Hasso-Plattner-Institut für IT-Systems Engineering

Bachelorarbeit

Eine Architektur für ein ereignisgesteuertes, webbasiertes Backend für Project-Zoom

Tom Bocklisch

tom.bocklischstudent.hpi.uni-potsdam.de

Betreut von Prof. Dr. Holger Giese,
Thomas Beyhl, M.Sc. und
Gregor Berg

Systemanalyse und Modellierung

Potsdam, 21. Juni 2013

Danksagung

Ich bedanke mich ...

Zusammenfassung

Die deutsche Zusammenfassung der Arbeit.

Abstract

The English abstract.

Inhaltsverzeichnis

Zusammenfassung V

Abstract VII

1. Motivation 1
1.1. Zielsetzung von Project-Zoom . . . 1
1.2. Abgrenzung . . . 2
1.3. Gliederung . . . 2

2. Überblick über die Backend-Architektur 3
2.1. Systemanforderungen . . . 3
2.2. Technologieauswahl . . . 4
2.2.1. Webanwendung vs. native Applikation . . . 4
2.2.2. Scala im Web: Play Framework . . . 4
2.2.3. MongoDB als Datenbanksystem . . . 5
2.2.4. Verwendete Bibliotheken . . . 6
2.2.5. Modularisierung . . . 7

3. Data Centric Design 9
3.1. Motivation . . . 9
3.1.1. NoSQL . . . 9
3.1.2. Datenbanktreiber . . . 9
3.1.3. Manipulation von JSON . . . 10
3.2. Idee hinter Data Centric Design . . . 11
3.3. Abgrenzung zum Objektrelationalen Abbildung . . . 11
3.3.1. Objektrelationale Unverträglichkeit . . . 12
3.3.2. Unterschiede zum Data Centric Design . . . 13
3.4. Abgrenzung zu Data Access Objekt . . . 13
3.4.1. Unterschiede zum Data Centric Design . . . 13

Literaturverzeichnis 15

Glossary 17

A. Anhang Kapitel A-1

B. Messdaten zur Evaluation B-1

1. Motivation

Die School of Design Thinking (D-School) lehrt die kreative Herangehensweise an Probleme und die Entwicklung einfallsreicher Lösungen. Die Lehre findet in verschiedenen Kursen statt. Studierenden können sich die Grundlagen im Basic-Track vermitteln lassen und anschließend bei erfolgreicher Teilnahme ihr Wissen im Advanced-Track vertiefen. Neben diesen studentischen Kursen bietet die D-School auch Weiterbildungskurse direkt für Unternehmen an.

In solch einem Kurs wird dann in kleingruppen von ca. 6 Teilnehmern an einem Projekt gearbeitet. Das Projektthema wird dabei entweder von der D-School selbst oder vor allem bei längeren Projekten von externen Projektpartnern vorgeschlagen.

Während der Arbeit an ihren Projekten durchlaufen die Teilnehmer verschiedene Phasen, welche vom Verstehen und Beobachten des Problemfeldes über die Definition eines Standpunktes und das Finden von Ideen bis hin zu einem Prototyp und dessen Tests reichen. Während dieser verschiedenen Phasen fallen unterschiedlichste Dokumente an, welche die Lösungsfindung dokumentieren.

Im Verlauf des Projektverlaufes kommt es durchaus vor, dass ein Team eine Phase mehrmals durchläuft oder in eine vorherige Phase zurückspringt. Dies erschwert die Organisation der Dokumente, z.B. in einer einfachen hierarchischen Struktur. Weiterhin ist es schwierig allein aus den Dokumenten deren Entstehungsreihenfolge und Bedeutung zu erfassen. Meist entstehen während eines 3 Monate andauernden Projektes verschiedenste Präsentationen, Zusammenfassungen, Prototypen, Bilder von Whiteboards und Interviewdokumentationen. Diese werden meist in der von den Studierenden bevorzugten Art und Weise gespeichert und verwaltet, z.B. mit Hilfe von Dropbox, Google Docs oder Box.

Das Verständnis der Dokumentation ist sowohl für das Projekt selbst zum verstehen und erlernen des Prozesses, als auch als Ideenquelle für zukünftige Projekte wichtig. Ferner sind die erstellten Artefakte nützlich zum Werben um neue Projektpartner, welche die Projekte betreuen.

1.1. Zielsetzung von Project-Zoom

Project-Zoom soll der Verbesserung der Dokumentation dienen. Dazu sollen die von den Studierenden erzeugten Artefakte durch manuelles Anordnen in eine Form gebracht werden, welche den Prozess der Gruppe visualisiert. Die Mitarbeiter der D-School kann anschließend diese entstandenen Graphen nutzen um die Projektverläufe zu analysieren und gegebenenfalls den D-School Prozess anpassen.

Für eine reibungslose Integration des Systems ist vor allem die Anbindung an bereits existierende IT-Systeme und die von den Studierenden für das Projekt verwendete Software wichtig. Die Software muss dabei mit moderatem Aufwand angepasst werden können, um andere externe Dienste anbinden zu können.

