Technische Universität Berlin

Institut für Telekommunikationssysteme
Fachgebiet Architektur der Vermittlungsknoten

Fakultät IV
Franklinstrasse 28-29
10587 Berlin
http://www.av.tu-berlin.de

Diploma Thesis

Contextual and Conditional Content Distribution

Abdulrahman Hamood

Matriculation Number: 227675
06.04.2013

Supervised by
Prof. Dr. Thomas Magedanz

Assistant Supervisor
Dr. Adel Al-Hezmi

FOKUS Institute
Kaiserin-Augusta-Allee 31
10589 Berlin

This thesis originated in cooperation with the Fraunhofer Institute for Open Communication Systems (FOKUS).

First of all I would like to thank Prof. Dr. Thomas Magedanz at the Fraunhofer Institute FOKUS for giving me the opportunity to carry out state of the art research in this field.

Special thanks to Dr. Adel Al-Hezmi and Hakan Coşkun for their guidance and support.

I would also like to thank my friends for their support and encouragement.

I am as ever, especially indebted to my family. Thank you for your unconditional love, support and understanding throughout my studies. I would like to dedicate this thesis to you.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 06.04.2013

. . . . . .

(Signature Abdulrahman Hamood)

Abstract

Context-aware systems can help people in many areas of daily life in order to plan their daily schedule, to make the right decisions and to perform other tasks for them. Due to the fact of the increased computing power of today's mobile devices, they perform tasks that were not possible a few years ago. For these devices, with their increasingly complex applications, context-aware behavior is of great importance. The explosive growth of content consumption from mobile devices and social networks and the consumer expectation of content availability on various devices should be taken as evidence, or at least as a strong indication, for the need of an efficient multimedia provisioning framework that supports efficient and personalized provisioning and discovery mechanisms of multimedia content information. According to a statistical result reported by The Nielsen Company, the growing popularity of smartphones has also led to a dramatic rise in mobile videos. While 23 million U.S mobile subscribers viewed videos on their phones in 2010, 31 million viewed mobile videos in 2011 - a 35% increase [21].

In this thesis, a framework is developed that facilitates the development of context-aware applications. The framework supports the correlation of context information and multimedia content and also provides an efficient and scaleable discovery and distribution mechanism. Thereby, the framework is not limited to specific context information or application domain and therefore suitable for many context-aware applications.

First of all, the terms context and context-awareness plus the relevant technologies are introduced. In the following, the requirements for such a framework are analyzed and defined. Based on these requirements, a conceptual design for the framework is developed and its feasibility is evaluated successfully by means of a prototypical implementation.

Zusammenfassung

Kontextbasierte Systeme können Menschen in vielen Bereichen des täglichen Lebens unterstützen, wie z.B. bei der Planung des Tagesablaufs, beim Treffen von wichtigen Entscheidungen oder beim Erledigen sonstiger Aufgaben anstelle des Benutzers. Die heutigen mobilen Geräte führen aufgrund ihrer hohen Rechenleistungen Aufgaben aus, die noch vor einigen Jahren nicht möglich waren. Für diese Geräte mit ihren immer komplexeren Anwendungen ist kontextsensitives Verhalten von Bedeutung. Das explosive Wachstum des Verbrauchs von digitalen Inhalten auf mobilen Geräten und sozialen Netzwerken und die Erwartungen der Verbraucher, dass die Inhalte auf verschiedenen Geräten verfügbar sein sollen, sollte als Beweis genommen oder zumindest als deutliches Zeichen angesehen werden, für die Notwendigkeit eines zuverlässigen, leichten, skalierbaren und kontextsensitiven Systems. Einem statistischen Bericht zur Folge, führte die wachsende Beliebtheit von Smartphones auch zu einem dramatischen Anstieg der mobilen Videos. Während im Jahr 2010 23 Millionen US-Mobilfunkkunden sich Videos auf ihren Handys ansahen, waren es im Jahr 2011 31 Millionen Kunden, was einer Zunahme von 35 % entspricht [21].

Im Rahmen dieser Diplomarbeit wird ein Framework entwickelt, dass die Entwicklung von kontextsensitiven Anwendungen erleichtert und einen effizienten und skalierbaren Mechanismus für das Suchen und Verteilung von Multimedia-Inhalten bereitstellt. Dabei ist das Framework nicht auf bestimmte Kontextinformationen oder Anwendungsdomänen beschränkt, sondern eignet sich für viele verschiedene kontextsensitive Anwendungen.

Zunächst werden die Begriffe Kontext und Kontextsensivität wie auch die relevanten Techniken eingeführt und erläutert. Anschließend werden die Anforderungen an ein solches Framework analysiert und definiert. Basierend auf diesen Anforderungen wird eine Konzeption für das Framework entwickelt und seine Machbarkeit dann erfolgreich durch eine prototypische Implementierung evaluiert.

Contents

List of Figures

List of Tables

1 Introduction

The Internet has become an essential part of our daily lives in different sectors business, social communication, healthcare, etc. It has revolutionized our economy and society and therefore considered to be at the top of the technological revolution in the current century. The success of the Internet can be seen by its traffic growth. The monthly global Internet traffic is expected to quadruple between 2010 and 2015 increasing from about 20.2 exabytes to 80.5 exabytes (one exabyte equals one billion gigabytes)[19]. Indeed, this growth indicates how huge the content (and rapidly increasing) that is uploaded and consumed is. Around one million minutes of video content will cross the network per second in 2015 [19]. About one hour of YouTube videos are being uploaded per second and more than four billion views per day[44]. The video-sharing content in YouTube is only one example of a huge number of distributed contents on the Internet provided by various content delivery platforms. These platforms provide different types of contents, i.e. contents are heterogeneous and can be anything, e.g. video, multimedia, books, etc.

The recent growth of multimedia content offered by multiple professional content providers (e.g. Internet Protocol Television (IPTV) or mobile TV provider), available in several multimedia-based social networking communities or distributed in various user devices, seems to be clear evidence for the need of an efficient multimedia provisioning framework that supports efficient and personalized provisioning and discovery mechanisms of multimedia content information compared to the classical client-server provisioning systems. This thesis will address the increase of the wealth of distributed multimedia content either in a controllable network or in a user private network. The challenge is to provide users with technical means for rapid and instant access to relevant, trustworthy multimedia content information and enriched personalization.

Nowadays, devices such as PCs, smartphones, positioning devices, health monitors in our environment, are expected to work with high levels of independence, performing programmed actions that benefit their users in everyday life. In order to meet this set of requirements, these different devices are connected together to perform certain tasks. This concept is known as Machine to Machine (M2M) communication. M2M is a concept that defines the rules and relations between devices while cooperating with each other. It implies a highly automated usage of a set of devices simultaneously, without much need for human interaction. With the increase of computational power, it is even possible to run different M2M

tasks on various consumer electronics (e.g. television sets, set-top boxes), however smartphones are still more frequently used in M2M domain.

The aim of this thesis is to develop a context-aware framework, in which the context information of multimedia content environment (such as location and time) is considered and embedded into the captured multimedia content as content information or metadata. The proposed framework uses M2M concept, which provides several resources such as sensor information like temperature, Global Positioning System (GPS) data, and so on. These resources can be used to enrich the metadata of the captured content. The motivation behind this work is due to the lack of context/content-aware storage management in the current Internet architecture which is one of its fundamental limitations [27]. The consumer, who is interested in a particular multimedia content service, submits the request with defined conditions (e.g. time, location, content provider, etc.) that are evaluated against multimedia context information in order to deliver the associated multimedia content information (e.g. content resources, description, etc.). User-specific conditions can be published once or updated regularly. The multimedia content service shall examine user-specific conditions and notifies the user with the matched multimedia content. Therefore, an efficient interaction model between consumer and the service will be developed. Furthermore, an end-to-end multimedia content service will be implemented in order to demonstrate the developed concept.

1.1 Motivation

Context-aware systems can help people in many areas of daily life in order to plan their daily schedule, to make the right decisions and to perform other tasks for them.

Due to the fact of increased computing power of today's mobile devices, tasks are now performed that were not possible a few years ago. For these devices, with their increasingly complex applications, context-aware behaviour is very important. Reliable, easy context-aware systems are required because of the explosive growth of content consumption from mobile devices and social networks and the consumer expectation of content availability on various devices. According to a statistical result reported by The Nielsen Company, U.S mobile video viewers have grown from 23 million in the third quarter of 2010 to 31 million in the third quarter of 2011 [21].

M2M technology is growing fast and will be available everywhere in the near future. Such technology will help to enrich context information associated with multimedia content.

There are several reasons for working on this diploma thesis. On one hand, there is no correlation nowadays between content and context supported by the current

Internet architecture. However, there is a demand for solutions or products that simplify the usage of the distributed content based on a specific type of context. The unavailability of such solutions or products is one of the main motivations behind this thesis. Within the context of this thesis, a new valid prototyping for context-aware content management framework will be developed. Therefore this thesis will set the ground for future investigations and will also be used as a cornerstone and give directions for better designed and acceptable solutions.

On the other hand, working on this thesis gives me the chance to get in-depth knowledge and hands-on experience of a hot topic that will evolve, improve and develop in the years to come and eventually become an integral part of normal lifestyle.

1.2 Objective

The main objective of this thesis is to develop a contextual content management framework in which the context information/metadata is created automatically during content capturing and relies on local, distributed sensing information. The framework will support the correlation and synchronization of context information and multimedia content stream. In order to deliver the content to various devices, the distribution mechanism will be implemented. The distribution mechanism should be aware of the device properties i.e screen resolution or internet speed. For context enrichment, the framework should provide a mechanism that uses the benefits of M2M concept.

Figure 1.1 shows the main component of the platform:

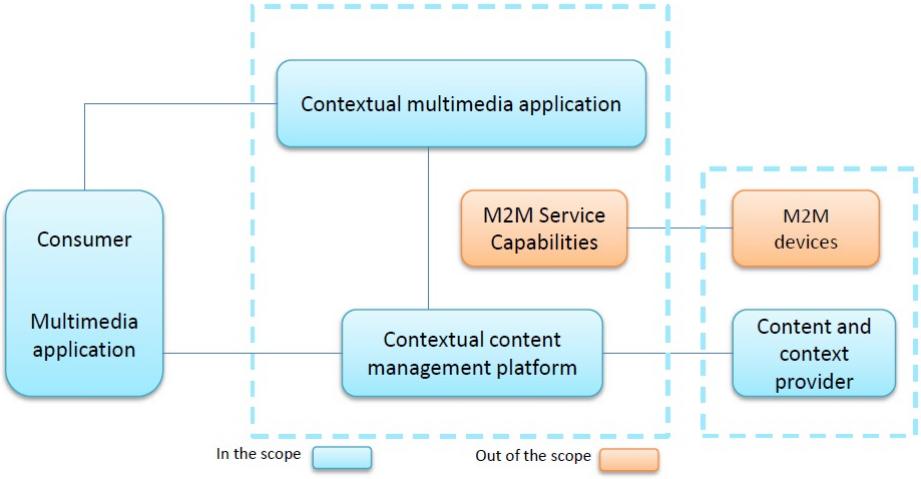

Figure 1.1: Overall Architecture

1.3 Scope

Due to the lack of time and possible wide range of technologies which have been mentioned above, the objective will be defined here in detail to decide what should be developed in this thesis.

The scope of this thesis is to manage the relationship between content and context. In order to manage this relationship, this thesis will study the state of the art of data model description and then choose the appropriate one. The thesis should integrate the M2M concept for context enrichment.

The activities in this thesis are outlined as follows:

- Act. 1: Study of the state of the art of data model format, context description language
- Act. 2: Investigate the process of automatic creation of context information and data fusion (correlation between context and content)
- Act. 3: Design the required management and delivery platforms
- Act. 4: Examine the available open sources for content management systems and HyperText Transfer Protocol (HTTP) based streaming servers
- Act. 5: Based on available open source solutions, develop an end-to-end framework that enables content provider(mobile app.) to capture multimedia content with the associated context information and publish this content to the server and allow consumers(mobile app. or web based) to discover and subscribe to multimedia content according to defined conditions.
- Act. 6: Develop an end-to-end application to evaluate the implemented functionality

Figure 1.2 shows inputs and outputs for this thesis:

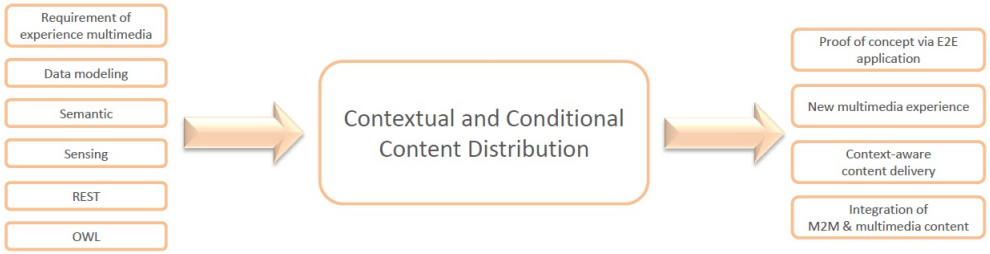

Figure 1.2: Inputs & Outputs

By studying these tasks, the goal is to draw conclusions about the best practices in this domain, and design the framework that will establish the basis for its further development and future implementation.

1.4 Outline

In chapter 2, the background and relevant technologies for this thesis will be presented. Chapter 3 deals mainly with the functional and non functional requirements for the framework to be developed. Chapter 4 represents the most comprehensive and important part of this thesis. This chapter provides a prototype design for the framework based on the requirements identified in chapter 3. Chapter 5 deals with the prototype implementation of the design presented in chapter 4. Next, the evaluation and conclusion follows in chapter 6 and chapter 7 respectively.

2 Background

As the main objective of this thesis is to develop a framework that manages the correlation between the content and its context and also to provide an efficient and scaleable discovery and distribution mechanism, this chapter gives an overview of the context and how it can be described in order to efficiently use these context data. Furthermore, this chapter deals with related technologies, which could be a part of the developed framework.

2.1 Context-awareness

In this section, the terms *context* and *context-aware computing* are discussed in more detail. Furthermore, the application possibilities of context awareness is demonstrated.

In order for computers to assist users in their everyday tasks, they should adapt themselves to the current user's situation, and then respond according to this.

Humans are quite successful at conveying ideas to each other and reacting appropriately. This is due to many factors including the richness of the language they use, the common understanding of how the world works, and an implicit understanding of every-day situations. When humans talk with each other, they are able to use implicit situational information, or context, to increase the conversational bandwidth. Unfortunately, this ability to convey ideas does not transfer well to humans interacting with computers [24].

2.1.1 What is Context?

The report that first introduces the term *context-aware*, [Schilit and Theimer] refers to context as location, identities of people and objects nearby, and changes to those objects. A similar definition, [Brown et al.] describes context as location, identities of the people around the user, the time of day, season, temperature, etc. [Ryan et al.] defines context as the user's location, environment, identity and time. [Dey] enumerates context as the user's emotional state, focus of attention, location and orientation, date and time, objects and people in the user's environment.

A recent definition of context-awareness is described by [Dey and Abowd] who defines it as, context is any information that can be used to characterize the

situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves.

This way, it is easier for an application developer to enumerate the context for a given application scenario. If a piece of information can be used to characterize the situation of a participant in an interaction, then that information is regarded as context.

After the term *context* has been defined the term *context-aware computing* is discussed in the following subsection.

2.1.2 Context-aware Computing

Context-aware computing was first described by [Schilit and Theimer] in 1994 to be software, that adapts itself according to the location of use, the collection of nearby people and objects, as well as changes to those objects over time. However, it is commonly agreed, that context-aware computing was first investigated in 1992 by [Want et al.].

[Ryan] has also defined the term context-aware applications as applications that allow users to select from a range of physical and logical contexts according to their current interests or activities and also monitor input from environmental sensors.

2.1.3 Context Examples

The orientation of the screen of a tablet computer is automatically changed, maps can orientate themselves according to the user's direction with the zoom level adapted to the current speed, and the backlight of the phone is switched on when used in the dark.

