André Schütz & Jens Grassel

Come out and Play

Webanwendungen mit Scala, Akka,
Scala.js und dem Play Framework

Come out and Play

Webanwendungen schreiben mit Scala, Scala.js, Akka und dem Play Framework.

Jens Grassel und Andre Schütz

Dieses Buch wird verkauft unter http://leanpub.com/comeoutandplay

Diese Version wurde veröffentlicht am 2019-03-19

Twittere dieses Buch!

Bitte unterstütze Jens Grassel und Andre Schütz, indem du dieses Buch auf Twitter weiterempfiehlst!

Vorschlag: Verwende den folgenden Hashtag, wenn du über dieses Buch twitterst: #comeoutandplay.

Was sagen andere über dieses Buch? Klicke hier, um nach diesem Hashtag auf Twitter zu suchen:

#comeoutandplay

Inhaltsverzeichnis

1. Vorwort

Die webbasierte Implementierung von Anwendungen hat sich durchgesetzt, um den Anwendern einen möglichst einfachen und der Zeit entsprechenden Zugang zu ermöglichen.

Für die Umsetzung kann man unter diversen Technologien und Herangehensweisen wählen, welche sich je nach Interesse der Entwickler oder durch Vorgaben der Auftraggeber ergeben. Eine Vorgehensweise ist die *Funktionale Programmierung*, welche in den letzten Jahren einen immer stärker werdenden Zulauf erfährt, auch wenn die Grundlagen für diese schon seit Jahrzehnten bestehen.

Ein häufiger Werdegang ist das Erlernen einer imperativen, prozeduralen oder objektorientierten Programmiersprache während der wissenschaftlichen oder beruflichen Ausbildung. Der Einstieg in die *Funktionale Programmierung* erfolgt dann zu einem späteren Zeitpunkt über Sprachen wie Clojure[1], Haskell[2], Lisp[3] oder Scala[4].

Die Autoren dieses Buches kamen teilweise auch über diesen Weg zur *Funktionalen Programmierung*, wobei beide auch schon in früheren Jahren diverse Berührungspunkte mit diesem Bereich hatten.
Den vollständigen Einstieg und Umstieg in den Bereich der *Funktionalen Programmierung* mittels Scala vollzogen sie in Zusammenhang mit einem Forschungs- und Entwicklungsprojekt, welches sich mit dem Gebiet der Datenintegration und -migration beschäftigte. Die *Funktionale Programmierung* bietet eine besonders gute Grundlage für das Verarbeiten, Modifizieren und Integrieren von unterschiedlichen Datenstrukturen.

Neben der rein funktionalen Programmierung, werden weitere Technologien in diesem Buch verwendet, die in unterschiedlichen Projekten eingesetzt wurden. Dazu zählen u.a. das *Akka Toolkit*, das *Play Framework* und *Scala.js*.

[1] Ein häufiger Grund ist beispielsweise, daß das Modell in einer API zur Verfügung gestellt wird. Eine einfache Änderung an Attributen würde so z.B. zu einem Bruch der API führen.

[2] https://www.haskell.org/

[3] https://de.wikipedia.org/wiki/Lisp

[4] https://www.scala-lang.org/

1.1 Einleitung

Webanwendungen haben sich in den letzten 20 Jahren von einem Nischendasein hin zu ernstzunehmenden Konkurrenten bzw. Ergänzungen von klassischen Anwendungen entwickelt.
Da sich die damit einhergehenden Technologien ebenfalls weiterentwickelt haben, eröffnen sich dementsprechend weitere Möglichkeiten.

Die Anzahl der Programmiersprachen, Basistechnologien und Frameworks, die in diesem Umfeld angeboten werden sind mehr als zahlreich und entsprechend schwierig ist es sich einen Überblick zu verschaffen bzw. geeignete Werkzeuge auszuwählen.

Wir haben im Laufe unserer Arbeit viele verschiedene Frameworks (inklusive Programmiersprachen) verwendet und haben uns schlußendlich in diesem Buch dazu entschieden das Play Framework zu nutzen.

Die Gründe hierfür sind vielfältig und teilweise subjektiv, da persönliche Vorlieben unsere Entscheidungen immer beeinflussen, schließlich wollen wir an dem, was wir tun, auch Freude haben. ;-)

Darüber hinaus basiert das Play Framework seinerseits auf soliden Basistechnologien wie Akka, Netty und Scala. Der Code der Anwendung wird compiliert und läuft auf der Java Virtual Machine (JVM).

1.2 Aufbau des Buches

Wir entwickeln im Verlauf dieses Buchs eine Beispielanwendung, was uns die Arbeit an konkreten und praxisnahen Problemstellungen erlaubt.

In *Teil 1* werden grundlegende Erklärungen zum Einrichten der Entwicklungsumgebung und Werkzeuge sowie eine kurze Einführung in Scala gegeben. Leser, denen diese Themen bereits vertraut sind, können diesen Teil somit getrost überspringen.

Die verwendeten Technologien werden in *Teil 2* beschrieben, um den Einstieg in die Beispielanwendung zu erleichtern. Dazu gehören das *Play-Framework*, *Akka* und *Scala.js*. Wiederum können diejenigen Leser, welche mit den Technologien bereits vertraut sind, diesen Teil überspringen und zu *Teil 3* voranschreiten.

Teil 3 dient dazu, das Szenario für eine Anwendung zu definieren, die im Verlauf dieses Buches entwickelt werden soll.

Eine grundlegende Komponente ist das *Frontend*, welches die Interaktion des Nutzers mit der Beispielanwendung ermöglicht und ihm Zugriff auf die Funktionalitäten gibt. Die Erstellung des Frontend auf Basis des *Play-Framework*, die Integration eines Authentifikations-Frameworks, die Implementierung der Anmeldung des Nutzers am System, das Durchsuchen und Verwalten von Freundeslisten und die Auswahl und Ausführung eines Spiels werden in *Teil 4* chronologisch erarbeitet. Darüber hinaus wurde eine Migrationsanleitung aufgenommen, in der die gesamte Beispielanwendung plus das integrierte Authentifikations-Framework auf Play 2.6 aktualisiert worden ist (Während der Arbeit an dem Buch gab es die Veröffentlichung von Version 2.6, so daß eine Migration von Play 2.5 und die damit verbundenen Erkenntnisse aufbereitet wurden).

In *Teil 5* wird das Spiel und dessen Umsetzung sowie Integration in das Frontend beschrieben, was die Lücke zwischen diesen beiden Komponenten schließt. Für die Ausführung der Beispielanwendung und das in diesem Zusammenhang notwendige Deployment wird eine Anleitung in *Teil 6* zusammengefaßt.

Zum Abschluß dieses Buches noch eine kurze Anmerkung hinsichtlich “Dokumentation im Code” und warum diese gewissenhaft und von Anfang an durchgeführt werden sollte.

1.2.1 Konventionen für den Quelltext

Wir folgen weitestgehend dem offiziellen Scala Style Guide. Ein Quelltext sieht beispielsweise wie folgt aus:

Beispiel für einen Quelltext

```
/**
  * Ein Kommentar...
  */
object Foo {
  def someFunction(param: Int): String = {
    /*
     * Noch ein Kommentar...
     */
    val someMagicValue = ??? // Kommentar...
    // Und wieder ein Kommentar
    val einEtwasLaengererBezeichner = someMagicValue.foldLeft(0)(_ + _) * outOfNowhereVal\
ue + 314
    ???
  }
}
```

Je nach Medium werden Zeilen im Quelltext unterschiedlich umgebrochen. In einer PDF-Datei können diese relativ lang sein (ca. 100 Zeichen). In einem E-Book dagegen werden sie meist schon nach 45-50 Zeichen umgebrochen, bei größeren Schriftarten (je nach Einstellung des Lesegerätes) auch deutlich früher. Dies erschwert es, größere Quelltextblöcke vorzuformatieren, aber wir bemühen uns um möglichst gute Lesbarkeit.

Es kann vorkommen, daß Zeilen im Quelltext, die sehr lang sind, mit einem Backslash (\) "zerteilt" umgebrochen werden. Wir werden uns bemühen den Code so vorzuformatieren, daß dies möglichst selten auftritt.

1.3 Schutzmarken und Copyrights

Folgende Technologien werden in diesem Buch genutzt. (Alphabetische Auflistung)

Technologie	Rechteinhaber
Akka	Lightbend
Java	Oracle Corporation
Java Virtual Machine (JVM)	Oracle Corporation
JavaScript	Oracle Corporation
Netty	The Netty Project
Play Framework	Lightbend
Sbt	Lightbend
Scala	EPFL - École polytechnique fédérale de Lausanne
Scala.js	EPFL - École polytechnique fédérale de Lausanne

1.4 Logos und Bilder

Wir bedanken uns recht herzlich für die Genehmigung zur Nutzung der folgenden Logos.

Logo	Eigentümer
Akka	Lightbend in Respekt der Trademark Policy
Play Framework	Lightbend in Respekt der Trademark Policy
Scala	Scala Center and Scala logo courtesy of EPFL, Switzerland
Scala.js	Scala Center and Scala logo courtesy of EPFL, Switzerland

Das Titelbild wurde für dieses Buch von André Schütz erstellt. Alle weiteren Abbildungen innerhalb dieses Buches wurden durch die Autoren angefertigt.

1.5 Quelltext zum Buch

Der Quelltext zu der in diesem Buch erstellten Anwendung und die jeweils in den Kapiteln angelegten Tags können im folgenden Repository eingesehen werden:
https://gitlab.com/comeoutandplay

Einrichtung der Werkzeuge und Einführung in Scala

Dieses Kapitel beschreibt grundlegende Arbeitsweisen und Werkzeuge und kann dementsprechend übersprungen werden, wenn diese Kenntnisse bereits vorliegen.
Wir beginnen mit einem kurzen Exkurs durch das Ökosystem der Programmiersprache Scala hinsichtlich der zur Verfügung stehenden Werkzeuge. Diese umfassen Java, SBT und diverse Entwicklungsumgebungen.

Danach erfolgt eine kurze Einführung in die Programmierung in Scala, welche einen grundlegenden Überblick über die Möglichkeiten und Vorgehensweisen in dieser Programmiersprache geben soll. Dazu zählen u.a. die interaktive Programmierung mit einer REPL, Hinweise zu vorhandenen Datenstrukturen, ein Schnelleinstieg in die funktionale Programmierung, Hilfsmittel, welche man zur Unterstützung nutzen kann und ein Überblick, wie man “Boilerplate” Code reduziert.

2. Werkzeuge

In der praktischen Anwendung geht letztlich nichts ohne die entsprechenden Werkzeuge. Daher folgt hier ein kurzer Exkurs zu den benötigten Hilfsmitteln.

2.1 Java

Da Scala eine Programmiersprache ist, die in der *Java Virtual Machine* (JVM) läuft und auch das Play-Framework und sonstige Werkzeuge Java benötigen, muß dieses installiert sein. Zum gegenwärtigen Zeitpunkt empfehlen wir die aktuellste Version aus der 1.8'er Reihe zu nutzen. Ob die Version von Oracle oder das OpenJDK zum Einsatz kommen ist fürs Erste nicht weiter von Belang. Wir selbst nutzen vorzugsweise das OpenJDK.

2.2 SBT

Als Werkzeug zur Projektverwaltung hat sich für Scala das "Simple Build Tool", kurz SBT, durchgesetzt. Da die Bezeichnung "Simple" durchaus zu Kontroversen führte, wird es mittlerweile eigentlich nur noch als SBT bezeichnet. ;-)

2.2.1 Installation

Für SBT bestehen ausreichende Pakete für verschiedenste Betriebssysteme. Die Installation desselben sollte also über die für das entsprechende System empfohlenen Kanäle erfolgen.

