließend nach Erhalt des HTTP-Response ausgeführt.

Wie danach ein Request gesendet werden kann, zeigt das Listing 7.11. In diesem wird eine Anforderung der Playerliste mittels `GET`-Request gezeigt. Dabei wird es in einer `launch`-Umgebung ausgeführt, welche das Warten auf die Antwort, mit dem Funktionsaufruf `await`, ermöglicht. Innerhalb von Kotlin besteht des Weiteren die Möglichkeit den letzten Parameter, sofern dieser ein Funktionsparameter ist, in geschweiften Klammern zu definieren. Daher stellt dieser Code die `callback`-Funktion dar, welche die Antwort zuerst deserialisiert und anschließend alle daraus

1 siehe Kapitel 7.1.4

```
class RequestUtility {
  companion object {
    fun get(
        url: String,
        vararg params: Pair<String, Any> = arrayOf(),
        callback: ((Event) -> dynamic)? = null
    ): Promise<XMLHttpRequest> {
      return Promise { resolve, _ ->
        val request = XMLHttpRequest()

        if (params.isNotEmpty()) request.open("GET", "$url?${parseParams(params)}")
        else request.open("GET", url)

        request.addEventListener("load", callback)
        request.addEventListener("load", { resolve(request) })
        request.send()
      }
    }

    fun post(
        url: String,
        vararg params: Pair<String, Any> = arrayOf(),
        callback: ((Event) -> dynamic)? = null
    ): Promise<XMLHttpRequest> {
      return Promise { resolve, _ ->
        val request = XMLHttpRequest()

        request.open("POST", url)
        request.addEventListener("load", callback)
        request.addEventListener("load", { resolve(request) })
        request.setRequestHeader("Content-type", "application/x-www-form-urlencoded")
        request.send(parseParams(params))
      }
    }

    private fun parseParams(params: Array<out Pair<String, Any>>): String {
      var paramsAsString = ""

      params.forEach { (key, value) ->
        if (paramsAsString != "") paramsAsString += "&"
        paramsAsString += "$key=$value"
      }

      // %2B is the '+' character. If using the '+' character it will parse into a space character
      paramsAsString = paramsAsString.replace("+", "%2B")

      return paramsAsString
    }
  }
}
```

Listing 7.10: Implementierung der Request-Methoden `GET` und `POST`, inklusiver der Parameterverarbeitung

resultierenden Player dem Client-Objekt hinzufügt.

```
launch {
  get("${client.config.serverRootUrl}players") {
    if (it.target is XMLHttpRequest) {
      val playerHashMap = JSON.parse<PlayerHashMap>((it.target as XMLHttpRequest).responseText)
      playerHashMap.player.forEach { client.addPlayer(it.value) }
    }
  }.await()
}
```

Listing 7.11: Funktionsaufruf eines `GET`-Requestes am Beispiel der Playerliste

7.3 Implementierung des Polling-Verfahrens

Für die Umsetzung des Polling-Verfahrens, welches in den Anforderungen aus dem Kapitel 5.1, verlangt wurde, muss mehreres gewährleistet werden. Zu aller erst muss das Polling gestartet und gestoppt werden können. Des Weiteren muss die Möglichkeit bestehen eine Aufgabe zu definieren, welche nach dem Start des Pollings rekursiv ausgeführt wird. Zwischen diesen rekursiven Aufrufen der Aufgabe soll ein Zeit-Wert definiert werden können, welcher den Prozess für diese Zeit unterbricht bevor sich dieser erneut aufruft. Sobald der Polling-Prozess gestoppt wurde, muss der rekursive Aufruf unterbrochen werden.

In dem Listing 7.12 wurden diese Anforderungen an das Polling-Verfahren implementiert. Die Eigenschaft `stopPolling` gibt dabei an, ob der rekursive Prozess unterbrochen werden soll. Mithilfe der Wartezeit und der auszuführende Aufgabe als Parameter, kann das Polling mit der `start`-Funktion eingeleitet werden. Dafür wird die Eigenschaft `stopPolling` auf `false` gesetzt und anschließend die Rekursion gestartet. Sobald die `stop`-Funktion aufgerufen wird, wird die Eigenschaft `stopPolling` auf `true` gesetzt. Das wiederum führt in der Methode `sendPollingRequest` zu einem Abbruch, was schließlich die Rekursion unterbricht.