These are examples of computers that are aware of their environment and their contextual use. However, such functions were not common 10 years ago and only existed on prototype devices in research labs which researched context-aware computing.

Below are some examples of context awareness in mobile and non-mobile environments. Although non-mobile environments for this thesis are not relevant, they are interesting at this point in order to show the diverse application areas which illustrate the usage context-awareness of systems.

- identity
- spatial information
  e.g. location, orientation, speed and acceleration

- temporal information
  e.g. time of the day, date and season of the year
- environmental information
  e.g. temperature, air quality and light or noise level
- social situation
  e.g. who you are with, and people nearby
- resources that are nearby
  e.g. accessible devices, and hosts
- availability of resources
  e.g. battery, display, network and bandwidth
- physiological measurements
  e.g. blood pressure, heart rate, respiration rate, muscle activity and tone of voice
- activity
  e.g. talking, reading, walking and running

2.2 Context Description

In order to efficiently use the context data after acquisition, it needs to be represented and/or stored in an appropriate form suitable for further processing. Now some of the different types of context modeling will be discussed.

- **Key-value model:** This modeling technique represents contextual information with key-value pairs which is one of the most simple data structures for modeling contextual information. This model was already used in 1994 by Schilit et al. to present the context by providing the value of context information (e.g. location information) to an application as an environment variable. Distributed service frameworks frequently use the key-value modeling approach. Although key-value pairs lack the capability for sophisticated structuring to enable efficient context retrieval algorithms, they are easy to manage.
- **Logic based model:** Logic-based models have a high degree of formality. Typically, facts, expressions and rules are used to define a context model [14].

  A logic based model defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inferencing) from

a set of other expressions or facts. In order to describe these conditions in a set of rules, a formal system is applied. In a logic based context model, the context is consequently defined as facts, expressions and rules. Usually contextual information is added to, updated or deleted from a logic based system in terms of facts or inferred from the rules in the system respectively. A high degree of formality is common to all logic based models [42].

In early 1993 Mccarthy and his group at Stanford University researched one of the first logic based context modeling approaches and published it as a *Notes on formalizing contexts*. They introduced contexts such as abstract mathematical entities with properties which are useful in artificial intelligence.

- **Ontology based model:** Ontologies are promising instruments to specify concepts and interrelations [28]. The Web Onotlogy Language (OWL) is one way of implementing these ontologies. This consists of a set of classes, class hierarchies, a set of property assertions, constraints on these elements and types of permitted relationships between them. Another way to implement the ontologies, is to use knowledge representation language - the Resource Description Framework (RDF). This is a promising model because of the possibility of applying reasoning techniques [36].

  Öztürk and Aamodt proposed one of the first approaches of modeling the context with ontologies. Psychological studies on the difference between recall and recognition of several issues in combination with contextual information were analyzed by them. The necessity of normalizing and combining the knowledge from different domains was derived from this examination. A context model based on ontologies due to their strengths in the field of normalization and formality was proposed by them.

- **Graphical models:** The Unified Modeling Language (UML) is a wide spread modeling tool for software systems. When using UML, the architectural aspects of software systems are defined as classes. Each class constitutes a set of objects with common services, properties and behavior. Services are described by methods and properties are described by attributes and associations [41].

- **Object-oriented models:** Object-oriented design of context benefits from common properties object-oriented programming, such as inheritance, encapsulation, reuse, and polymorphism. For example, in a model of a payroll system, a company is an object, an employee is another object and employment is a relationship or association. An employee class (or object for simplicity) has attributes like name, birthdate, etc. The association itself may

be considered as an object, having attributes, or qualifiers such as position, etc[35].

- **Markup languages:** All markup scheme modeling approaches share a hierarchical data structure consisting of markup tags with attributes and content. In particular, the content of the markup tags is usually recursively defined by other markup tags. Typical representatives of this kind of context modeling approach are profiles. They are usually based on a serialization of a derivative of Standard Generic Markup Language (SGML), the superclass of all markup languages such as the popular Extensible Markup Language (XML)[42].

2.3 Related Technologies

This section covers a wide of technologies which are relevant for this thesis.

2.3.1 Web Services

There are many definitions for the term Web service, the World Wide Web Consortium (W3C) defines it as follows:[1]

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language (WSDL)). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The W3C also states:[1]

> We can identify two major classes of Web services, REpresentational State Transfer (REST)-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations.

By using Web Services, it is now very easy to make existing data and functions from existing applications available to consumers. Web services are considered

as machine to machine communication, which exchange messages via standard protocols.

The best known web services technologies SOAP and REST are discussed in the following subsections.

SOAP

SOAP stands for "Simple Object Access Protocol" and it relies on XML for its message format. For message negotiation and transmission, it is dependent on other Application Layer protocols, particularly on HTTP or Simple Mail Transfer Protocol (SMTP), however HTTP has gained a wider acceptance, as it works well with today's Internet infrastructure and also with network firewalls.

A SOAP message is a type of envelope or container, which may contain an optional header element and a mandatory body element, see figure 2.1. Meta-data for this message are located in the header and the user data are stored in the body.

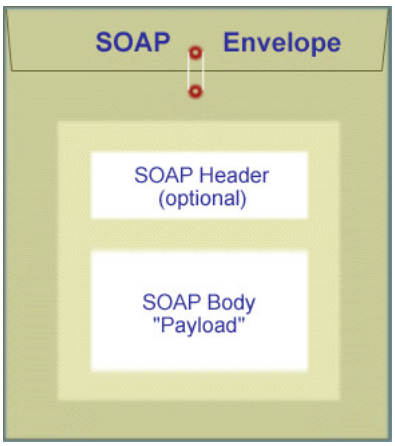

Figure 2.1: SOAP Envelop
[22]

SOAP Message Example: Listing 1 is an example, which gives an overview of a SOAP message:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
```

```
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Listing 1: SOAP message example

REST

In addition to SOAP, there is another alternative for the implementation of Web services. Fielding in his dissertation describes an architectural style that he calls REpresentational State Transfer architecture or short REST.

REST is based on principles that are used in the largest distributed application - the World Wide Web. The World Wide Web is itself a gigantic REST application. Many search engines, shops or booking systems have unintentionally been based on REST web services.

The REpresentational State Transfer Architecture is an architectural model, which describes how the Web should work. The model will serve as a guide and reference for future enhancements. REST is not a product or standard. REST describes how web standards in a Web-friendly manner can be used.

REST Example: An online store will serve here as an example of a RESTful application. In this application, there are customers who can place items in shopping carts.

Each object of the application, such as the product or the customer is a resource that is externally accessible via a Uniform Resource Locator (URL). With the following request in the example application, the shopping cart with the number 7621 is retrieved.

```
GET /cart/7621
```

It is not specified in REST how the result of a request is represented. Client and server must have a shared understanding how the data is represented, i.e. in XML or JavaScript Object Notation (JSON). Listing 2 is a response example in JSON format.

```
{
"customer": 7621,
"articles":[
  {"position":1,
  "articleNumber":89,
  "description":"iPhone5",
  "price":200},
```

```
  {"position":2,
  "articleNumber":76,
  "description":"Samsung Galaxy S III",
  "price":150}
  ]
}
```

Listing 2: JSON response example

SOAP vs. REST

The main advantages of REST web services are that they are lightweight, without a lot of extra XML markup. As well REST has easy to read results and is easy to build, requiring no special tool-kits.

SOAP also has some advantages. Usually it is easy to use, provides relatively strong typing, since it has a fixed set of supported data types and as well, many different kinds of development tools are available.

Next some aspects of SOAP and REST will be compared.

API Flexibility & Simplicity The key to the REST methodology is to use an interface that is already well known and widely used, the Uniform Resource Identifier (URI), in order to write web services. For example, providing a currency converter service, in which a user types-in the desired currencies for input and output and the specific amount in order to receive a real-time conversion, could be as simple as making a script accessible on a web server via the following URI: http://www.currencyconverter.com/convert?in=us-dollar&value=100&out=euro

This service could easily be requested with an HTTP GET command by any client or server application with HTTP support. The resulting HTTP response depends on how the service provider wrote the script and it might be as simple as some standard headers and a text string containing the current price for the given currencies, or it might be an XML document.

The significant advantages of this interface method over SOAP-based services are as follows:-

The creation and modification of a URI in order to access different web resources can easily be figured out by any developer. However, in order for SOAP to be used, most developers would need a SOAP toolkit to form requests and obtain the results, as it requires specific knowledge of a new XML specification.

Bandwidth Usage The RESTful interface has short requests and responses, which is another advantage, whereas, an XML wrapper around every request and response is required for SOAP. For a four- or five-digit stock quote, a SOAP response may require more than 10 times the number of bytes for the same response in REST, as SOAP requires namespaces and typing to be present.

Security The security perspective debate is probably the most interesting aspect of the comparison between REST and SOAP.

Sending Remote Procedure Call (RPC) through standard HTTP ports is seen by the SOAP camp as being a good way to ensure Web services support across organizational boundaries. In contrast however, REST followers see this as compromising network safety and considers this practice a major design flaw.

With REST, the administrator (or firewall) can discern the intent of each message by analyzing the HTTP command used in the request, even though the REST calls also go over HTTP or HyperText Transfer Protocol Secure (HTTPS). For example, a GET request is always seen as being safe, because by definition, the data cannot be modified, it can only query data.

On the other hand, HTTP POST is used by a typical SOAP request to communicate with a given service. Without looking into the SOAP envelope, it is not possible to know whether the request simply wants to query data or delete entire tables from the database. This task is resource-consuming and it is not built into most firewalls.

On the downside, with SOAP, the difficult task of authentication and authorization is left up to the application developer. However, the fact that the web servers already have support for these tasks is taken into account by the REST methodology. REST methodology developers can make the network layer do all the heavy work by using industry-standard certificates and a common identity management system, such as an Lightweight Directory Access Protocol (LDAP) server.

However, REST is not perfect. It is not always the best solution for all web services. Data should never be sent as parameters in URIs in order to be kept secure.

Type Handling Due to its fixed set of supported data types, SOAP provides stronger typing. In this way, it ensures that a return value will be given in the corresponding native type in a specific platform. For example, when an Application Programming Interface (API) is HTTP based, the return value will need to be deserialized from its original XML format before being type-casted. However, handling complex data-types proves to be the main challenge and is mainly achieved

by defining a serialization and deserialization mechanism, therefore, there ease of client-side coding has no definitive advantage.

Client-side Complexity Making calls to an REST API poses less of a challenge than making calls to a SOAP API. While REST is elementary to all programming languages and merely implies constructing an HTTP request with the appropriate parameters, the latter requires a client library, a Stub and involves additional learning.

Testing and Troubleshooting A further characteristic of REST APIs is their easy testing and troubleshooting ability, requiring no more than a browser, the response appearing in the browser window itself. Generating a request does not require special test tools which is a major advantage of REST based APIs.

Server-side Complexity The majority of programming languages provide easy to operate mechanisms in order to expose a method using SOAP. However, exposing a method using REST based APIs involves additional effort due to the task of mapping the URI path to specific handlers. Though various frameworks facilitate this task, the exposition of methods is still easier to achieve using SOAP than REST.

Caching To consume a REST based API service, a simple GET request is needed, therewith allowing proxy servers to cache their response very easily. In contrast, SOAP requests use POST and requires a complex XML format, producing difficulties for response-caching.

2.3.2 NoSQL Databases

Structured Query Language (SQL) databases have been used to solve storage problems for a long time, including cases in which there is a high discrepancy between the object model and its relational model. The conversion of graphs to tables represents yet another dis-functional use of data mapping. The complex structure of these kinds of mismanagement causes, depends on mapping frameworks and complex algorithms. The rigid relational scheme characteristic for SQL becomes especially inefficient for such web applications as blogs, due to their multifaceted range of attributes that need to be stored in their respective tables, e.g. comments, pictures, audios, videos and source codes. Therefore, adding or removing a new feature to this sort of website could result in system unavailability.

Nowadays of course, web sites are developing towards more interactive models, obliging databases to perform real-time schema updates, thereby paving the way for Not Only SQL (NoSQL) to provide a database molded for modern demands.

There are a variety of ideas surrounding the NoSQL movement, however, the core idea is to provide more flexible data models, as opposed to the SQL approach, in order to provide live schema updates. The ever increasing amount of data streaming through the web implies challenges, which any competitive website wishing to stay in business will have to meet. Besides dealing with vast amounts of data, these sites have to respond to constant requests around the globe without allowing any noticeable latency.

To this end, many companies have developed their own storage systems, according to their specific needs, which have been classified as NoSQL databases. Considering the fact that these stores are set up to fulfill the individualized requirements of the companies they belong to, there can be no final answer as to which one works most efficiently. For example, Facebook implemented the NoSQL database Cassandra, in order to solve the so called "Inbox Search Problem" - the challenge of allowing Facebook users to search through their sent and received messages - caused by the multitude of stored data along with the high number of active users.

A selection of the most known NoSQL systems are shown in Table 2.1, which are categorized into the following groups: Key Value Stores, Document Stores, Column Family Stores and Graph Databases

Key Value Stores	Document Stores	Column Stores	Graph Databases
Riak **Amazon SimpleDB** **Voldemort** **Redis**	**CouchDB** **MongoDB** **Couchbase**	**HBase** **Hypertable** **Cassandra**	**Neo4J** **AllegroGraph**

Table 2.1: Most known NoSQL systems

The following subsections will examine these groups, each accompanied by one or more exemplary implementations.

Key Value Stores

Key value stores provide suitable storage systems for simple operations, based on key attributes only. They can be compared to maps or dictionaries due to the fact that data is identified by a unique key. They allow for a user specific webpage to be partially calculated beforehand and consequently be served quickly to the user when requested. Most key value stores save their data in memory, so they are frequently used for the caching of more time intensive SQL queries. Examples for key value stores are Project Voldemort, Redis, Membase and Riak and the latter will be described in detail in the following paragraph.

Riak: Riak is a distributed, scalable, open source key/value store. Riak scales predictably and easily and simplifies development by giving users the ability to quickly prototype, test, and deploy their applications. One of two ways to access data in Riak is by using a REST API. The other way to access data is through a fully-featured Protocol Buffers API. This is a simple binary protocol based on the library Google's open source project of the same name.

The only one way to organize data inside of Riak is by using buckets and keys. Data is stored and referenced by bucket/key pairs. These buckets are used to define a virtual keyspace and provide the ability to define isolated non-default configuration. They might be compared to tables or folders in relational databases or file systems respectively [15].

Interactions with Riak are either setting or retrieving the value of a key. The following describes this using the Riak HTTP API. In order to read an object, a basic request is used to retrieve a specific key from a bucket.

```
GET /riak/bucket/key
```

The body of the response will contain the content of the object (if it exists).

For storing an object with a user-defined key, the basic request looks like this:

```
PUT /riak/bucket/key
```

There is no need to explicitly create a bucket because it is automatically created when keys are added to them.

Document Stores

Document Stores interlace key value pairs in JSON or JSON like documents. Each of these documents contains a special key *ID*, which is unique throughout a collection of documents and therefore identifies a document explicitly.

Key value stores don't allow values to be queried, because the values are only accessible through their respective keys. Document stores on the other hand, provide mechanisms for this additional function. They therefore allow complex data structures to be handled more efficiently. The interpretable JSON format applied by document stores ensures a developer frien dly usage by supporting data types. In contrast to key value stores which focus on high performance for read and write concurrent, document stores concentrate on storing big data efficiently while also providing high query performance. The most known document stores are CouchDB and MongoDB which are described in detail in the following paragraphs.

CouchDB: CouchDB is a free open source document oriented database developed by Apache Software Foundation. It is written in Erlang, a programming language aimed at concurrent and distributed applications. It was first released in 2005 and in 2008 it became an Apache project.

CouchDB implements a Multi Version Concurrency Control (MVCC) to avoid the need to lock the database file while writing. The application has to be aware of resolving conflicts, which generally involves merging data into one of the documents, then deleting the stale one.