1.2. Abgrenzung

Die Arbeit beschreibt und bezieht sich auf das Bachelorprojekt „From Creative Ideas to Well-Founded Engineering“ und das umgesetzte Softwaresystem Project-Zoom. An diesem Projekt haben 6 Studierenden gearbeitet und beschreiben in ihren Bachelorarbeiten das Projekt aus verschiedenen Blickwinkeln mit verschiedenen Schwerpunkten.

Tom Herolds Arbeit [Her13] behandelt die Interaktion mit kontextsensitiven Graphen. Es gilt die Interaktion des Studierenden mit der Nutzeroberfläche so intuitiv wie möglich zu gestallten und den Nutzer bei der Erfassung dokumentationsrelevanter Eigenschaften zu unterstützen.

Die Ausführungen von Anita Diekhoff [Die13] beschäftigen sich mit der ...

Norman Rzepka's Bachelorarbeit [Rze13] thematisiert die webbasierte eventgesteuerte clientseitige Architektur von Project-Zoom. Hier wird näher darauf eingegangen, wie die Daten von der DB, über das Backend asynchron an den Client ausgeliefert werden.

Die Bachelorarbeit von Dominic Bräunlein [Brä13] erläutert das generieren und bereitstellen von semantischen Thumbnails, um dem Nutzer das Erkennen der Dokumente seines Projektes zu erleichtern und somit selbst bei wenig verfügbarem Platz so viele Informationen eines Dokumentes anzeigen zu können.

Thomas Werkmeister befasst sich in seiner Arbeit [Wer13] mit der Anbindung externer Systeme zur Integration von Daten. Diese aggregierte Datenbasis ist die Grundlage für die Wissensbasis und die einzelnen Projekte.

Die Komponenten die in den Arbeiten [Wer13] und [Brä13] beschrieben sind, sind mit dem hier erläuterten Systemteil mittels eines Eventsystems verbunden. Das Client-Frontend ist über eine REST Anbindung an das Server-Backend angeschlossen. Mit der clientseitigen Implementierung der REST Schnittstelle beschäftigt sich [Rze13].

1.3. Gliederung

In dieser Arbeit wird zuerst ein Überblick über das Gesamtsystem gegeben. Dazu werden die Anforderungen der D-School an das Backend analysiert und dienen als Grundlage für die Begründung der Verwendeten Technologien. Anschließend wird die Architektur des Backends näher erläutert. Hier liegt der Hauptfokus zunächst auf einer neuen Art und Weise eine Datenbank an eine Webapplikation anzubinden. Dann werden einige Feinheiten und interessante Stellen der Datenmodellierung von Project-Zoom beleuchtet. Den Abschluss bildet die Architektur technische Grundlage für die Anbindung externer Systeme für die Erweiterung des Systems.

2. Überblick über die Backend-Architektur

Im Verlauf des Projektes wurden in Zusammenarbeit mit den D-School Studierenden und dem D-School Staff verschiedene Anforderungen an Project-Zoom als Projektverwaltungs und Dokumentationstool entwickelt [BBD+13]. Aus diesen leiten sich dann die verwendeten Technologien und die umgesetzte Architektur ab. Hier aufgeführt sind nur die für diesen Teil des Projektes wichtigsten und relevanten Anforderungen. Diese beziehen sich Hauptsächlich auf die Erweiterbarkeit des Systems.

2.1. Systemanforderungen

Eine wichtige Anforderung ist die übersichtliche Verwaltung der Artefakte. Hierbei sollen die von den Teams erstellten Dokumente aus der jeweiligen Box für die Studierenden in Project-Zoom zur Verfügung stehen. Diese Verfügbarkeit soll schnellstmöglich nach Hinzufügen einer neuen Datei zur Box gegeben sein. Hierfür ist es sinnvoll ein asynchrones System zu bauen. Neben der Anbindung von Box ist für die Zukunft auch der Zugriff auf facebook, Dropbox und Fileshare vorgesehen. Es zeigt sich, dass ein paralleles, unabhängiges Abfragen der einzelnen Dienste notwendig ist.

Die Studierenden sollen in der Lage sein die Anwendung von außerhalb der D-School verwenden zu können. Dies ist wichtig, da die Studierenden meist nur 2 Tage der Woche in der D-School verbringen. Zum Teil treffen sich die Projekt-Teilnehmer außerhalb der D-School mit Interviewpartnern und Projektpartnern, welche am Stand des Projektes interessiert sind.

Das System muss sowohl erweitert, als auch gewartet werden können. Es ist sehr wahrscheinlich, dass ein Folgeprojekt an diesem Projekt weiterarbeiten wird. Aus diesem Grund soll das System verständlich sein und nach einer Einarbeitung von anderen Entwicklern weiterentwickelt werden.

Neben diesen priorisierten Anforderungen gibt es noch eine Reihe weiterer Punkte, die beim Architekturentwurf berücksichtigt werden müssen:

- Es muss sichergestellt werden, dass kein unautorisierter Nutzer auf Daten Zugriff hat, die geschützt sind.
- Es sollen 100 parallele Anfragen an das Backend pro Sekunde möglich sein.
- Aggregierte Daten sollen nicht gelöscht werden. Es kann durchaus vorkommen das Daten in aggregierten Quellen nach einiger Zeit gelöscht werden müssen. Project-Zoom soll die Daten dann weiterhin vorhalten können.

