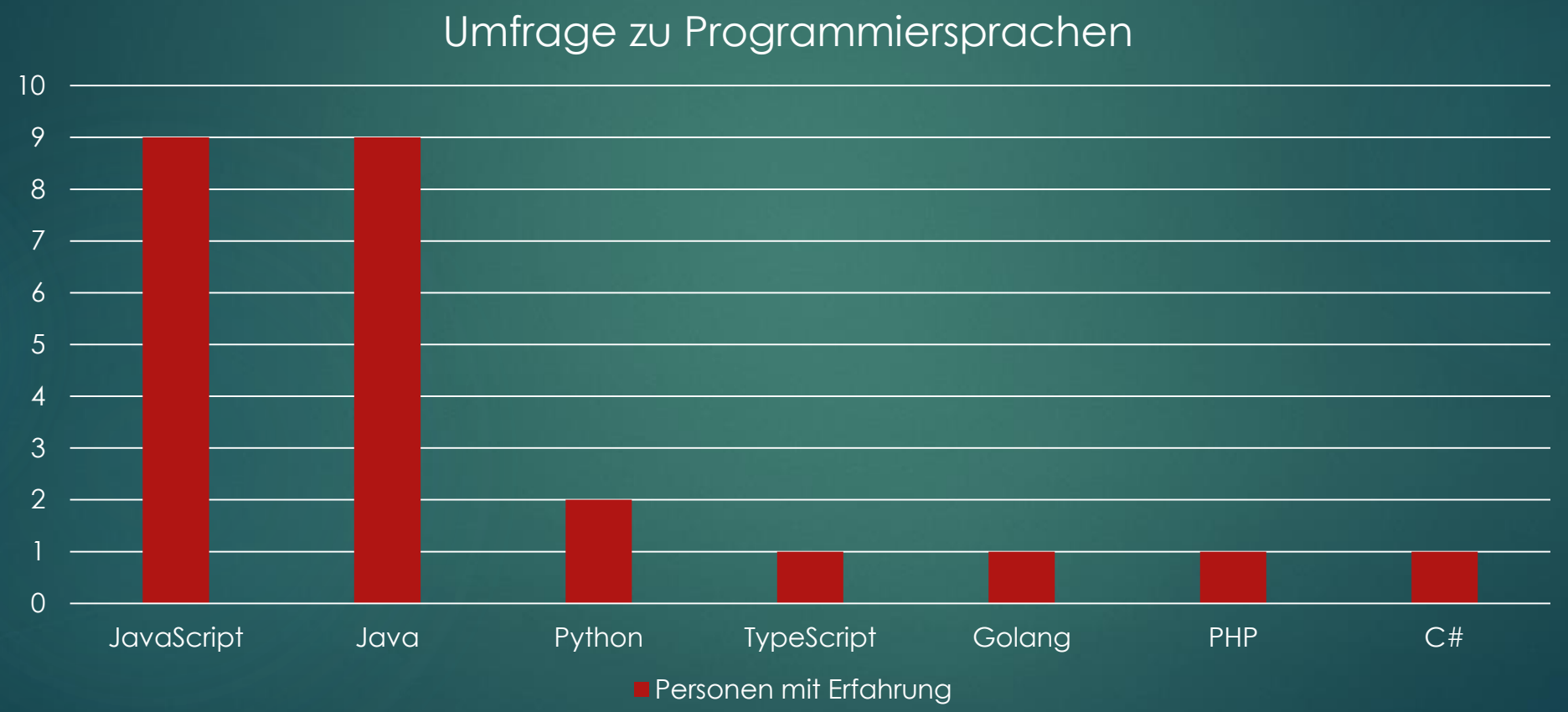
Technologie-Evaluation

Erarbeitet von:
Daniel Weidle
Vladislav Chumak
Rimac Valdez
Martin Starman
Jochen Schwander
Alexander Schramm

