



UNIVERSITÄT LEIPZIG

Digital Text Forensics Search

Information Retrieval

WS 2017/2018

Betreuer: Martin Potthast
martin.potthast@uni-leipzig.de

Autor: David Drost
dd42cequ@studserv.uni-leipzig.de
Matrikelnummer:

Autor: Edward Kupfer
ek96foje@studserv.uni-leipzig.de
Matrikelnummer:

Autor: Hendrik Sawade
hs34byhe@studserv.uni-leipzig.de
Matrikelnummer: 3745956

Autor: Tobias Wenzel
tw54byka@studserv.uni-leipzig.de
Matrikelnummer: 3733301

Abgabedatum: Leipzig, den 2. März 2018

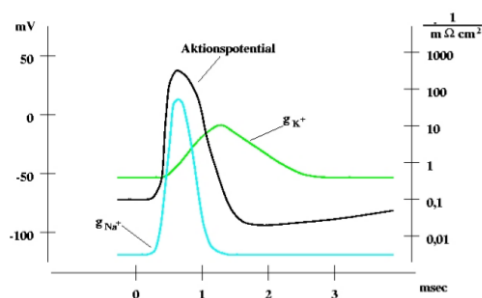
Inhaltsverzeichnis

1	Einleitung	2
2	Vorverarbeitung	3
3	Indexierung	4
4	Backend	5
4.1	Auswahl eines Backend-Frameworks	5
4.2	Auswahl einer Datenbank	6
4.3	Erstellung des Backends	7
4.3.1	Kommunikation mit der Datenbank	7
4.3.2	Controller	8
5	Evaluation und Fazit	9

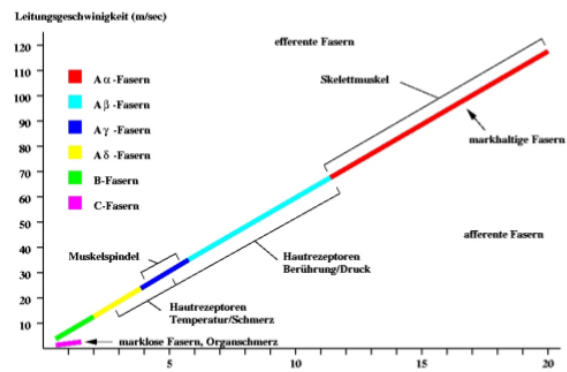
1 Einleitung

Hier schreibt man dann was. Und so . Offenbar sieht das nicht so schön aus.

kommentiere
ihre Sa-
chen.



(a) Pyramidenzelle



(b) Neocortex

Abbildung 1.1: Links: AP. Rechts: Leitungsgeschwindigkeit von verschiedenen Fasern.

2 Vorverarbeitung

3 Indexierung

4 Backend

4.1 Auswahl eines Backend-Frameworks

Zuerst wurde für die Programmierung einer Webanwendung, um besser abstrahieren und Programmier-Paradigmen umsetzen zu können, ein geeignetes Framework gesucht. Hierfür wurden zahlreiche Frameworks mit der Unterstützung von Dependency Injection (DI) und Aspect-Oriented Programming (AOP) untersucht. Aufgrund der Auswahl der Programmiersprache Java für die Umsetzung der Anwendung schränkte sich die Anzahl der Frameworks ein. Die Recherche ergab folgende vier Frameworks:

Frameworks	Sprache	Eigenschaft
PicoContainer	Java	DI, AOP, schlank
HiveMid	Java	DI, AOP-ähnliches Feature, IoC Container
Google Guice	Java	DI, AOP, IoC Container, Annotations, Generics, modular
Spring	Java	DI, AOP, IoC Container, Annotations, Generics, modular

Tabelle 4.1: DI-Frameworks

HiveMind und Guice bieten leichter verständliche Programmierungstechniken sowie einen prägnanteren und lesbaren Code gegenüber Spring. HiveMind fokussiert sich auf das Verbinden von Services. Seine Konfiguration erfolgt über eine XML-Datei oder eine eigene Definitions-Sprache. Hierdurch ist HiveMind ein kleiner und simpel gestalteter DI-Container. Guice hingegen unterstützt die neuen Features, wie Annotations und Generics die ab Java 1.5 zur Verfügung stehen. Dies hilft dabei, eine weitgehend aufgeräumte und einfache Konfiguration zu ermöglichen. HiveMind bietet zusätzlich die Möglichkeit, mit AOP zu arbeiten. Guice und Spring bieten sehr ähnliche Ansätze und kommen mit vielen Anforderungen, die Unternehmenssoftware erfüllen müssen, zurecht. Spring's Modularität bietet dabei einen größeren Vorteil. Die Module können in Spring, je nachdem welche Module der Entwickler benötigt, ohne viel Aufwand hinzugefügt werden. Guice ist durch

die geringere Komplexität zwar leichter zu verstehen und insgesamt kleiner als Spring. Dieser Punkt kann aber vernachlässigt werden, da durch die Modularität von Spring das Framework nach den Voraussetzungen des Entwicklers zusammengestellt werden kann. Schlussendlich wurde sich für das Spring-Framework entschieden, da die Flexibilität und Modularität von Spring die Entwicklung von Anwendungen stark vereinfacht und daher die beste Wahl darstellt.