2.2. Technologieauswahl

2.2.1. Webanwendung vs. native Applikation

Bevor die Implementierung des Projektes beginnen konnte, galt es zu klären welche Technologien verwendet werden. Eine der grundlegenden Entscheidungen war, ob ein Webanwendung oder eine native Anwendung entwickelt wird.

Auf Grund der Anforderung der D-School, dass die Anwendung auch außerhalb der D-School Gebäude verwendbar sein muss, liegt eine Webanwendung nahe. Hinzu kommt, dass die Projekt-Mitglieder bereits mit dem Umgang von Webseiten und deren Navigation vertraut sind. Für die Erweiterbarkeit ist dies ebenfalls ein enormer Vorteil, da ein zentrales System gewartet werden kann und neue Funktionen einfach eingespielt werden können. Zudem dreht sich das Projekt um das Thema Dokumentation, bei der oft dazu geneigt wird sie aufzuschieben. Eine Webseite senkt hier die Hemmschwelle und umgeht die Notwendigkeit einer Verteilung und Installation des Programms.

Eine Trennung der Applikation in Frontend und Backend erlaubt eine klare Funktionstrennung. Für Webapplikationen befindet sich das Backend auf Serverseite und das Frontend befindet sich im Browser auf Clientseite. Die klassischen Aufgaben er beiden Teile sind:

- Backend Aufgaben:
  - Sammeln und zur Verfügung stellen von Daten
  - Kommunikation mit anderen Servern
  - Validierung eingehender Daten vom Client
  - Speicherung von Daten
  - Business Logik
- Frontend Aufgaben:
  - Kommunikation mit dem Backend zur Daten Synchronisation
  - Visualisierung der Daten
  - Interaktion mit dem Nutzer
  - Feedback an den Nutzer

2.2.2. Scala im Web: Play Framework

Die Auswahl der Programmiersprache prägte vor allem der Gedanke eine Sprache zu finden, die einerseits auf benötigte Bibliotheken zugreifen kann und andererseits eine kurze Einarbeitungszeit benötigt. Für die Arbeit mit Thumbnails hat sich die Bibliothek Apache Tika als besonders wertvoll erwiesen. Nähere Informationen zu dieser Bibliothek finden sich in [Brä13]. Da diese Bibliothek in Java geschrieben ist, musste die Wahl auf eine Sprache fallen welche in der Lage ist Java Bibliotheken anzusprechen.

Um die verschiedenen Sprachen und ihr jeweiliges Webframework zu evaluieren wurde sich mit den in Tabelle 2.1 aufgelisteten Kombinationen näher beschäftigt. Dazu wurde von den

	Java + Spring	Groovy + Grails	Scala + Play
Authentifizierung	Ja mit Spring-WS	Ja mit Authentication Plugin	Ja mit SecureSocial
Json Unterstützung	Ja mit Jersey	Ja	Ja
Vorhandene Erfahrung im Entwicklerteam	Gering	Gut	Sehr Gut
Dokumentation	Gut	Gut	Gut
Asynchron	Nein	Nein	Ja
Zustandslos	Nein	Nein	Ja
Produktiv Einsatz	Einsatz	Trial	Trial
Wichtigkeit	Am wichtigsten	Wichtig	Wichtiger

Tabelle 2.1. – Vergleich von verschiedenen Webframeworks miteinander

Backend-Entwicklern jeweils eine kurze Hello-World Anwendung aufgesetzt. Dadurch konnte festgestellt werden, ob der Arbeitsablauf, der Aufwand und der zu erzeugende Quellcode angemessen ist. Neben dieser praktischen kurz Evaluierung wurden einige Fakten zusammengetragen, um die Entscheidung zu erleichtern.

In die Nähere Auswahl kamen Java, Groovy und Scala mit den jeweiligen Webframeworks Spring MVC, Grails und Play. Im Gegensatz zu Spring und Grails ist Play ein zustandsloses, schlankes Framework was eine ideale Voraussetzung für ein REST Backend darstellt. Zusätzlich spielten die persönlichen Präferenzen der Entwickler eine Rolle, die bereits Erfahrung in der Kombination Scala und Play hatten.

Scala ist eine Programmiersprache deren Syntax stark an Java orientiert ist. Programme welche in Scala geschrieben sind können bidirektional mit Java Code interagieren und so auf den vollen Funktionsumfang der Java Bibliotheken zurückgreifen. Dabei werden in der Sprache Konzepte der funktionalen mit gewohnten Elementen der objektorientierten Programmierung verknüpft. Es eignet sich somit gut für einen allmählichen Einstieg in die funktionale Programmierung. Die Lernkurve hängt stark von den Vorkenntnissen in Java ab. Durch die starke Ähnlichkeit ist ein Umstieg einfach. Die Verwendung der hinzugekommenen funktionalen Aspekte der Programmiersprache erfolgen nach und nach. Hier empfiehlt es sich frei verfügbare Bücher wie [Ode11] zu nutzen.

