|  |  |
| --- | --- |
|  | **kan** |

**Architecture Documentation Knownana**

**Authors**Vladislav Chumak  
Alexander Schramm  
Jochen Schwander

# Table of Contents

Table of Contents 2

List of Figures 3

List of Tables 3

List of Abbreviations 4

Version History 5

Preface 6

1 Introduction and Goals 7

2 Architecture Constraints 10

3 System Scope and Context 11

4 Solution Strategy 12

5 Building Block View 13

5.1 Level 1 13

5.2 Level 2 13

6 Runtime View 14

6.1 Create Article 14

7 Deployment View 16

8 Concepts 17

9 Design Decisions 18

9.1 Search Engine 18

9.2 Persistence 18

9.3 Backend Technology 18

9.3.1 Influencing Factors 18

9.3.2 Constraints 18

9.3.3 Considered Alternatives 18

9.3.4 Decision 19

9.4 Frontend 19

9.5 Deployment 19

10 Quality Scenarios 20

11 Technical Risks 21

12 Glossary 22

# List of Figures

1: level 1 building block view 13

2: create article sequence diagram 15

# List of Tables

1: AD-01-Search 7

2: AD-02-Usability 7

3: AD-03-Generic 8

4: AD-04-Testability 8

5: AD-05-Persistence 9

# List of Abbreviations

HTML HyperText Markup Language

MVC Model View Controller

PDF Portable Document Format

REST Representational State Transfer

# Version History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 0.1 | 24.05.2016 | Arc42 structure & architecture drivers & backend technology design decision |
|  |  |  |

# Preface

TODO arc42

# Introduction and Goals

TODO Drivers

1: AD-01-Search

|  |  |  |
| --- | --- | --- |
| **Categorization** | | |
| **Driver Name** | Full text search | |
| **Driver ID** | AD-01-Search | |
| **Status** | Under Design | |
| **Priority** | High | |
| **Description** | | **Quantification** |
| **Environment** | The system contains articles. | At least one article |
| **Stimulus** | The user wants to find information about a specific problem. | Enters at least one keyword |
| **Response** | The system presents relevant articles. | All articles containing respective keywords are presented |

2: AD-02-Usability

|  |  |  |
| --- | --- | --- |
| **Categorization** | | |
| **Driver Name** | Usability | |
| **Driver ID** | AD-02-Usability | |
| **Status** | Under Design | |
| **Priority** | Medium | |
| **Description** | | **Quantification** |
| **Environment** | The system is running. | - |
| **Stimulus** | The user interacts with the system. | The user didn’t use the system. |
| **Response** | The user can use the system intuitively and productively. | The user understands the system and can create his first article within 15 minutes. |

3: AD-03-Generic

|  |  |  |
| --- | --- | --- |
| **Categorization** | | |
| **Driver Name** | Generic Usage | |
| **Driver ID** | AD-03-Generic | |
| **Status** | Under Design | |
| **Priority** | High | |
| **Description** | | **Quantification** |
| **Environment** | The system is running. | - |
| **Stimulus** | The user wants to create an article containing documents. | User uploads at least one file. |
| **Response** | The system supports PDF, Microsoft Office and text files regardless of their format. | The system accepts and indexes an unlimited number docx, doc, pdf and txt files. As well as a text of any length, containing tables, bullet points, text formatting and images. |

4: AD-04-Testability

|  |  |  |
| --- | --- | --- |
| **Categorization** | | |
| **Driver Name** | Testability | |
| **Driver ID** | AD-04-Testability | |
| **Status** | Under Design | |
| **Priority** | Medium | |
| **Description** | | **Quantification** |
| **Environment** | The system implementation started. | Source code is available. |
| **Stimulus** | The tester wants to test the system. | - |
| **Response** | The code is testable in an automated way. | 85% branch coverage can be achieved. |

5: AD-05-Persistence

|  |  |  |
| --- | --- | --- |
| **Categorization** | | |
| **Driver Name** | Persistence of articles | |
| **Driver ID** | AD-05-Persistence | |
| **Status** | Under Design | |
| **Priority** | High | |
| **Description** | | **Quantification** |
| **Environment** | The system contains articles with attached documents. | At least one article with one document. |
| **Stimulus** | The user wants to read an article including its documents. | The user has found the article. |
| **Response** | The system displays the articles content and offers the documents for download. | The retrieved documents are in their original format. |

# Architecture Constraints

TODO hardware constraints?