2.2.2 Nützliches für SBT

Nach derzeitigem Stand kann man SBT global über das Verzeichnis `~/.sbt/VERSION/` konfigurieren. Hierbei steht `VERSION` entweder für `0.13` (für ältere Projekte) oder für `1.0`. Einige Einstellungen, die sehr nützlich sind, sollten in der folgenden Datei abgelegt werden: `~/.sbt/VERSION/global.sbt`

Nützliche globale SBT-Einstellungen

```
// Prevent Strg+C from killing SBT.
cancelable in Global := true
// Use a coloured scala console if possible.
initialize ~= (_ =>
  if (ConsoleLogger.formatEnabled)
    sys.props("scala.color") = "true"
)
```

Die erste Einstellung verhindert, daß beim Beenden einer aus SBT heraus gestarteten Anwendung durch `Strg+C`, SBT mit beendet wird. Die nächste Einstellung sorgt dafür, daß Syntax-Highlighting in der Konsole aktiviert wird, wenn man diese aus SBT heraus startet.

2.2.2.1 Globale Plugins

Im Verzeichnis `~/.sbt/0.13/plugins` bzw. `~/.sbt/1.0/plugins` können globale Plugins eingebunden werden. Zwei sehr nützliche Plugins sind sbt-updates und sbt-dependency-graph. Wer sich mit Ensime beschäftigen möchte, kann dort auch das entsprechende SBT-Plugin installieren.

2.2.2.2 Schnelleres Herunterladen von Abhängigkeiten

Wer mit mehreren Projekten parallel arbeitet, wird recht schnell über die Meldung 'Waiting for /.ivy2/.sbt.ivy.lock to be available' stolpern. Darüber hinaus lädt SBT via Ivy Abhängigkeiten nur einzeln herunter. Eine sch

Achtung! Da einige andere SBT-Plugins ebenfalls Coursier nutzen, kann es zu Problemen kommen, wenn verschiedene Versionen davon im Klassenpfad zu finden sind!

2.2.2.3 Projektplugins

Innerhalb eines Projekts können Plugins im Verzeichnis `project` eingebunden werden. Der Übersichtlichkeit halber sollte man einfach eine Datei `project/plugins.sbt` anlegen, in der man die gewünschten Plugins einbindet. Nützliche Plugins auf Projektebene sind beispielsweise:

1. Scalafmt für automatisiertes Formatieren von Quelltext (Style Guide) oder alternativ Scalariform
2. Wartremover für das Erzwingen von strengeren Regeln hinsichtlich funktionaler Programmierung
3. sbt-git für nützliche Funktionen rund um Git wie z.B. die automatische Ableitung einer Versionsnummer aus Tags

2.2.2.4 SBT-Version fixieren

Die SBT-Version kann für ein Projekt fest eingestellt werden, indem man diese in der Datei `project/build.properties` konfiguriert:

SBT-Version in build.properties einstellen

```
sbt.version=1.2.8
```

2.3 Entwicklungsumgebung

Mittlerweile gibt es einige Entwicklungsumgebungen (IDEs) für Scala wie Scala IDE und IntelliJ Idea. Es ist jedoch auch möglich mit Texteditoren zu arbeiten. Das Projekt

Ensime bietet Plugins für SBT und diverse Editoren an, um fortgeschrittene Funktionalitäten zu ermöglichen. Beliebte Editoren im Umfeld von Scala sind: Emacs, Vim bzw. Neovim, Sublime Text 2 und Atom.

Seit einiger Zeit gewinnt auch das Projekt Metals immer mehr an Fahrt und macht einen sehr vielversprechenden Eindruck. Insbesondere in Kombination mit Visual Studio Code ist es ein sehr schlankes, aber wirkungsvolles Werkzeug. Es gibt auch Integrationen in andere Editoren.
Wer sich Metals ansehen möchte, sollte zudem einen Blick auf Bloop werfen, da diese Projekte Hand in Hand gehen.

Die Nutzung von Texteditoren für größere Projekte ist nur fortgeschrittenen Anwendern zu empfehlen. Für den Einstieg sollte eine IDE gewählt werden. Wir nutzen vorzugsweise IntelliJ Idea, daher bezieht sich die Beschreibung zur Einrichtung der Entwicklungsumgebung darauf. Damit IntelliJ Idea genutzt werden kann, muß das Scala-Plugin installiert werden.

Das Erstellen von Projektstrukturen erfolgt jedoch mittels SBT. Diese Strukturen werden dann in die IDE importiert.

3. Programmierung in Scala

Scala ist eine objektorientierte und ebenso eine funktionale Programmiersprache. Für Ein- und Umsteiger, die Erfahrungen in objektorientierter Programmierung (insbesondere mit Java) haben, bietet sich so die Möglichkeit eines recht einfachen Übergangs. Man kann erstmal “besseres Java” schreiben und ist nicht gezwungen gleich komplett in die funktionale Programmierung einzutauchen. Ein interessantes Feature von Scala ist auch die Möglichkeit Java- und Scalaklassen beliebig zu mischen bzw. Bibliotheken gegenseitig zu nutzen.

Es gibt zahlreiche Literatur zur Programmierung in Scala. Die folgenden Bücher sind unserer Ansicht nach zu empfehlen:

1. Programming in Scala - Martin Odersky, Lex Spoon, Bill Venners
2. Scala for the Impatient - Cay S. Horstmann
3. Functional Programming in Scala - Runar Bjarnason, Paul Chiusano
4. Functional Programming for Mortals - Sam Halliday
5. Scala with Cats - Noel Welsh, Dave Gurnell

“Functional Programming in Scala” ist definitiv kein Einsteigerbuch, aber nichtsdestotrotz ein sehr gutes Buch für jemanden, der sich intensiv mit funktionaler Programmierung auseinandersetzen möchte. Die beiden letztgenannten Bücher bieten einen guten Einstieg mit dem Schwerpunkt jeweils auf Scalaz bzw. Cats als Bibliotheken.

Eine umfassende Einführung in Scala würde den Rahmen dieses Kapitels sprengen, daher werden wir hier nur kurz auf einige im Rahmen des Buches wichtige Aspekte eingehen und verweisen auf die bereits erwähnte Literatur.

3.1 Interaktive Programmierung via REPL

Das Kürzel REPL steht für "read eval print loop" und wertet Nutzereingaben direkt in der verwendeten Programmiersprache aus. Dieses Prinzip der interaktiven Programmierung ist je nach bisherigem Erfahrungshintergrund eventuell bekannt. Sollte dieses Konstrukt Neuland sein, können wir nur empfehlen, sich damit auseinanderzusetzen. Auch wenn wir für das Entwickeln größerer Anwendungen eine IDE nutzen, so bietet eine REPL jedoch den Vorteil schnell Code auszuprobieren und zu verfeinern.

Scala bietet eine Standard-REPL, die einfach via `scala` bzw. `scala -Dscala.color` gestartet werden kann.

Zum Kopieren größerer Quelltextausschnitte ist es hilfreich die Scala-REPL in den Einfügemodus zu schalten. Dies geschieht durch den Befehl `:paste`.

Eine Alternative zur Scala-REPL ist die Ammonite-REPL. Sie bietet einige interessante Erweiterungen zur Standard-REPL, ist aber für das Ausprobieren der Codeschnipseln in diesem Buch nicht zwingend erforderlich.

Des Weiteren besteht die Möglichkeit in IDEs (IntelliJ Idea, Scala IDE) ein sogenanntes "Worksheet" anzulegen. Hierzu muß ein Scalaprojekt erstellt werden, in welchem dann der Worksheet erzeugt werden kann. Diese Möglichkeit bietet alle Annehmlichkeiten einer integrierten Entwicklungsumgebung und weitaus bessere Speichermöglichkeiten als eine REPL.

3.2 Hinweise zu Datenstrukturen (var, val)

Man kann Datenstrukturen in den folgenden Varianten definieren:

Immutable
: Datenstrukturen, die nicht veränderbar sind. Scala stellt eine Reihe von Datentypen bereit, die unter `scala.collection.immutable` zu finden sind. Ein solcher Datentyp kann nach seiner Initialisierung nicht wieder verändert werden. Man kann lediglich eine veränderte Kopie von selbigem erzeugen.

Mutable
: Datenstrukturen, die veränderbar sind. Ein solcher Datentyp ist nach seiner Initialisierung beliebig veränderbar und kann mehrmals zugewiesen werden. Auch hierfür bietet Scala eine Reihe von Datentypen, die analog zu den vorherigen unter `scala.collection.mutable` zu finden sind.

Var Eine Variable, deren Inhalt geändert werden kann. Sie kann einen beliebigen Datentyp (immutable oder mutable) enthalten und kann jederzeit mit einem neuen Wert belegt werden.

Val Eine Variable, deren Inhalt nicht geändert werden kann. Auch sie kann einen beliebigen Datentyp (immutable oder mutable) enthalten, aber nach ihrer Initialisierung nicht wieder mit einem neuen Wert belegt werden.

Hieraus ergeben sich die folgenden Kombinationsmöglichkeiten:

3.2.0.1 Immutable Val

Perfekt

Die optimale Kombination im Hinblick auf Datensicherheit. Man muß sich keine Gedanken darum machen, daß ein Wert, den man weitergibt oder erhalten hat von einer anderen Stelle aus geändert werden könnte.

3.2.0.2 Immutable Var

In Ordnung

Die Nutzung eines unveränderbaren Datentyps in einer Variablen kann sinnvoll sein, wenn diese nur innerhalb eines bestimmten Kontextes (z.B. für den Zustand eines Aktors) genutzt wird.

3.2.0.3 Mutable Val

Möglichst vermeiden

Die Nutzung von veränderbaren Datentypen kann unter Umständen sinnvoll sein, aber man muß sicherstellen, daß man diese niemals weitergibt! Wer sich nicht sicher ist, sollte diese Kombination vermeiden.

3.2.0.4 Mutable Var

Niemals!

Um es kurz zu machen: Nie, aber auch wirklich nie sollte diese Kombination verwendet werden!

3.3 Schnelleinstieg in die funktionale Programmierung

Unter funktionaler Programmierung versteht man gemeinhin den Aufbau von Programmen aus Funktionen unter der Prämisse, daß Nebenwirkungen (Seiteneffekte) bewußt vermieden werden. Sogenannte “reine funktionale” Programmiersprachen schließen die Verwendung von Elementen aus, die diesem Paradigma zuwider laufen. Scala ist keine reine (pure) funktionale Programmiersprache. Allerdings ist Scala eine rein objektorientierte Sprache. Im Gegensatz zu Java bedeutet dies, daß auch primitive Datentypen Objekte sind. Interfaces können über sogenannte Traits realisiert werden, die jedoch nicht nur die Deklarationen sondern auch konkrete Implementierungen beinhalten können. Des Weiteren können Klassen in Scala mehrere Traits erweitern,

was jedoch keine Mehrfachvererbung ist sondern ein Mixin-Mechanismus. Statische Felder und Methoden können nicht in Klassen definiert werden, allerdings in einem `object`. Ein solches `object` stellt ein Singleton dar und bildet im Zusammenhang mit einer gleichnamigen Klasse ein sogenanntes Companion-Object.
Die funktionale Programmierung wird in Scala dadurch möglich, daß Funktionen sogenannte First-Class-Objects sind. Auch Funktionen höherer Ordnung werden unterstützt. Darüber hinaus werden wichtige Eigenschaften wie Pattern-Matching und Closures unterstützt. Scala ist eine statisch typisierte Programmiersprache mit einem sehr umfangreichen Typisierungssystem.

3.3.1 Auswertungsstrategien (evaluation strategies)

Man unterscheidet die beiden Strategien “Call by value” und “Call by name”.
Erstere wertet den angegebenen Ausdruck immer aus, aber dafür nur einmal. Letztere wertet den Ausdruck nur aus, wenn er auch benutzt wird, dafür wird er immer wieder ausgewertet.
Beide Strategien setzen voraus, daß die angegebenen Ausdrücke reine Funktionen sind und auch terminieren.
Scala verwendet “Call by value”, unterstützt aber auch “Call by name” dessen Verwendung sinnvoll sein kann, wenn teure Auswertungen evtl. nicht notwendig sind.

call-by-value

```
scala> val a = 1
a: Int = 1
```

call-by-name

```
scala> def f(a: Int, b: => Int): Int = {
  if (a == 0)
    a + 1
  else
    b
}
f: (a: Int, b: => Int)Int

scala> f(0, extremLangsameFunktion)
res4: Int = 1
```

Jede Funktion, die für “Call by value” terminiert, terminiert auch für “Call by name”. Umgekehrt gilt dies nicht.