2.2.3. MongoDB als Datenbanksystem

Während der Prototypen Phasen, in der wir die Idee der Umsetzung verfeinerten, stellten wir ein Daten Modell auf. Dieses Modell wird im späteren Verlauf der Arbeit näher beleuchtet. Ein wichtiger Aspekt, der für die Wahl der Datenbank entscheidend ist, sind die verschiedenen anbindbaren externen Systeme. Das Daten Modell muss hier sicherstellen, das Informationen die aus allen Datenquellen verfügbar sind einfach zugänglich sind. Gleichzeitig muss es dafür sorge tragen, dass keine Informationen verloren gehen. Aus diesen Gründen fiel die Entscheidung auf eine NoSQL Datenbank, welche die benötigte Flexibilität ohne großen Aufwand ermöglicht.

Neben MongoDB stand Apache CouchDB als Alternative zur Verfügung. Die Wahl von MongoDB als persistenten Speicher für Project-Zoom fiel vorrangig aufgrund des sehr guten Daten-

bank Treibers ReactiveMongo für Scala. Dieser erlaubt komplett asynchrone Datenbankzugriffe und gliedert sich deshalb perfekt in das asynchrone Play Framework ein.

Als Dokumentorientierter Datenspeicher passt MongoDB sehr gut zu einer REST Architektur. Die Daten, welche in der Datenbank abgelegt werden, werden im Binary JSON (BSON) format gespeichert. Dieses BSON format ist eine binär encodierte serialisierung von JSON ähnlichen Dokumenten. BSON unterstützt die repräsentation aller Datentypen, welche auch von JSON unterstützt werden, und erlaubt zusätzlich weitere Datentypen. Erweitert wurden die JSON Datentypen unter Anderem um die Unterstützung von Binärdaten und Datumsangaben [Dir10]. Somit ähnelt die Datenform, die auf Clientseite verarbeitet wird, dem Format der Datenspeicherung in der Datenbank. Diese Konstellation erlaubt die Implementierung einer schlanken Model Schicht, wie sie im Abschnitt Json-Coast-To-Coast 3 erklärt wird.

2.2.4. Verwendete Bibliotheken

Die Umfangreichsten Bibliotheken, welche im Project-Zoom Backend Core verwendet werden sind Akka, Play und SecureSocial. Nicht hier aufgeführt ist ReactiveMongo als Datenbanktreiber. Diese Bibliothek wird im Abschnitt XXXXX näher erläutert.

Akka[1] stellt die Grundlage für das Play Framework dar. Die Library ermöglicht die Nutzung verschiedenester Konzepte des asynchronen Programmierens. Das Ziel ist das schreiben von parallelen, fehler-toleranten und skalierbaren Anwendungen zu vereinfachen [Akk12].

Die sogenannten *Actors* der Library sind für dieses Projekt am relevantesten. Sie stellen eine Implementierung des Aktorenmodells dar. Aktoren sind abgeschlossene Einheiten, welche nur über Nachrichten kommunizieren. Dabei erfolgt die Abarbeitung der Nachrichten eines Aktors sequenziell, die Kommunikation mit anderen Aktoren aber asynchron. Dadurch können mehrere Nachrichten in unterschiedlichen Aktoren gleichzeitig abgearbeitet werden. Verschiedene Aktoren teilen sich nur über Nachrichten ausgetauschte Variablen. Damit das System also Thread sicher arbeitet, müssen diese Nachrichten Thread sicher sein. In Scala ist es deshalb üblich für Nachrichten *Case Classes*[2] zu verwenden. Diese sind von vorneherein unveränderbar und somit Thread sicher.

Neben *Actors* finden auch *Agents* in Project-Zoom Verwendung. Ein Agent bildet eine Kapselung um einen Status. Dieser Status kann unverzüglich synchron gelesen und asynchron überschrieben werden. Bei einem Update wird dem Agent eine Funktion übergeben, welche den neuen Status des Agents berechnet. Einen Agent kann man zum Beispiel verwenden um Status zwischen verschiedenen *Actors* zu teilen.

Play Framework 2.0[3] ist die Weiterentwicklung und Portierung eines früheren Java Web Frameworks nach Scala. Es ist zwar in Scala programmiert, kann aber sowohl mit Java als auch Scala als Backend Sprache verwendet werden. Play liegt ein Model-View-Controller (MVC) Architektur zugrunde. Hierbei werden in jedem Controller sogenannte Actions definiert und im View verschiedene Templates angelegt. Der Ablauf einer Anfrage an den Webserver verläuft wie folgt:

[1] http://akka.io
[2] `http://www.scala-lang.org/node/107`
[3] `http://playframework.com`

1. Der default HTTP-Router leitet die Anfrage an eine Action weiter. Diese Weiterleitung basiert auf der Routen die in der Datei conf/routes definiert sind.

   ```
   GET /projects/:id controllers.ProjectController.read(id: String)
   ```

   Eine Route besteht dabei immer aus der HTTP Methode (GET, POST, HEAD, PUT, DELETE)[Pla13], einem URI Pattern und einer Action die den Request beantwortet.

2. Die Action im Controller ist für die Beantwortung zuständig. Dazu können Informationen im Model abgefragt werden. Die Templates können benutzt werden um ein dynamisches Ergebnis für den Client zu erzeugen.