# System Scope and Context

TODO kontextdiagramm

# Solution Strategy

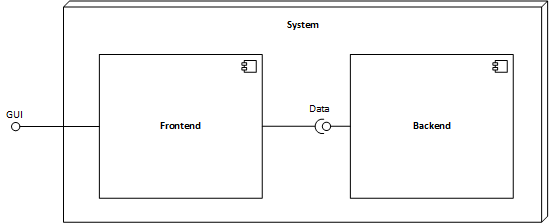
TODO Client Server

# Building Block View

TODO

## Level 1

TODO



1: level 1 building block view

## Level 2

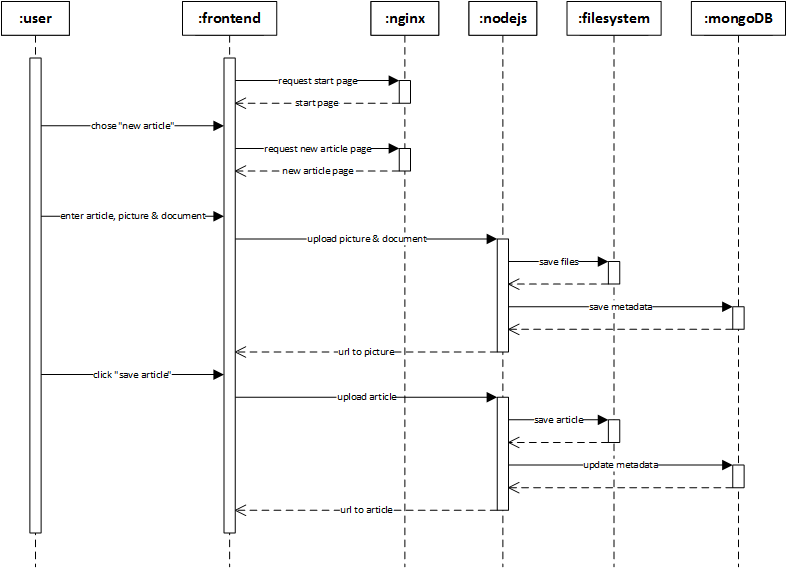
TODO

# Runtime View

TODO

## Create Article

TODO



2: create article sequence diagram

# Deployment View

TODO

# Concepts

TODO

# Design Decisions

TODO

## Search Engine

TODO

## Persistence

TODO fs, hybrid, db

TODO mongo

## Backend Technology

The backend has to provide the users web browser with the frontend. The backend has to supply the frontend with the articles, including attached documents and images.

### Influencing Factors

This decision is influenced by the architecture drivers *AD-04-Testability* and *AD-05-Persistence*, as well as by the design decisions *9.1 Search Engine* and *9.2 Persistence*.

### Constraints

All developers are capable of the two programing languages Java and JavaScript.

### Considered Alternatives

The two following alternatives were considered as backend technologies.

#### Node.js with Nginx

Node.js is a lightweight and performant JavaScript server. It allows easy file handling and simple REST calls. As a downside, pure JavaScript does not provide type safety. It’s extremely easy to integrate with mongoDB.

Nginx is a fast and lightweight webserver well suited for static content delivery. It’s a stable, mature server technology that has proven itself in the web for several years.

The combination of those two technologies allows for fast and stable delivery of static content with Nginx, while Node.js handles dynamic content.

#### Spring Web MVC

Spring Web MVC is an industry proven Java Framework for dynamic web content. It’s well documented and wide spread. On the downside, Java is relatively heavyweight and memory hungry. Doing REST calls with Java requires additional effort.

### Decision

Both technologies are capable to fulfill the requirements. The decision was made for Node.js with Nginx because they provide easier integration with REST and the team competences favored Node.js.

## Frontend

TODO angular

## Deployment

TODO docker?

# Quality Scenarios

Todo?

# Technical Risks

TODO

# Glossary

**Article** An article includes structured text as its content and attached documents.

**Document** A document is an external file, like text, PDF or Word.

**Dynamic Webserver** A dynamic webserver, in contrast to a static webserver, only delivers dynamic content build from different sources like for example a database.

**REST** sTODO

**Static Webserver** A static webserver, in contrast to a dynamic webserver, only delivers static content, like HTML pages, pictures or JavaScript files.