CouchDB adopts a semi structured data model based on JSON format, which is a collection of named key-value pairs. The value can either be a number, string, boolean, list or a dictionary. The data model is completely schema-free and it only needs to be conformed to the JSON specification. An example document is shown in listing 3, which represent the apple price in various supermarkets.

```
{
    "_id" : "bc2a98726578c326ec68382f846d7629",
    "_rev" : "8763642898",
    "item" : "apple",
    "prices" : {
        "ALDI" : 1.59,
        "LIDL" : 5.99,
        "Kaufland" : 0.79
    }
}
```

Listing 3: Example of a CouchDB document

Each document is identified by a unique ID (the *id* field) within a CouchDB database . CouchDB can simply be seen as a collection of documents and it does not know or track relationships between documents [13]. It provides a RESTful

HTTP API to perform basic CRUD operations on all stored documents, which can easily be done via the HTTP methods POST, GET, PUT and DELETE.

Figure 2.2 illustrates how easy the data synchronization is between any two databases. After replication, each database is able to work independently.

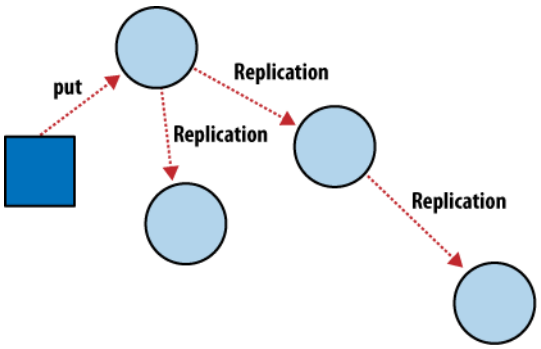

Figure 2.2: CouchDB replication
[26]

CouchDB relies on the MVCC model for handling conflicts. Each document has a revision id and on each update of a document, the old version is kept and the new version is given a different revision id. In case of a conflict, the desired version is always saved as the most recent version and the old version is saved in the document's history. Thereby, the application handles the conflict by itself and decides which version is the desired one to keep and which is the old one [13].

MongoDB: MongoDB is a free and open source document oriented database written in C++ that provides high performance, high availability, and easy scalability. The development is supported by the open source community and also by the company 10gen. As in CouchDB, it is completely schema-free and its data model is based on JSON format that is easy to code, easy to manage, and yields excellent performance by grouping relevant data together internally [10].

The document structure in MongoDB are BSON objects, short for Binary JSON, which is a binary-encoded serialization of JSON-like documents. Binary JSON (BSON) supports all JSON data types but also defines new types, i.e. the Date data type and the BinData type [3].

Like CouchDB, the documents are saved within collections, which can be compared to tables in relational databases. All kinds of documents can be stored within collections but in order to provide a logical way of organizing data, the documents within a collection usually have the same structure.

MongoDB provides two ways for replication, namely Replica Sets and Master-Slave, see figure 2.3. Master-Slave replication has been deprecated since version 1.6 therefore Replica Sets are used for all new production deployments.

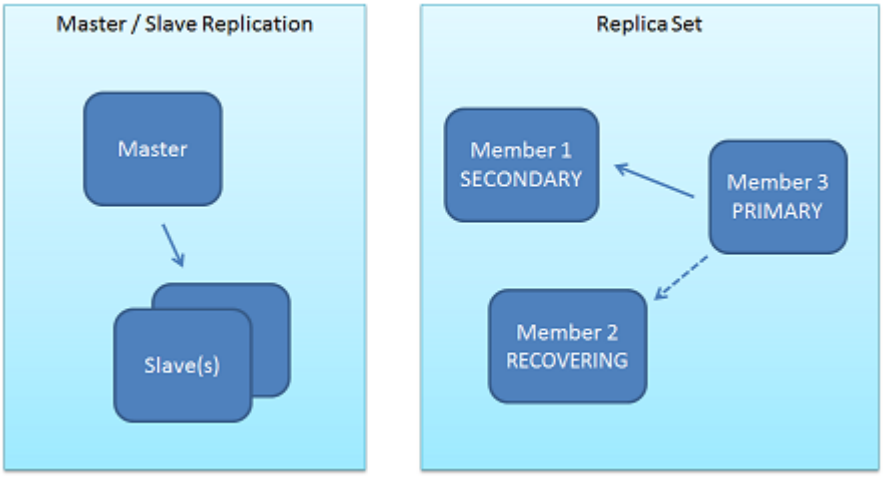

Figure 2.3: MongoDB replication
[12]

Column Stores

Column Family Stores are known as column oriented stores, extensible record stores or wide columnar stores. All stores are inspired by Googles Bigtable, which is a distributed storage system for managing structured data that is designed to scale to a very large size.[30, 18]

Bigtable has been used by Google in many projects, which require high throughput and low latency data serving. The data model is described as a sparse, distributed, persistent multidimensional sorted map [18] . Any number of key value pairs can be stored in rows within a map. In contrast to document stores, column stores cannot be queried by value and therefore one is required to write the data the way one intends to read it, meaning one must think about the data model in terms of ones queries. In order to achieve both versioning and consistency, multiple versions of a value are stored chronologically.

Big companies such as Google and Facebook, use their respective Column Family Stores (BigTable and Cassandra) to store data in the same way, as it will be required when requested later on. This sort of storage ensures that only one data retrieval is needed when a request is launched, therefore saving considerable effort and maximizing performance as compared to the conventional process via MySQL which often requires multiple queries.

Cassandra: Apache Cassandra is a free and open source distributed, structured key-value store with eventual consistency. It was initially developed by Facebook

and now is developed by the Apache Foundation. It is designed to handle very large amounts of data, while providing high availability and scalability.

A table in Cassandra is a distributed multi dimensional map indexed by a key. The value is an object which is highly structured [16]. Columns are grouped together into sets called column families very similar to the Bigtable system. Cassandra exposes two kinds of these columns families, Simple and Super column families. Super column families can be visualized as a column family within a column family[16].

Furthermore, applications can specify the sort order of columns within a Super Column or Simple Column family. The system allows columns to be sorted either by time or name. Time sorting of columns is exploited by the application Inbox Search where the results are always displayed in time sorted order.

Lakshman and Malik have introduced in their article [33] the Facebook Inbox Search in which they maintain a per user index of all messages that have been exchanged between the sender and the recipients of the message. They introduced two kinds of search features, namely *term search* and *interactions* - given the name of a person returning all messages that the user might have ever sent or received from that person. The schema consists of two column families. For query a *term search*, the user id is the key and the words that make up the message become the super column. Individual message identifiers of the messages that contain the word become the columns within the super column. For a query an *interactions* again the user id is the key and the recipients id's are the super columns. For each of these super columns the individual message identifiers are the columns. In order to make the searches faster, Cassandra provides certain hooks for intelligent caching of data. For instance, when a user clicks into the search bar, an asynchronous message is sent to the Cassandra cluster to prime the buffer cache with that user's index. This way, when the actual search query is executed, the search results are likely to already be in memory. The system currently stores about 50+TB of data on a 150 node cluster, which is spread out between east and west coast data centers. Table 2.2 shows some production measurement for read performance.

Latency Stat	Search Interactions	Term Search
Min	7.69ms	7.78ms
Median	15.69ms	18.27ms
Max	26.13ms	44.41ms

Table 2.2: Cassandra production measurement for read performance by Facebook [33]

Graph Databases

Graph databases are specialized with efficient management of heavily linked data and therefore are more suitable for applications based on data with many relationships, thereby cost intensive operations such as recursive joins can be replaced by efficient traversals [30].

Neo4j and GraphDB are examples of graph databases. They are based on directed and multi relational property graphs. Nodes and edges consist of objects with embedded key value pairs. The range of keys and values can be defined in a schema, whereby the expression of more complex constraints can be easily described. Therefore, it is possible to define that a specific edge is only applicable between a certain types of nodes. Use cases for graph databases are location based services, knowledge representation and path finding problems raised in navigation systems, recommendation systems and all other use cases which involve complex relationships [30].

Twitter has implemented their own graph database FlockDB in which they store a lot of relationships between people in order to provide their tweet following service. These one-way relationships are handled by FlockDB which is optimized for very large adjacency lists, fast reads and writes.

Neo4J: Neo4J is an open source project implemented in Java and has been developed by Neo Technology since 2007. In contrast to relational databases, which map nodes and edges in tables, Neo4J supports these graph elements natively which involves an easy structure of the data model.

The node is the basic component within a graph in Neo4J, which has a unique and continuous ID. Nodes can optionally have schema-less properties such as key-value pairs which means that each node can have different keys. This is more flexible than relational databases, which need a separate table for each node type.

A special node within Neo4J is the reference node or the so-called root node, which is the entry point of each graph. This node is automatically created while creating a new database. Each node should be indirectly connected to the reference node in order to be reachable to the reference node.

The edge is another component of the graph in Neo4J, which connects nodes together and provides a mechanism to traverse over the database tree. Similar to nodes, edges have an unique ID and optional schema-less properties such as key-value pairs.

Neo4J supports two types of deployment, either as embedded or as a server, see figure 2.4.

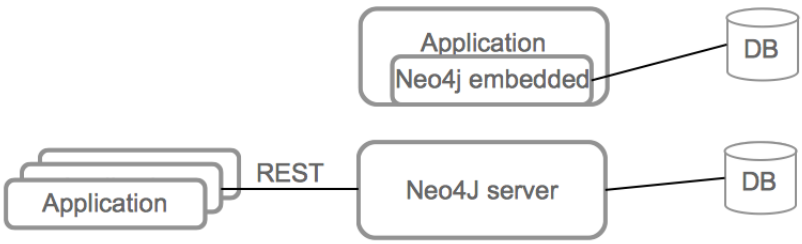

Figure 2.4: Embedded vs. Server deployment of Neo4J

2.3.3 Message broker

A Message Broker is a centralized hub that simplifies communication among heterogeneous systems [29]. It hosts messaging destinations like queues and topics for the purposes of asynchronous communication, meaning that the sender and receiver of a message do not need to interact with the message queue at the same time. Messages placed in a queue are stored until the recipient retrieves them. There are a number of open source choices of message brokers, including JBoss Messaging, Apache ActiveMQ, WebSphere Message Broker, BizTalk and RabbitMQ.

RabbitMQ RabbitMQ is an open source message broker which implements the Advanced Message Queuing Protocol (AMQP) standard. It has been developed by Rabbit Technologies Ltd. which was acquired in April 2010 by VMWare.

Figure 2.5 shows the four basic components of the AMQP concept which consist of a producer, exchange, queue and a consumer. Producers publish messages to exchanges, which can be compared to post offices or mailboxes. Exchanges then distribute these messages to queues using rules called bindings. Messages are then delivered to consumers subscribed to queues, or the consumers fetch/pull messages from queues on demand [2].

Thereby, the producer never sends the message directly to a message queue, rather to the exchange. Hereby, the exchange must know what to do with the message and to which message queue it should be forwarded to. This is the task of the rules that are defined by four different exchange types: topic, headers, direct and fanout, the last two types being described below.

- **Direct:** Direct exchanges are often used to distribute tasks between multiple workers (instances of the same application) in a round robin manner. When doing so, it is important to understand that, in AMQP , messages are load

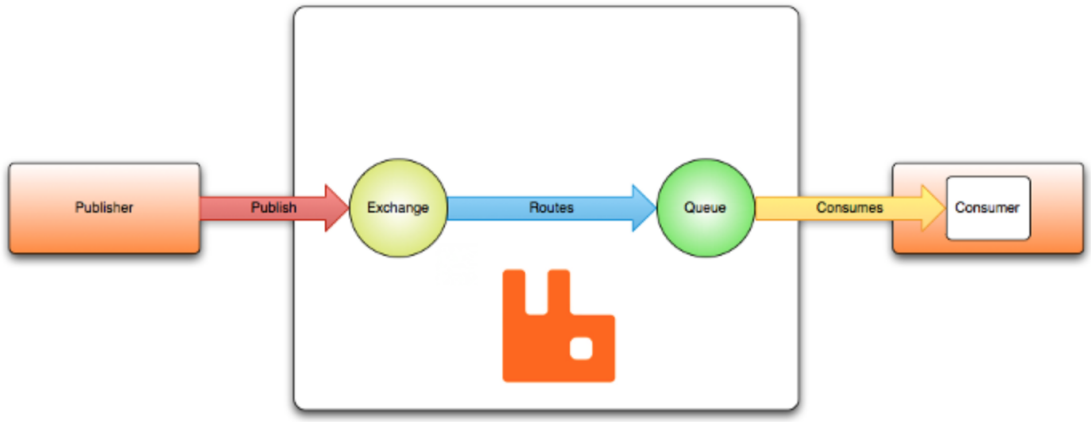

Figure 2.5: RabbitMQ Components
[2]

balanced between consumers and not between queues [2], see figure 2.6 for an illustration.

- **Fanout:** Fanout exchange routes messages to all of the queues that are bound to it and the routing key is ignored [2]. A fanout exchange can be represented graphically in figure 2.7.

ActiveMQ ActiveMQ is an open source message broker developed by the Apache Foundation and is implemented in Java. ActiveMQ has implemented the Java Message Service (JMS) protocol. The advantage over RabbitMQ is that these brokers can be embedded in a Java application and this simplifies the delivery of a software package. On the other hand, ActiveMQ implemented the AMQP protocol first in the version 5.8.0, which was released in February 2013 and therefore their implementation of the AMQP protocol have not been widely used or tested in production environments.

2.3.4 Search Platforms

The mean idea behind search engines is to help users to quickly find useful information from the web. There are many available solutions that perform the task of information retrieval, however, only some of them are popular.

Over time, user queries are becoming more complex and personalized. Where once only a SQL LIKE clause was good enough, today's usage sometimes calls for sophisticated algorithms. Fortunately, a number of open source and commercial

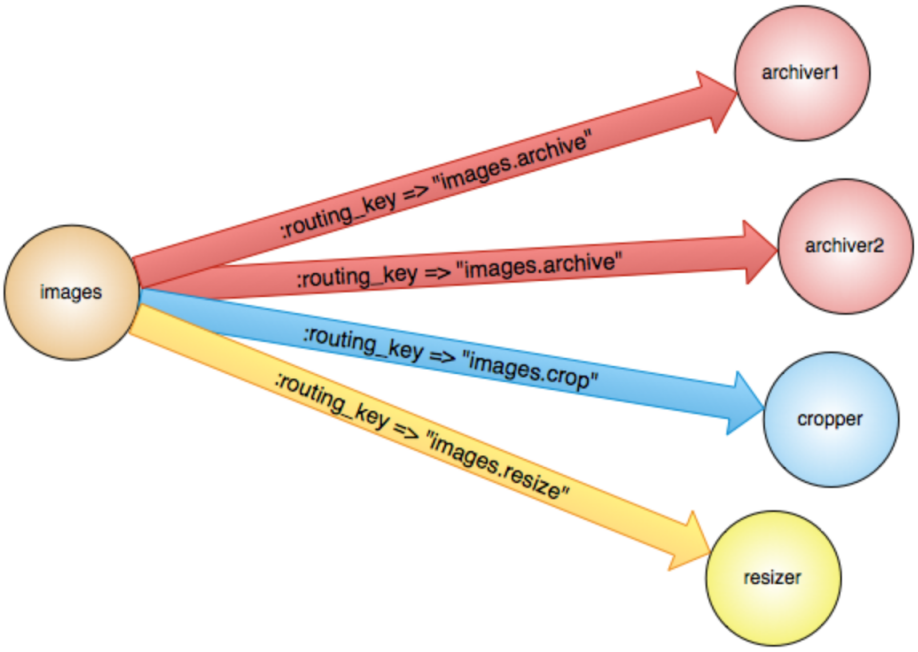

Figure 2.6: Direct exchange routing
[2]

platforms address the need for more powerful and more widely distributed search technology, including Apache Solr, Amazon's CloudSearch, and Elasticsearch.

Apache Solr

Solr is a popular, blazing fast open source enterprise search platform based on the Apache Lucene project. It is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat.

Solr has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly scalable, providing distributed search and index replication and it powers the search and navigation features of many of the world's largest internet sites [8].