SecureSocial[4] handelt den User login. Dieses Paket ist ein Authentifizierungsmodul mit Support für OAuth, OAuth2, OpenID und Username/Passwort Authentifizierung.

2.2.5. Modularisierung

Play erlaubt die Modularisierung des Codes in sogenannte Subprojekte. Ein solches Subprojekt ist dabei eine abgeschlossene Einheit. Diese kann alleine kompiliert, getestet und ausgeführt werden. Dabei können neben Play-Projekten auch Java oder Scala Projekte als Subprojekte verwendet werden. Die Einzelnen Projekte und deren Abhängigkeiten werden in der *Build.scala* Datei angegeben. Project-Zoom besteht aus drei verschiedenen Subprojekten:

common enthält den Code der sowohl von den Projekten *main* als auch *admin* benötigt wird. In diesem Projekt sind die Datenmodelle definiert und hier befindet sich auch das Eventsystem und die Erweiterungen zum Daten aggregieren und zum Thumbnail generieren. Weiterhin befindet sich in diesem Modul die Authentifizierung.

main schließt alle Controller ein, die für den normalen Nutzer ansprechbar sind. Es wurden verschiedenen Actions definiert und für die jeweiligen Sichten Templates angelegt. Der Frontend Code, welcher näher in den Arbeiten [Rze13, Her13, Die13] beschrieben ist, wird ebenfalls in diesem Projekt verwaltet.

admin definiert jede Interaktion, die nur für privilegierte Nutzer sichtbar sein soll. Dies sorgt für eine klare Trennung zwischen User und Admin Anfragen und sorgt somit für mehr Sicherheit. Ein weiterer Vorteil ist, das durch das abschalten dieses Subprojektes jedwede Admin Aktion unterbunden werden kann.

In der Grafik (????) sind die Abhängigkeiten der Pakte voneinander dargestellt. Das Anlegen eines Hauptprojektes, in diesem Fall *Project-Zoom*, erleichtert die Arbeit mit dem gesamt Projekt. Durch die Abhängigkeit zu *main* und *admin* muss nur noch das Hauptprojekt kompiliert werden und die Abhängigkeiten werden bei Source-Code Änderungen automatisch mit kompiliert.

[4] `http://securesocial.ws`

3. Data Centric Design

In diesem Kapitel wird die grundlegende Architektur der Anbindung eines persistenten Speichers im Core von Project-Zoom näher erläutert. Dieser Befindet sich im *Common* Subprojekt im Paket *projectZoom.core* und *models*. Das gewählte Design hat vorrangig Auswirkungen auf die Art und Weise wie der Code für Datenmodelle geschrieben wird, erstreckt sich aber als Betrachtungsweise über die gesamte Architektur.

3.1. Motivation

Das JSON Coast-To-Coast Design als Umsetzung des Data Centric Designs ist erstmals zusammenhängend und beispielunterlegt dargestellt wurden durch Pascal Voitet in seinem Blog Mandubian [Voi13]. Voitet ist selbst Mitwirkender am open Source Projekt Play und aktiver Scala Bibliotheken Autor (*play-reactivemongo*[1], *play-autosource*[2]).

Ein Webapplikation Backend ist zunehmend eine Verbindung zwischen verschiedenen anderen Backends und Frontends. Diese Entwicklung geht auf die Bereitstellung von APIs für viele große und kleine im Web verfügbare Dienste zurück. Anwendungen, welche Daten vorrangig aus anderen Quellen aggregieren, sind zum großen Teil mit der Datenmanipulation beschäftigt. Hier bietet sich ein Data Centric Design an. Für das Design grundlegend sind drei Entwicklungen. Zum einen das Aufstreben von NoSQL Datenbanken und zum anderen asynchrone Datenbanktreiber mit einfachen Methoden zur JSON Manipulation.

3.1.1. NoSQL

Die Entwicklung der Datenbank Management Systeme bestand viele Jahre lang in der Optimierung und Verbesserung bestehender relationaler Datenbank Modelle. Im Jahr 1998 kam dann der Term Not-only SQL (NoSQL) auf [LM10]. Heute gibt es verschiedene etablierte Datenbanken die keine reinen SQL Datenbanken mehr sind, bekannte Beispiele sind MongoDB, CouchDB, Apache Cassandra. MongoDB ist eine Open-Source Datenbank, welche ihre Daten Dokumenten basiert speichert.

3.1.2. Datenbanktreiber

Für die Anbindung von relationalen Datenbanken gibt es in Java die Java Database Connectivity (JDBC) Schnittstelle. Diese abstrahiert über Datenbanken und deren Treiber indem eine einheitliche API angeboten wird. Ausgerichtet ist JDBC auf relationale Datenbanken.

Um NoSQL Datenbanken anzubinden benötigt man, wie bei JDBC, einen eigenen Treiber. Der Unterschied ist, das es hier keine Abstraktionsebene über verschiedene NoSQL Datenbanken

[1] `https://github.com/zenexity/Play-ReactiveMongo`
[2] `https://github.com/mandubian/play-autosource`

gibt. Dies liegt vorrangig an der sich stark unterscheidenden Struktur der einzelnen Speichersysteme. Die unterschiedlichen NoSQL Datenbanken sind jeweils speziell auf eine bestimmte Aufgabe ausgerichtet, wie z.B. Durchsatz, Verteilte Umgebungen oder Flexible Daten Schemas.

