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

**Architecture Documentation  
Knownana**

**Authors**Vladislav Chumak  
Alexander Schramm  
Jochen Schwander

# Table of Contents

Table of Contents 2

List of Figures 4

List of Tables 4

List of Abbreviations 5

Version History 6

Preface 7

1 Introduction and Goals 8

2 Architecture Constraints 11

3 System Scope and Context 12

4 Solution Strategy 13

5 Building Block View 14

5.1 Level 1 14

5.2 Level 2 14

6 Runtime View 15

6.1 Create Article 15

6.2 Read Article 17

6.3 Search Article 18

7 Deployment View 20

8 Concepts 21

9 Design Decisions 22

9.1 Search Engine 22

9.2 Persistence 22

9.3 Backend Technology 22

9.3.1 Influencing Factors 22

9.3.2 Constraints 22

9.3.3 Considered Alternatives 22

9.3.4 Decision 23

9.4 Frontend 23

9.5 Deployment 23

10 Quality Scenarios 24

11 Technical Risks 25

12 Glossary 26

# List of Figures

Figure 1: context diagram 12

Figure 2: level 1 building block view 14

Figure 3: create article sequence diagram 16

Figure 4: read article sequence diagram 17

Figure 5: search article sequence diagram 19

# List of Tables

Table 1: AD-01-Search 8

Table 2: AD-02-Usability 8

Table 3: AD-03-Generic 9

Table 4: AD-04-Testability 9

Table 5: AD-05-Persistence 10

# List of Abbreviations

HTML HyperText Markup Language

MVC Model View Controller

PDF Portable Document Format

REST Representational State Transfer

UML Unified Modeling Language

URL Uniform Resource Locator

# Version History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 0.1 | 24.05.2016 | Arc42 structure & architecture drivers & backend technology design decision |
| 0.2 |  |  |

# Preface

TODO arc42

# Introduction and Goals

TODO Drivers

|  |  |  |
| --- | --- | --- |
| **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 |

Table : AD-01-Search

|  |  |  |
| --- | --- | --- |
| **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. |

Table : AD-02-Usability

|  |  |  |
| --- | --- | --- |
| **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. |

Table : AD-03-Generic

|  |  |  |
| --- | --- | --- |
| **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. |

Table : AD-04-Testability

|  |  |  |
| --- | --- | --- |
| **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. |

Table : AD-05-Persistence

# Architecture Constraints

TODO hardware constraints?

# System Scope and Context

Figure 1: context diagram

The system has two roles interacting with it. The first role is the user. He can create, read, update, delete and search articles. This includes attaching documents to an article and downloading them. The second role is the admin. He can trigger the system to re-index all articles. The system does not interact with any other system.

# Solution Strategy

TODO Client Server

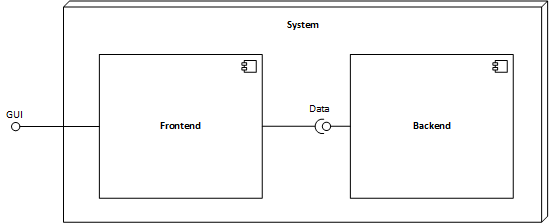
# Building Block View

TODO

## Level 1

TODO

Figure 2: level 1 building block view



## Level 2

TODO

# Runtime View

This chapter contains the runtime view of the system, represented by multiple runtime scenarios. Each section represents one runtime scenario. These are show with UML sequence diagrams.

## Create Article

The user wants to make his knowledge available to other users. He starts his browser and creates a new article in the knowledge base. *Figure 3* shows the interaction between user and knowledgebase.

First the user navigates his browser onto the knowledgebase webpage from the static webserver. After it is loaded, the user clicks on the button to create a new article. The frontend loads the new article page from the static webserver and offers the user the possibility to enter a formatted article with pictures and attached documents.

As his second action, the user writes the article and decides to use pictures in it, as well as to attach a document. The frontend forwards the picture and document to the dynamic webserver. The dynamic webserver saves the document and picture into the file system. It also saves metadata about which document and picture belong to the article in the database. When the picture upload is done, the dynamic webserver returns an URL pointing to the picture to the frontend. After this, the frontend loads the picture from the static webserver.

The user thirdly clicks on the button to save the article after he is content with it. The frontend sends the article to the dynamic webserver. The dynamic webserver saves the article on the file system and updates the metadata in the database. After the article is saved, the dynamic webserver sends an URL to the article back to the frontend. The frontend then allows to read the article (see *6.2 Read Article*).