Elasticsearch

Elasticsearch is an open source search engine implemented in Java and is based on the Apache Lucene project. It provides both an indexing service as well as a data store and does not require a detailed schema. This means that the data which

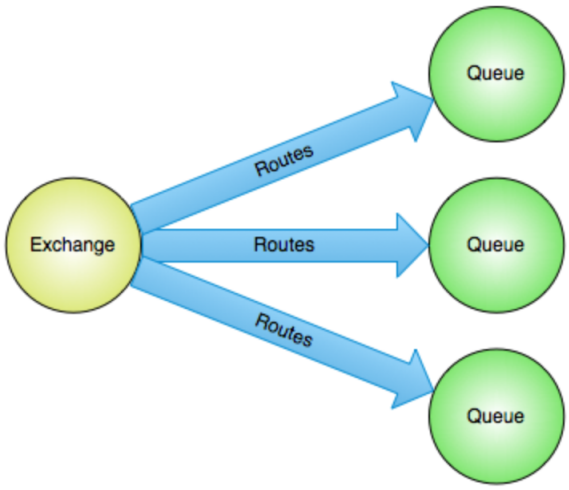

Figure 2.7: Fanout exchange routing
[2]

can be expressed as JSON can be easily stored and indexed within Elasticsearch. Listing 4 shows two examples of indexing data that are in JSON format. These examples show how easy the schema can be extended, i.e. the second JSON data has a new tag called *author*.

```
curl -XPUT http://localhost:9200/blogs/blog/1 -d '{
    "post_date": "2012-12-15T09:00:00",
    "content": "This is the first blog"
}'

curl -XPUT http://localhost:9200/blogs/blog/2 -d '{
    "author": "tom",
    "post_date": "2012-12-16T11:00:00",
    "content": "Second blog"
}'
```

Listing 4: Schema-less Elasticsearch

Elasticsearch uses an internal NoSQL database system supporting JSON documents. Due to its multi-tenancy, an Elasticsearch instance can have more than one index which consists of types, which again consist of fields. Comparable to relational database systems, an index is equivalent to a database, a type is comparable to a table and a field is comparable to a column. No type schema definition is necessary which makes Elasticsearch highly flexible.

One of the unique features of Elasticsearch that makes it especially well suited to distributed systems, is the ease in which it can can be scaled. It provides the ability to either add or remove resources, which are individual machines running Elasticsearch, at any time in order to support a growing data set or perhaps to satisfy an increasing amount of requests and also improve the performance of the desired solution.

2.3.5 Spring Framework

The Spring framework is a lightweight solution and a potential one-stop-shop for building enterprise-ready applications. Due to its modularity, one need only use those parts that one needs, without having to include all other modules i.e. one can use the IoC container, with Struts on top, or else one can use only the Hibernate integration code or the Java Data Base Connectivity (JDBC) abstraction layer. The Spring Framework supports declarative transaction management, remote access to the application logic through Remote Method Invocation (RMI) or web services, and various options for persisting the data. It offers a full-featured Model View Controller (MVC) framework, and enables integrating Aspect Oriented Programming (AOP) transparently into softwares [6].

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP, Instrumentation, and Test [6], see figure 2.8 for illustration.

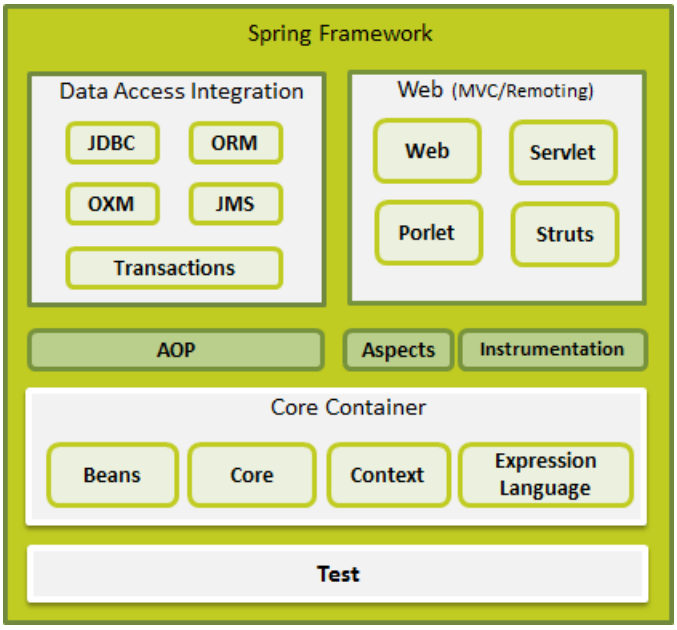

Figure 2.8: Overview of the Spring Framework [6]

3 Requirements

As described in section 1.2, the objective of this thesis is to design and develop a generic contextual content management framework, which supports the correlation of context information and multimedia content and also provides an efficient and scaleable discovery and distribution mechanism. In the following sections, two instructive use cases are provided. These use cases illustrate the functional assets of the framework and facilitate a deduction of its functional and non-functional requirements.

3.1 Scenarios

For better comprehension of the functionality of the proposed framework, two examples are given in this section as use case scenarios.

Mobile capturing of live event A user - as a content provider - captures a live event (e.g. demonstration, car race, marathon, Tour de France, etc.) using an application on a GPS capable smart phone. While filming, the application also collects context information (e.g. location, acceleration, temperature, time, etc.). Later, the user uploads the captured content with its context information to the framework.

Let's consider a video of Tour de France as an example of the uploaded content. Any consumer, who is interested in a specific uploaded video that has been taken in a specific place on the road of the tour (e.g. Les Essarts: town which is located in western France), searches for the video by specifying some related information such as time range, location and "Tour de France" as a search string. The framework will then give the user a list of all videos that match the specified criteria. The user can select any of the listed videos and begin streaming.

Restaurant guide Another approach would be a domain oriented search (e.g. restaurants, gas-station's, public libraries...). Considering a hungry user looking for a suitable restaurant; the restaurant guide will help users find facilities based on search criteria e.g name, place, cuisine, ratings ... etc. and then display images or videos of the selected restaurant and other information e.g price list, daily menu etc.

3.2 Functional requirements

In 'normal' applications, the functional requirements serve the purpose of describing what the system should do. This applies to both internal processes, and the interaction of the system with its environment. These requirements are derived commonly from use cases. Frameworks are different, thereby making the identification of functional requirements more difficult. Frameworks usually do not address specific use cases, but are supposed to be open for varied scenarios. The functional requirements of the framework are therefore rather abstract.

User Management Administrators of the framework can grant access to other administrators or providers, thereby ensuring a simple user management. Once a developer has been registered, he can administrate the flow and accessibility of the multimedia content via his respective applications.

Applications Management The framework should provide an easy mechanism for creating and deleting applications. When a developer wishes to create a new application, he sends a request including global configurations. A developer may also grant access to other developers to use his applications. This access either grants full administrative control over the applications or can be limited to uploading/downloading and searching activities.

Content & Context Data Store Furthermore, the framework should provide a mechanism for creating data stores for multimedia contents and their related context. In order to ensure legitimate context-based search results, the framework facilitates the correlation between content and its related context.

Content Discovery In order for the developer to discover context-based contents, an adjustable search engine is required. To this end, the developer defines the parameters relevant to potential search requests. The developer can respond to new challenges and refine search options as time goes on by adding new parameters to the original set.

Content Adaptation Based on the global configuration of the application, the framework should support content transcoding. Transcoding refers to optimizing processes, e.g. quality adjustments for efficient use in varying networks (mobile, Wifi), size adjustment for individual displays in the area of video contents or Word to PDF conversions for textual contents.

Content Distribution The framework should support content delivery to most widespread internet devices. It should also optimize delivery by streaming contents from the nearest available server, thereby minimizing the network latency and reducing bandwidth costs.

3.3 Non functional requirements

Non-functional requirements are mainly related to the quality aspects of a system. As the implementation of the design presented in 4 is considered in this work as a prototype, the non-functional requirements play a subordinate role. However, some non-functional requirements have quite an influence on fundamental architectural decisions. Nevertheless, it is important to analyze these requirements. The following section outlines the non-functional requirements for the development of the framework.

3.3.1 Usability

Since this thesis is concerned with the development of a framework and not a concrete application or Graphical User Interface (GUI), this requirement is limited. However there are certain ease-of-use requirements that are relevant, constituting the degree of effort needed to comprehend, evaluate and effectively use the software.

All functionalities of the framework shall be accessible in a simple way. From a developer's point of view, who in a sense represents the "user" of the framework, it can be said that a good structure and readability of the source code is desirable. Changes and enhancements to the framework shall always be restricted to as few logical components as possible. More importantly however, the standard use of the framework shall not require deep knowledge of the internal structure of the framework's components. Creating a new application for instance, shall be possible without further knowledge of the framework via the external interface of the framework.

3.3.2 Efficiency

Efficiency describes the response time for inquiries, as well as the consumption of resources. The framework shall be capable of serving multiple applications simultaneously. The creation of an additional application shall only marginally affect the performance of the overall system.

3.3.3 Scalability

The framework shall be scaled easily. Thereby, it shall provide a mechanism to either add or remove resources at any time, which are individual machines, in order to support a growing data set or perhaps satisfy an increasing amount of requests and improve the performance of the desired solution.

4 Design

Against the background of the requirements described previously in chapter 3, this chapter provides a design concept for the framework. Section 4.1 will describe a basic architectural concept of the framework to be refined in section 4.2 which will provide a more detailed description of the framework's components and their interactions.

4.1 Architecture Overview

Figure 4.1 shows an architecture overview of the framework, which consists mainly of six components. The components with three dots(...) means that the framework can later be extended with new components for adding new services to the framework such as Subscribe/Notification or context enrichment through M2M etc.

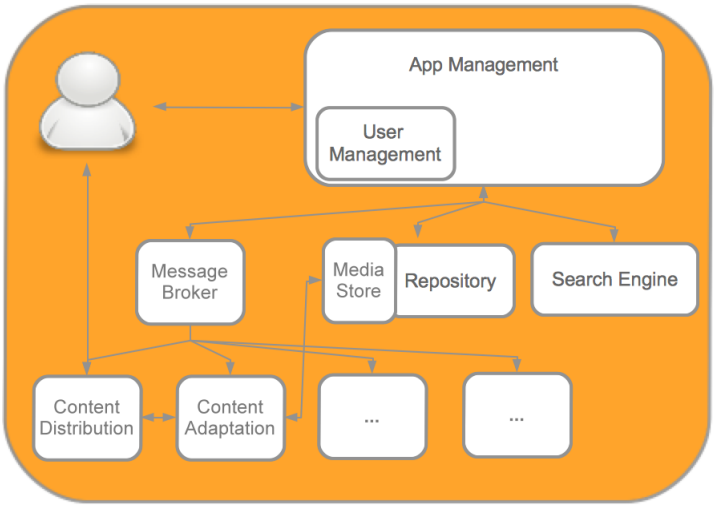

Figure 4.1: Architecture Overview

4.2 Framework Components

The app management component is the main entry for each developer to interact with the framework. These developers need to authenticate themselves in order to use the framework and therefore an appropriate user management is needed. Furthermore, the repository/media store is needed in order to store the data and the multimedia files. For efficient data discovery, the search engine component is required. The task of the content adaptation component is to convert the stored files to other formats and then upload these converted files to the content distribution component which will then serve them to various devices.

Those components are described in more detail in the next subsections.

4.2.1 App Management

The app management component is the core of the entire framework and it interacts with almost all components in the framework. The following subsections describe the design decision that have been made for implementing this component.

JSON

The data format to interact with the framework is JSON. It has been chosen due its low-overhead compared with XML and it is also the data format for many of the known NoSQL systems, one of which will be used in the framework.

User Management

In order to allow only registered developers to use the framework, a user management design concept is described in the following.

The user management component provides two levels of management. The first one is to provide a role based user management for using the entire framework. The role can be either an administrator or a normal user - which here means a developer who uses the framework. Both administrator and user are allowed to use the framework i.e. creating new applications, modeling the app and storing, obtaining or deleting content etc. Only administrators can add a new administrator or user.

The second level of the user management component provides a mechanism for managing who can modify an existing app or store/get contents from this app. The user who created the app is theoretically the owner of it and he is the only one allowed to add other users for using his app. Thereby, he can add users with the same rights as his own rights, meaning they can modify the whole app, deleting content etc., or add other users, who only allowed to store/get/search content. For example, this is useful for a developer, who has deployed/configured a new app

and then lets a service provider use his app by only storing, obtaining, or searching the contents in this app.

REST Interfaces

For interacting with the outside world, a standard and well defined interface is needed. The decision for choosing REST as an API interface is due to its flexibility, simplicity, less bandwidth usage and easy way to scale for large deployment, see section 2.3.1 for more detailed comparison.

The following paragraphs describe the various interfaces needed in order to use the framework:

Framework User Management Interface /users/ : This interface provides the first level of user management described in 4.2.1. Only administrators are allowed to use this interface for managing users.

- **Create:** The method *addUser* requires the following parameters, *username*, *password*, *firstname*, *lastname*, *email* and *role*. The role parameter is an integer and specifies if the user has ADMIN - 1 - or USER - 2 - rights. These parameters are passed through URI parameters, see the following example for adding a new administrator.

```
POST https://user1:pass@107.23.121.185:8443/cccd/
/users?username=frank&password=pass2&firstname=frank&
lastname=schulz&email=frank21@yahoo.de&role=1
```

- **Delete:** The method *deleteUser* requires only a *username* as a parameter. It does not allow one to delete the *username* if he is the last administrator in the framework, otherwise no one can administrate the framework.

```
DELETE https://user1:pass@107.23.121.185:8443/cccd/
/users?username=frank
```

App interface /app/ : This interface provides the four Create Read Update Delete (CRUD) operations described below, which allow developers to interact with their apps.

- **Create(POST):** This method *createApp* creates a new app and it requires a name and global configuration parameters for the app. For example, these configuration parameters can be a mandatory secret word to be used later for securing the contents which belongs to this app or a list of encoding profiles, which will be then used to transcode each uploaded video within this app

in order to support various devices i.e. iPad, iPhone, PC and etc. Listing 5 shows a JSON example for creating a new app with **vod1** as a name.

```
POST https://user1:pass@107.23.121.185:8443/cccd/app/vod1
```

Payload:

```
{
"secret":"pass123",
"profiles":["cell_4x3_150k","wifi_4x3_640k","wifi_4x3_1240k"]
}
```

Response:

```
{
"ok": "1",
"debug": "app: vod1 created"
}
```

Listing 5: Creating a new app

- **Read(GET):** There are two methods in this app interface, which can be consumed through a HTTP GET. The first one *listApps* is for listing all apps which belongs to the user, see listing 6 as an example.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app
```

Response:

```
{
"data":["app3","vod","vod1","app1"],
"ok":"1"
}
```

Listing 6: Listing all apps which belong to a user

The second method *checkApp* returns either the configuration of the specified app, or the data that reflects the amount of storage used and data contained in a specific app, as well as object, collection, and index counters. This data can be used to check and track the state and storage of a specific app. The example below will return this data for an app called *vod1*.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app/vod1
GET https://user1:pass@107.23.121.185:8443/cccd/app/vod1?op=stats
```

- **Update(PUT):** This method *updateApp* is for updating/adding the configuration parameters of the app. i.e changing the secret word or adding new configuration parameters which could be needed from other components, i.e. adding the user credentials for a Justin.TV account in order to stream a video to it.

```
PUT https://user1:pass@107.23.121.185:8443/cccd/app/vod1
```

Payload:

```
{
"secret":"secret2"
}
```

- **Delete(DELETE):** This method *deleteApp* deletes the app along with its related contents which might be in the repository, media store or in the content distributor component.

App User Management Interface /app/appName/users : This interface provides the second level of user management described in 4.2.1. The users table is saved on the repository within the app. In order to add a user to the app with WRITE or READ only rights, the user must be already registered in the framework as a valid user.

- **Read:** The method *listUsers* takes only the *appName* as a parameter and returns a list of all users who have READ or WRITE rights to this app.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app/vod1/users
```

- **Update:** The method *addUser* takes the following parameters, *appName*, *username* which needs to be added to the app and the parameter *readonly* that states which rights the user will have. The *username* and *readonly* parameters are passed through URI parameters, see the following example.