ReactiveMongo ist ein asynchroner Datenbanktreiber für MongoDB und die Programmiersprache Scala. Die Vorteile eines asynchronen Treibers liegen auf der Hand: Für jede synchrone Datenbank Abfrage wird normalerweise ein Thread verwendet, der bis zur Antwort blockiert ist. Bei mehreren Datenbankabfragen pro Request werden bei Last viele Threads benötigt um Datenbank abfragen auszuführen. Asynchrone Treiber umgehen dieses Problem indem sie Threads, welche Datenbank Abfragen ausführen, nicht blockieren. Für dieses Konzept ist es Notwendig Platzhalter einzuführen. Diese ersetzen das Ergebnis solange es noch nicht vorhanden ist.

Quelltext 3.1 – Funktionssignatur für asynchronen Datenbankzugriff

```
def findOneById(bid: BSONObjectID): Future[Option[Graph]]
```

Im Beispiel XXX zeigt die Funktionssignatur den Rückgabetyp *Future[Option[Graph]]*. *Future* ist hierbei genau dieser Platzhalter für ein Ergebnis welches noch nicht existiert. Arbeiten kann man mit einem *Future* durch das Festlegen von Callbacks.

3.1.3. Manipulation von JSON

Das Data Centric Design basiert auf der Idee der direkten Manipulation von Daten. Für eine Umsetzung des Data Centric Designs mit Hilfe von JSON benötigt man somit eine Bibliothek, welche die Möglichkeit bietet, JSON zu transformieren und zu validieren.

Ein Beispiel für solch eine Bibliothek ist Play-JSON. Sie erlaubt das Validieren, Transformieren und Serialisieren. Ein Beispiel ist im Quelltextauszug 3.2 zu sehen. Dort wird ein JSON Objekt erzeugt, welches anschließend zuerst Transformiert wird indem das Attribut *role* angehangen wird und danach Validiert wird. Bei der Validierung wird überprüft ob der User über 18 Jahre alt ist.

Quelltext 3.2 – Play JSON Transformations / Validierungs Beispiel

```
val json = Json.obj(
  "firstName" -> "max", "latsName" -> "Mustermann", "age" -> 17)

val addUserRole = JsPath.json.update(
  (JsPath \ "role").json.put(JsString("User")))
// addUserRole : Reads[JsObject]

val isMajor =
  (JsPath \ "age").read[Int](min(18))
//> isMajor : Reads[Int]

json.transform(addUserRole)
//> res0: JsResult[JsObject] =
//     JsSuccess({
//       "firstName":"max",
//       "latsName":"Mustermann",
```

```
//         "age": 17,
//         "role": "User"},)

json.validate(isMajor)
//> res1: JsResult[Int] =
//     JsError(List((
//       /age, List(
//         ValidationError(validate.error.min, WrappedArray(18))))))
```

3.2. Idee hinter Data Centric Design

Das Ziel im Data Centric Design ist die direkte Manipulation von Daten. Der Fokus des Designs liegt auf der Daten Manipulation und dem Datenfluss. Wie bereits in 3.1 beschrieben, sind Webapplikation Backends immer Häufiger Knotenpunkte in einem mehrere Server durchlaufenden Datenfluss. Ein Teil dieses Datenflusses ist die Kommunikation des Backends mit der Datenbank und dem Client 3.1.

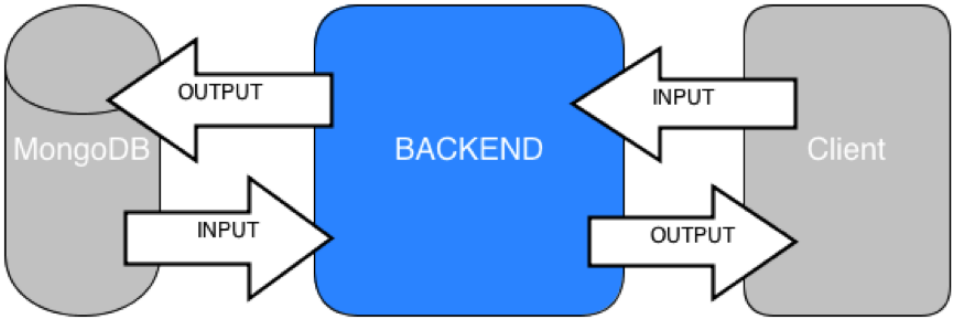

Abbildung 3.1. – Datenfluss zwischen Datenbank, Backend und Client

In diesem Datenfluss soll es keine implizite Umwandlung der Daten in Objekte der Programmiersprache geben. Für ein einfaches durchleiten der Daten aus der Datenbank zum Client oder von einer anderen Webressource zum Client eine Umwandlung nicht notwendig und sinnvoll ist.