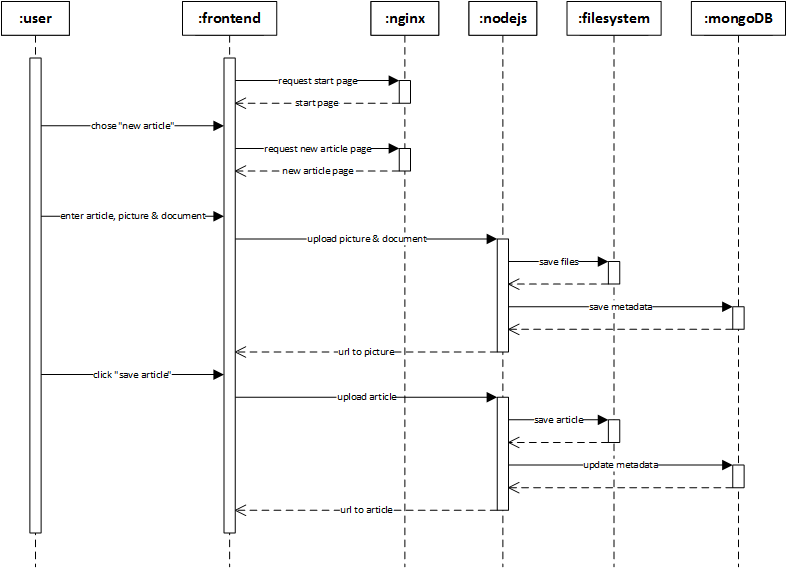


Figure 3: create article sequence diagram

## Read Article

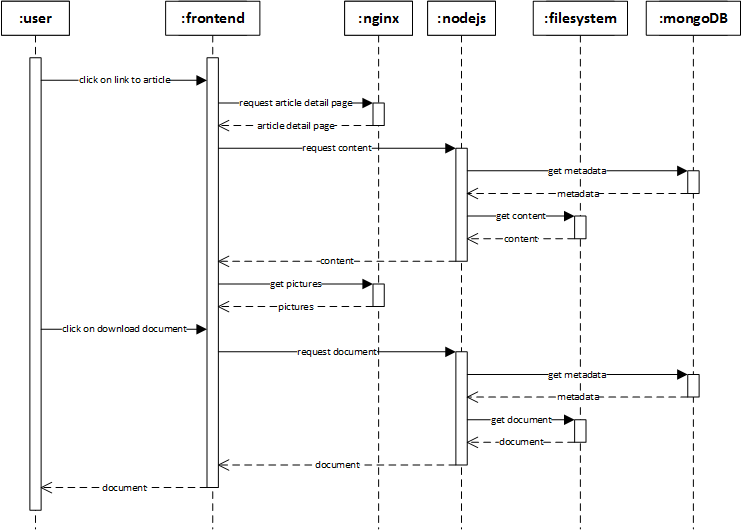
The user wants to read about a specific problem. He opens his browser and reads the according knowledgebase article. *Figure 4* shows the interaction between the user and the knowledgebase.

Figure 4: read article sequence diagram

First the user navigates his browser to the article page. After it is loaded from the static webserver, the frontend requests the article’s content from the dynamic webserver. The dynamic webserver accesses the database to read the article’s metadata. With the information in the metadata, the dynamic webserver can then load the article’s content from the file system. The content is send to the frontend. The frontend displays it, loading pictures from the static webserver.

Reading in the article the user decides to download an attached document for further information. He clicks on the document to download it. The frontend requests the document form the dynamic sever which reads the article’s metadata from the database. With the metadata the dynamic webserver reads the document from the file system and sends it to the frontend with its original name. The user can then download the document.

## Search Article

The user has a problem and needs a solution. He starts his browser and searches for solutions the in the knowledgebase. *Figure 5* shows the interaction between the user and the knowledgebase.

First the user navigates his browser onto the knowledgebase webpage from the static webserver. After it is loaded, the user enters keywords matching his problem in the search field. The frontend sends the keywords to the dynamic webserver. The dynamic webserver then queries the search engine. With the search results the dynamic webserver accesses the database for metadata to match articles and documents together. Afterwards it returns matching articles to the frontend. The frontend displays those articles to the user.



Figure 5: search 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.

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