```
PUT https://user1:pass@107.23.121.185:8443/cccd/app/vod1
/users?username=frank&readonly=true
```

- **Delete:** The method *deleteUser* takes two parameters, namely *appName* and the name of the user *username* which needs to be deleted from the app. It does not allow the deletion of the *username* if this user is the last user who has WRITE rights, otherwise no one else can administrate the app.

```
DELETE https://user1:pass@107.23.121.185:8443/cccd/app/vod1
/users?username=frank
```

Collection Interface /app/appName/collections/ : The collection within an app can be compared with tables in SQL systems and it contains the real data, i.e. the metadata for content. This interface provides only Create, Read, and Delete operations.

- **Create:** The method *createCollection* requires two argument, the *appName* and the name of the new collection *collName*. This method automatically creates an empty index in the search engine, which can later be used for searching within this collection. The name of the created index is based on the name of the app combined with '_' and the name of the collection, i.e if the name of the app is *vod1* and the name of the collection is *collection1*, the created index name would be *vod1_collection1*.
- **Read:** The method *listCollections* is for listing all collections within an app, see listing 7 for an example.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app/vod1
/collections
```

Response:

```
{
"data":["collection1","collection2"],
"ok":"1"
}
```

Listing 7: Listing all collections within an app

- **Delete:** Deleting a collection would delete the entire data in it and also delete its index in the search engine.

Mapping Interface /app/appName/collections/collectionName/mapping : Before adding data to any collection, the fields should be mapped to a proper object, i.e. string, integer, date, array or geo_point.

Mapping is the process of defining how a document should be mapped to the search engine, including its searchable characteristics such as which fields are searchable and if/how they are tokenized. By default, there is no need to define explicit mapping, since one is automatically created and registered when a new type or new field is introduced and has sensible defaults. A mapping definition only needs to be provided when the defaults need to be overridden [31]. Only Get and Update operations are provided through this interface.

- **Read:** The method *listAllMappings* takes as parameters the *appName* and the *collName* and returns the entire mapping list for this collection.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app/vod1
/collections/collection1/mapping
```

Payload:

```
{
"data": {
  "properties": {
    "date": {
      "type": "date",
      "format": "dateOptionalTime"
      },
    "name": {
      "type": "string"
    },
    "owner": {
      "type": "string"
    }
  }
},
"ok": "1"
}
```

- **Update:** The method *updateMapping* takes as parameters the *appName*, the *collName* and the mapping description *body* as a payload in JSON format and allows one to register a specific mapping definition for a specific collection.

```
PUT https://user1:pass@107.23.121.185:8443/cccd/app/vod1
/collections/collection1/mapping
```

Payload:

```
{
  "collection1" : {
    "properties" : {
      "name" : {"type" : "string"}
    }
  }
}
```

Document Interface /app/appName/collections/collectionName/doc : The repository of the framework is a document-based database, and as a result, all records, or data are documents. Documents are the default representation of most user accessible data structures in the repository.

- **Create:** The method *createDocument* takes as parameters the *appName*, *collName* and the document itself as a payload in JSON format *body* and saves this document in the repository and also in the search engine in order to make the document searchable.

```
POST https://user1:pass@107.23.121.185:8443/cccd/app/vod1
/collections/collection1/doc
```

Payload:

```
{
"name":"video in Berlin",
"date":"2013-01-01T01:12:12",
"owner":"Tom"
}
```

- **Read:** The method *getDocument* requires the *appName*, *collName* and an *objectId* and returns the corresponding document. The example below shows how to obtain the document with the *objectId* 512d3258e4b0acc3647858f2.

```
GET https://abdul:abdul@107.23.121.185:8443/cccd/app/vod1
/collections/collection1/doc/512d3258e4b0acc3647858f2
```

Response:

```
{
"data":{
  "_id":{
      "machine":-458183485,
      "timeSecond":1361916504,
      "inc":1685608690,
      "time":1361916504000,
      "new":false
  },
  "name":"video in Berlin",
  "owner":"Tom",
  "date":"2013-01-01T01:12:12"
},
"ok":"1"
}
```

- **Update:** The method *updateDocument* requires the *appName*, *collName*, *objectid* and the JSON formatted payload *body*. The body contains the fields, which need to be updated within a document.

```
POST https://abdul:abdul@107.23.121.185:8443/cccd/app/vod1
/collections/collection1/doc/512d3258e4b0acc3647858f2
```

Payload:

```
{
"name":"video in Potsdam"
}
```

- **Delete:** The method *deleteDocument* deletes the specific document from the repository and also from the search engine.

```
DELETE https://abdul:abdul@107.23.121.185:8443/cccd/app/vod1
/collections/collection1/doc/512d3258e4b0acc3647858f2
```

Buckets Interface /app/appName/buckets : Just as a bucket holds water, buckets - known in Amazon Web Services - are containers for files. The name of the bucket must be unique within an app. The four CRUD operations are described below.

- **Create:** The method *createBucket* requires two parameters, namely the *appName* and the bucket name *bucketName* and it creates a bucket within an app. The following example is for creating a bucket name *bucket1* within the app *vod1*.

```
POST https://user1:pass@107.23.121.185:8443/cccd/app/vod1
/buckets/bucket1
```

- **Read:** Within this interface there are two methods, which can be consumed through a HTTP GET. The first method *listBuckets* lists all buckets which belong to a specific app.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app/bookstore
/buckets
```

The second method *listAllFilesInBucket* takes as parameters the *appName* and the *bucketName* and lists all the files which are in the bucket *bucketName*.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app/bookstore
/buckets/books
```

- **Delete:** The method *deleteBucket* deletes a bucket along with its files from an app

```
DELETE https://user1:pass@107.23.121.185:8443/cccd/app/bookstore
/buckets/books
```

Files Interface /app/appName/buckets/bucketName/files : This interface manages files within a bucket.

- **Create:** The method *addFileToBucket* allow one to upload a file to a bucket and it requires the parameters *appName* and *bucketName* along with the file itself. Thereby, the name of the uploaded file (not the original name of the file) must be set to *file* in the uploaded form(*file*=@./testDoc.txt). This method returns the *objectid* of the newly created file.

```
curl -F "file=@./testDoc.txt" https://user1:pass1@107.23.121.185:8080/cccd
  /app/vod/buckets/bucket1/files
```

- **Read:** The method *getFileFromBucket* allows one to obtain a file from a bucket and it requires the parameters *appName*, *bucketName* and *objectid*. An example for downloading a file with the *objectid 515ae2dee4b0997c69141cad* from the bucket *books* and app *bookstore* is listed below.

```
GET https://user1:pass@107.23.121.185:8443/cccd/app/bookstore
  /buckets/books/files?objectid=515ae2dee4b0997c69141cad
```

- **Update:** The method *updateFileInBucket* allows one to update a file within a bucket and it requires the parameters *appName*, *bucketName*, *objectid* and the file itself. The update here is theoretically removes the old file and adds the new one.

```
PUT https://user1:pass@107.23.121.185:8443/cccd/app/bookstore
/buckets/books/files?objectid=515ae2dee4b0997c69141cad
```

- **Remove:** The method *deleteFileFromBucket* deletes a file from a bucket.

Informative Response:

In order to know if a request to the framework has succeeded or failed, the API should always send a tag in each response which shows whether the request was processed successfully or not, i.e. "ok":1 in case of success or "ok":0 in case of failure. Furthermore, a configuration parameter will be implemented to enable more debug information, i.e. when creating a new app, if its name exist already, then the response includes the error as a string i.e. "ok":0,"debug":"app name exist already".

4.2.2 Repository/Media Store

The repository/media store is accessible through the REST interfaces described above. The design decisions for this component are listed below.

- JSON data format: As described in section 4.2.1, the REST API data format is JSON and therefor the repository should be based on JSON format too in order to avoid unnecessary data conversations.
- Schema-less: The framework is generic and should support a wide variety of applications and therefore the repository should be based on schema-less solutions in order to allow the application developer to store schema-less data.
- Replication: Data replication ensures redundancy, backup, and automatic failover.
- Distribution: While replication provides basic protection against single-instance failure, when all of the members of a replication group are in a single facility, the replication is still susceptible to some errors in that facility including power outages, networking distortions, and natural disasters. As protection against these failures, the repository/media store component should be easy to distribute in a geographically distinct facility or data center.
- Media Store: In order to store or retrieve content, a file server is needed. This file server can either be a component within the repository itself or a separated component. In order to ensure the correlation between the content stored in the media store and its metadata within the repository, the URI of the content in the media store is embedded in its metadata in the repository.

4.2.3 Search Engine

The search engine is accessible through following REST interfaces, *search* and *mapping*. The first one is for searching withing a collection. The later is for configuring the mapping of fields within a collection. The design decisions for this component are mostly similar to the repository/media store component described in 4.2.2 such as JSON data format, schema-less and easy to replicate and distribute content.

4.2.4 Message Broker

The main idea behind the message broker is to avoid doing a resource-intensive task immediately and having to wait for its completion. Instead, the task will be

scheduled to be done later. The task is encapsulated as a message and will be sent to a queue. A worker process running in other components such as 4.2.5 or 4.2.6 will pop the tasks from the queue and eventually execute the job. When there are a few more of the same worker processes, the tasks will be shared between them.

4.2.5 Content Adaptation

The content adaptation component is a process, which listens for new tasks from the message broker. This component is responsible for transcoding or scaling videos. Furthermore, it splits the transcoded/scaled videos into segments compatible with Apple HTTP Live Streaming (HLS).

Figure 4.2 illustrates how this component works internally. As soon as it receives a message notification from the message broker component, which tells that a new video has been stored and needs further processing i.e. transcoding or scaling, it fetches the video file from the media store and provides this file as an input for the media encoder. The media encoder transcodes and scales this video and returns a MPEG-2 transport stream which is used by the stream segmenter as an input, The stream segmenter breaks this stream into segments and saves these segments as a series of one or more *.ts* media files. The segmenter also creates an index file which contains a list of media files and metadata. This index file is saved as an *.m3u8* playlist. The index file along with the *.ts* media files are then uploaded to the proper path in the content distribution component.

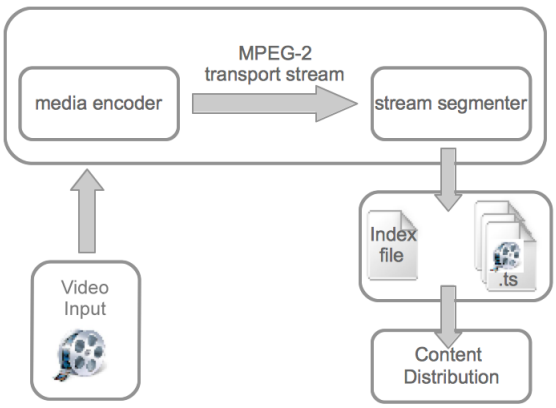

Figure 4.2: Content Adaptation Overview

4.2.6 Content Distribution

The content distribution component consists internally of two components. The first one is an HTTP server to serve contents to consumers, see figure 4.3. The second one is a worker process, which listens for message notifications from the message broker component. These messages tell when a new app is created or deleted. In case of a newly created app, the worker process creates a new dictionary within the HTTP server and secures it with a secret word to grant access only to this app. In the case of a deleted app, the worker process deletes the dictionary for this app along with its entire contents.

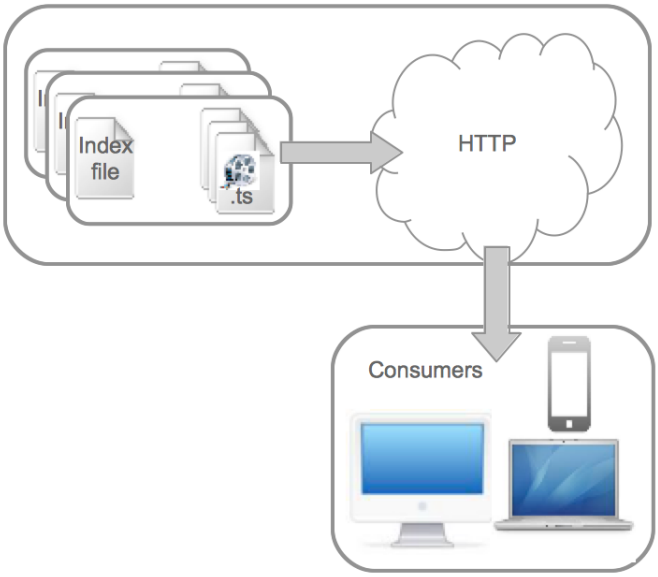

Figure 4.3: Content Distribution

5 Implementation

This chapter deals with the implementation of the design presented in chapter 4. It is not the purpose of this thesis to implement a market-ready product. The focus is clearly on the conceptual part. Therefore, in order to demonstrate the general realization of the proposed design, the implementation is only done in the form of a prototype.

5.1 Tools & Technologies

This section gives an overview of the tools and technologies that have been used to simplify the implementation. The programming languages, which have been used to implement the framework, are Java and Python.

5.1.1 Tools

The following tools have been used to simplify, organize and test the implementation.

Eclipse Juno 4.2 IDE: Eclipse is one of the most widely used IDE for Java. This tool makes it easier to develop Java applications.

Maven 3: Maven is an open source build automation tool developed by the Apache Software Foundation. It uses an XML file called *pom.xml* to describe the software project being built, its dependencies, the build order, directories, and the required plug-ins. It comes with predefined targets for performing certain well-defined tasks such as the compilation of code, its packaging and how and where to deploy the project.

Advanced REST client: Advanced REST client is a plugin within the Chrome browser and can help developers create and test custom HTTP requests. It has been mainly used to test the different REST APIs, which have been developed within the framework.

5.1.2 Technologies

So that the wheel is not reinvented again, most of the components in the framework are based on existing open source technologies.

Spring Framework 3.1.3: As described in 2.3.5, the Spring framework is a lightweight solution and a good base for building enterprise-ready applications. It provides an incredibly powerful and flexible collection of technologies and projects to improve enterprise Java application development. The following are some projects from the Spring framework, which have been used in developing the framework.

Spring Security 3.1.3: Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Data MongoDB 1.2.0: Spring Data for MongoDB is part of the umbrella Spring Data project which aims to provide a familiar and consistent Spring-based programming model for new datastores while retaining store-specific features and capabilities. The Spring Data MongoDB project provides integration with the MongoDB document database.

Spring AMQP 1.1.3: The Spring AMQP project applies core Spring concepts to the development of AMQP-based messaging solutions. It provides a template as a high-level abstraction for sending and receiving messages.

MongoDB 2.4.1: As described in 2.3.2, MongoDB is a free and open-source document-oriented database which is completely schema-free and manages JSON-style documents.

Elasticsearch 0.20.6: Elasticsearch is an open-source, distributed, RESTful, search engine built on top of Apache Lucene. Its data model roots lie with schema-free and document-oriented databases, and as shown by the NoSQL movement, this model proves very effective for building applications.

RabbitMQ 3.0.4: RabbitMQ is an open-source message broker, which implements the AMQP standard. It provides robust messaging services for applications and is reliable and highly scalable.

NginX 1.2.7: Nginx- engine-x pronounced - is a free, open-source, high-performance HTTP server. Unlike traditional servers, Nginx does not rely on threads to handle requests. Instead, it uses more scalable event-driven (asynchronous) architecture. This architecture uses small, but more importantly, predictable amounts of memory under load.

FFmpeg 0.9.2: FFmpeg is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play most things that humans and machines have created. It is an open source project licensed under LGPL version 2.1.

Tomcat 7.0.37: Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies. It powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations.

5.2 Framework Components

This section describes how each component in the framework has been implemented and realized.

5.2.1 App Managment

The app management component is developed completely in Java and it is based on the Spring framework. Figure 5.1 shows the project structure of this component within the Eclipse IDE.

As shown in figure 5.1, the project consists of three java packages, namely *de.fhg.fokus.ngni.cccd.model*, *de.fhg.fokus.ngni.cccd.rest* and *de.fhg.fokus.ngni.cccd.services*, and also four configuration files, which are, the maven configuration file *pom.xml* 20, the *web.xml* 21 file, the logging configuration file *log4j.xml* 22 and the Spring configuration file *cccd-config.xml* 23.

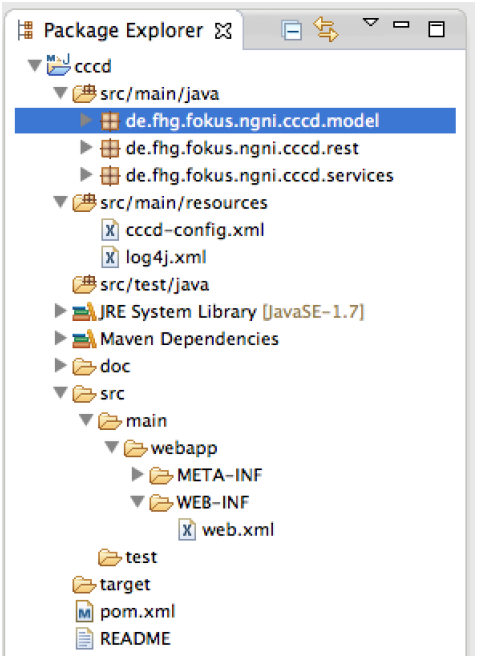

Figure 5.1: Eclipse Project Structure

The maven configuration file *pom.xml* contains information about the project and configuration details used by Maven to build the project and also manages the dependencies which are needed for building the project.

Within the *web.xml* file, one can set how to map a specific URL to a particular servlet and Spring provides this in the form as in the listing 8.