Um anschließend die Klasse `PollingUtility` nutzen zu können, muss ausschließlich eine Instanz dieser angelegt und die `start`-Funktion aufgerufen werden. Dabei wird, wie im Listing 7.13 zu sehen, die Wartezeit in Millisekunden und die Aufgabe der Rekursion übergeben. Dafür muss die Wartezeit als Integer und die Aufgabe als Lambda-Ausdruck übergeben werden. Im gezeigten Beispiel wird dabei im übergebenen Intervall ein Request an den Server gesendet, welcher alle Draws eines Matches anfordert. Der HTTP-Response wird daraufhin in eine Draw-Liste deserialisiert, aus welcher zu aller erst alle bereits im Match existierenden Draws entfernt werden. Sollten anschließend noch Draws enthalten sein, so werden diese im dazugehörigen Match ergänzt.

```
class PollingUtility {
  private var stopPolling = true

  fun start(delayTime: Int, pollingTask: () -> Unit) {
    stopPolling = false
    sendPollingRequest(delayTime, pollingTask)
  }

  fun stop() {
    stopPolling = true
  }

  private fun sendPollingRequest(delayTime: Int, pollingTask: () -> Unit) {
    if (stopPolling) return
    launch {
      pollingTask()
      delay(delayTime)
      sendPollingRequest(delayTime, pollingTask)
    }
  }
}
```

Listing 7.12: Implementierung der Klasse `PollingUtility` für die Umsetzung des Polling-Verfahrens

```
val pollingDraw = PollingUtility()

pollingDraw.start(client.config.pollingDelayTime) {
  get("${client.config.serverRootUrl}matches/$matchId/draws") {
    if (it.target is XMLHttpRequest) {
      val drawList = JSON.parse<DrawList>((it.target as XMLHttpRequest).responseText)
      if (match.history.size != drawList.draws.size) {
        match.history.forEach { draw ->
          drawList.draws.removeAll { it.id == draw.id }
        }
        drawList.draws.forEach { draw ->
          match.addDraw(draw, true)
        }
      }
    }
  }
}
```

Listing 7.13: Einbindung bzw. Nutzung der `PollingUtility` Klasse

KAPITEL 8

Fazit

Zielstellung dieser Arbeit war es herauszufinden ob es möglich ist mit der Programmiersprache Kotlin sowohl den Server als auch den Client, am Beispiel eines Schachspiels, zu programmieren. Dabei wurde durch diese Arbeit gezeigt, dass dies keinerlei Problem darstellt. Dennoch stellt sich die Frage ob sich der Einsatz von Kotlin lohnt. Zur Beantwortung dieser Frage wird in den Abschnitten 8.1 und 8.2 zwischen Java und JavaScript, als Ziel der Kompilierung, unterschieden.

8.1 Verwendung von Kotlin für serverseitige Programmierung (Java)

Grundlegend lässt sich sagen das die Programmierung mit Kotlin als Ersatz für Java sehr angenehm ist. Im Vergleich zu Kotlin wirkt Java teilweise sehr aufgebläht, was möglicherweise durch die benötigten `getter`- und `setter`-Methoden geschuldet sein kann. In Kotlin werden diese automatisch anhand der Klasseneigenschaften generiert, können aber bei Bedarf überschrieben werden[1].

Ein weiterer Pluspunkt ist das die Entwickler von Kotlin versucht haben aus den Fehlern von Java zu lernen und diese zu beseitigen. Ein großer Aspekt dabei ist die Null-Sicherheit, welche dafür sorgt das keine `NullPointerException` mehr geworfen werden muss. Diesen Fehler bezeichnet selbst Tony Hoare, der Entwickler der Null-Referenz, in seiner Präsentation [Hoa09] als „billion-dollar mistake“. Sollte diese Funktionalität dennoch notwendig sein so kann der Datentyp einer Variable so definiert werden, dass dieser auch den Wert `null` annimmt [sroi]. Eine ganze Reihe weiterer behobener Fehler bzw. Verbesserungen können auf der offiziellen Referenzseite [srob] nachgelesen werden.

Da Java, wie schon im Kapitel 3.2 erwähnt, seit 1995 existiert und sich sehr großer Beliebtheit erfreut, existieren logischerweise derzeit eine ganze Menge an Bibliotheken. Diese bei einem Umstieg neu- bzw. umzuschreiben liegt in Niemandes Interesse, deshalb stellt Kotlin für dieses Problem die Möglichkeit der Interoperabilität mit Java-Code bereit. Damit ist es möglich, alle in Java geschriebenen Bibliotheken „out-of-the-box“ zu nutzen.

1 siehe Listing 6.10

Letztlich lässt sich meiner Meinung nach sagen, dass ich für eine Programmierung auf der JVM Kotlin Java vorziehen würde, da das meiste bis auf schon genannte Fehler unterstützt wird und noch einiges darüber hinaus.

8.2 Verwendung von Kotlin für clientseitige Programmierung (JavaScript)

Bei der Kompilierung nach JavaScript-Code konnte Kotlin mich nicht ganz überzeugen. Prinzipiell ist die Erweiterung der Objektorientierung und der statischen Typisierung durch Kotlin sehr nützlich, es stellt sich jedoch die Frage ob dies durch den EcmaScript 6[1] Standard noch notwendig ist. Durch diesen wird zumindest die Objektorientierung und eine Reihe weiterer Features, welche Kotlin gegenüber dem EcmaScript 5 Standard besitzt, ergänzt.