Funktionsparameter können als “by value” oder “by name” übergebene werden. Dies gilt gleichermaßen für die Definition von Ausdrücken. Wobei `def` für “by name” und `val` für “by value” genutzt wird.

3.3.2 Scopes und Blöcke

In Scala können Funktionen auch innerhalb anderer Funktionen definiert werden. Dies ist sehr hilfreich, wenn sehr spezifische Hilfsfunktionen genutzt werden.

Funktionen in Funktionen

```
def foo(a: Int) = {
  def bar(x: Int, y: Int) = ???
  bar(a, 1)
}
```

Blöcke werden durch geschweifte Klammern geöffnet und geschlossen. Sie können eine Reihe von Definitionen oder Ausdrücken enthalten und das letzte Element des Blocks bestimmt dessen Rückgabewert. Ein Block ist wiederum ein Ausdruck und kann an allen Stellen genutzt werden, an denen auch ein Ausdruck stehen kann.
Bei der Schachtelung von Funktionen und bei Blöcken, ist die Sichtbarkeit von Werten zu beachten.

Sichtbarkeit in geschachteltem Code

```
def foo(a: Int) = {
  val x = 23
  def bar(a: Int) = {
    x + a // x ist gleich 23
  }
  val r = {
    val x = bar(42) + a
    x * x // x ist gleich 42 + 23 + a
  }
}
```

Definitionen und Werte innerhalb eines Blocks sind auch nur in diesem sichtbar. Ausdrücke von außerhalb des Blocks sind in diesem sichtbar, es sei denn, darin sind identisch benannte definiert. Dies nennt man “shadowing”.

3.3.3 Semikolons und Infix-Operatoren

In Scala ist die Verwendung von Semikolons im Gegensatz zu Java optional, d.h. man kann eine Zeile mit einem Semikolon beenden, muß dies aber nicht tun. Gemeinhin lässt man das Semikolon weg. Lediglich wenn zwei Befehle hintereinander geschrieben werden, müssen diese durch ein Semikolon getrennt werden.

Semikolon

```
// Korrekte Syntax
val a = 1
// Auch korrekt, aber nicht notwendig.
val b = 2;
// Semikolon notwendig
val c = a + b; c * 2
```

Bei Ausdrücken, die über mehrere Zeilen gehen, müssen die Infix-Operatoren an das Ende der vorhergehenden Zeile gesetzt werden. Alternativ könnte man den gesamten Ausdruck in Klammern setzen.

Ausdrücke über mehrere Zeilen

```
// Wird interpretiert als a; + b
a
+ b
// Wird interpretiert als a + b
a +
b
// Wird interpretiert als a + b
(a
+ b)
```

3.3.4 Tail-Rekursion

Wenn der letzte Befehl einer Funktion der Aufruf einer anderen ist, nennt man dies einen Tail-Call, da dann der Stack für beide Funktionen genügt. Nutzt man diese Technik in rekursiven Funktionen, werden diese *tail-recursive* genannt. Eine solche Funktion ist ein iterativer Prozess und kann vom Compiler zu einer einfachen Schleife optimiert werden.
Tail-Rekursion bringt erhebliche Leistungsvorteile. Allerdings sind tail-rekursive Funktionen meist nicht so leicht lesbar wie rein rekursive. Deswegen sollte man abwägen ob der Aufwand für die Implementierung einer Tail-Rekursion notwendig ist. Manchmal ist es auch schlichtweg nicht möglich einen Algorithmus tail-rekursiv zu implementieren.

Fakultät mit Rekursion

```
object Factorial {
  def fac(n: Long): Long = {
    if (n == 0)
      1L
    else
      n * fac(n - 1)
  }
}
```

Fakultät mit Tail-Rekursion

```
import scala.annotation.tailrec
object Factorial {
  def fac(n: Long): Long = {
    @tailrec
    def fact(acc: Long, x: Long): Long = {
      if (x == 0)
        acc
      else
        fact(acc * x, x - 1)
    }
    fact(1, n)
  }
}
```

3.3.5 Funktionen höherer Ordnung (Higher Order Functions)

Durch die Behandlung von Funktionen als First-Class-Objects können diese wie jeder andere Wert verwendet werden. Dies bedeutet, daß Funktionen implementiert werden, die andere Funktionen als Parameter empfangen und Funktionen als Rückgabewerte liefern können. Solche Funktionen nennt man Funktionen höherer Ordnung (Higher-Order Functions).
Der Typ für Funktionen lautet `A => B`, demnach impliziert die Notation `String => Int` eine Funktion, die einen String empfängt und eine Ganzzahl (Integer) zurückgibt. Damit man nicht jede benutzte Funktion explizit definieren muß, kann man auch sogenannte anonyme Funktionen definieren.

Funktionen und anonyme Funktionen

```
@ def check(x: Int)(f: Int => Boolean) = f(x)
defined function check
@ def isEven(a: Int) = a % 2 == 0
defined function isEven
@ // Funktion explizit übergeben.
@ check(3)(isEven)
res2: Boolean = false
@ // Eine anonyme Funktion übergeben.
@ check(3)((a: Int) => a % 2 != 0)
res3: Boolean = true
```

3.3.6 Currying

Mit Currying meint man die Umwandlung einer Funktion mit mehreren Argumenten in eine mit einem Argument. Praktisch bedeutet dies, daß durch die Implementierung von Funktionen, die wiederum Funktionen zurückgeben weitere Generalisierungen und Vereinfachungen möglich sind. Wenn also eine Funktion definiert ist, die `n` Argumente erfordert und auf *ein* Argument angewendet wird, so verarbeitet sie dieses und gibt eine Funktion zurück, die ihrerseits noch `n - 1` Argumente verlangt.
Der Aufruf `foo(bar)(42, "Die Antwort.")` wird linksassoziativ ausgewertet, d.h. er löst auf zu `(foo (bar)) (42, "Die Antwort.")`. Der Ausdruck `foo(bar)` wendet die Funktion `foo` auf `bar` an und die daraus zurückgegebene Funktion wird mit den übrigen Parametern aufgerufen.

Beispiel für Currying

```scala
/**
  * Multipliziere alle Zahlen beginnend mit
  * `x` und endend mit `y`. Hierbei wird auf
  * jede Zahl die übergebene Funktion `f`
  * angewendet bevor die Multiplikation
  * durchgeführt wird.
  */
def mult(f: Int => Int): (Int, Int) => Int = {
  def applyF(x: Int, y: Int): Int = {
    if (x > y)
      1
    else
      f(x) * applyF(x + 1, y)
  }
  applyF
}
// Identität
mult((a: Int) => a)(1, 1) // => 1
mult((a: Int) => a)(1, 2) // => 2
mult((a: Int) => a)(1, 3) // => 6
// Quadrieren
mult((a: Int) => a * a)(1, 1) // => 1
mult((a: Int) => a * a)(1, 2) // => 4
mult((a: Int) => a * a)(1, 3) // => 36
```

Scala unterstützt eine spezielle Syntax, um die Definition derartiger Funktionen zu vereinfachen. Es können mehrere Parameterlisten angegeben werden.

Currying mit spezieller Syntax

```
/**
  * Vereinfachte Definition durch multiple
  * Parameterlisten.
  */
def mult(f: Int => Int)(x: Int, y: Int): Int = {
  if (x > y)
    1
  else
    f(x) * mult(f)(x + 1, y)
}
// Identität
mult((a: Int) => a)(1, 1) // => 1
mult((a: Int) => a)(1, 2) // => 2
mult((a: Int) => a)(1, 3) // => 6
// Quadrieren
mult((a: Int) => a * a)(1, 1) // => 1
mult((a: Int) => a * a)(1, 2) // => 4
mult((a: Int) => a * a)(1, 3) // => 36
```

3.3.7 Polymorphismus

Scala unterstützt polymorphe Funktionen, d.h. man kann angeben für welche Typen eine Funktion anwendbar ist. Nehmen wir z.B. die folgende Funktion, die generisch auf einem Datentyp `Person` arbeitet und das Gehalt der Person zurückgibt, wenn diese ein Angestellter (`"EMPLOYEE"`) ist. Anderenfalls wird ein Ausnahmefehler (Exception) geworfen.

```
final case class Person(...)

def getSalary(p: Person) = {
  if (p.category == "EMPLOYEE") {
    ...
  }
  else
    throw new IllegalArgumentException("Cannot get salary of non-employee!")
}
```

Dieser Ansatz hat sicherlich mehr als ein Problem, aber wir konzentrieren uns darauf, daß die Funktion nur auf Datentypen anwendbar sein sollte, welche die erforderlichen Bedingungen erfüllen. Mit Hilfe der Typisierung ist eine andere Lösung denkbar:

```
trait Person {
  ...
}

final case class Stranger(...) extends Person

final case class Employee(...) extends Person

def getSalary[A <: Employee](p: A) = {
  ...
}
```

Nun verlangt die Funktion `getSalary` einen Datentyp, der ein untergeordneter Typ von `Employee` ist. Anwender der Funktion sehen nun direkt an deren Signatur, daß ein solcher erforderlich ist. Darüber hinaus können Tests eingespart werden, da der Compiler nun prüft ob ein korrekter Typ übergeben wurde. Des Weiteren kann dieser Optimierungen vornehmen, da nur bestimmte Typen zugelassen sind.

Auch komplett generische Angaben sind möglich:

```
def apply[A, B](a: A)(f: A => B): B = f(a)
```

Es lohnt sich dieses Thema zu vertiefen, da es hiermit möglich ist den Implementierungsraum einer Funktionsdefinition einzuschränken.
Man vergleiche z.B. die möglichen Implementierungen für `def f(a: String): String` mit denen für `def f[A,B](a: A)(b: A => B): B`.

3.3.8 Pattern-Matching

Das Problem der Dekomposition in der Programmierung lässt sich in funktionalen Programmiersprachen mit Hilfe von Pattern-Matching (Musterabgleich) lösen. Generell geht es um die Fragestellung, welche Klasse bzw. Unterklasse mit welchen Konstruktorparametern benutzt wurde.

Folgendes Beispiel soll das Prinzip verdeutlichen.

Beispiel für Pattern-Matching

```
scala> def fn(x: Any): String = x match {
     |   case Some(value) => value.toString
     |   case None        => "None"
     |   case (v1, v2)    => s"Pair($v1, $v2)"
     |   case xh :: xs    => "List"
     |   case _           => "..."
     | }
fn: (x: Any)String
scala> fn(Option(123))
res0: String = 123
scala> fn(None)
res1: String = None
scala> fn(Option(List(1,2,3)))
res2: String = List(1, 2, 3)
scala> fn(List(1,2,3))
res3: String = List
scala> fn((1,2))
res4: String = Pair(1, 2)
scala> fn((1,2,3))
res5: String = ...
```

Noch einige Anmerkungen zum Beispiel:

1. Die Verwendung von `Any` als Datentyp dient hier lediglich der Demonstration.
2. Der Ausdruck `Some(...)` wird genutzt um auf eine `Option` zu treffen.
3. Mit `(v1, v2)` trifft man ein Paar (Tuple) und extrahiert die einzelnen Elemente direkt.
4. Der letzte Ausdruck _ trifft auf alles.

Die Unterstrichnotation (_) im Pattern-Matching bedeutet, daß der Wert selbst nicht verwendet werden soll. So trifft z.B. das Pattern `Some(_)` auf eine `Option`, aber der Inhalt derselben ist nicht relevant.