```
<web-app>
  <servlet>
    <servlet-name>cccd</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>
        contextConfigLocation
      </param-name>
      <param-value></param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

```
  <servlet-mapping>
    <servlet-name>cccd</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Listing 8: Excerpt from the web.xml configuration file

The above example at the servlet-mapping tag says that all URL("/") which are requested should be handled by the *cccd* servlet. The servlet tag tells Tomcat which Java class the *cccd* servlet should be resolved to. In other applications, one could just directly specify a class which inherits from the HttpServlet class. However, in Spring, the servlet tag actually enters the Spring Framework. Instead of defining the class, which needs to be executed for this servlet directly, one needs to specify only the class *org.springframework.web.servlet.DispatcherServlet*. From this point onwards, the requests and responses are known by the Spring framework, so that one can apply Spring pre-processing and post-processing with special Spring modules, i.e. Security, Aspect Oriented Programming and so on.

The Spring configuration file *cccd-config.xml* contains all of the Spring Web MVC-specific components (beans). In order sor the Spring framework to detect all *@Controller* beans, which provide all REST interfaces, one needs to set the tag *context:component-scan* as follows:

```
<context:component-scan base-package="de.fhg.fokus.ngni.cccd.rest" />
```

User Management: The Spring Security project is used for securing all REST interfaces. Listing 9 is an excerpt from the configuration file *cccd-config.xml* and it shows how to secure specific URI's with a specific role.

```
<sec:http create-session="stateless">
   <sec:intercept-url pattern="/app/**" access="ROLE_USER" />
  <sec:intercept-url pattern="/users/**" access="ROLE_ADMIN" />
  <sec:http-basic />
</sec:http>
```

Listing 9: Excerpt from the security part of the cccd-config.xml configuration file

The parameter *create-session="stateless"* in the listing 9 forces re-authentication with each request, in order to make it easier for load balancer as then the request can go to any server machines, making the location of the server machines completely transparent and this leads to better scalability. The authentication mechanism is set through the tag *sec:http-basic* in the listing 9 and it enables the HTTP

Basic Auth, which should be used over HTTPS of course, as it does not require a particular complex negotiation.

By default in Spring, the user information is added to the application context file. However, in order to provide a more scalable source of user information, another authentication provider is implemented. This authentication provider is implemented in the class *de.fhg.fokus.ngni.cccd.services.CustomUserDetailsService*, which implements the class *org.springframework.security.core.userdetails.UserDetailsService*. In order to configure the framework to use the implemented authentication provider, the listing 10 shows how it should be set within the configuration file *cccd-config.xml*.

```
<bean id="customUserDetailsService"
  class="de.fhg.fokus.ngni.cccd.services.CustomUserDetailsService" />

<bean id="saltSource"
  class="org.springframework.security.authentication.dao.ReflectionSaltSource">
  <property name="userPropertyToUse" value="username"/>
</bean>

<bean id="passwordEncoder"
  class="org.springframework.security.authentication.encoding.Md5PasswordEncoder" />

<sec:authentication-manager  alias="authenticationManager"
  erase-credentials="false">
  <sec:authentication-provider
    user-service-ref="customUserDetailsService">
     <sec:password-encoder ref="passwordEncoder">
       <sec:salt-source ref="saltSource" />
     </sec:password-encoder>
   </sec:authentication-provider>
 </sec:authentication-manager>
```

Listing 10: Configuring the authentication-provider within the cccd-config.xml configuration file

The class *CustomUserDetailsService* saves the user information in the repository. Thus it does not save the password of the user as it is in a plain format within the repository, rather it merges the password first with a salt, which is the username here, see the tag *sec:password-encoder ref="passwordEncoder"* in the listing 10, and then encodes it with an MD5 algorithm, which is provided through the class *org.springframework.security.authentication.encoding.Md5PasswordEncoder*.

The following excerpt from the configuration file *cccd-config.xml* shows where to save the user information exactly within the repository.

```
<bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
  <constructor-arg ref="mongoDb" />
```

```
  <constructor-arg name="databaseName" value="users"/>
</bean>
```

REST Interfaces: All REST interfaces extend the class *BaseCtrl.java*, which contains the Java driver instances for some components used in the REST interfaces, such as the repository, the search engine and the message broker, see the listing 11, which is an excerpt from the class *BaseCtrl.java*. The *@Autowired* annotation will tell Spring to search within the configuration file *cccd-config.xml* for a Spring bean which implements the required interface and place it automatically into the specified object, see the listing 12. The next subsections will show how those Java driver instances are configured.

```
/* MongoDB Java Driver */
@Autowired
protected Mongo mongoDb;

/* Elasticsearch Java Driver */
@Autowired
Client esClient;

/* RabbitMQ Java Driver */
@Autowired
protected RabbitTemplate amqpTemplate;
```

Listing 11: Java driver Instances within BaseCtrl.java

```
<bean id="baseCtrl" class="de.fhg.fokus.ngni.cccd.rest.BaseCtrl">
     <property name="mongoDb" ref="mongoDb"/>
     <property name="esClient" ref="esClient"/>
     <property name="debugResponse" ref="debugResponse"/>
     <property name="customUserDetailsService" ref="customUserDetailsService"/>
     <property name="amqpTemplate" ref="amqpTemplate"/>
</bean>
```

Listing 12: Properties Injection of Class BaseCtrl.java

The class *BaseCtrl.java* also contains two methods, namely *canRead* and *canWrite*. These methods provide the second level of the user management described in the subsection 4.2.1.

5.2.2 Repository/Media Store

The open source solution MongoDB is used as a repository within the framework. The specification GridFS, which is implemented in almost each MongoDB driver,

is for storing and retrieving files within MongoDB. The GridFS is used as a media store for the framework.

The listing 13 shows how to configure the framework for using MongoDB. Thereby a replication set of three MongoDB nodes are configured.

```
<bean id="mongoDb" class="com.mongodb.Mongo">
    <constructor-arg index="0">
        <list value-type="com.mongodb.ServerAddress">
            <bean class="com.mongodb.ServerAddress">
                <constructor-arg index="0" value="10.0.0.73"/>
                <constructor-arg index="1" value="27017"/>
            </bean>
            <bean class="com.mongodb.ServerAddress">
                <constructor-arg index="0" value="10.0.0.74"/>
                <constructor-arg index="1" value="27017"/>
            </bean>
            <bean class="com.mongodb.ServerAddress">
                <constructor-arg index="0" value="10.0.0.75"/>
                <constructor-arg index="1" value="27017"/>
            </bean>
        </list>
    </constructor-arg>
</bean>
```

Listing 13: Configuring the Java driver of MongoDB

By default, the read and write operations are done in the primary node. However, in order to enable read operations from secondary nodes, within the method *setMongoDb* in Java class *BaseCtrl.java* the following method is called.

```
mongoDb.setReadPreference(ReadPreference.secondaryPreferred());
```

The installation and configuration of MongoDB on Ubuntu is described in section 6.1.2.

5.2.3 Search Engine

The Elasticsearch project is used as an open source solution for the search engine. Listing 14 shows how to configure the Java driver in the configuration file *cccd-config.xml*.

```
<bean id="esClient" class="org.elasticsearch.client.transport.TransportClient" />

<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
  <property name="targetObject"><ref local="esClient"/></property>
```

```
    <property name="targetMethod"><value>addTransportAddresses</value></property>
    <property name="arguments">
      <list value-type="org.elasticsearch.common.transport.InetSocketTransportAddress">
        <bean class="org.elasticsearch.common.transport.InetSocketTransportAddress">
          <constructor-arg index="0" value="10.0.0.186"/>
          <constructor-arg index="1" value="9300"/>
        </bean>
       </list>
    </property>
  </bean>
```

Listing 14: Configuring the Java driver of Elasticsearch

The installation and configuration of Elasticsearch on Ubuntu is described in section 6.1.3.

5.2.4 Message Broker

The open source project RabittMQ is used as a message broker. Listing 15 shows how to configure the framework for using RabittMQ as a message broker. The default username/password for RabbitMQ is *guest*/*guest*.

```
  <bean id="connectionFactory" class="org.springframework.amqp.rabbit.connection.CachingConnectionFa
      <constructor-arg value="10.0.0.140" />
    <property name="username" value="guest" />
      <property name="password" value="guest"/>
  </bean>

  <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"/>
```

Listing 15: Configuring the Java driver of RabbitMQ

The installation and configuration of RabittMQ on Ubuntu is described in section 6.1.4.

5.2.5 Content Adaptation

As shown in figure 4.2 in the design chapter, the content adaptation component consists internally of two components, namely the media encoder and the media segmenter.

For the media encoder, the open source project FFmpeg is used.

As a media segmenter, this open source media segmenter/distributor [5] is used. This project makes it easier to set up a live streaming server using Apple's HTTP streaming protocol. The project includes ruby scripts and a C program that use

FFMpeg to encode and segment an input video stream in the correct format for use with the HTTP streaming protocol.

In order to consume messages from the message broker and process them, the python script *cccdCA.py* is implemented. This script is started automatically with Linux and listens for new messages from the message broker.

After receiving a message from the message broker, which contains the *ObjectId* of the video file,the app name *appName* and the bucket name *bucketName* the python script downloads the video file which needs to be processed from the GridFS media store to the */tmp/* directory. The file is then passed to the media segmenter/distributor along with the directory name within the content distribution component where to upload the output files to. This directory name is basically built from *appName/bucketName/ObjectId*. The segmenter/distributor uploads the output files via FTP, which need to be already configured on the content distribution component. If the message is successfully processed, the python script updates the metadata of the video file in the repository and adds a new tag which points to the index file of the transcoded/scaled video.

The installation and configuration of this component on Ubuntu is described in section 6.1.5.

5.2.6 Content Distribution

The content distribution component consists internally of two components, which are a python script called *cccdCD.py* and the open source project NginX as a HTTP server.

In order to secure the video file link within the content distribution component, the *HttpSecureLinkModule* from NginX is used. This module is not compiled by default and must be enabled while compiling the source code of NginX.

In order to consume messages from the message broker and process them, the python script *cccdCD.py* is implemented. This script is started automatically with Linux and listens for new messages from either a queue called *AppEvents* or a queue called *DocDelete* within the message broker.

Listing 16 shows a method from the script *cccdCD.py* which process a message coming from the queue *AppEvents*. The message can either be a notification of a newly created app(*status==created*) or a notification of a deleted app(*status==deleted*). In the case of the creation of a new app, the python script creates a configuration file for the NginX server and also creates a directory with the same name as the *appName* within the WWW directory of the NginX server. In the case of a deletion of an app, the python script deletes the app directory along with all of its contents.

```
jsonBody = json.loads(body)
if(jsonBody['status']=="created"):
  file = open(locations_dir+jsonBody['appName']+'.conf', 'w+')
  file.write(str(location_tmpl.safe_substitute(dict(app=jsonBody['appName']
    ,secret=jsonBody['secret']))))
  file.close()
  if not os.path.exists(www_dir+jsonBody['appName']):
    os.makedirs(www_dir+jsonBody['appName'])
    #reload the nginx server
    os.system("%s %s %s"%(nginx_bin, '-s', 'reload'))
  if(jsonBody['status']=="deleted"):
    os.remove(locations_dir+jsonBody['appName']+'.conf')
    shutil.rmtree(www_dir+jsonBody['appName'],
      ignore_errors=False, onerror=handleRemoveReadonly)
    #reload the nginx server
    os.system("%s %s %s"%(nginx_bin, '-s', 'reload'))
```

Listing 16: cccdCD.py python script

Another message notification will be sent to the queue *DocDelete* by deleting a document from an app so that its related contents within the content distribution component will be deleted.

Listing 17 shows an example of a created configuration file for a new app called *app1* with *pass123* as a secret word. From now on, each request to the URI */app1/* must have the MD5 hash as a parmeter, which is built from the secret word, the URI(the path of the requested file) and the expiration time. The documentation site of NginX [20] shows how this MD5 hash can be calculated.

```
location /app1/ {
        secure_link $arg_st,$arg_e;
        secure_link_md5 pass123$uri$arg_e;
        if ($secure_link = "") {
                return 403;
        }
        if ($secure_link = "0") {
                return 403;
        }
}
```

Listing 17: Sample configuration file for locations within NginX

6 Evaluation

The evaluation of the implemented prototype of the framework will be explained in this chapter. Thereby section 6.1 provides an overview of the test environment along with a detail information on how to install and configure each component and section 6.2 evaluates the framework by means of two test scenarios which are a usability and performance test.

6.1 Test Environment

Normally, in the case of a concrete application there is no need to show in detail how to install and configure the application. However the result of this thesis is a framework which consists of many components. In order to make it easier for developers who need to deploy this framework, this section provides detailed information on how to install and configure each component. Each component will be installed on a Linux machine running in the Amazon Web Services (AWS) platform. Each of these machines has 7 GB RAM and 20 Elastic Compute Cloud (EC2) Compute Units (8 virtual cores with 2.5 EC2 Compute Units each) and running Ubuntu 12.04. The name of this machine within AWS platform is *High-CPU Extra Large Instance/c1.xlarge*. In order to have these machines in the same private network, these machines are created within the Amazon Virtual Private Clouds (VPC).

Amazon VPC enables provisioning of a logically isolated section of the AWS Cloud where one can launch AWS resources in a virtual network. One can have complete control over the virtual networking environment, including the selection of the IP address range, creation of subnets, and configuration of route tables.

Table 6.1 shows the IP address of each component. Thereby, only the components, the app management and the content distribution have a private as well as a public IP address because these components as seen in figure 4.1 are the only components that the developer can interact with.

The source code along with the documentation of the entire framework can be obtained from the following GitHub link: `https://github.com/abdul7383/cccd`.

Name	IP address
App Management	**public:107.23.121.185 private:10.0.0.91**
Repository	**10.0.0.136**
Search Engine	**10.0.0.249**
Message Broker	**10.0.0.235**
Content Adaptation	**10.0.0.138**
Content Distribution	**public:107.23.180.173 private:10.0.0.139**

Table 6.1: IP addresses of test machines

6.1.1 App Management

The source code of the app management component can be obtained with the following command:

```
cd
git clone git://github.com/abdul7383/cccd.git
```

In order to deploy the app management component, one needs firstly to install and configure the following software:

- **Java Development Kit (JDK) 7:** Installing JDK 7 on Ubuntu is just easy as typing the following command.

  ```
  sudo apt-get install openjdk-7-jdk
  ```

- **Maven 3:** After downloading and extracting Maven, one needs to add the *bin* directory of Maven to the Linux PATH variable. Furthermore one needs to add the following tag within the *servers* tag to the setting file of Maven *apache-maven-3.0.5/conf/settings.xml*.