Genau wie bei der Kompilierung nach Java-Bytecode ist es auch bei der clientseitigen Programmierung wichtig, bereits existierende Bibliotheken verwenden zu können. Kotlin bietet dafür, wie schon im Kapitel 3.2 erwähnt, mittels JsInterop die Möglichkeit Programm-APIs manuell nachzubauen oder mit dem Tool „ts2kt“ zu generieren. Es kann dabei aber vorkommen, dass der resultierende Kotlin-Code nicht ganz korrekt erzeugt wird und dadurch manuelle Optimierungen notwendig sind, was wiederum Mehraufwand bedeutet.

Wer allerdings erwartet das Java-Bibliotheken, bei der Kompilierung nach JavaScript, verwendet werden können, wird leider enttäuscht. Im Forumsbeitrag [Zei] wird das durch Dmitry Jemerov, einem JetBrains-Programmierer, bestätigt. Nach seiner Aussage hin ist die Konvertierung von Java- nach Kotlin-Code kein 100%iger automatisierter Prozess. Auf dieses Problem bin ich bei der Umsetzung des Schach-Clients ebenfalls gelegentlich gestoßen, da viele Kotlin-Bibliotheken Funktionen aus der Java-API verwenden und so nicht eingesetzt werden konnten.

Schlussendlich lässt sich sagen, dass es derzeit noch ein paar Stolpersteine gibt, wofür aber bereits teilweise Lösungen bzw. Lösungsansätze existieren. Meiner Meinung nach hat Kotlin Potenzial und es lohnt sich die Entwicklung im Auge zu behalten. Derzeit würde ich von der Verwendung in kleinen Projekten abraten, da sich dafür der etwas größere Konfigurationsaufwand nicht lohnt. Das betrifft beispielsweise die Konfiguration des Build-Tools. Für größere Projekte hingegen, welche von der Objekt-Orientierung und Typisierung profitieren können, lohnt sich gegebenenfalls ein Blick auf Kotlin.

1 Dieser Standard wird durch so gut wie alle gängigen Browser fast zu 100% unterstützt. (siehe https://caniuse.com/#search=es6)

KAPITEL 9

Ausblick

Dieses Kapitel soll dazu dienen mögliche Ausblicke für fortführende wissenschaftliche Arbeiten zu präsentieren. Des Weiteren werden Ideen zusammengetragen wie die derzeitige Version der Implementierung sinnvoll erweitert bzw. verbessert werden kann.

9.1 Ideen zur Erweiterung der Implementierung

Um aus dem schon etwas fortgeschrittenen Prototypen, welcher im Rahmen dieser Arbeit entstanden ist, eine vollwertige und nutzbare Applikation zumachen, bedarf es noch einiger Features welche implementiert werden müssten.

Ein Punkt dieser Erweiterungen wäre eine vollwertige Nutzerverwaltung. Dafür müsste dem Nutzer in erster Linie eine Möglichkeit zur Anmeldung bereitgestellt werden. Nach einer erfolgreichen Authentifizierung sollten dem jeweiligen Nutzer nur noch seine eigenen Matches angezeigt werden. Um anschließend auch gegen andere Player spielen zu können, müsste der Nutzer die Möglichkeit haben einen anderen einzuladen. Erst nach einer Annahme der Einladung sollte das Match gestartet werden können. Damit sich ein Nutzer überhaupt erst anmelden kann, müsste dieser sich vorerst über ein Formular registrieren. Die derzeitige Übersicht der Player würde durch eine solche Implementation überflüssig werden.

Mit einer Nutzerverwaltung gehen natürlich auch Sicherheitsrisiken einher, weshalb ein Authentifizierungsverfahren verwendet werden sollte. Zur Auswahl stehen dabei beispielsweise die „basic access authentication“, welche bereits durch den HTTP-Standard mitgeliefert wird oder aber das offene Protokoll Open Authorization (OAuth).

Als letzte Idee zur Erweiterung, bzw. ist es in diesem Fall eine Verbesserung der Implementierung, wäre die Verwendung eines Multiplattform Projektes [sroh]. Dieses Feature wurde mit der Version 1.2 von Kotlin veröffentlicht, befindet sich aber derzeit noch im experimentellem Stadium. Mithilfe dieses Features ist es möglich ein und denselben Kotlin-Code nach Java und JavaScript zu kompilieren. Damit wäre eine deutliche Ersparnis des zu schreibenden Quellcodes möglich. Da

sich im gezeigten Beispiel viele Funktionen doppeln und auch die Klassenmodelle weitestgehend übereinstimmen, würde sich eine Verwendung dieses Features auf jeden Fall lohnen.

9.2 Genauere Analyse des Konzeptes HATEOAS

Wie bereits im Kapitel 4.3 erwähnt, gibt es noch weitere Verfahren um dieses Design-Konzept umzusetzen. Daher könnte sich eine fortführende wissenschaftliche Arbeit mit der Analyse dieser Varianten befassen. Dabei könnten die jeweiligen Vor- und Nachteile verglichen und gegebenenfalls ergründet werden für welchen Einsatzzweck welches Verfahren am besten geeignet ist.