3.3.9 Implizite Parameter

Bei der Nutzung von Currying mit mehreren Parameterlisten ist es manchmal hinderlich, wenn alle Parameter jeweils explizit angegeben werden müssen. Durch die Definition eines Wertes als `implicit` innerhalb eines Bereichs (Scope) wird der entsprechende Ausdruck automatisch genutzt. Allerdings dürfen nicht zwei implizite Ausdrücke im selben Bereich definiert sein, wenn die Funktion genutzt werden soll. Implizite Parameter können auch explizit belegt werden, um dieses Problem zu umgehen.

Beispiel für implizite Parameter

```
scala> def increment(n: Int)(incBy: Int) =
     |   n + incBy
increment: (n: Int)(incBy: Int)Int
scala> increment(3)
<console>:14: error: missing argument list for method increment
...
scala> increment(3)(4)
res0: Int = 7
scala> def increment(n: Int)
     |   (implicit incBy: Int): Int = n + incBy
increment: (n: Int)(implicit incBy: Int)Int
scala> increment(3)(4)
res1: Int = 7
scala> implicit val i = 10
i: Int = 10
scala> increment(3)(4)
res2: Int = 7
scala> increment(3)
res3: Int = 13
scala> implicit val foo = 1
foo: Int = 1
scala> increment(3)
<console>:15: error: ambiguous implicit values:
```

```
 both value i of type => Int
 and value foo of type => Int
 match expected type Int
       increment(3)
scala> increment(3)(foo)
res7: Int = 4
```

3.4 Hilfsmittel zur Unterstützung

Es gibt einige nützliche Plugins für SBT, die es erlauben, die Codequalität bzw. die Konformität hinsichtlich funktionaler Standards, zu überprüfen. Für uns hat sich Wartremover als überaus hilfreich erwiesen. Wenn Akka genutzt wird, muß über der Implementierung von `receive` jedoch immer die Annotation stehen, die Warnungen für `Any` unterdrückt. Dies sieht dann zum Beispiel so aus:

Annotation zum Unterdrücken von Any-Warnungen bei Aktoren

```
class FancyActor extends Actor {

  @SuppressWarnings(
    Array("org.wartremover.warts.Any")
  )
  override def receive: Receive = ???

}
```

Man kann mit Wartremover einen funktionalen Programmierstil forcieren, ohne diesen unausweichlich zu erzwingen. Insbesondere in einer Übergangs- bzw. Lernphase kann dies praktisch sein, sollte jedoch nicht dazu verleiten, ein `SuppressWarnings` einer sauberen Lösung vorzuziehen.

In einigen Fällen (wie z.B. im vorher erwähnten Aktor) kann es zu Fehlalarmen kommen, die dann entsprechend unterdrückt werden können.

Des Weiteren steht mit Scalafix ein neues Werkzeug zur Verfügung. Es überschneidet sich etwas mit Wartremover, stellt aber auch sehr gute andere Möglichkeiten zur Verfügung. Als Beispiel sei hier nur das Unterdrücken des generischen Vergleichs via == genannt.

3.5 Reduzierung von “Boilerplate” Code

Eine sehr nützliche Eigenschaft von Scala ist die Definition von Case-Classes. Dadurch lassen sich einfach Datencontainer implementieren, ohne den von Java gewöhnten umfangreichen Code zu schreiben (oft “Boilerplate” genannt). Im Folgenden zwei kleine Beispiele, wovon eines in Java und das andere in Scala umgesetzt sind:

Datencontainer in Java

```
class Person {
  private String firstname = "";
  private String surname = "";
  private String phone = "";

  public Person(String fn,
                String sn,
                String ph) {
    this.firstname = fn;
    this.surname = fn;
    this.phone = ph;
  }

  public String getFirstname() {
    return firstname;
  }

  public String getSurname() {
    return surname;
  }

  public String getPhone() {
    return phone;
  }
}
```

Datencontainer in Scala

```
final case class Person(firstname: String,
                        surname: String,
                        phone: String)
```

Der geringere Aufwand ist deutlich ersichtlich und darüber hinaus bieten Case-Classes noch weitere nützliche Funktionen wie z.B. Nichtveränderbarkeit (Immutablility) und Hilfsfunktionen wie beispielsweise `copy`. Damit kann man einfach eine modifizierte Kopie der Daten erzeugen.

Copy mit Case-Classes

```
@ final case class Person(firstname: String,
  surname: String,
  phone: String)
defined class Person
@ val p = Person(
  "Max",
  "Mustermann",
  "555-12345")
p: Person = Person("Max", "Mustermann", "555-12345")
@ p.copy(firstname = "Franz")
res2: Person = Person("Franz", "Mustermann", "555-12345")
```

> Bei tief verschachtelten Datenstrukturen wird die Verwendung von `copy` sehr umständlich. Eine mögliche Lösung bieten "Optics" (Lenses). Für Scala empfiehlt sich die Bibliothek Monocle.

Einführung und Grundlagen zu den verwendeten Technologien

Dieses Kapitel beschreibt die grundlegenden Technologien, welche bei der Implementierung der späteren Beispielanwendung verwendet werden.

Play Framework
: Das Play Framework (im Folgenden oft auch nur einfach `Play` genannt) ist ein Web-Framework, welches die Erstellung einer Oberfläche zur Interaktion zwischen der Anwendung und dem Nutzer erleichtert.

Akka
: Akka ist ein Toolkit, welches die Erstellung verteilter, asynchroner und paralleler Anwendungen ermöglicht, die zudem hochperformante Aufgabenstellungen mittels einfacher Skalierung bereitstellt.

Scala.js
: Scala.js kombiniert die Typisierung von Scala Code mit den vielfältigen Möglichkeiten und vorhandenen Bibliotheken von JavaScript. Dadurch wird die Erstellung von Frontend-Anwendungen erleichtert und in eine vorhandene Scala Umgebung fließend integriert.

Je nach vorhandenen Vorkenntnissen kann das folgende Kapitel oder Teile davon übersprungen werden.

4. Play Framework

Dieses Kapitel beschreibt die Grundlagen für die Arbeit mit dem *Play Framework* und kann bei entsprechenden Vorkenntnissen übersprungen werden.

Das *Play Framework* ist ein Web Framework für Java und Scala, kann in beiden Programmiersprachen genutzt und hinsichtlich der Anforderungen angepasst werden. Das zu Grunde liegende asynchrone Modell wurde auf Grundlage von Akka konzipiert und bietet nicht-blockierende (asynchrone), zustandslose Anwendungen (stateless), welche eine planbare und robuste Skalierung ermöglichen.

4.1 Erstellen einer Play Anwendung

Play Anwendungen können auf unterschiedlichem Wege mittels *sbt* erstellt werden. Seit Version `0.13.13` von *sbt* ist es möglich vorgefertigte Projektschablonen über den Befehl `sbt new` zu nutzen.

4.1.1 Play Anwendung über Schablonen erstellen

Für die Erstellung eines Projekts mit Play und Scala genügt der folgende Befehl:

Erstellung einer play-scala Anwendung via sbt new

```
sbt new playframework/play-scala-seed.g8
```

4.1.2 Play Anwendung von Hand erstellen

Eine neue Play Anwendung kann direkt mittels **sbt** erstellt und nach den eigenen Bedürfnissen konfiguriert werden.

Nach der Erstellung eines neuen Ordners, welcher die Grundlage für das Projekt bildet, müssen die folgenden Zeilen in die Datei *project/plugins.sbt* innerhalb eines *project* Ordners eingetragen werden.

Erstellung einer Play Anwendung mit SBT: plugins.sbt

```
// Repository of the Typesafe plugins
resolvers +=
  "Typesafe repository" at
  "https://repo.typesafe.com/typesafe/maven-releases/"

// The Play sbt plugin for the creation of Play projects
// Replace the `x` for the actual version of the plugin
// example: `2.5.15` or `2.6.3`
addSbtPlugin("com.typesafe.play" % "sbt-plugin" %  "2.5.x")
```

Die zu verwendende SBT Version kann in der *project/build.properties* definiert werden.

Erstellung einer Play Anwendung mit SBT: build.properties

```
sbt.version = 0.13.16
```

Abhängig von der Version des Play Framework kann die Version von SBT variieren. Daraus ergeben sich die folgenden Kombinationen aus SBT und Play Framework.

- Play 2.5 und SBT 0.13.x
- Play 2.6 und SBT 1.x

Eine grundlegende *build.sbt* wird rudimentär folgendermaßen aussehen.

Erstellung einer Play Anwendung mit SBT: build.sbt

```
name := "play-test"

version := "0.0.1"

lazy val root = (project in file(".")).enablePlugins(PlayScala)

scalaVersion := "2.11.11"
```

Die Version von Scala variiert wie die SBT Version in Abhängigkeit von der gewählten Play Version. Daraus ergeben sich die folgenden Kombinationen.

- Play 2.5 und Scala 2.11.x
- Play 2.6 und Scala 2.12.x

Die gesamte bisherige Verzeichnisstruktur ergibt sich daraus wie folgt.

Erstellung einer Play Anwendung mit SBT: Verzeichnisstruktur

```
play-test
 |_ build.sbt
 |_ project
   |_ build.properties
   |_ plugins.sbt
```

Es werden noch diverse SBT-Plugins für ein Play-Projekt benötigt, welche in der Datei *project/plugins.sbt* eingetragen werden.

SBT-Plugins für Play-Projekte

```
// The Play sbt plugin for the creation of Play projects
// Replace the `x` for the actual version of the plugin
// example: `2.5.9`
addSbtPlugin("com.typesafe.play" % "sbt-plugin" %  "2.5.x")

// web plugins
addSbtPlugin("com.typesafe.sbt" % "sbt-coffeescript" % "1.0.0")
addSbtPlugin("com.typesafe.sbt" % "sbt-less" % "1.1.0")
addSbtPlugin("com.typesafe.sbt" % "sbt-jshint" % "1.0.4")
addSbtPlugin("com.typesafe.sbt" % "sbt-rjs" % "1.0.8")
addSbtPlugin("com.typesafe.sbt" % "sbt-digest" % "1.1.1")
addSbtPlugin("com.typesafe.sbt" % "sbt-mocha" % "1.1.0")
// If you enable sassify then you need to have libsass installed.
//addSbtPlugin("org.irundaia.sbt" % "sbt-sassify" % "1.4.6")
```

Darüber hinaus kann die *build.sbt* erweitert werden, um weitere externe Abhängigkeiten hinzuzufügen und in das Projekt einzubinden.

Einbindungen in der build.sbt für Play-Projekte

```
name := "play-test"

version := "0.0.1"

lazy val root = (project in file(".")).enablePlugins(PlayScala)

scalaVersion := "2.11.11"

libraryDependencies ++= Seq(
  jdbc,
  cache,
  ws,
  "org.scalatestplus.play" %% "scalatestplus-play" % "1.5.1" % Test
)
```

Die Variable *libraryDependencies* enthält die eingebundenen Abhängigkeiten. Angefangen mit Zeile (10) werden diese eingebunden, so daß sie in der Anwendung zur Verfügung stehen. Darunter sind der Zugriff auf Datenbanken (jdbc), die Nutzung eines internen Cache (cache) und web services (ws).

4.2 Projektstruktur

Der Aufbau einer Play Anwendung ist standardisiert und trennt wichtige Teile der Kernanwendung, Konfiguration und Administration in separate Projektpfade. Im folgenden Abschnitt wird die Standardstruktur um einige Ordner erweitert, die für die konzeptionelle Planung von Vorteil sind und die Administration des Projektes erleichtern.