Eine Deserialisierung der Daten in Objekte ist für komplizierte Business Logik von Vorteil. Statische Daten Modelle sollten immer dann verwendet werden, wenn die Manipulation der Daten kompliziert wird oder wenn die Verwendung externer Bibliotheken eine Objekt Repräsentation benötigt.

Nach Voitot ist die Frage also nicht, statische Daten Modelle zu vergessen, sondern sie nur dann zu benutzen, wenn sie notwendig sind. Einfache und dynamische Strukturen sollten so oft es geht erhalten bleiben [Voi13].

3.3. Abgrenzung zum Objektrelationalen Abbildung

Das Object-relational Mapping (ORM) bezeichnet man auch als All-Model-Approach. Hierbei erfolgt die Kommunikation mit der Datenbankschnittstelle über Objekte der Programmiersprache. Abfragen an die Datenbank resultieren in Objekten der Programmiersprache. Der Fokus

der Objektrelationalen Abbildung liegt in der Überführung von Objekten der Objektorientierten Programmierung in ein relationales, tabellenorientiertes Schema [Amb03].

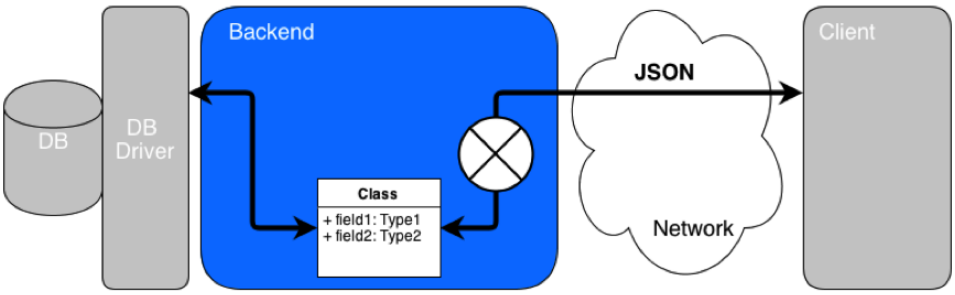

Abbildung 3.2. – Datenfluss einer Applikation bei Verwendung eines ORM Frameworks

Der Hauptvorteil eines ORM liegt in der Abstraktion von der Datenbank. Für die Business Logik gibt es keinen Unterschied zwischen Objekten aus der Datenbank und Objekten der Programmiersprache. Die Datenbank kann meist ohne Änderungen an der Applikation ausgetauscht werden.

3.3.1. Objektrelationale Unverträglichkeit

Ein ORM bringt einige konzeptionelle Probleme mit sich. Diese werden im allgemeinen in der Literatur als *Object-relational Impedance Mismatch* (ORIM) bezeichnet. Die objektrelationale Unverträglichkeit resultiert aus den Unterschieden im zugrunde liegenden Konzept, sowie Differenzen in der Darstellung und den Erwartungen des Entwicklers [Bow10].

Folgende Herausforderungen ergeben sich für die praktische Arbeit mit einem ORM:

- Die Abbildung der Relationen zwischen den Objekten ist komplex. Es müssen One-to-One, One-to-Many und Many-to-Many Abhängikeiten ins Relationale Schema übertragen werden.
- Ein weiterer wichtiger Punkt ist die Abbildung von Hierarchien und Vererbungen aus der Objekt Orientierten Programmierung in das physische Daten Modell.
- Innerhalb des ORM muss es ein Objekt Caching geben. Dieses ist notwendig um die Illusion zu erzeugen, dass es sich um ein Objekt der Programmiersprache handelt. Denn wird ein Objekt mehrmals aus der Datenbank abgefragt, so muss ein referenz-gleiches Objekt zurückgegeben werden [HVE03].
- Der Entwickler muss die ORM Limitierungen die sich aus dem ORIM ergeben akzeptieren. Die Probleme müssen bekannt sein, um sie bereits bei der Daten Modellierung zu umgehen [Atw06].

Ted Neward, Autor von „The Busy Java Developers Guide to Scala“, macht den frühen Erfolg von ORMs und die Erleichterungen, welche die ersten ORMs mit sich brachten, dafür verantwortlich, dass ORMs verwendet werden ohne deren Limitationen zu beachten. So kommt es zu Mehraufwand an Zeit und Energie für die Erhaltung und Anpassung an alle Nutzungsfälle [Atw06].

3.3.2. Unterschiede zum Data Centric Design

Das Data Centric Design ist viel stärker an die Datenbank gebunden und deren Austausch ist deutlich schwieriger. Dies sorgt aber dafür, dass keine Illusion einer abstrahierten Datenbank entsteht. Da das Data Centric Design ein Ansatz mit funktionalem Hintergrund ist, sind die Datenstrukturen in der Regel unveränderbar. Ein Caching zur Sicherung der Objektidendität ist daher nicht notwendig. Die Objektgleichheit ist in diesem Fall über Wertgleichheit definiert (vgl. Scala Case Klassen [Sca08]). Im Gegensatz zum ORM muss der Entwickler sich beim Data Centric Design selbst um die Darstellung und Umwandlung seiner Daten in das Datenbank Format kümmern. Durch die Verwendung einer objekt- oder dokumentenbasierten Datenbank ist diese Umwandlung einfacher und mit weniger Problemen behaftet als eine Konvertierung in ein relationales Schema.