9.3 Lastverteilung von REST-APIs

Im Kapitel 2.1.5 wurde erwähnt das sich REST-APIs, durch ihre statuslose Kommunikation, besonders gut für eine Lastverteilung auf mehrere Instanzen eignen. Interessant könnte hierbei eine Analyse der Möglichkeiten zur Umsetzung dieser Verteilung sein. Beispielsweise wäre da ein Aufbau eines Clusters mithilfe von Docker Swarm oder Kubernetes möglich. Da sich aber diese beiden Technologien auf Container-Anwendungen beziehen, wäre eine weitere Lösung ohne die Verwendung von Container denkbar.

9.4 Weitere Vergleichsmöglichkeiten

Kapitel 3 zeigte bereits GWT als Alternative zu Kotlin, aber es gibt durchaus noch eine ganze Reihe weiterer Optionen welche näher betrachtet werden könnten. Zum einen wäre da die Möglichkeit mithilfe von Node.js [Fou] auch mit JavaScript Serveranwendungen zu programmieren. Zum anderen gibt es noch alternativ die Programmiersprache Dart [Incb], welche von Google entwickelt wird. Mit dieser Sprache sollen sich Client-, Server- und Mobil-Anwendungen erstellen lassen. Da sich die genannten Alternativen ebenso wie Kotlin für die Programmierung von Server-Client-Anwendungen verwenden lassen, stellen diese genau wie GWT einen sinnvollen Vergleich dar.

Eine weitere Programmiersprache, welche sich als Vergleichspartner anbietet, ist TypeScript [Cora]. Diese erfreut sich laut den Indizes [BV], [OGr18] und [Car18] ebenfalls sehr großer Beliebtheit. Sie liegt dabei in zwei von drei Fällen vor Kotlin. Anders als Kotlin ist sie aber für die clientseitige Programmierung ausgelegt, da im Fall von TypeScript nach JavaScript kompiliert wird. Allerdings ist dabei anzumerken, dass auch an dieser Stelle eine Verwendung von Node.js zur Erstellung von Server-Anwendungen benutzt werden kann.

Literaturverzeichnis

[ANO18] ANONYM: *Fasterxml*. 27. März 2018. URL: https://github.com/FasterXML/jackson-core (besucht am 19. 05. 2018) (siehe S. 43).

[ANOa] ANONYM: *Require.js. A JavaScript Module Loader*. URL: http://requirejs.org/ (besucht am 13. 04. 2018) (siehe S. 51).

[ANOb] ANONYM: *Why AMD?* URL: http://requirejs.org/docs/whyamd.html (besucht am 13. 04. 2018) (siehe S. xi, 52).

[ARD17a] ARD/ZDF-MEDIENKOMMISSION: „ARD/ZDF-Onlinestudie 2017: Neun von zehn Deutschen sind online. Bewegtbild insgesamt stagniert, während Streamingdienste zunehmen - im Vergleich zu klassischem Fernsehen jedoch eine geringe Rolle spielen." *Media Perspektiven* (Sep. 2017), Bd. URL: http://www.ard-zdf-onlinestudie.de/ardzdf-onlinestudie-2017/ (besucht am 22. 03. 2018) (siehe S. 1).

[ARD17b] ARD/ZDF-MEDIENKOMMISSION: „Kern-Ergebnisse der ARD/ZDF-Onlinestudie 2017". *Media Perspektiven* (Sep. 2017), Bd. URL: http://www.ard-zdf-onlinestudie.de/files/2017/Artikel/Kern-Ergebnisse_ARDZDF-Onlinestudie_2017.pdf (besucht am 26. 03. 2018) (siehe S. 1).

[Bra17] BRAUN, HERBERT: „Modul.js. Formate und Werkzeuge für JavaScript-Module". *c't Heft 3/2017* (2017), Bd.: S. 128–133 (siehe S. 52).

[Bre16] BRESLAV, ANDREY: *Kotlin 1.0 Released: Pragmatic Language for JVM and Android*. 15. Feb. 2016. URL: https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/ (besucht am 14. 06. 2018) (siehe S. 22).

[BV] BV, TIOBE SOFTWARE: *TIOBE Index for May 2018*. URL: https://www.tiobe.com/tiobe-index/ (besucht am 08. 06. 2018) (siehe S. 23, 62).

[Byo03] BYOUS, JON: *Java Technology: The Early Years*. Apr. 2003. URL: https://web.archive.org/web/20100105045840/http://java.sun.com/features/1998/05/birthday.html (besucht am 14. 06. 2018) (siehe S. 23).

[Car18] CARBONNELLE, PIERRE: *PYPL PopularitY of Programming Language*. Juni 2018. URL: http://pypl.github.io/PYPL.html (besucht am 08. 06. 2018) (siehe S. 23, 62).

[Cora] CORPORATION, MICROSOFT: *TypeScript*. URL: https://www.typescriptlang.org/ (besucht am 27. 06. 2018) (siehe S. 62).

[Corb] CORPORATION, ORACLE: *GNU General Public License, version 2, with the Classpath Exception*. URL: http://openjdk.java.net/legal/gplv2+ce.html (besucht am 13. 06. 2018) (siehe S. 22).

[Corc] CORPORATION, ORACLE: *JDK 10 Documentation*. URL: https://docs.oracle.com/javase/10/ (besucht am 13. 06. 2018) (siehe S. 21).