4.2.1 Verzeichnisse, SBT-Einstellungen und Abhängigkeiten

Die Verzeichnisstruktur einer Play Anwendung[1] gliedert sich grundlegend in die folgenden Teile (Ordner mit einem * wurden hinzugefügt):

Struktur einer Play Anwendung

```
app                   -> Anwendungsdateien
  |_ actors           -> Actor Definitionen
  |_ adt              -> Abstrakte Datentypen
  |_ assets
      |_ stylesheets  -> Normalerweise LESS CSS Dateien
      |_ javascripts  -> Normalerweise Coffeescript Dateien
  |_ controllers      -> Anwendungscontroller
  |_ dao*             -> Datenzugriffsobjekte
  |_ forms*           -> Formulardefinitionen
  |_ models           -> Anwendungsgeschäftsschicht
  |_ views            -> Templates
conf                  -> Konfigurationsdateien
  |_ application.conf -> Hauptkonfigurationsdatei
  |_ routes           -> Routing
dist                  -> Diverse weitere Projektdateien
public                -> Öffentliche Dateien
  |_ stylesheets      -> CSS Dateien
  |_ javascripts      -> Javascript Dateien
  |_ images           -> Bilddateien
project               -> SBT Konfigurationsdateien
  |_ build.properties -> Grundeinstellungen des SBT Projektes
  |_ plugins.sbt      -> SBT Plugins
```

[1] https://www.playframework.com/documentation/2.5.x/Anatomy

```
lib                   -> Manuelle Bibliotheksabhängigkeiten
logs                  -> Log-Datei Ordner
  |_ application.log   -> Standard Log-Datei
target                -> Erstellte Projektdateien
  |_ resolution-cache -> Informationen über Abhängigkeiten
  |_ scala-2.11
       |_ api          -> Erstellte API Dokumentation
       |_ classes      -> Kompilierte Class Dateien
       |_ routes       -> Von `routes` erstellt
       |_ twirl        -> Von `templates` erstellt
  |_ universal         -> Packaging
  |_ web               -> Kompilierte Web Ressourcen
test                  -> Ordner für diverse Testdateien
build.sbt             -> Skript zum Erstellen der Anwendung
```

Die folgende Übersicht stellt einen Überblick über die Verwendung der einzelnen Verzeichnisse und ihrer Bedeutung im Projekt dar. (Eine ausführliche Beschreibung der einzelnen Verzeichnisse findet sich in der Play Dokumentation[2].)

Das /app Verzeichnis

Dieses Verzeichnis enthält alle ausführbaren Java und Scala Dateien, Templates und kompilierte Medieninhalte. Die grundlegende MVC (Model-View-Controller) Architektur gliedert sich in die drei Grundverzeichnisse app/models, app/views und app/controllers. Darüber hinaus wurden einige Verzeichnisse zur Standardstruktur hinzu gefügt, welche die folgenden Bedeutungen haben. Das app/actors Verzeichnis beherbergt Actor Definitionen, im app/adt Verzeichnis werden abstrakte Datentypen definiert, welche nicht in das app/models Verzeichnis gehören, das app/dao Verzeichnis beinhaltet Datenzugriffsobjekte (Data Access Objects), welche den Datenzugriff auf verschiedene Datenquellen regeln und das app/forms Verzeichnis beinhaltet eigene Formulardefinitionen, welche aus den anderen Codeteilen ausgelagert wurden.

Das /conf Verzeichnis

Dieses Verzeichnis enthält die Konfigurationsdateien für die Anwendung.

Das /public Verzeichnis

Im /public Verzeichnis sind statische Ressourcen hinterlegt, welche direkt vom

[2] https://www.playframework.com/documentation/2.5.x/Anatomy

Webserver ausgeliefert werden. Dazu zählen unter anderem CSS Dateien, Bilder und Javascript Dateien.

Das /project Verzeichnis
In diesem Verzeichnis werden die *sbt* Informationen hinterlegt, welche für die Erstellung der finalen Anwendung notwendig sind. Dazu gehören u.a. verwendete Plugins und die Version von *sbt*, welche für die Kompilierung der Anwendung genutzt wird.

Das /lib Verzeichnis
Dieses optionale Verzeichnis beinhaltet alle manuell hinterlegten JAR Bibliotheksdateien, welche automatisch zum Klassenpfad (Classpath) hinzu gefügt werden.

Das /logs Verzeichnis
Dieses Verzeichnis enthält Log-Daten der Anwendung, welche automatisch in die Standard-Logdatei geschrieben werden.

Das /target Verzeichnis
Das /target Verzeichnis enthält alle Dateien, welche durch den Kompilierungsprozeß vom System erstellt werden. Dazu gehören u.a. kompilierte Klassen der Java und Scala Dateien, kompilierte CSS und Javascript Dateien oder die erstellten Template Inhalte.

Diese Grundstruktur ist bei allen Play Projekten grundsätzlich ähnlich und kann darüber hinaus an den persönlichen Programmierstil angepasst werden.

4.2.2 Unterprojekte

Oftmals ist es sinnvoll, ein Projekt in mehrere Unterprojekte (Multiprojekt) aufzuteilen, um eine Trennung zwischen den einzelnen Komponenten der Anwendung zu erhalten und die Wartbarkeit der Code-Basis zu erleichtern. Ein Grund kann u.a. die Aufteilung der Anwendung in diverse Teilkomponenten sein, welche sich über definierte Schnittstellen miteinander unterhalten und eine getrennte Skalierung ermöglichen.

Unterprojekte teilen sich die **build.sbt** das Hauptprojektes, indem die einzelnen Projektdefinitionen aufgenommen werden. Dadurch wird der komplette Erstellungsprozeß über eine zentrale Datei ermöglicht.

Hauptprojekte können als Play Anwendung erstellt werden oder mittels SBT Multiprojekt die Play Anwendung als Unterprojekt beinhalten. Dadurch ergeben sich

unterschiedliche Projektstrukturen und Definitionen für die Anwendung.

4.2.2.1 Play Anwendung als Hauptprojekt

Eine Play Anwendung kann selbst das Hauptprojekt sein und diverse Unterprojekte enthalten, die über die `build.sbt` definiert werden. Im folgenden Beispiel wird eine Play Anwendung um zwei weitere Unterprojekte erweitert, die einen Datenbankservice und einen Authentifikationsservice via Akka bereitstellen.

Definition einer Play Anwendung mit 2 Akka-Unterprojekten in der build.sbt

```
name := "main-play-project"
version := "0.9"

lazy val commonSettings = Seq(
    organization := "com.my.organization",
    scalaVersion := "2.11.11",
    scalaOptions ++= Seq( ... ),
    javaOptions ++= Seq( ... ),
    ...
)

lazy val mainPlayProject = project.in(file("."))
    .settings(commonSettings: _*)
    .aggregate(subProjectDatabase, subProjectAuthentication)
    .enablePlugins( ... )

lazy val subProjectDatabase = project
    .in(file("subProjectDatabase"))
    .settings(commonSettings: _*)
    .settings(
        name := "sub-project-database",
        libraryDependencies ++= List( ... )
    )

lazy val subProjectAuthentication = project
    .in(file("subProjectAuthentication"))
    .settings(commonSettings: _*)
    .settings(
        name := "sub-project-authentication",
```

```
        libraryDependencies ++= List( ... )
    )
```

Das *mainPlayProject* (Zeile 16) stellt das Hauptprojekt der Anwendung dar und gliedert sich in die zwei Unterprojekte *subProjectDatabase* (Zeile 21) und *subProjectAuthentication* (Zeile 31).

Die Verzeichnisstruktur würde sich wie folgt darstellen.

Verzeichnisstruktur einer Play Anwendung mit 2 Akka-Unterprojekten

```
mainPlayProject
  |_ build.sbt
  |_ app
  |_ conf
  |_ logs
  |_ project
  |_ public
  |_ subProjectDatabase
    |_ packaging.sbt
    |_ project
    |_ src
    |_ target
  |_ subProjectAuthentication
    |_ packaging.sbt
    |_ project
    |_ src
    |_ target
  |_ target
  |_ test
```

Wie in einem *normalen* Projekt gibt es eine `build.sbt` und diverse weitere Ordner, die für alle Unterprojekte gültig sind. Darüber hinaus werden spezifische Dateien, welche in die jeweiligen Unterprojekte gehören, jeweils unter einem Order angelegt, der gleich dem Namen des Unterprojektes ist.

4.2.2.2 Play Anwendung als Unterprojekt eines SBT Multiprojektes

Bei der Wahl eines SBT Multiprojektes als Hauptprojekt, werden alle Komponenten der Anwendung als Unterprojekte definiert. Dadurch wird das Hauptprojekt von den

Verzeichnisstrukturen der Unterprojekte befreit und die Trennung zwischen den einzelnen Komponenten erweitert.
Im folgenden Beispiel wird ein SBT Multiprojekt erstellt, welches 3 Unterprojekte beinhaltet. Das Projekt gliedert sich in eine Play Anwendung, welche das Frontend darstellt und zwei Akka Anwendungen, die einen Datenbankservice und einen Authentifikationsservice bereitstellen.

Definition eines SBT Multiprojektes mit 3 Unterprojekten

```
name := "main-project"
version := "0.9"

lazy val commonSettings = Seq(
    organization := "com.my.organization",
    scalaVersion := "2.11.11",
    scalaOptions ++= Seq(
        ...
    ),
    javaOptions ++= Seq(
        ...
    ),
    ...
)

lazy val mainProject = project.in(file("."))
    .settings(commonSettings: _*)
    .aggregate(subProjectPlay,
      subProjectDatabase, subProjectAuthentication)
    .enablePlugins(...)

lazy val subProjectPlay = project
    .in(file("subProjectPlay"))
    .settings(commonSettings: _*)
    .settings(
        name := "sub-project-play",
        libraryDependencies ++= List(
            ...
        )
    )

lazy val subProjectDatabase = project
```

```
    .in(file("subProjectDatabase"))
    .settings(commonSettings: _*)
    .settings(
        name := "sub-project-database",
        libraryDependencies ++= List(
            ...
        )
    )

lazy val subProjectAuthentication = project
    .in(file("subProjectAuthentication"))
    .settings(commonSettings: _*)
    .settings(
        name := "sub-project-authentication",
        libraryDependencies ++= List(
            ...
        )
    )
```

Das *mainProject* bildet das Gerüst und beinhaltet die Play Anwendung *subProjectPlay* und die beiden Akka Anwendungen *subProjectDatabase* und *subProjectAuthentication*.

Die Verzeichnisstruktur würde sich wie folgt darstellen.

Verzeichnisstruktur eines SBT Multiprojektes mit 3 Unterprojekten

```
mainProject
  |_ bin
  |_ build.sbt
  |_ logs
  |_ project
  |_ subProjectAuthentication
    |_ packaging.sbt
    |_ project
    |_ src
    |_ target
  |_ subProjectDatabase
    |_ packaging.sbt
    |_ project
    |_ src
    |_ target
```

```
|_ subProjectPlay
  |_ app
  |_ conf
  |_ logs
  |_ project
  |_ public
  |_ target
  |_ test
```

Das Hauptprojekt beinhaltet gemeinsame Dateien, welche von allen Unterprojekten genutzt werden. Spezifische Dateien, welche speziell zu den Unterprojekten gehören, werden in Ordnern angelegt, welche den Namen des Unterprojektes tragen.