4.2 Auswahl einer Datenbank

Als nächstes wurde für die spätere Speicherung der Interaktion zwischen Backend, Suchergebnissen und User, also das User-Feedback eine geeignetes eingebettetes Datenbanksystem gesucht. Die Recherche ergab folgende Datenbanken:

Datenbank	Sprache	Eigenschaft
SQLite	C	SQL-92-Standard, Transaktionen, Unterabfragen (Subselects), Sichten (Views), Trigger und benutzerdefinierte Funktionen, direkte Integrierung in Anwendungen, In-Memory-DB
H2	Java	Schnell, Referenzielle Integrität, Transaktionen, Clustering, Datenkompression, Verschlüsselung und SSL ,kann direkt in Java-Anwendungen eingebettet oder als Server betrieben werden, Direkte Unterstützung in Spring, In-Memory-DB
Apache Cassandra	Java	Spaltenorientierte NoSQL-Datenbank, Für sehr große strukturierte Datenbanken, Hohe Skalierbarkeit und Ausfallsicherheit bei großen, verteilten Systemen

Tabelle 4.2: Datenbanken

SQLite bietet einen leichten Einstieg in die SQL-Welt. Dabei bietet es den größten Teil des SQL-92-Standards und kann Transaktionen, Unterabfragen und viele weitere Funktionen. Außerdem ist es eine In-Memory-DB. Leider unterstützt Spring Boot diese Datenbank nicht von Haus aus und es müssten Anpassungen in der Konfiguration vorgenommen werden. Apache Cassandra ist eine Spaltenorientierte NoSQL-Datenbank und ist für große strukturierte Daten, hohe Skalierbarkeit und Ausfallsicherheit ausgelegt. Da diese Datenbank für große Datenmengen ausgelegt ist, ist diese Datenbank für dieses Backend zu groß ausgelegt. H2 ist eine In-Memory-DB, welche schnell ist, Referenzielle Integrität,

Transaktionen, Clustering, Datenkompression unterstützt. Außerdem kann Spring Boot mit dieser Datenbank ohne besondere Maßnahmen verwendet und in diese Anwendung integriert werden. Daher wurde sich Schlussendlich für die H2 Datenbank entschieden.

4.3 Erstellung des Backends

Nach der Auswahl der Backendtechnologien wurde die Grundarchitektur des Backends konzipiert und implementiert.

4.3.1 Kommunikation mit der Datenbank

Zunächst wurden Datenmodels, wie die Query oder LoggingDocument erstellt. Hieraus werden später die Tabellen der Datenbank generiert. Um mit der Datenbank kommunizieren zu können, gehört die Erstellung von DAOs dazu, welche die Kommunikationsschnittstelle bilden. Ein DAO generiert eine SQL Query und übermittelt diese an die Datenbank.

```
1 public interface LoggingDocDao extends JpaRepository<
    LoggingDocument, Long> {
2     LoggingDocument findByDocId(Long docId);
3 }
```

Beispiele hierfür sind die Speicherung und Abrufen von LoggingDocument Daten, welche ein Teil des User-Feedbacks darstellen. In dem zu betrachtenden Code Ausschnitt sieht man wie mit Hilfe von Spring die SQL Query findByDocId von einem LoggingDocument generiert wird. Dies findet über den Namen eines Interfaces statt. Die einzelnen Komponenten, welche implementiert wurden, kommunizieren nicht direkt über die DAOs mit der Datenbank, sondern dazwischen befindet sich noch ein Abstrahiertes Service Interface. Dadurch ist eine lose Kopplung zwischen den Komponenten, DAO und Datenbank möglich. Dies dient dazu, dass die Datenbank ohne große Änderungen in den Implementierungen austauschbar ist. Es ist dadurch nur notwendig, in den Konfigurationen und eventuell in den DAOs Änderungen vorzunehmen. Ein Beispiel für einen Service ist der UserLoggingService.

```
1 public class LoggingDocServiceImpl implements
    LoggingDocService {
```


4 Backend

```
2 public LoggingDocument findById(Long id) {  
3     return loggingDocDao.findOne(id);  
4 }
```

In der Implantation des Interfaces wird nun das DAO aufgerufen. Wie hier im Beispiel zu sehen die Methode findById.

4.3.2 Controller

Die nächste Schritt ist es sogenannte Controller zu erstellen. Diese bilden eine wichtige Schnittstelle für die Kommunikation mit dem Frontend und dem Backend. Controller reagieren auf HTTP-Requests, welche von dem Frontend oder anderen Clients gesendet werden. Die Aufgabe ist es für die bestimmten Ressource-URLs bestimmte Ereignisse auszuführen. Ein Beispiel hierfür ist die Suchanfrage der Search Zeile im Frontend auszuwerten und die Suchergebnisse zurück zu senden.

```
1 @RequestMapping(method = RequestMethod.GET, path = "/")  
2 public ModelAndView searchPage(  
3     @RequestParam(defaultValue = "")  
4     String query) {  
5     ModelAndView modelAndView = new ModelAndView("search");  
6     ...  
7     List<ScoreDoc> list = querySearcher.search(query);  
8     ...  
9     modelAndView.addObject("searchResultPage", searchResultPage);  
10    return modelAndView;}
```

In diesem Code Beispielabschnitt sieht man, dass wenn ein Request bei der Path-URL „/“ ausgelöst wird, wird die Funktion SearchPage aufgerufen und eine Suche ausgeführt. Hierfür wird der RequestParameter query ausgewertet, gesucht und anschließend die Suchergebnisse in einem modelAndView Objekt dem Frontend übergeben.

5 Evaluation und Fazit