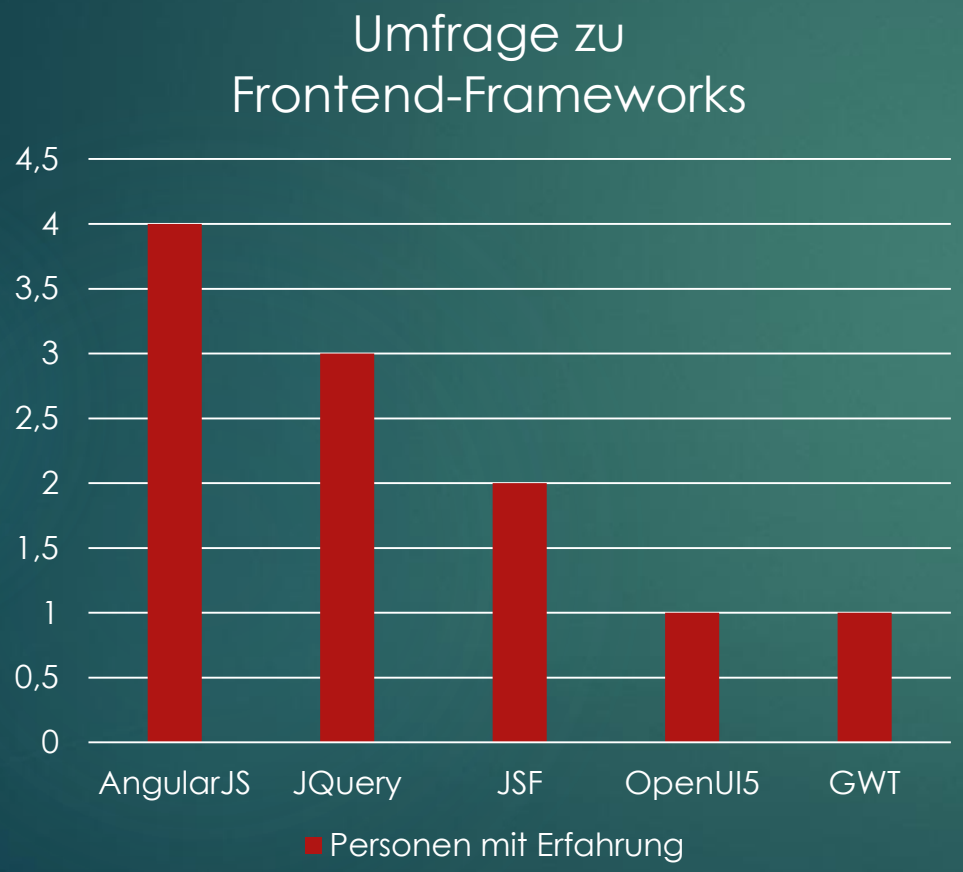
Inhalt

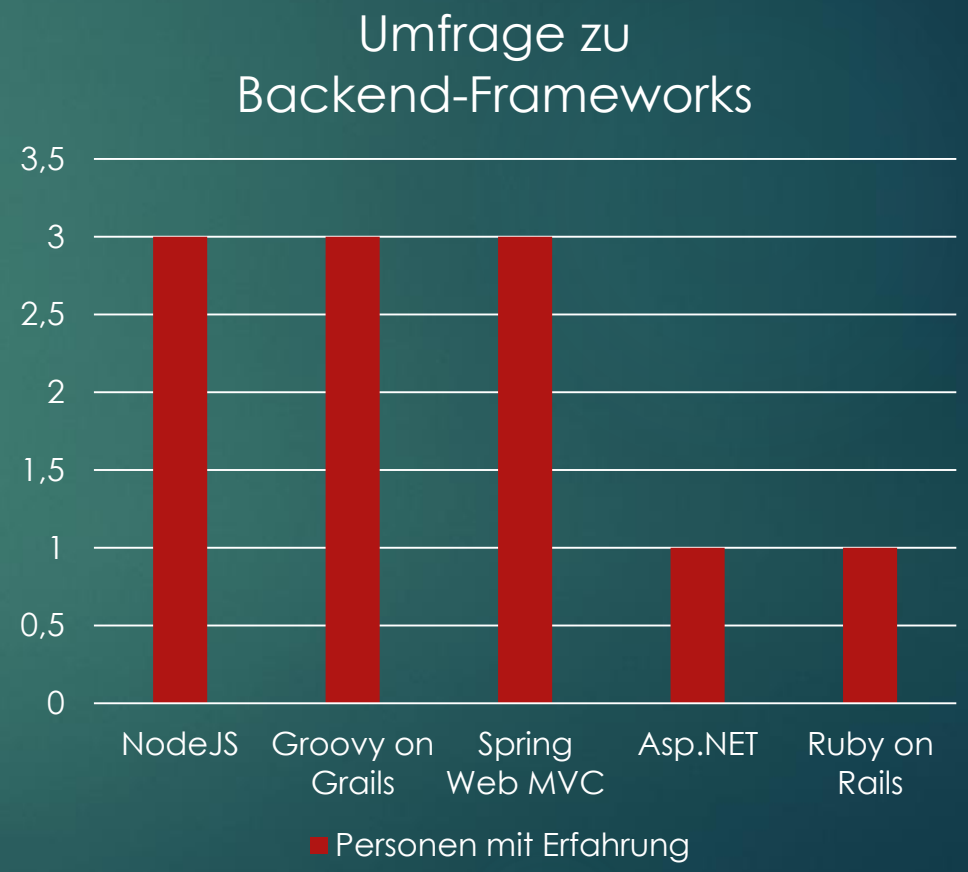
- Programmiersprache Abstimmungen
- Dokumenten-Persistenz Evaluation
- Search Engine Evaluation
- Komponenten-Modell
- Technologie Auswahl