[Cos17] COSMINA, JULIANA u. a.: *Pro Spring 5. An In-Depth Guide to the Spring Framework and Its Tools*. English. 5th. Apress, 11. Nov. 2017 (siehe S. 39).

[Dew17] DEWANTO, DR. LOFI: *GWTcon 2017 in Florenz - GWT 3 Reloaded*. 13. Okt. 2017. URL: https://www.heise.de/developer/artikel/GWTcon-2017-in-Florenz-GWT-3-Reloaded-3853937.html (besucht am 11. 06. 2018) (siehe S. 18).

[Elia] ELIZAROV, ROMAN u. a.: *Guide to kotlinx.coroutines by example*. URL: https://github.com/Kotlin/kotlinx.coroutines (besucht am 06. 06. 2018) (siehe S. 54).

[Elib] ELIZAROV, ROMAN u. a.: *Guide to kotlinx.coroutines by example*. URL: https://github.com/Kotlin/kotlinx.coroutines/blob/master/coroutines-guide.md (besucht am 06. 06. 2018) (siehe S. 54).

[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).

[Fie08] FIELDING, ROY THOMAS: *REST APIs must be hypertext-driven*. 20. Okt. 2008. URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (besucht am 14. 05. 2018) (siehe S. 3, 9).

[Fie] FIELDING, ROY THOMAS u. a.: *Hypertext Transfer Protocol – HTTP/1.1*. URL: https://tools.ietf.org/html/rfc2616 (besucht am 12. 05. 2018) (siehe S. 8).

[Fou] FOUNDATION, NODE.JS: *Node.js*. URL: https://nodejs.org/en/ (besucht am 27. 06. 2018) (siehe S. 62).

[Gos15] GOSLING, JAMES u. a.: *The Java® Language Specification. Java SE 8 Edition*. Version 8. Feb. 2015. URL: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf (besucht am 13. 06. 2018) (siehe S. 19).

[Har] HARIRI, HADI u. a.: *Creating a RESTful Web Service with Spring Boot*. URL: https://kotlinlang.org/docs/tutorials/spring-boot-restful.html (besucht am 04. 04. 2018) (siehe S. 41).

[Hip] HIPP, WYRICK & COMPANY INC.: *About SQLite*. URL: https://www.sqlite.org/about.html (besucht am 09. 04. 2018) (siehe S. 41).

[Hoa09] HOARE, TONY: *Null References: The Billion Dollar Mistake*. 25. Aug. 2009. URL: https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare (besucht am 22. 06. 2018) (siehe S. 59).

[Inca] INC., GOOGLE: *Community*. URL: http://www.gwtproject.org/community.html (besucht am 13. 06. 2018) (siehe S. 21).

[Incb] INC., GOOGLE: *Dart programming language | Dart*. URL: https://www.dartlang.org/ (besucht am 27. 06. 2018) (siehe S. 62).

[Incc] INC., GOOGLE: *Developer's Guide*. URL: http://www.gwtproject.org/doc/latest/DevGuide.html (besucht am 13. 06. 2018) (siehe S. 21).

[Incd] INC., GOOGLE: *Getting Started*. URL: http://www.gwtproject.org/gettingstarted.html (besucht am 12. 06. 2018) (siehe S. 19, 20).

[Ince] INC., GOOGLE: *JsInterop*. URL: http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJsInterop.html (besucht am 12. 06. 2018) (siehe S. 19).

[Incf] INC., GOOGLE: *JSNI*. URL: http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJSNI.html (besucht am 12. 06. 2018) (siehe S. 19).

[Incg] INC., GOOGLE: *JSON*. URL: http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsJSON.html (besucht am 13. 06. 2018) (siehe S. 20).

[Inch] INC., GOOGLE: *Terms of Service*. URL: http://www.gwtproject.org/terms.html (besucht am 13. 06. 2018) (siehe S. 21, 22).

[Inc06] INC., GOOGLE: *Versions*. 16. Mai 2006. URL: http://www.gwtproject.org/versions.html (besucht am 14. 06. 2018) (siehe S. 23).

[Inci] INC., GOOGLE: *XML*. URL: http://www.gwtproject.org/doc/latest/DevGuideCodingBasicsXML.html (besucht am 13. 06. 2018) (siehe S. 20).

[Incj] INC., PIVOTAL SOFTWARE: *Building a RESTful Web Service*. URL: https://spring.io/guides/gs/rest-service/ (besucht am 04. 04. 2018) (siehe S. 41).

[Inc17] INC., STACK EXCHANGE: *Developer Survey 2017*. 2017. URL: https://insights.stackoverflow.com/survey/2017#technology (besucht am 26. 03. 2018) (siehe S. 1).

[Jad11] JADHAV, ANIL S. u. a.: „Framework for evaluation and selection of the software packages: A hybrid knowledge based system approach“. *Journal of Systems and Software* (2011), Bd. 84(8): S. 1394–1407. URL: http://www.sciencedirect.com/science/article/pii/S016412121100077X (besucht am 08. 06. 2018) (siehe S. 15, 16).