4.3 Requests, Routing und Controller

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.3.1 Requests

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.3.2 Routing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.3.3 Controller

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.4 Templates (Twirl)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.4.1 Wiederverwendung von Templates

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.5 Mehrsprachigkeit (Internationalisierung)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.5.1 Messages Objekt

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.6 Formulare

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.6.1 Formdefinition

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.6.2 Form-Objekte und ihre Typen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.6.3 Beispiele für Formulare

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.6.4 Verarbeitung von Formularen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.6.5 Formulardarstellung in Template View

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.6.6 Beispiel mit sich wiederholenden Elementen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.7 Datenbankkonfiguration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.7.1 Konfiguration von Slick für Play

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.8 Datenbankzugriff

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.9 Asynchrone Programmierung mit Play

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.9.1 Websockets

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.9.1.1 Websockets mit Akka Stream und Aktoren

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.10 Webservices

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11 Migration von Play 2.5 auf 2.6

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.1 Was hat sich geändert

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.2 SBT 0.13.15 erforderlich

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.3 `Guice` und `OpenId` Unterstützung ausgelagert

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.4 Bereitstellung neuer `Controller` Klassen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.5 Assets

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.6 Play WS

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.7 Anpassungen bei `i18n`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.7.1 Entfernung von Implicit Default Lang

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.7.2 Refactoring der Message API zu Traits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.7.3 I18nSupport benötigt impliziten Request

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.7.4 Einfachere Einbindung von I18nSupport

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.8 Cache

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.9 Veränderungen an der `Scala Configuration API`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.10 Entfernung diverser APIs und Bibliotheken

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.11 `play.api.libs.concurrent.Execution` ist nun veraltet

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.11.12 Neue `Standardfilter`

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.12 Konfiguration von Ehcache

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

4.13 Ausführen mit IntelliJ IDEA und Debuggen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5. Akka

Akka ist ein Toolkit für die Erstellung von asynchronen, parallelen und verteilten Anwendungen, die von kleinen Anwendungsfällen bis hin zu hochperformanten Aufgabenstellungen skaliert werden können.
Das Aktormodell ermöglicht die Abstraktion komplexer Aufgabenstellungen hin zu fehlertoleranten, belastbaren Komponenten, die untereinander kommunizieren und ein transparentes Konstrukt darstellen.

Zusammengefasst ist Akka:

- Parallelisierung und Verteilung von Anwendungen durch Aktoren
- Asynchroner, nicht-blockierender Nachrichtenaustausch zwischen den Aktoren
- Fehlertoleranz durch Supervision und dem *let it crash* Modell
- Verteilte Anwendungen durch reinen Nachrichtenaustausch
- Persistenz durch Recovery Strategien
- JVM

5.1 Einrichten einer Akka Anwendung

Die Einrichtung einer Akka Anwendung kann via Maven, SBT, Gradle, etc. erfolgen. Anleitungen für diverse Wege gibt es in dem Getting Started Abschnitt der offiziellen Akka Dokumentation.

Das folgende Beispiel zeigt die Installation einer Akka Anwendung via SBT. Zuerst wird ein Projektordner angelegt und eine grundsätzliche *build.sbt* Datei erstellt.

SBT-Konfiguration für ein Akka-Projekt

```
name := "AkkaProject"

version := "0.1"

scalaVersion := "2.11.11"

libraryDependencies += "com.typesafe.akka" %% "akka-actor" % "2.4.12"
```

5.2 Akka Grundlagen

Nebenläufigkeit und Parallelismus

Nebenläufigkeit (*concurrency*) und Parallelismus haben einige feine Unterschiede, welche sich in einem Akka System widerspiegeln. Nebenläufigkeit bedeutet, daß zwei Prozesse eigenständig voranschreiten können, auch wenn sie nicht parallel ausgeführt werden. Parallelismus hingegen bedeutet, daß die Prozesse wirklich parallel ausgeführt werden.

Synchron und Asynchron

Ein synchroner Methodenaufruf bedeutet, daß der aufrufende Prozess erst weitermachen kann, wenn die aufgerufene Methode ein Ergebnis geliefert hat. Wohingegen ein asynchroner Methodenaufruf den aufrufenden Prozess nicht blockiert, so daß dieser weiter in seiner Abarbeitung voranschreiten kann.

Blockierend und nicht-blockierend

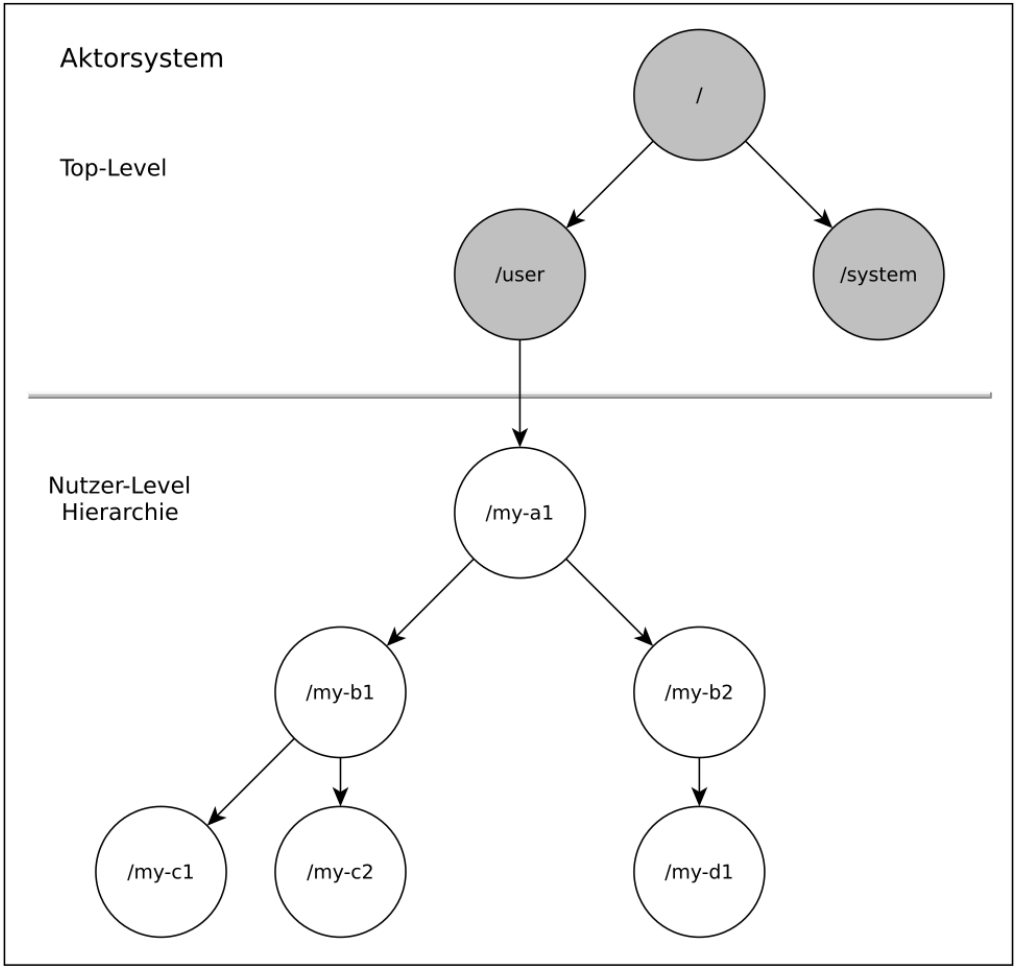
Blockierend bedeutet, daß eine Ressource exklusiv von einem Thread genutzt und dadurch den Zugriff anderer Threads auf diese Ressource verhindert. Im Kontrast dazu verhindern nicht-blockierende Prozesse den exklusiven Zugriff nur eines Threads auf eine Ressource und werden generell blockierenden Strukturen vorgezogen.

5.2.1 Aktorsystem und Aktoren

Ein Aktorsystem verwaltet die in ihm laufenden Aktoren und gibt der Anwendung eine hierarchische Struktur. Darüber hinaus werden grundlegende Konfigurationen wie

das Logging, die Fehlerbehandlung oder das Verhalten des Systems im Vergleich zu anderen Aktorsystemen, definiert.

Aktoren sind Teile eines Aktorsystems und gliedern sich in dessen Hierarchie. In dieser Hierarchie gibt es Aufsichtsaktoren, welche die Erstellung anderer Aktoren und deren Fehlerbehandlung überwachen und steuern. Weitere Aktoren übernehmen die ihnen definierten Funktionalitäten und kommunizieren untereinander mittels Nachrichten.



Aktorsystem und Aktoren

Dadurch bilden Aktoren die kleinste Einheit in einem Aktorensystem und sollten

im Gesamtkonzept möglichst einfach gehalten und mit einer klar definierten Aufgabe versehen sein. Dieses herunterbrechen der Komplexität in einfache, in sich geschlossene Einheiten, ermöglicht eine klare Trennung von sich überschneidenden Verantwortlichkeiten und Abhängigkeiten.

Ein Aktor kann über seine **Aktorreferenz** aufgelöst und angesprochen werden. Dahinter verbirgt sich ein Container aus *Zustand*, *Verhalten*, einer *Nachrichtenbox*, *Kindaktoren*, die von diesem Aktor erstellt wurden und eine Strategie für den eigenen *Lebenszyklus und die Fehlerbehandlung* der von ihm erstellten Kindaktoren.

Aktorreferenz
Eine Aktorreferenz ist ein Objekt, welches den Aktor nach außen hin abschottet und frei übergeben werden kann. Dadurch ist das Ansprechen der Aktoren von diversen Orten aus möglich und ermöglicht eine lose Kopplung im gesamten Aktorensystem. Diese dezentrale Haltung der Aktoreinheiten ermöglicht das Durchstarten eines Aktors ohne Erneuerung seiner Aktorenreferenz, das Ansprechen eines Aktors auf entfernten Systemen (Remote) oder die Kommunikation mit Aktoren aus anderen Applikationen.

Eine Aktorenreferenz kann auch als eine Art Abschottung angesehen werden, durch welche alle Aktoren ihre internen Zustände nach außen hin verbergen und nur preisgeben, was sie preisgeben wollen.

Zustand
Jeder Aktor verfügt über seinen eigenen leichtgewichteten Thread, welcher die Daten und damit den Zustand des Aktors gegenüber anderen Aktoren verbirgt. Solch ein Zustand kann direkt über eine Zustandsmaschine (FMS - Finite State Machine) abgebildet werden oder aus internen Variablen, Nachrichten oder Anfragen bestehen.

Verhalten
Das Verhalten eines Aktors spiegelt sich in den Aktionen wieder, welche als Reaktion auf erhaltene Nachrichten ausgeführt werden. Dabei kann sich das Verhalten im Verlaufe der Zeit ändern und je nach Zustand anpassen.

Nachrichtenbox
Aktoren erhalten Nachrichten von anderen Aktoren oder aus anderen Systemen. Diese Nachrichten werden in der Nachrichtenbox des empfangenden Aktors abgelegt und in der Reihenfolge ihres Eintreffens abgearbeitet.

Werden Nachrichten von einem Aktor an einen anderen versendet, bleibt die Reihen-

folge, in der die Nachrichten versendet worden sind, beim empfangenden Aktor gleich. Versenden hingegen verschiedene Aktoren Nachrichten an einen Aktor, kann sich die Reihenfolge der Nachrichten im empfangenden Aktor unterscheiden, da diese diversen Aktoren innerhalb unterschiedlicher Threads agieren.

Kindaktoren
Jeder Aktor kann mehrere Kindaktoren erstellen und über diese die Aufsicht haben. Er wird dann zum Aufsichtsaktor über seine Kindaktoren. Wenn dieser Aufsichtsaktor beendet wird, beenden sich auch seine Kindaktoren.

Stategie für den Lebenszyklus und die Fehlerbehandlung
Während des Lebenszyklus von Kindaktoren eines Aktors können sich Fehler ergeben, welche von dem Aktor je nach vorgesehener Strategie behandelt werden.
Je nach Strategie kann dies zum Neustart der Kindaktoren, dem Weiterführen ihres Prozesses, dem Anhalten der Kindaktoren oder der Eskalation des Fehlers führen, so daß das gesamte System beendet wird.
Darüber hinaus ist es auch möglich, eigene Strategien und Ablaufpfade zu definieren, welche unter bestimmten eintretenden Situationen durchgeführt werden.

Das Kapitel "Supervision and Monitoring" in der Akka Dokumentation gibt einen ausführlichen Überblick über die vorhandenen Strategien und deren Bedeutung.

Die Strategie, welche für Kindaktoren genutzt werden soll, kann nach der Erstellung des Kindaktors nicht mehr geändert werden. Sollen in diesem Zusammenhang verschiedene Strategien genutzt werden, um diverse Fehler bei verschiedenen Aktoren zu behandeln, muß man die Aktorhierarchie dahingehend anpassen.
Aktoren, welche die gleiche Fehlerbehandlungsstrategie erhalten, sollten unter Aufsichtsaktoren gruppiert werden, welche diese definieren und diese Aktoren erzeugen.