  ```
  <server>
    <id>tomcat7</id>
    <username>tomcat</username>
    <password>tomcat</password>
  </server>
  ```

- **Tomcat 7:** After downloading and extracting the Apache Tomcat, one needs to modify the file *apache-tomcat-7.0.37/conf/tomcat-users.xml* as follows in order to enable Maven to directly deploy the project into it.

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager-script"/>
  <user username="tomcat" password="tomcat" roles="manager-script"/>
</tomcat-users>
```

To start the Tomcat server from any location in Linux, one also needs to add the *bin* directory to the Linux PATH. The following example updates the PATH variable for Maven and Tomcat assuming that the home directory of the user is *ubuntu* and both Maven and Tomcat have been extracted to the home directory:

```
export PATH=/home/ubuntu/apache-maven-3.0.5/bin:/home/ubuntu/apache-tomcat-7.0.37/bin:$PATH
```

This sets the PATH variable only for the current Linux session, however, in order to have this configuration valid for each new Linux sessions, one needs to add the export line above at the end of the file */home/ubuntu/.bashrc* .

Furthermore, the SSL support of Tomcat needs to be enabled and this is shown in detail in the official documentation of Apache Tomcat here [7].

After having installed and configured everything as described above, the Tomcat server can be started with the following command:

```
catalina.sh start
```

All debug messages of the Tomcat server can be seen in the log file *apache-tomcat-7.0.37/logs/catalina.out*.

Before deploying the app management component, one needs firstly to install and start all other components and then to update the IP addresses of these components in the *cccd-config.xml* file of the project as described in 5.2.2, 5.2.3 and 5.2.4. The following command will then deploy the app management component to the Tomcat server:

```
cd cccd/cccdAppManagement/
mvn tomcat:deploy
```

It will take sometime for the first run, as Maven will download and compile all needed libraries and classes.

At this point, there is no user or administrator for the framework and in order to add the first administrator to the framework, one needs to run the following Maven command:

```
mvn exec:java -Dexec.mainClass="de.fhg.fokus.ngni.cccd.services.AddAdmin"
```

This program will ask for a username and password and then will add this user information in a new collection called *user* within a new database called *users*. After having at least one administrator then the RESTful framework user management interface */users* described in 4.2.1 can be used for adding new administrators or users.

6.1.2 Repository/Media Store

The installation of MongoDB on Ubuntu is described in the listing 18, see this reference for more details [9].

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart \
  dist 10gen" | sudo tee /etc/apt/sources.list.d/10gen.list
sudo apt-get update
sudo apt-get install mongodb-10gen
```

Listing 18: Installing MongoDB on Ubuntu

In order to provide a sufficient capacity for many distributed read operations, a replication set of three MongoDB instances needs to be deployed. This can be seen in the official documentation of MongoDB here [11].

6.1.3 Search Engine

After downloading and extracting the version 0.20.6 of Elasticsearch, it can be started using:

```
bin/elasticsearch
```

ElasticSearch is built using Java, and requires at least Java 6 in order to run. Therefore, one needs to install the JDK as described in subsection 6.1.1.

Under Unix system, the command will start the process in the background. To run it in the foreground, one need to add the -f switch to it:

```
bin/elasticsearch -f
```

The most important setting for the script is the -Xmx to control the maximum allowed memory for the process and -Xms to control the minimum allocated memory for the process. The following command starts Elasticsearch in the foreground and sets the max/min memory to 4GB and stores the index in the memory.

```
bin/elasticsearch -f  -Xmx4g -Xms4g -Des.index.storage.type=memory
```

Further configuration parameters can be seen in the setup documentation of Elasticsearch here [32].

6.1.4 Message Broker

The installation of the last release of RabbitMQ is shown in the following steps:

- One needs to add the following line to the file */etc/apt/sources.list*:

```
deb http://www.rabbitmq.com/debian/ testing main
```

- To avoid warnings about unsigned packages, add the public key of RabbitMQ to the trusted key list using apt-key:

```
wget http://www.rabbitmq.com/rabbitmq-signing-key-public.asc
sudo apt-key add rabbitmq-signing-key-public.asc
```

- Next install the package as usual, for instance:

```
sudo apt-get update
sudo apt-get install rabbitmq-server
```

To start or stop the server or check its status, etc., one can use the script rabbitmqctl as an administrator.

```
sudo rabbitmqctl status
sudo rabbitmqctl stop
sudo rabbitmqctl start
```

6.1.5 Content Adaptation

As described in 4.1, this component consists internally of the media encoder, the media segmenter and a python script called *cccdCA.py*. The following describes how to install and configure these components.

The following steps are for installing FFmpeg (media encoder) along with some essential libraries [4]. Some of the libraries are in the Ubuntu multiverse repository. Therfore, one needs to edit */etc/apt/sources.list* to include the multiverse repository as well.

- First of all, the following command will install some of the dependencies needed by FFmpeg

```
sudo apt-get update
sudo apt-get install autoconf automake build-essential checkinstall \
  git libass-dev libfaac-dev libgpac-dev libmp3lame-dev \
  libopencore-amrnb-dev libopencore-amrwb-dev librtmp-dev libspeex-dev \
  libtheora-dev libtool libvorbis-dev pkg-config texi2html zlib1g-dev
```

- **Yasm**: Yasm is an assembler and is recommended for x264 and FFmpeg

```
cd
wget http://www.tortall.net/projects/yasm/releases/yasm-1.2.0.tar.gz
tar xzvf yasm-1.2.0.tar.gz
cd yasm-1.2.0
./configure
make
sudo checkinstall --pkgname=yasm --pkgversion="1.2.0" \
  --backup=no --deldoc=yes --fstrans=no --default
```

- **x264**: H.264 video encoder. The following commands will get the current source files, compile, and install x264.

```
cd
git clone --depth 1 git://git.videolan.org/x264.git
cd x264
./configure --enable-shared
make
sudo checkinstall --pkgname=x264 \
  --pkgversion="3:$(./version.sh | awk -F'[" ]' '/POINT/{print $4"+git"$5}')" \
  --backup=no --deldoc=yes --fstrans=no --default
```

- **fdk-aac:** AAC audio encoder.

```
cd
git clone --depth 1 git://github.com/mstorsjo/fdk-aac.git
cd fdk-aac
autoreconf -fiv
./configure --disable-shared
make
sudo checkinstall --pkgname=fdk-aac --pkgversion="$(date +%Y%m%d%H%M)-git" \
 --backup=no --deldoc=yes --fstrans=no --default
```

- **libvpx:** VP8 video encoder and decoder.

```
cd
git clone --depth 1 http://git.chromium.org/webm/libvpx.git
cd libvpx
./configure --disable-examples --disable-unit-tests
make
sudo checkinstall --pkgname=libvpx --pkgversion="1:$(date +%Y%m%d%H%M)-git" \
  --backup=no --deldoc=yes --fstrans=no --default
```

- **FFmpeg:**

```
cd
wget http://ffmpeg.org/releases/ffmpeg-0.9.2.tar.bz2
tar -xvvf ffmpeg-0.9.2.tar.bz2
cd ffmpeg-0.9.2

./configure --enable-gpl --enable-libass --enable-libfaac \
  --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb \
  --enable-libspeex --enable-librtmp --enable-libtheora --enable-libvorbis \
  --enable-libvpx --enable-libx264 --enable-nonfree --enable-version3
make
sudo checkinstall --pkgname=ffmpeg --pkgversion="0.9.2" \
  --backup=no --deldoc=yes --fstrans=no --default
sudo ldconfig
```

The following steps describe how to compile the media segmenter/distributor.

- The source code can be obtained with following command.

```
cd
git clone git://github.com/abdul7383/cccd.git
```

- The following command will install some of the dependencies needed by the media segmenter/distributor and the python script *cccdCA.py*.

```
sudo apt-get install ruby rubygem python-pika python-pip python-dev build-essential
sudo pip install pymongo
```

- In order to compile the media segmenter/distributor, the following commands are needed.

```
cd ~/cccd/cccdContentAdaption/hls
make
```

The python script *cccdCA.py* and its configuration *cccdCA.conf* are located in the directory */cccd/cccdContentAdaption*. The most important configuration in *cccdCA.conf* is the FTP transfer profile. One needs to set the IP address and the user credentials of the FTP server, which is installed on the content distribution component 6.1.6. The default configuration contains 8 encoding profiles shown below. These encoding profiles are for devices with 16x9 or 4x3 screen resolution and they are in four different streams bitrate.

```
[ 'cell_16x9_150k', 'cell_16x9_240k', 'wifi_16x9_440k', 'wifi_16x9_640k' ,
   'cell_4x3_150k', 'cell_4x3_240k', 'wifi_4x3_440k', 'wifi_4x3_640']
```

Listing 19: List of supported encoding profiles

Finally, to start the content adaptation component, one needs simply to run the following commands.

```
cd ~/cccd/cccdContentAdaption
sudo ./cccdCA.py
```

Before running the python script, the parameter *message_broker_ip* needs to be set to the IP address of the message broker component.

6.1.6 Content Distribution

As described in 5.2.6, the content distribution component consists internally of two components, which are a python script called *cccdCD.py* and the open source project NginX as a HTTP server. The installation of the NginX server is described below.

```
cd
wget http://nginx.org/download/nginx-1.2.7.tar.gz
tar -xvvf nginx-1.2.7.tar.gz
cd nginx-1.2.7/
sudo apt-get update
sudo apt-get install gcc make
./configure --with-http_secure_link_module \
  --without-http_rewrite_module --without-http_gzip_module
make
sudo make install
```

The following command starts Nginx in the background.

```
sudo /usr/local/nginx/sbin/nginx
```

The following commands are for the installing and the running of the python script *cccdCD.py*.

```
cd
wget https://raw.github.com/abdul7383/cccd/master/cccdContentDistribution/cccdCDN.py
chmod +x cccdCDN.py
sudo apt-get install python-pika
sudo ./cccdCDN.py
```

Before running the python script, the parameter *message_broker_ip* needs to be set to the IP address of the message broker component.

6.2 Test Scenarios

In order to evaluate the framework, the next two subsections provide a usability test in which a deployment scenario of a specific application is described and a performance test in which the document REST interface */app/appName/collections/collectionName/doc* is tested.

6.2.1 Usability

As a usability test, the first scenario mentioned in section 3.1 will be deployed and configured in the framework. Thereby, this test will only show how the JSON requests and what its responses look like. It also shows the interaction between the components.

In order to send custom HTTP requests and show their responses, the Advance Rest Client described here 5.1.1 is used.

After creating the first administrator as described at the end of this subsection 6.1.1, the administrator creates a new user without administration rights with the following request.

```
POST https://admin:pass@107.23.121.185:8443/cccd/users?username=user1
  &password=pass1&email=user1@email.com&role=2
```

Response:

```
{
"ok": "1",
"debug": "user: user1 created"
}
```

From now on, the created user *user1* will be used for all following requests.

The following request is for creating a new application named *vod*. Within the payload will be contained, a secret word *secret1* and a list of video encoding profiles, so that each uploaded video will be available in these profiles. Listing 19 lists all encoding profiles.

```
POST https://user1:pass1@107.23.121.185:8443/cccd/app/vod
```

Payload:

```
{
"secret":"secret1",
"profiles":["cell_16x9_150k","wifi_16x9_640k"],

}
```

Response:

```
{
"ok": "1",
"debug": "app: vod created"
}
```

Figure 6.1 shows how the request is processed. A new database within the repository is created. Thereby, a new table called *user* is created, which contains all the users and their rights, and also another table called *conf* which contains all configuration parameters that belongs to this app.

Furthermore, a notification message is sent to the message broker component to forward this message to the content distribution component. This message notifies the content distribution component about the newly created app, so that it creates a new directory for this new app within the HTTP server and secures it with the secure word provided by the user while creating the app.

For checking if the app *vod* was created successfully, the following request lists all apps that belongs to the user *user1*.

```
GET https://user1:pass1@107.23.121.185:8443/cccd/app/
```

Response:

```
{
"data": [
  "vod"
],
"ok": "1"
}
```

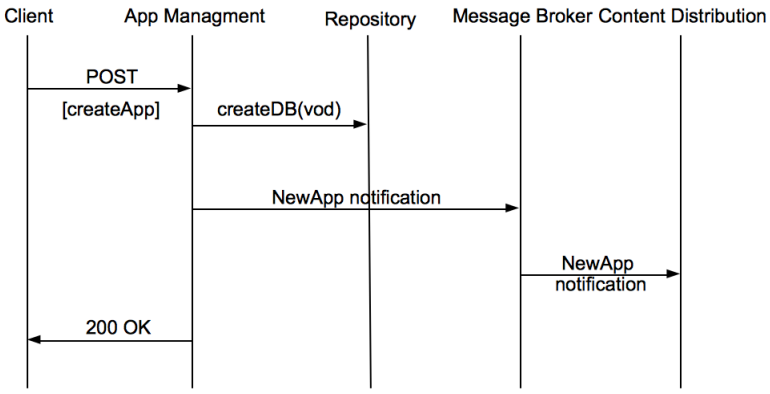

Figure 6.1: Creating a new app

In order to upload files, i.e. videos or images, to the newly created app, one needs firstly to create a bucket, which will hold these files, within this app. The following request creates a new bucket called *bucket1*.

```
POST https://user1:pass1@107.23.121.185:8443/cccd/app/vod/buckets/bucket1
```

Response:

```
{
"ok": "1",
"debug": "bucket: bucket1 created"
}
```

Listing all buckets within an app can be done with the following request.

```
GET https://user1:pass1@107.23.121.185:8443/cccd/app/vod/buckets
```

Response:

```
{
"data": [
  "bucket1"
],
"ok": "1"
}
```

Now the bucket *bucket1* is created. However, before uploading files to it, one needs to create a collection within the app *vod* to hold the context/metadata of the file. The following request creates a new collection called *videos*. Thereby, an empty index is automatically created in the search engine component. Figure 6.2 illustrates how the request is processed internally in the framework.

```
POST https://user1:pass1@107.23.121.185:8443/cccd/app/vod/collections/videos
```

Response:

```
{
"ok": "1",
"debug": "collection: videos and an empty index created"
}
```

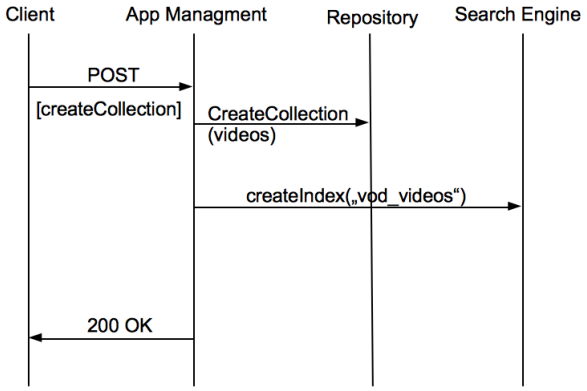

Figure 6.2: Creating a new collection

In order to configure the app *vod* to send a new message notification to a queue called *contetnt_adaptation* for each new inserted document to the collection *videos*, the following request is needed. Thereby, one can configure the app to send the message notification to more than one queue for supporting more services like M2M context enrichment or Subscribe/Notification.

```
POST https://user1:pass1@107.23.121.185:8443/cccd/app/vod
```

Payload:

```
{
"videos" : ["contetnt_adaptation"]
}
```

The following request lists all configurations for the app *vod*.

```
GET https://user1:pass1@107.23.121.185:8443/cccd/app/vod
```

Response:

```
{
"data": {
  "profiles": [
    "cell_16x9_150k",
    "wifi_16x9_640k"
  ],
  "secret": "secret1",
  "videos": [
    "contetnt_adaptation"
  ]
},
"ok": "1"
}
```

Before adding data to the collection *videos*, the data fields should be mapped to a proper object, i.e. string, integer, date, array or geo_point. One can also specify if a specific field will be indexed or stored in the search engine. At first, the mapping of the created collection is empty, which can be seen with the following request.