[Jem] JEMEROV, DMITRY: *Compile Java to Javascript*. URL: https://discuss.kotlinlang.org/t/compile-java-to-javascript/2034 (besucht am 12. 06. 2018) (siehe S. 19).

[Kel] KELLY, MIKE: *JSON Hypertext Application Language*. URL: https://tools.ietf.org/html/draft-kelly-json-hal-08 (besucht am 04. 06. 2018) (siehe S. 32).

[Kre15] KRETZSCHMAR, CHRISTOPH: „Demonstration eines RESTful Webservices am Beispiel eines Schachservers“. Bachelor. Hochschule für Technik und Wirtschaft Dresden, 2015 (siehe S. 13, 14, 32).

[Kri11] KRILL, PAUL: *JetBrains readies JVM-based language. The Kotlin language will be statically typed and free of legacy troubles, according to the company*. 22. Juli 2011. URL: https://www.infoworld.com/article/2622405/java/jetbrains-readies-jvm-based-language.html (besucht am 14. 06. 2018) (siehe S. 22).

[Led] LEDBETTER, LETTY: *Oracle and IBM Collaborate to Accelerate Java Innovation Through OpenJDK*. URL: http://www.oracle.com/us/corporate/press/176988 (besucht am 13. 06. 2018) (siehe S. 21).

[Lim] LIMITED, TUTORIALS POINT INDIA PRIVATE: *SQLite - Java*. URL: https://www.tutorialspoint.com/sqlite/sqlite_java.htm (besucht am 09. 04. 2018) (siehe S. 41).

[Los08] LOSSA, GÜNTER: *Schach lernen. Ein Leitfaden für Anfänger des königlichen Spiels; Der entscheidene Zug zum zwingenden Mattangriff*. German. Joachim Beyer Verlag, 2008 (siehe S. 13).

[Masa] MASHKOV, SERGEY: *DOM trees*. URL: https://github.com/Kotlin/kotlinx.html/wiki/DOM-trees (besucht am 13. 04. 2018) (siehe S. xi, 53).

[Masb] MASHKOV, SERGEY: *kotlinx.html*. URL: https://github.com/Kotlin/kotlinx.html (besucht am 06. 06. 2018) (siehe S. 52).

[OGr18] O'GRADY, STEPHEN: *The RedMonk Programming Language Rankings: January 2018*. 7. März 2018. URL: http://redmonk.com/sogrady/2018/03/07/language-rankings-1-18 (besucht am 08. 06. 2018) (siehe S. 23, 62).

[sroa] S.R.O., JETBRAINS: *Calling JavaScript from Kotlin*. URL: https://kotlinlang.org/docs/reference/js-interop.html (besucht am 12. 06. 2018) (siehe S. 19).

[srob] S.R.O., JETBRAINS: *Comparison to Java Programming Language*. URL: https://kotlinlang.org/docs/reference/comparison-to-java.html (besucht am 22. 06. 2018) (siehe S. 59).

[sroc] S.R.O., JETBRAINS: *FAQ: Is Kotlin an object-oriented language or a functional one?* URL: https://kotlinlang.org/docs/reference/faq.html#is-kotlin-an-object-oriented-language-or-a-functional-one (besucht am 13. 06. 2018) (siehe S. 19).

[srod] S.R.O., JETBRAINS: *Is Kotlin free?* URL: https://kotlinlang.org/docs/reference/faq.html#is-kotlin-free (besucht am 13. 06. 2018) (siehe S. 21).

[sroe] S.R.O., JETBRAINS: *JavaScript Modules*. URL: https://kotlinlang.org/docs/reference/js-modules.html (besucht am 11. 06. 2018) (siehe S. 18).

[srof] S.R.O., JETBRAINS: *Kotlin JavaScript Overview*. URL: https://kotlinlang.org/docs/reference/js-overview.html (besucht am 11. 06. 2018) (siehe S. 18).

[srog] S.R.O., JETBRAINS: *Kotlin to JavaScript*. URL: https://kotlinlang.org/docs/tutorials/javascript/kotlin-to-javascript/kotlin-to-javascript.html (besucht am 12. 06. 2018) (siehe S. 19).

[sroh] S.R.O., JETBRAINS: *Multiplatform Projects*. URL: https://kotlinlang.org/docs/reference/multiplatform.html (besucht am 27. 06. 2018) (siehe S. 61).

[sroi] S.R.O., JETBRAINS: *Null Safety*. URL: https://kotlinlang.org/docs/reference/null-safety.html (besucht am 22. 06. 2018) (siehe S. 59).

[sroj] S.R.O., JETBRAINS: *Reference*. URL: https://kotlinlang.org/docs/reference/ (besucht am 13. 06. 2018) (siehe S. 20, 21).

[srok] S.R.O., JETBRAINS: *We're all Kotlin*. URL: https://kotlinlang.org/community/ (besucht am 13. 06. 2018) (siehe S. 21).