5.2.2 Supervision

5.2.3 Aktorreferenzen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.2.4 Nachrichten und deren Auslieferung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.2.5 Konfiguration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3 Aktoren

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3.1 DeathWatch

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3.2 Nachrichten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3.3 Aktoren beenden

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3.4 FSM

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3.5 Persistenz

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3.5.1 Aufbau

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.3.6 Tests

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.4 Aktorenhilfsmittel

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.4.1 Event-Bus

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.4.2 Logging

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.4.3 Scheduler

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.4.4 Zeitdauer (Duration)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.4.5 Unterbrecher (Circuit Breaker)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

5.5 Streams

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

6. Scala.js

Scala.js ist ein Compiler, der Scala-Quelltext in entsprechendes Javascript übersetzt. Dies ermöglicht es Scala zu schreiben und das Programm in einem Webbrowser oder anderen Javascriptumgebungen (z.B. Node.js) auszuführen.

Javascript wird von den gängigen Webbrowsern unterstützt und ist letztlich die einzige Wahl, wenn man interaktive Webanwendungen schreiben möchte. Obwohl es zahlreiche Versuche gab, andere Technologien über Browserplugins hierfür zu nutzen (z.B. Flash, Java Applets, Silverlight), konnten sich diese nicht durchsetzen. Darüber hinaus ist Javascript die einzige Technologie, die auf mobilen Browsern verfügbar ist.

Als Programmiersprache ist Javascript geeignet für kleinere bis mittlere Projekte. In größeren leidet das Projekt unter diversen Eigenheiten und Schwächen der Sprache. Andererseits ist Javascript auch eine Plattform mit sehr interessanten Eigenschaften:

1. Die Anwendung muß nicht mehr heruntergeladen und installiert werden.
2. Sandbox, d.h. die Anwendung läuft per se abgesichert.
3. Verweise zu anderen Anwendungen sind dank Hyperlinks trivial.

Trotz aller Probleme der Sprache und zugehöriger Werkzeuge (HTML, CSS) bietet es sich an, die Stärken der Webplattform zu nutzen. Hierfür kommt Scala.js gelegen, das es ermöglicht in einer statisch typisierten funktionalen Programmiersprache Webanwendungen zu schreiben.

Dies ist sicherlich nicht notwendig bei kleinen Anwendungen, aber je größer ein Projekt wird, desto mehr sind Fehlerursachen nicht in externen Bibliotheken, sondern innerhalb des eigenen Codes zu suchen und zu finden. Wer je mit einer größeren Code-Basis in Javascript konfrontiert war, wird wissen wie aufwendig und schwierig dies ist. Die Nutzung von typisierten Sprachen ermöglicht es, einen Teil dieser Arbeit bereits beim Schreiben des Codes vom Compiler erledigen zu lassen.

Das dies in der Tat ein gewichtiger Punkt ist, kann man an den Bemühungen aller namhaften Internetkonzerne ersehen, die seit einiger Zeit versuchen, typisierte Varianten von Javascript zu schaffen (z.B. Dart, Flow, Typescript).

Darüber hinaus bietet Scala.js eine gute Möglichkeit für geteilten Code, d.h. Client und Server können gemeinsam Code verwenden, der nur einmal geschrieben und dann jeweils in JVM-Bytecode bzw. Javascript übersetzt wird. Zahlreiche etablierte Bibliotheken für Scala werden mittlerweile auch für Scala.js angeboten.

Es ergeben sich also die folgenden Vorteile durch geteilten Code:

1. Man muß nicht länger zwei Bibliotheken finden, welche die gleiche Funktionalität bieten.
2. Man muß nicht länger die gleiche Sache auf zwei verschiedene Arten tun.
3. Man muß nicht länger den gleichen Algorithmus in zwei verschiedenen Programmiersprachen implementieren und danach schwer zu findende Fehler suchen, die eben daraus resultieren.
4. Man muß nicht länger komplexe Konstrukte bauen, um Logikdoppelungen zwischen Client und Server zu vermeiden.

Im folgenden erklären wir kurz das Aufsetzen eines einfachen Projekts mit Scala.js.

6.1 Erstellen einer Scala.js Anwendung

Hierfür nutzt man Plugins für SBT, welche unter `project/plugins.sbt` eingebunden werden:

Scala.js Plugins für SBT

```
addSbtPlugin("org.scala-js"       % "sbt-scalajs"              % "0.6.26")
addSbtPlugin("org.portable-scala" % "sbt-scalajs-crossproject" % "0.6.0")
```

Anschließend muß das Plugin noch aktiviert werden, dies geschieht in der `build.sbt` z.B. durch folgende Einstellung:

Scala.js SBT Plugin aktivieren

```
enablePlugins(ScalaJSPlugin)
```

Das Compilieren erfolgt ganz normal via `compile` an der SBT-Konsole, allerdings können die daraus generierten Dateien (`.sjsir` und `.class`) so nicht in einem JVM-Projekt benutzt werden!

Damit eine Javascriptdatei generiert wird muß der Befehl `fastOptJS` genutzt werden. Alternativ dazu kann man auch `fullOptJS` nutzen, was jedoch während der Entwicklung nicht zu empfehlen ist, da es deutlich länger dauert.

Insofern die entwickelte Anwendung keine Bibliothek ist, sondern ausgeführt werden soll, muß noch die folgende Einstellung getätigt werden:

```
scalaJSUseMainModuleInitializer := true
```

Zusammen mit einem Top-Level-Objekt, das eine Methode `main` hat, kann die Anwendung via `run` von der SBT-Konsole gestartet werden.

Haupteinsprungspunkt für eine Javascriptanwendung

```
object Main {
  def main(args: Array[String]): Unit = {
    println("Hallo Welt!")
  }
}
```

6.2 Abhängigkeiten

Verwendete Bibliotheken werden wie gewohnt in der SBT-Konfiguration eingetragen jedoch mit drei(!) statt zwei Prozentzeichen.

Abhängigkeiten für Scala.js-Projekte

```
libraryDependencies ++= Seq(
  "org.scala-js"  %%% "scalajs-dom" % "0.9.6",
  "org.typelevel" %%% "cats-core"   % "1.5.0",
  "org.scalatest" %%% "scalatest"   % "3.0.5"  % Test
)
```

Möchte man Javascriptbibliotheken einbinden, können diese via Webjars integriert werden:

Einbinden von Webjars-Bibliotheken

```
libraryDependencies += "org.webjars" % "jquery" % "2.1.4"
```

Zusätzlich ist es erforderlich die Bibliotheken in der Direktive `jsDependencies` zu definieren, damit sie korrekt verfügbar gemacht werden:

```
jsDependencies += "org.webjars" % "jquery" % "2.1.4" / "2.1.4/jquery.js"
```

Das Scoping funktioniert für `jsDependencies` genauso wie für die "klassischen" Abhängigkeiten, d.h. `jsDependencies += "org.webjars" % "jquery" % "2.1.4" / "jquery.js" % "test"` grenzt in diesem Fall JQuery auf den Testmodus ein.

Lokale Javascriptbibliotheken können über den Helfer `ProvidedJS` spezifiziert werden. Die folgende Konfiguration sucht in den Projektressourcen nach der Datei `foo.js`:

Lokale Javascriptbibliotheken einbinden

```
jsDependencies += ProvidedJS / "foo.js"
```

6.3 Module exportieren

6.4 Cross-Compile

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

6.5 Testen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

Anwendungsszenario

Das Ziel der zu entwickelnden Anwendung ist es, eine Spieleplattform zu entwickeln. Auf dieser soll es die Möglichkeit geben, daß sich Nutzer registrieren, ihre Freundeslisten verwalten und Online-Spiele gegen andere Nutzer spielen.
Die verfügbaren Online-Spiele sind zudem auch ein Teil der Anwendung, werden separat entwickelt und auf der Plattform eingebunden.

Daraus ergibt sich eine Einteilung in folgende Unterprojekte:

- `Frontend` für die Nutzerinteraktion (`frontend`)
- `Online-Spiel`, welches über das Frontend ausgewählt werden kann (`seabattle`) und sich unterteilt in:
  - serverseitigen Code (`server`)
  - clientseitigen Code (`client`)
  - gemeinsamen Code (`shared`)

Die Implementierung der Anwendung erfolgt in iterativen Schritten:

- Erstellung der grundlegenden Play Anwendung
- Einrichtung des Projektverzeichnisses
- Konfiguration des Projektes in Unterprojekten
- Erstellung der grundlegenden `Frontend` Methoden für Autorisierung und Authentifikation
- Implementierung der Nutzerverwaltung
- Erstellung eines Spiels
- Integration des Spiels in die Anwendung

- Ausführung des Spiels
- Deployment der Anwendung auf einen Server

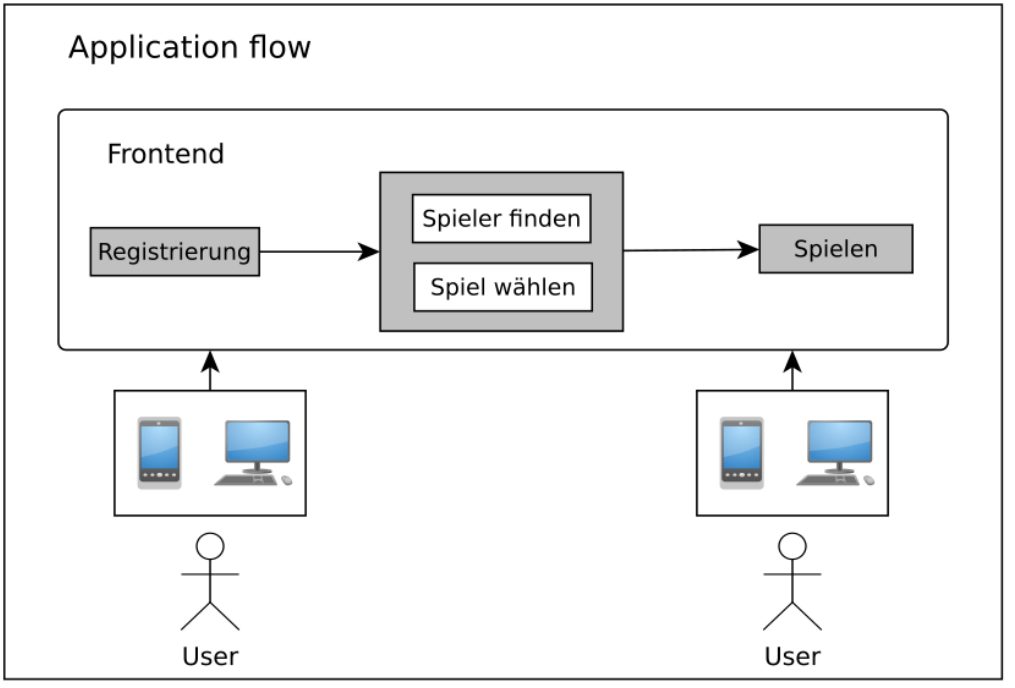
Im Anschluß werden einige Erkenntnisse aufgezeigt, welche sich aus der Entwicklung der Anwendung ergeben und zu einem interessanten Wissensschatz für kommende Projekte entwickelt haben.