Programmiersprache
1. Umfrage



Programmiersprache 2. Umfrage





Programmiersprache Ergebnis

- Denkbare Technologie-Kompositionen:
  - Java Backend mit Groovy & Grails und Groovy Server Pages Frontend
  - Java Backend mit Spring und JavaScript Frontend mit AngularJS
  - JavaScript Backend mit NodeJS und JavaScript Frontend mit AngularJS

Dokumenten-Persistenz Grundlagen

Datenbank:
- Query Language
- Concurrency Handling
- Performantes Datenhandling

File System:
- Geringe Komplexität
- Direkte Dateiablage
- Performantes Dateihandling

Dokumenten-Persistenz Optionen

Datenbank

File System

Hybrid

Dokumenten-Persistenz Hands-On

Datendank	• MongoDB & NodeJS • CouchDB
File System	• File System & NodeJS
Hybrid	• File System & MySQL

Dokumenten-Persistenz Datenbank

Pro	Con
• Handling für konkurrierende Zugriffe • Dokumente und Artikel können zusammen gespeichert werden • (Versionierung out of the box)	• Komplexe Wartung und Portierung • (Search Engine Indizierung evtl. nicht möglich) • (Zusätzliche Frameworks erforderlich)

Dokumenten-Persistenz File System

Pro	Con
• Leichte Wartung und Portierung • Geringe architekturelle Komplexität • Search Engine Indizierung direkt auf Persistenz-Verzeichnis	• Versionierung nur durch Redundanz • Konkurrierende Zugriffe nicht behandelt • Komplexes Mapping in der Webserver Logik

Dokumenten-Persistenz Hybrid

Pro	Con
• „Das Beste aus beiden Welten“ • Sauber strukturiertes Datenschema (SQL) • Dokumente belasten Datenbank-Performance nicht	• Doppelte Konfiguration • Konsistenz zwischen Datenbank und File System • Doppelte Abfrage (erst Datenbank, dann File System)

Dokumenten-Persistenz Ergebnis

- Hybrid-Ansatz
  - Entscheidende Vorteile
    - Einfaches Dokumenten-Handling im File System
    - Es müssen keine Dateien für Meta-Informationen und Artikel erstellt werden
    - Mit jeder Search Engine Technologie kombinierbar
  - Nachteile & Lösungsansätze
    - **Doppelte Konfiguration:**
      Leichtgewichtige Datenbank mit wenig Konfiguration wählen
    - **Konsistenz:**
      Muss in der Business-Logik überprüft werden, keine anderen Lösungsansätze gefunden
    - **Doppelte Anfrage:**
      Doppelte Anfrage als Vorteil -> Concurrency Handling der Datenbank übernehmen

Search Engine Optionen

- Apache Lucene & Tika
- Apache Solr
- Open Search Server
- Elastic Search

Search Engine Vergleichs-Kriterien

- **Stand-alone:**
  Kann die Search Engine eigenständig betrieben werden?
- **Integrierbar:**
  Kann die Search Engine in ein Programm eingebettet werden?
- **API:**
  Wie können andere Komponenten mit der Search Engine kommunizieren?
- **Daten-Format:**
  Welche Daten-Formate können von der Search Engine indiziert und gesucht werden?
- **Daten-Quellen:**
  Wie (explizit/implizit) und woher kann die Search Engine Daten beziehen?
- **Lizenz:**
  Unter welcher Lizenz ist die Search Engine lizenziert?

Search Engine Vergleich

	Stand-alone	Integrier-bar	API	Daten-Format	Daten-Quelle	Lizenz
Apache Lucene & Tika	Ja	Java	Nativ Java	org.apache.lucene.document.Document	Explizit aus Java	Apache Lizenz 2
Apache Solr	Ja	Java	REST API; Java API	XML, JSON, (JavaBIN)	Explizit über REST API oder JDBC Link	Apache Lizenz 2
Open Search Server	Ja	(WAR Datei)	REST API; Java, PHP, Ruby, Perl und C# APIs	XML, JSON	Explizit über REST API oder implizit über einen Crawler	GNU GPL 3
Elastic Search	Ja	Nein	RESTful API	JSON	Explizit über RESTful API	Apache Lizenz 2

Zu aufwendig!