[srol] S.R.O., JETBRAINS: *Who develops Kotlin*. URL: https://kotlinlang.org/docs/reference/faq.html#who-develops-kotlin (besucht am 13. 06. 2018) (siehe S. 21).

[srom] S.R.O., JETBRAINS: *Working with Build Tools*. URL: https://kotlinlang.org/docs/tutorials/build-tools.html (besucht am 12. 06. 2018) (siehe S. 19).

[Sha17] SHAFIROV, MAXIM: *Kotlin on Android. Now official*. Mai 2017. URL: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/ (besucht am 26. 03. 2018) (siehe S. 1).

[Spi16] SPICHALE, KAI: *API-Design. Praxishandbuch für Java- und Webservice-Entwickler*. German. 1st. dpunkt.verlag GmbH, Dez. 2016 (siehe S. 3–5, 9, 33).

[SRL16] SRL., BAELDUNG: *Spring MVC Content Negotiation*. 20. Aug. 2016. URL: http://www.baeldung.com/spring-mvc-content-negotiation-json-xml (siehe S. 45).

[Staa] STARTSEV, LEONID u. a.: *Kotlin cross-platform / multi-format reflectionless serialization*. URL: https://github.com/Kotlin/kotlinx.serialization (besucht am 05. 06. 2018) (siehe S. 20, 53).

[Stab] STARTSEV, LEONID u. a.: *Quick Example*. URL: https://github.com/Kotlin/kotlinx.serialization#quick-example (besucht am 05. 06. 2018) (siehe S. xi, 54).

[Var15] VARANASI, BALAJI u. a.: *Introducing Gradle*. English. 1st. Apress, 23. Dez. 2015 (siehe S. 10).

[Wat] WATSON, GRAY: *OrmLite - Lightweight Object Relational Mapping (ORM) Java Package*. URL: http://ormlite.com/ (besucht am 10. 04. 2018) (siehe S. xi, 42, 43).

[Zei] ZEINDL, FABIAN u. a.: *Compile Java to Javascript*. URL: https://discuss.kotlinlang.org/t/compile-java-to-javascript/2034 (besucht am 22. 06. 2018) (siehe S. 60).

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Dresden, den 19. Juli 2018 Felix Dimmel

A Praktikumsaufgabe

Mithilfe dieses Praktikums soll Ihr Verständnis über Webservices, insbesondere solche welche den REST Architekturstil umsetzen, schulen. Des Weiteren zielt dieses Praktikum auf eine Einführung in das Kommandozeilen Tool cURL ab, welches ein mächtiges Werkzeug für die Kommunikation mit Webservices darstellt.

Die nachfolgenden Aufgaben sind chronologisch angeordnet und sollten daher in der Vorgegebenen Reihenfolge bearbeitet werden. Die ersten vier Punkte dienen dabei als Erläuterung der Einrichtung und interaktiven Einführung in der Benutzung des Webservices.

A.1 Einrichtung

Laden Sie sich das Repository `git@github.com:GagaMen/chessgame.git` mithilfe des Tools git herunter und wechseln Sie zum Tag „v1.0.0“. Alternativ können Sie das Projekt auch gezippt, direkt über die URL `https://github.com/GagaMen/chessgame/archive/master.zip`, herunterladen und anschließend entpacken. Haben Sie den richtigen Tag ausgecheckt, können Sie das Projekt mit dem mitgelieferten Gradle-Wrapper erstellen. Nach erfolgreicher Erstellung kann die Anwendung mit Docker ausgeführt werden. Dafür müssen Sie allerdings vorerst die Docker-Services erstellen lassen. Führen Sie für diese Schritte folgende Befehlsabfolge in einem Terminal aus:

```
1 git clone git@github.com:GagaMen/chessgame.git
2 cd chessgame
3 git checkout v1.0.0
4 ./gradlew build
5 docker-compose build
6 docker-compose up -d
```

Listing A.1: Befehlsabfolge zur Einrichtung des Webservices

Möchten Sie den Server herunterfahren, so können Sie dies über den Befehl `docker-compose down` erreichen. Für einen erneuten Start, genügt anschließend der letzte Befehl aus dem Listing A.1.

A.2 Schachregeln und Schachnotationen

1. Sofern Sie die Spielregeln von Schach noch nicht kennen sollten, schauen Sie sich zuallererst diese an. Werfen Sie dafür einen Blick auf die Seite http://www.schachtrainer.de/learn/fide01.php.
2. Machen Sie sich mit der Forsyth-Edwards-Notation (FEN) vertraut. Benutzen Sie dafür folgende URL https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation.
3. Untersuchen Sie als nächstes die Standard Algebraic Notation (SAN) unter folgender URL https://en.wikipedia.org/wiki/Algebraic_notation_%28chess%29

A.3 Web-Applikation

1. Rufen Sie die Web-Applikation, nachdem Sie diese mittels Docker gestartet haben, unter der URL http://localhost:8080 auf.[1]
2. Wechseln Sie zur Player-Ansicht(Players) und legen sich über das Formular einen eigenen Player an. Gehen Sie anschließend über den Return-Button zurück zur Start-Ansicht.[2]
3. Wechseln Sie nun zur Match-Ansicht(Matches) und legen ein Match mit dem AI-Player und Ihrem an.
4. Starten Sie das eben angelegte Match, über den ▶-Button in der Übersichtstabelle.
5. Spielen Sie nun ein paar Züge, indem Sie Ihre Figuren per Drag&Drop bewegen. Mögliche Züge werden dabei per Mouseover angezeigt.