- Nutzung der Bibliothek `Silhouette` für die Autorisierung und Authentifikation
- Nutzung der Bibliothek `Circe` für den Umgang mit JSON

Darüberhinaus wird die Migration der Anwendung von einer vorherigen auf eine aktuellere Version des Play Framework durchgeführt. Diese häufig vorkommende Aufgabe im Laufe eines Softwarelebenszyklusses beinhaltet diverse Schritte wie u.a.:

- Aktualisierung der genutzten Bibliotheken (Abhängigkeiten)
- Anpassungen des bestehenden Codes, wenn abhängige Bibliotheken interne Methoden und Konzepte verändern
- Anpassungen an Änderungen in den Konzepten des Play Frameworks
- generelles Refactoring

7. Das Frontend



Anwendungsfluss im Frontend

Das *Frontend* dient der Interaktion zwischen den Nutzern untereinander und des Nutzers mit der Anwendung selbst. Folgende Funktionalitäten sollen über das *Frontend* dem Nutzer zur Verfügung stehen:

- Registrierung auf der Webseite via E-Mail und Paßwort oder via Facebook
- Ändern des Paßworts
- Löschen des eigenen Accounts
- Abmelden von der Webseite
- Erstellen und Verwalten von Freundeslisten

- Suchen nach potentiellen Freunden
- Anzeigen der eigenen Freunde
- Freundschaftsanfrage senden
- Freundschaftsanfragen annehmen oder abbrechen
- Nutzer blockieren
- Übersicht der verfügbaren Online-Spiele
- Auswahl und Starten eines Online-Spiels
- Anzeigen der gespielten Spiele

Bei der Implementierung des *Frontend* wurden die folgenden Aspekte besonders berücksichtigt, da sie einen entscheidenden Einfluss auf die Funktionalitäten der Anwendung haben.

- Die Wahl einer geeigneten Bibliothek ermöglicht die Registrierung und Anmeldung der Nutzer über diverse Authentifikationsmethoden.
- Das Speichern der Daten in die unterliegende Datenbank sollte durch einen unterstützenden Datenbank-Layer erfolgen, welcher den Zugriff und die Arbeit mit den Daten erleichtert.
- Das dynamische Laden von Inhalten in Bezug auf die durch den Nutzer durchgeführten Aktionen führt zu einer Minimierung von zu ladenden Komponenten und einer Beschleunigung der Seite.

Eine beispielhafte Nutzung des Frontend durch einen Nutzer soll nach der Implementierung folgendermaßen möglich sein:

- Der Nutzer kann die Startseite aufrufen und bekommt die Möglichkeit, einen Account zu erstellen.
- Ein Registrierungsformular ermöglicht dem Nutzer alle notwendigen Informationen einzugeben und den Account zu erstellen.
- Der Account muß durch das Aufrufen eines Bestätigungslinks freigeschaltet werden.
- Der Nutzer kann sich mit seinen Anmeldedaten einloggen.
- Im *Freundebereich* können andere Nutzer zu einer persönlichen Freundesliste hinzugefügt werden.

- Freundesanfragen können abgelehnt oder bestätigt werden.
- Der Nutzer kann sich die auf der Plattform verfügbaren Spiele anzeigen lassen.
- Der Nutzer kann ein Spiel auswählen und einen anderen Spieler einladen gegen ihn zu spielen.
- Das Spiel wird eingerichtet und die Spieler können gegeneinander spielen.
- Gespielte Spiele werden aufgelistet.
- Der Nutzer kann sich von der Plattform abmelden.
- Der Nutzer kann seinen Account von der Plattform löschen.

8. Das Online-Spiel

Im Rahmen dieses Buches wird das allseits bekannte Spiel “Schiffe versenken” implementiert und in das Frontend integriert. Dieses Spiel bietet diverse interessante Herausforderungen, welche während der Implementierung gelöst werden müssen:

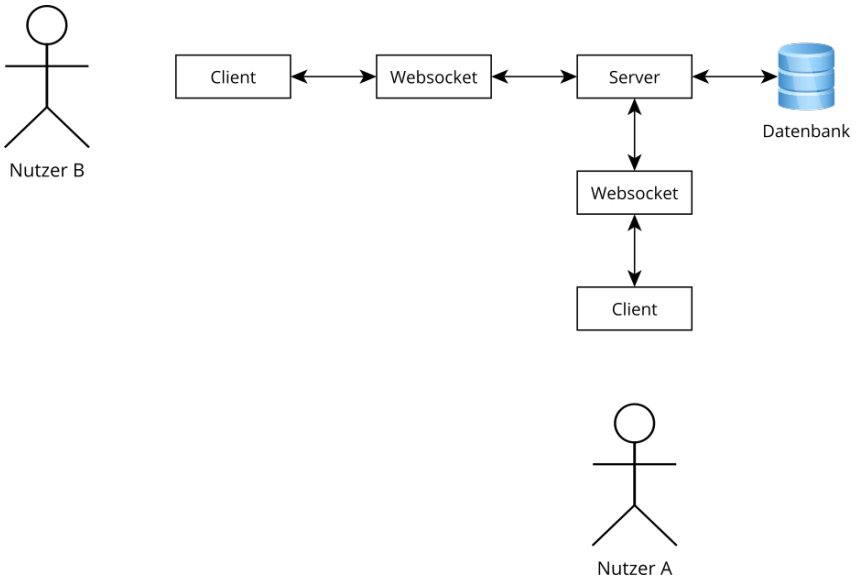
- Mehrspielerbetrieb
- Rundenbasierte Dynamik
- Benachrichtigung der Spieler über die Aktionen der Gegenspieler
- Erstellung eines eigenständigen Spiels, welches in die Hauptanwendung integriert wird
- Nachrichtenaustausch zwischen diversen Komponenten

Das Spiel muß verschiedene Komponenten zur Verfügung stellen, die entweder auf dem Server (JVM), auf dem Client im Webbrowser (Javascript) oder in beiden Umgebungen laufen. Letzteres sind hauptsächlich Datentypen, damit man diese nicht mehrfach implementieren muß. Dadurch ergibt sich eine verringerte Fehleranfälligkeit hinsichtlich Typisierung und eine Reduzierung des zu schreibenden Codes.

Auf Seiten des Servers müssen Funktionen für die Spiellogik und die Speicherung der relevanten Informationen (Spielstand) implementiert werden.

Damit der Anwender auch spielen kann, müssen im Client alle Funktionalitäten rund um die Darstellung, sowie Nutzerinteraktion und Nachrichtenfluß mit dem Server, implementiert werden. Die Spieler nutzen also einen Webbrowser, der vom Frontend eine HTML-Seite bzw. Seiten serviert bekommt, die den notwendigen Code enthalten. Die Hintergrundkommunikation läuft über Websockets, um regelmäßiges Neuladen der Seite zu vermeiden. Hierbei soll die wesentliche Spiellogik auf dem Server abgearbeitet werden, damit auf Seiten des Clients eigentlich nur noch Daten angezeigt und Aktionen der Spieler entgegengenommen werden.

Die visuelle Darstellung der Komponenten wird mittels `Scala.js` im Browser erstellt.



Schema des Spielflusses

Zum Schluß wird das Zusammenbringen des Spiels in das bereits erstellte `Frontend` beschrieben, so daß der Nutzer später auch eigene Spiele entwickeln und in die Anwendung integrieren kann. Zu diesen Schritten gehören u.a.:

- Anpassungen und Aktualisierung der Verzeichnisstruktur
- Erstellung geeigneter Websockets für die Kommunikation
- Implementierung eines generischen Controllers für das Einbinden diverser Spiele
- Anpassungen von Template-Dateien

Nachdem das `Spiel` mit dem `Frontend` verbunden worden ist, werden verschiedene Möglichkeiten des Deployment in eine ausführbare Umgebung aufgezeigt.

Das Frontend

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

9. Erstellung und Konfiguration einer *Basis-Play-Anwendung*

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

10. Einbindung von *Silhouette* als Authentifikations-Framework

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11. Anmeldung der Nutzer am System

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.1 Konfiguration des Backend Store (PostgreSQL)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.2 Definition des Nutzermodells

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.3 Erstellen einer Datenbank-Evolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.4 Tabellendefinition innerhalb der Anwendung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.5 DAOs für den Zugriff auf die Nutzerdaten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.6 Silhouette Konfiguration auf eigene DAOs umstellen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.7 Konfiguration der Social-Provider

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

11.8 Funktionalität für das Löschen eines Accounts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12. Suchen und Verwalten von Freunden

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.1 Erweiterung des Nutzermodells um einen Nutzernamen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.2 Registrierung der Nutzer mit Nutzernamen und E-Mail

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.3 Evolution und Tabellendefinitionen für Freundeslisten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.4 Funktionalitäten für Freundeslisten in einem DAO

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.5 Erstellen von WebSockets zur dynamischen Interaktion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.5.1 Erstellen des WebSocket auf Basis eines Actors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.5.2 Controller als Endpunkt für das WebSocket

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.5.3 Verbinden der Action innerhalb des Routing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.5.4 Erstellen von Funktionen innerhalb des Javascript, welche mit dem WebSocket zusammen arbeiten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.6 Erweiterung des CSR für WebSockets

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.7 Visualisierung der Freundeslisten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

12.8 Erweiterung der Views zur Übergabe von Skripten und CSS

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13. Migration auf Play 2.6 und Silhouette 5

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.1 Upgrade der benötigten Abhängigkeiten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.2 Anpassungen für das Upgrade von Silhouette

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.3 Änderungen im CustomPostgresDriver

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.4 Neue Controller-Klassen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.5 Von WebJarAssets zu AssetsFinder

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.6 Anpassungen für die Änderungen in i18n

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.7 Impliziter ExecutionContext

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

13.8 Refactoring (Compiler-Warnungen)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

Das Spiel

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

14. Regeln und Spielverlauf

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15. Umsetzung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.1 Grundlegende Datentypen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.2 Operationen auf einem Spielstand

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.3 Operationen auf einem Spielfeld

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.4 Nutzung von Eq (Cats)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.5 Datenbank (Repository)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.6 Zeichnen von Spielfeldern im Client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.7 Hilfsfunktionen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.7.1 Websocket-URL berechnen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.7.2 Feldgröße zum Zeichnen berechnen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.7.3 Berechnen der Klickposition in einem Spielfeld

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.7.4 Logging

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.8 Spielvorbereitung (`Preparation`)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.8.1 Globale Variablen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.8.2 Struktur der HTML-Datei

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.8.3 Funktionen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.8.4 Websocket

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.8.5 Aufruf und Initialisierung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.9 Spielablauf (`Game`)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.9.1 Globale Variablen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.9.2 Struktur der HTML-Datei

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.9.3 Funktionen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.9.4 Websocket

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

15.9.5 Aufruf und Initialisierung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16. Integration ins Frontend

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.1 Verzeichnisstruktur

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.1.1 Aktoren, Controller, DAO und Modelle

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.1.2 View-Templates

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.2 Datenbankschicht (Repository) als DAO

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.3 Websocket

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.3.1 Eine Websocket-Algebra

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.3.2 Komposition zum fertigen Websocket

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4 Controller und Routing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4.1 Übersichtsseite

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4.2 Spielerstellung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4.3 Löschen eines Spielstandes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4.4 Dem Spiel beitreten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4.5 Das Spiel

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4.6 Spielvorbereitung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.4.7 Websocket

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

16.5 Views

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

Deployment (Auslieferung)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

17. Konfiguration für den Produktivbetrieb

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

18. Erstellen eines Artefakts mit allen Abhängigkeiten

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

19. Erstellen von Paketen für Debian

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

19.1 Systemstart-Skripte

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

20. Auslieferung zu einem Cloud Service

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

20.1 Deployment via Remote Repository

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

20.2 Deployment mittels des Plugins sbt-heroku

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

20.3 Datenbankzugriff bei Heroku

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

Erkenntnisse

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

21. Silhouette

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

21.1 Abhängigkeiten von anderen Bibliotheken

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

21.2 Aufwand durch inkompatible Änderungen

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

22. Circe

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

22.1 Erstellung von Codecs

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

22.1.1 Vollautomatische Ableitung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

22.1.2 Halbautomatische Ableitung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

22.1.3 Manuelle Implementierung

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

22.2 Geschwindigkeit des Compilers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

22.3 Fehlerhäufigkeit

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

23. WTFM - Write that fucking manual!

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

23.1 Vorteile für bereits involvierte Entwickler

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

23.2 Vorteile für neue Entwickler

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.

24. Danke

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/comeoutandplay.