3.4. Abgrenzung zu Data Access Objekt

Implementierungen eines *Data Access Object* (DAO, oft auch *Data Access Procedure*) basiert auf dem *Data Access Object* Pattern [ACM03]. Das Pattern hat das Ziel, den Datenzugriff und die Datenmanipulation in einen separaten Layer auszulagern. Es ergibt sich eine Kapselung der Datenbankzugriffe an einem Ort der Applikation. Bei der Umsetzung gibt es se Möglichkeit ein DAO zur Abstraktion der Datenbank als ganzes zu verwenden oder für jede einzelne Datenbankstruktur ein DAO anzulegen.

Der Hauptfokus liegt in der Abstrahierung der Datenbank. Bei Änderungen an der Datenquelle müssen in der Regel nur die Data Access Objects angepasst werden, der Anwendungscode bleibt in der Regel unberührt. Die Aufgabe der Serialisierung der Daten von den Strukturen der Anwendung in Strukturen der Datenbank sind nicht Aufgabe des DAO sondern des Data Transfer Object (DTO) [3].

Im DAO findet über einen Datenbank Treiber direkter Zugriff auf die Datenbank statt. Man kann dieses Objekt somit als Mittelsmann zwischen Datenbank und Applikation sehen. Durch die direkte Kommunikation mit der Datenquelle können alle unterstützten Funktionen ausgenutzt werden.

3.4.1. Unterschiede zum Data Centric Design

[3] vgl. J2EE Patterns – Data Transfer Object `http://www.oracle.com/technetwork/java/transferobject-139757.html`

Literaturverzeichnis

[ACM03] Deepak Alur, John Crupi, and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall / Sun Microsystems Press, USA, California, June 2003.

[Akk12] Akka. What is akka? Webseite, 2012. Erreichbar unter: `http://doc.akka.io/docs/akka/2.0/intro/what-is-akka.html`; besucht am 19. Juni 2013.

[Amb03] Scott W. Ambler. Mapping objects to relational databases. Webseite, 2003. Erreichbar unter: `http://www.agiledata.org/essays/mappingObjects.html`; besucht am 15. Mai 2013.

[Atw06] Jeff Atwood. Object-relational mapping is the vietnam of computer science. Webseite, 2006. Erreichbar unter: `http://www.codinghorror.com/blog/2006/06/object-relational-mapping-is-the-vietnam-of-computer-science.html`; besucht am 10. Juni 2013.

[BBD+13] Dominic Bräunlein, Tom Bocklisch, Anita Diekhoff, Norman Rzepka, Tom Herold, and Thomas Werkmeister. Software requirements specification. 2013.

[Bow10] David Bowers. Object relational database design – impedance mismatch or expectation mismatch? Webseite, 2010. Erreichbar unter: `http://www.dmsg.bcs.org/web/images/stories/PDFs/2010-10-13-david-bowers.pdf`; besucht am 10. Juni 2013.

[Brä13] Dominic Bräunlein. Generierung und bereitstellung von semantischen thumbnails aus heterogenen daten für project-zoom, 2013.

[Die13] Anita Diekhoff. Automatisches layouten in interaktiven graphen, 2013.

[Dir10] Mike Dirolf. Bson. Webseite, 2010. Erreichbar unter: `http://bsonspec.org`; besucht am 19. Juni 2013.

[Her13] Tom Herold. Kontextsensitver assistent für interaktive graphen, 2013.

[HVE03] Anders Hejlsberg, Bill Venners, and Bruce Eckel. Inappropriate abstractions - a conversation with anders hejlsberg, part vi. Webseite, 2003. Erreichbar unter: `http://www.artima.com/intv/abstract3.html`; besucht am 10. Juni 2013.

[LM10] Adam Lith and Jakob Mattsson. Investigating storage solutions for large data. Master's thesis, Chalmers University Of Technology, 2010.

[Ode11] Martin Odersky. *Scala By Example*. EPFL, Switzerland, May 2011.

[Pla13] Play. Http routing. Webseite, 2013. Erreichbar unter: `http://www.playframework.com/documentation/2.1.1/ScalaRouting`; besucht am 19. Juni 2013.

[Rze13] Norman Rzepka. The design of an web-based event-driven client application architecture for project-zoom, 2013.

[Sca08] Scala. A tour of scala: Case classes. Webseite, 2008. Erreichbar unter: `http://www.scala-lang.org/node/107`; besucht am 17. Juni 2013.

[Voi13] Pascal Voitet. Json coast-to-coast. Webseite, 2013. Erreichbar unter: `http://mandubian.com/2013/01/13/JSON-Coast-to-Coast/`; besucht am 21. März 2013.

[Wer13] Thomas Werkmeister. Extensible backend plugin architecture for data aggregation of heterogeneous sources for project-zoom, 2013.

Glossary

Box Cloud storage and collaboration solution for enterprises. 3

Anhang A.

Anhang Kapitel

Anhang B.

Messdaten zur Evaluation

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und dafür keine anderen als die genannten Quellen und Hilfsmittel verwendet habe.

Tom Bocklisch

Potsdam 21. Juni 2013