Fokus passt nicht zu Projekt-Scope

Search Engine Benchmarks

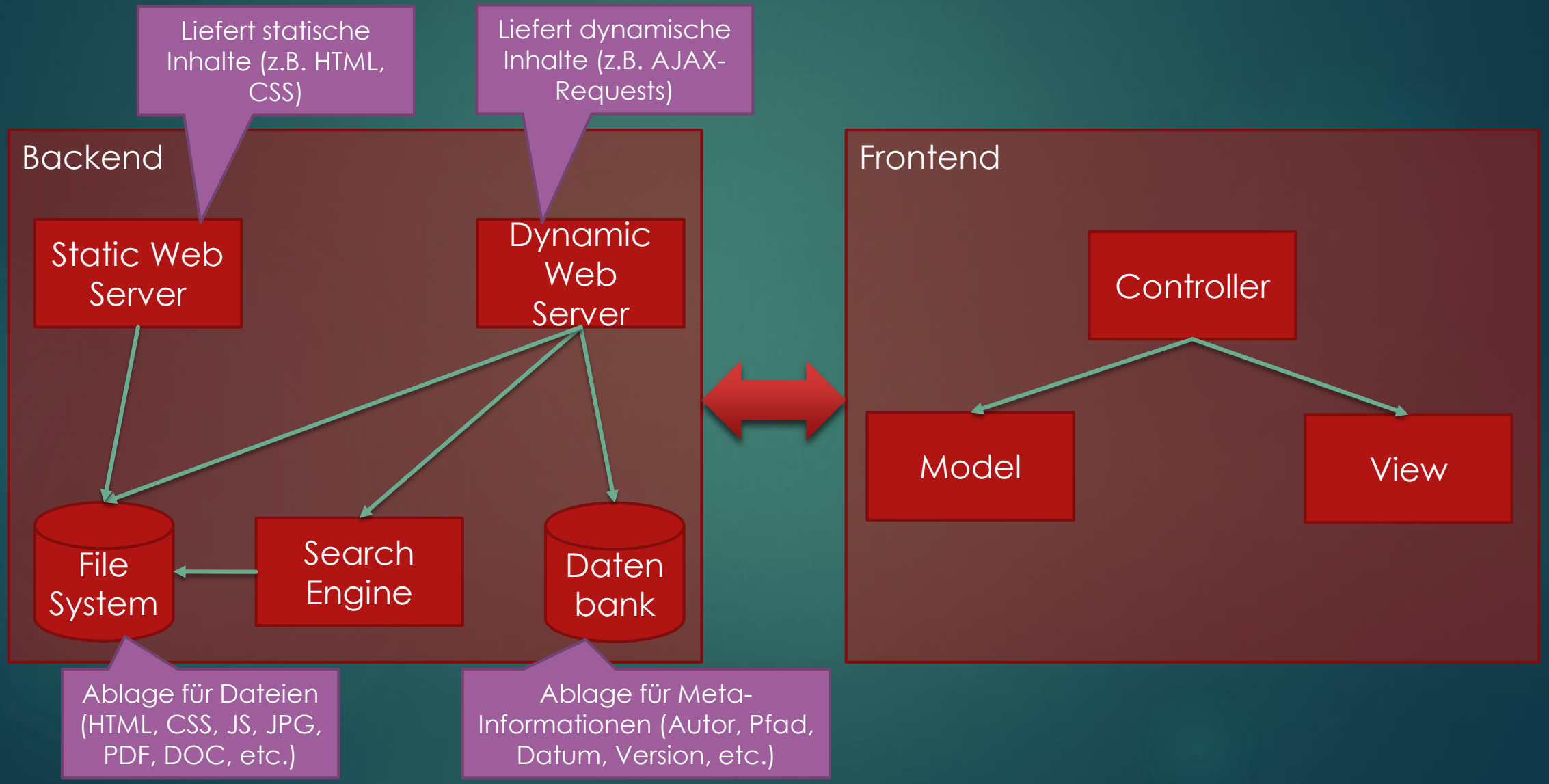
min:sec:ms	Initial index	Reindex	Reindex	Reindex
	45 PDFs (461 MB)	1 PDF (520 KB)	1 PDF (12 MB)	1 DOCX (78 kB)
Open Search Server	00:05:22	00:00:01	00:00:05	00:00:08
Apache Solr	00:02:35	00:00:00	00:00:02	00:00:01

Ergebnis: Beide Technologien indizieren schnell genug für den Knowledge Base Projekt-Kontext. Da beide im Kern die selbe Search Engine verwenden (Apache Lucene) ist der Performance-Unterschied wohl hauptsächlich auf den Dokument-Parser zurück zu führen.