A.4 Verwendung der REST-API mithilfe des Browsers

1. Lassen Sie sich alle angelegten Player mithilfe der REST-API anzeigen. Rufen Sie dafür die URL http://localhost:8080/api/players auf.
2. In den meisten Browsern wird standardmäßig, im Accept-Header des HTTP-Requests, Extensible Markup Language (XML) als Rückgabeformat angefordert. Mithilfe der Technik Content Negotiation ist es möglich sich Daten auch in der JavaScript Object Notation (JSON) ausgeben zu lassen. Fügen Sie dafür den Suffix „.json“ oder den Parameter „?mediaType=json“ an die URL an.[3]

1 Beachten Sie dabei, dass es ein paar Sekunden dauern kann bis der Server komplett hochgefahren ist, da der Docker-Befehl nicht darauf wartet.
2 Beachten Sie hierbei, dass die Browser-Funktion zum zurückkehren zur letzten Seite nicht funktioniert.
3 Für ein besseres Verständnis dieser Technik, schauen Sie auch in die README-Datei des Github Repositorys unter https://github.com/GagaMen/chessgame#content-negotiation.

3. Machen Sie sich mit den bereitgestellten Ressourcen der REST-API vertraut und fordern Sie weitere Daten an. Schauen Sie hierfür in der README-Datei unter https://github.com/GagaMen/chessgame#entry-points nach, welche URLs für einen GET-Request bereitstehen. Die anderen Request-Arten können vorerst vernachlässigt werden.

A.5 Verwendung der REST-API mithilfe des Tools cURL

Das Tool cUrl ist ein Kommandozeilenprogramm und steht für „Client for URLs“. Es dient zur Übertragung von Daten in Rechnernetzen und unterstützt dabei eine ganze Reihe von Übertragungsprotokollen. Darunter vertreten sind zum Beispiel HTTP, HTTPS, FTP oder auch FTPS. Nachfolgend soll dieses Tool dazu genutzt werden mit der bereitgestellten REST-API zu kommunizieren und so Ressourcen abzurufen, zu erstellen, zu aktualisieren oder zu löschen. Verwenden Sie daher für die Erfüllung der nachfolgenden Aufgaben ausschließlich das Tool cURL.

1. Lernen Sie das Tool Curl kennen und achten Sie dabei besonders auf die Parameter `-d`, `-v`, `-H` und `-X`, welche für die Erfüllung der nachfolgenden Aufgaben benötigt werden.
2. Wie Sie schon im vorhergehenden Aufgabenblock gesehen haben, gibt es unterschiedliche HTTP-Methoden, welche für den Aufruf einer URI benutzt werden können. Machen Sie sich mit diesen Vertraut und lernen Sie deren Bedeutung kennen bzw. für welchen Zweck diese gedacht sind.[1]
3. Wegen des Designkonzeptes HATEOAS, welches für REST-APIs, zwingend umgesetzt werden muss, stellt jeder Response eines Requestes Links zur Navigation bereit. Im vorliegenden Beispiel befinden sich diese Links im Link-Header des Response. Einzelne Links werden dabei durch Semikolons getrennt und bestehen aus der URI in spitzen Klammern, der Relation und dem HTTP-Verb, welche die HTTP-Methode darstellt. Analysieren Sie die API weiter, indem Sie beginnend mit der http://localhost:8080/api und den gelieferten Link-Headern durch diese navigieren.[2]
4. Legen Sie weitere neue Player an.[3] Schicken Sie zunächst die benötigten Daten im Format `application/x-www-form-urlencoded` und anschließend im Format `application/json`. Benutzen Sie dafür den HTTP-Header „Content-Type“.
5. Aktualisieren Sie das Passwort eines Players.[3]
6. Löschen Sie ein beliebigen zuvor angelegten Player.

1 Schauen Sie hierfür unter https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods nach.
2 Wollen Sie sich ausschließlich die empfangenen Header anzeigen, so können Sie folgenden Befehl benutzen: „curl -sSL -D - localhost:8080/api -o /dev/null“
3 Schauen Sie für weitere Informationen zu den Parametern in der README unter https://github.com/GagaMen/chessgame#entry-points nach.

7. Legen Sie ein weiteres Match an. Lassen Sie sich anschließend alle Matches im XML-Format anzeigen.[3] Nutzen Sie dafür den Accept-Header des HTTP-Requestes.
8. Verfestigen Sie Ihre Kenntnisse über die Schachnotationen, indem Sie mehrere Züge dem zuvor angelegten Match hinzufügen.[3] Schauen Sie sich dafür nach jedem hinzugefügten Zug das Resultat im Match an.

B Lösung der Praktikumsaufgabe