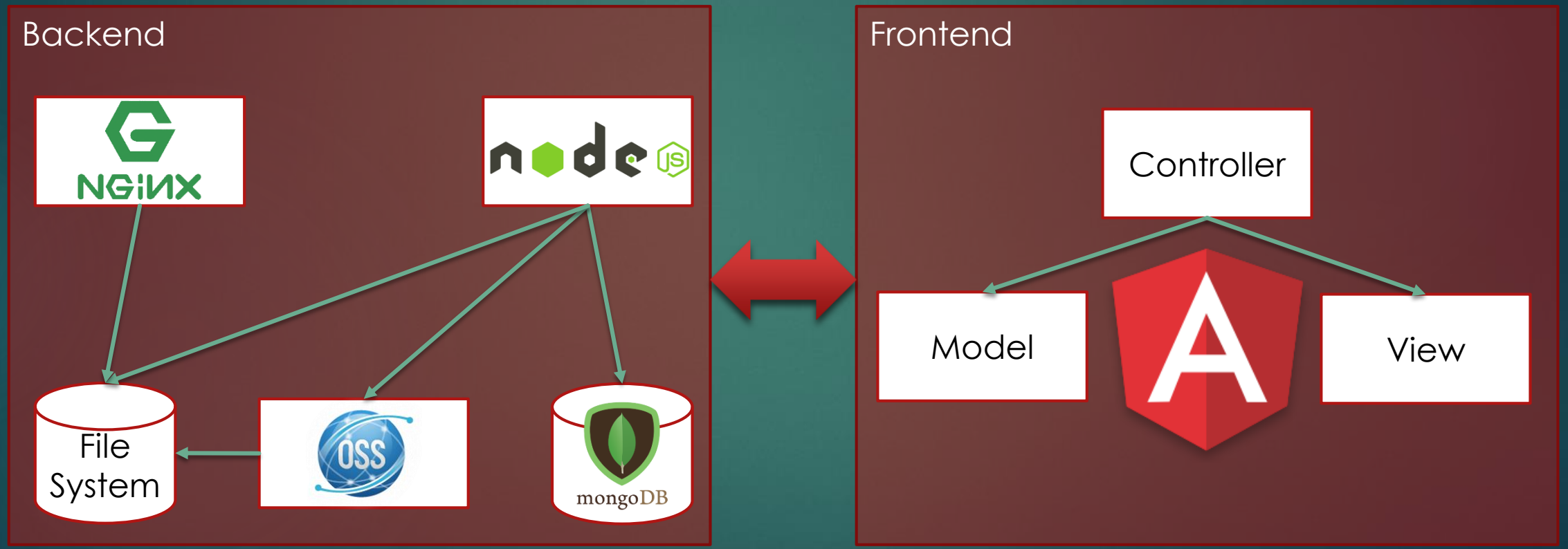
Search Engine Ergebnis

- Open Search Server
  - Entscheidende Vorteile
    - Geringer Implementierungs-Aufwand -> Geringer Test-Aufwand
    - Integrierbar -> Stand-alone, viele APIs und Crawler
    - Crawler übernimmt Indizierungs-Management
  - Nachteile & Lösungsansätze
    - **Hoher Konfigurations-Aufwand:**
      Einplanung in Aufwandseinschätzung des Konfigurations-Management
    - **Geringere Indizierungsperformanz als Solr:**
      Performanz ausreichend für Kontext, Search Engine kann durch Clustering aber auch skalieren, falls der Kontext wächst
    - **Komplexes Tool:**
      Technologie-Team mit Open Search Server Erfahrung ist Teil des Entwicklungs-Teams und bringen damit die meiste Erfahrung mit

Komponenten-Modell



Technologie-Auswahl Überblick



Technologie-Auswahl Rationale

- Schnell, einfach und leichtgewichtig
- *Seperation of Concerns* (dynamische vs. Statische Inhalte)

- Einfach und leichtgewichtig
- Erfahrung im Team (JavaScript und NodeJS)

- Wenig Implementierungs-Aufwand
- Crawler übernimmt Indizierungs-Management

- Unterstützt anfallende Datenformate
- Abfragemöglichkeit nach verschiedensten Aspekten der Meta-Daten

- Erfahrung im Team
- Bewährte Umgebung für Frontends