```
GET https://user1:pass1@107.23.121.185:8443/cccd/app/vod/collections/videos/mapping
```

Response:

```
{
"ok": "0",
"debug": "no mapping found"
}
```

As soon as some data is inserted into the collection, the mapping is automatically updated, however it can not be updated anymore, i.e. setting a specific filed to be indexed or stored in the search engine. Therefore, the mapping for each field should be set before adding any data to the collection, see the official documentation of the search engine Elasticsearch for further information on the mapping interface [31]. The following request updates the mapping of the collection *videos*.

```
PUT https://user1:pass1@107.23.121.185:8443/cccd/app/vod/collections/videos/mapping
```

Payload:

```
{
"videos" : {
```

```
  "properties" : {
    "user" : {"type" : "string", "store" : "yes", "index" : "not_analyzed"},
    "titel" : {"type" : "string", "store" : "yes"},
    "description" : {"type" : "string", "store" : "yes"},
    "tags" : {"type" : "string", "store" : "yes"},
    "uploadTime" : {"type" : "date", "store" : "yes"},
    "location" : {"type" : "geo_point", "store" : "yes"},
    "fileLink" : {
      "type" : "object",
      "properties" : {
        "app" : {"type" : "string", "store" : "yes", "index" : "not_analyzed"},
        "bucket" : {"type" : "string", "store" : "yes", "index" : "not_analyzed"},
        "objectid" : {"type" : "string", "store" : "yes", "index" : "not_analyzed"}
      }
    }
  }
}
}
```

Now listing the mapping of the collection *video* will show all fields with their respective types, see the following request.

```
GET https://user1:pass1@107.23.121.185:8443/cccd/app/vod/collections/videos/mapping
```

Response:

```
{
"data": {
  "properties": {
    "description": {
      "type": "string",
      "store": "yes"
    },
    "fileLink": {
      "properties": {
        "app": {
          "type": "string",
          "index": "not_analyzed",
          "store": "yes",
          "omit_norms": true,
          "index_options": "docs"
        },
        "bucket": {
          "type": "string",
          "index": "not_analyzed",
          "store": "yes",
          "omit_norms": true,
          "index_options": "docs"
```

```
        },
        "objectid": {
          "type": "string",
          "index": "not_analyzed",
          "store": "yes",
          "omit_norms": true,
          "index_options": "docs"
        }
      }
    },
    "location": {
      "type": "geo_point",
      "store": "yes"
    },
    "tags": {
      "type": "string",
      "store": "yes"
    },
    "titel": {
      "type": "string",
      "store": "yes"
    },
    "uploadTime": {
      "type": "date",
      "store": "yes",
      "format": "dateOptionalTime"
    },
    "user": {
      "type": "string",
      "index": "not_analyzed",
      "store": "yes",
      "omit_norms": true,
      "index_options": "docs"
    }
  }
},
"ok": "1"
}
```


Now the bucket, the collection and its mapping are all ready and one can now upload the videos and their related context. In order to have users who are developers or service providers that need to use the created app for only uploading, searching, downloading, or streaming videos, the following request adds a read-only user to the created app *vod*. Thereby, the user to be added to the app user list must already be a registered user in the framework (previously created through */cccd/users*).

```
PUT https://user1:pass1@107.23.121.185:8443/cccd/app/vod
```

```
  /users?username=user2&readonly=true
```

To check if the user has been added successfully to the user list of the app *vod*, the following request is needed.

```
GET https://user1:pass1@107.23.121.185:8443/cccd/app/vod/users
```

Response:


```
{
"data": [
  {
    "username": "user1",
    "readOnly": false
  },
  {
    "username": "user2",
    "readOnly": true
  }
],
"ok": "1"
}
```


Next, uploading a video to the new created app can be done in two steps. Firstly one needs to upload the video file to a bucket within the app. The bucket *bucket1*, which is created above, will be used in the following request. Thereby ,the *objectid* of the uploaded file is returned in the response, which will be then used for ensuring the correlation between the video file and its context/metadata.

```
POST https://user1:pass1@107.23.121.185:8443/cccd/app/vod
  /buckets/bucket1/files
```

Response:


```
{
"data": {
  "app": "vod",
  "bucket": "bucket1",
  "objectid": "515b2e2ae4b0da706f84c1a7"
},
"ok": "1"
}
```


The video file is now saved within the bucket *bucket1*. The second step is to upload the context/metadata of the video file along with the file-link within the app *vod*.

```
POST https://user1:pass1@107.23.121.185:8443/cccd/app/vod
  /collections/videos/doc
```

Payload:

```
{
  "title" : "Brandenburger Tor New year 2013 ",
  "description" : "Berlin Brandenburger Tor - New year 2013",
  "tags" : "Berlin,Brandenburger Tor,new year 2013,fireworks",
  "uploadTime" : "2013-01-15T14:12:12",
  "location" : "52.516275,13.377227",
  "user" : "abdul",
  "fileLink" : {
    "app" : "vod",
    "bucket" : "bucket1",
    "objectid" : "515b2e2ae4b0da706f84c1a7"
  }
}
```

These two steps are needed for uploading videos and in the same way two other videos have been uploaded, which have been recorded in Berlin Tempelhof and Frankfurt am Main Bahnhofsviertel.

```
{
  "title": "Happy New Year 2013 Berlin Tempelhof ",
  "location": "52.469635,13.385658",
  "description": "New Year 2013 Silvester in Berlin Tempelhof",
  "fileLink": {
    "app": "vod",
    "bucket": "bucket1",
    "objectid": "515b0a99e4b0262de3bb2c7d"
  },
  "uploadTime": "2013-01-02T14:12:12",
  "tags": "Berlin,Tempelhof, new year 2013,firework",
  "user": "tom"
}
{
  "title": "Frankfurt Germany Fireworks New Years 2013",
  "location": "50.108806,8.664432",
  "description": "From our balcony the fireworks for 2013",
  "fileLink": {
    "app": "vod",
    "bucket": "bucket1",
    "objectid": "515ad4afe4b08cc9e8560330"
  },
  "uploadTime": "2013-01-05T14:12:12",
  "tags": "frankfurt,,new year 2013,firework",
```

```
  "user": "frank"
}
```

After a short time, depending of how big the video files are, the framework will update the context/metadata of each uploaded video file with the links for each HLS encoded videos based on the encoding profiles which have been set while creating the app (*cell_16x9_150k,wifi_16x9_640k*).

The following request is for searching for all videos which have been captured within a maximum distance of 10 km from Potsdamer Platz(lat,lon: 52.509693,13.376218) in Berlin. Thereby, the response must be ordered by the distance.

```
POST https://user1:pass1@107.23.121.185:8443/cccd/app/vod
  /collections/videos/search
```

Payload:

```
{
    "sort" : [
        {
            "_geo_distance" : {
                "location" : "52.509693,13.376218",
                "order" : "asc",
                "unit" : "km"
            }
        }
    ],
    "query": {
        "filtered" : {
            "query" : {
                "match_all" : {}
            },
            "filter" : {
                "geo_distance" : {
                    "distance" : "10km",
                    "location" : "52.509693,13.376218"
                }
            }
        }
    }
}
```

Response:

```
{
"data": {
  "hits": {
```

```
    "total": 2,
    "max_score": null,
    "hits": [
      {
        "_index": "vod_videos",
        "_type": "videos",
        "_id": "515b4008e4b0da706f84c1a9",
        "_score": null,
        "_source": {
          "title": "Brandenburger Tor New year 2013 ",
          "location": "52.516275,13.377227",
          "description": "Berlin Brandenburger Tor - New year 2013",
          "fileLink": {
            "app": "vod",
            "bucket": "bucket1",
            "objectid": "515b2e2ae4b0da706f84c1a7"
          },
          "uploadTime": "2013-01-15T14:12:12",
          "tags": "Berlin,Brandenburger Tor, new year 2013,firework",
          "user": "abdul"
        },
        "sort": [
          0.7350631627891678
        ]
      },
      {
        "_index": "vod_videos",
        "_type": "videos",
        "_id": "515b429ce4b0da706f84c1aa",
        "_score": null,
        "_source": {
          "title": "Happy New Year 2013 Berlin Tempelhof ",
          "location": "52.469635,13.385658",
          "description": "New Year 2013 Silvester in Berlin Tempelhof",
          "fileLink": {
            "app": "vod",
            "bucket": "bucket1",
            "objectid": "515b0a99e4b0262de3bb2c7d"
          },
          "uploadTime": "2013-01-02T14:12:12",
          "tags": "Berlin,Tempelhof, new year 2013,firework",
          "user": "tom",
          "hls_videos": {
            "cell_16x9_150k":
              "http://107.23.180.173/vod/515b4008e4b0da706f84c1a9/cell_16x9_150k.m3u8",
            "wifi_16x9_640k":
              "http://107.23.180.173/vod/515b4008e4b0da706f84c1a9/wifi_16x9_640k.m3u8",
            "multi_rate_16x9":
              "http://107.23.180.173/vod/515b4008e4b0da706f84c1a9/multi_16x9.m3u8"
```

```
          }
        },
        "sort": [
          4.499869967538772
        ]
      }
    ]
  }
},
"ok": "1"
}
```


For each matched video in the response above, a *sort* tag that contains the distance of the matched video from the searched location is returned. Furthermore, the matched videos above contain a new tag *hls_videos* and they can be used to start streaming the video file directly on a mobile device, i.e. iPad, iPhone and etc. These video links are secured and can be only accessed through a security token and the generation of this token is shown here [20]. Thereby the secret word is the secret word provided while creating the app.

6.2.2 Performance

It is not easy to do a performance evaluation for the entire framework, therefore, only one REST interface is tested, which has interactions with more components. The interface to be tested is the document interface */app/appName/collections/collectionName/doc*. This interface has interactions with the repository, search engine and the message broker component.

The tool used for testing the interface is the open source tool Apache ab and for drawing the graphs the open source tool Gnuplot.

The tests are done in two scenarios, the first one with only one instance of the app management component and the second one with two instances of the app management and also an AWS load balancer to distribute the incoming traffic across both app management instances. Only the operations CREATE(POST) and READ(GET) are tested.

GET The following Apache ab command is used for testing the interface, where the -n option is the number of requests to perform and the -c option is the number of multiple requests to make(concurrency level). The -n option is set always to 10000 requests in the following tests and the -c option is changed in every test.

```
ab -A user1:pass1 -n 10000 -c 50 -g doc_50.dat
 http://10.0.0.91:8080/cccd/app/vod/collections/videos/doc/515b4008e4b0da706f84c1a9
```

- **Without load balancer:** Figure 6.3 shows three tests with the -c option set to 50, 70 and 100. Thereby, only the test with the concurrency level set to 50 was successful without any failed requests. By setting the -c option to 70 there were a total of 4996 failed requests and by setting it to 100 a total of 7722 failed requests.

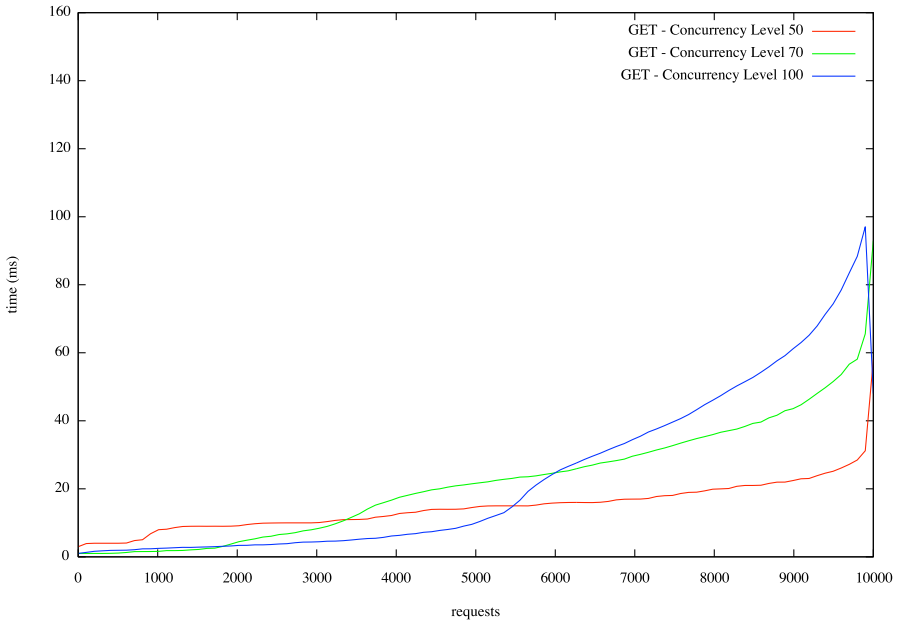

Figure 6.3: GET tests without load balancer

- **With load balancer:** Figure 6.6 shows four tests with the -c option set to 200, 250, 300 and 400. The expected result was that both instances were capable of serving double traffic as the test above(-c 50) with one instance, without a failed request, with the concurrency level set to 100. However, the two instances with the load balancer were unexpectedly capable of serving up to a concurrency level of 400 without noticeable failed requests(less than 100 requests).

POST The following Apache ab command is used for testing the CREATE operation. The payload of the POST request is saved in the file called *post.json*.

```
ab -A user1:pass1 -T "application/json" -p post.json -n 10000 -c 50 -g doc_50.dat
 http://10.0.0.91:8080/cccd/app/vod/collections/videos/doc
```

The file *post.json* contains:

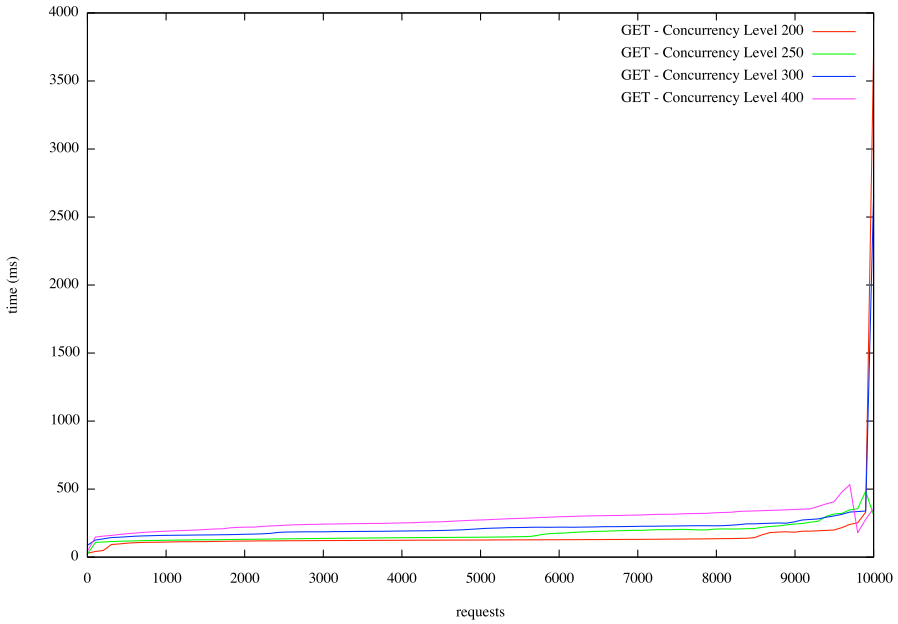

Figure 6.4: GET tests with load balancer

```
{
  "title": "Happy New Year 2013 Berlin Tempelhof ",
  "location": "52.469635,13.385658",
  "description": "New Year 2013 Silvester in Berlin Tempelhof",
  "fileLink": {
    "app": "vod",
    "bucket": "bucket1",
    "objectid": "515b0a99e4b0262de3bb2c7d"
  },
  "uploadTime": "2013-01-02T14:12:12",
  "tags": "Berlin,Tempelhof, new year 2013,firework",
  "user": "tom"
}
```

- **Without load balancer:** Figure 6.5 shows three tests with the -c option set to 50, 70 and 100. Thereby only the test with the concurrency level set to 50 was successful without any failed requests. By setting the -c option to 70 there were a total of 5167 failed requests and by setting it to 100 a total of 9040 failed requests.

- **With load balancer:** Figure 6.6 shows three tests with the -c option set to 200, 250 and 300. The failed requests within the concurrency level 200

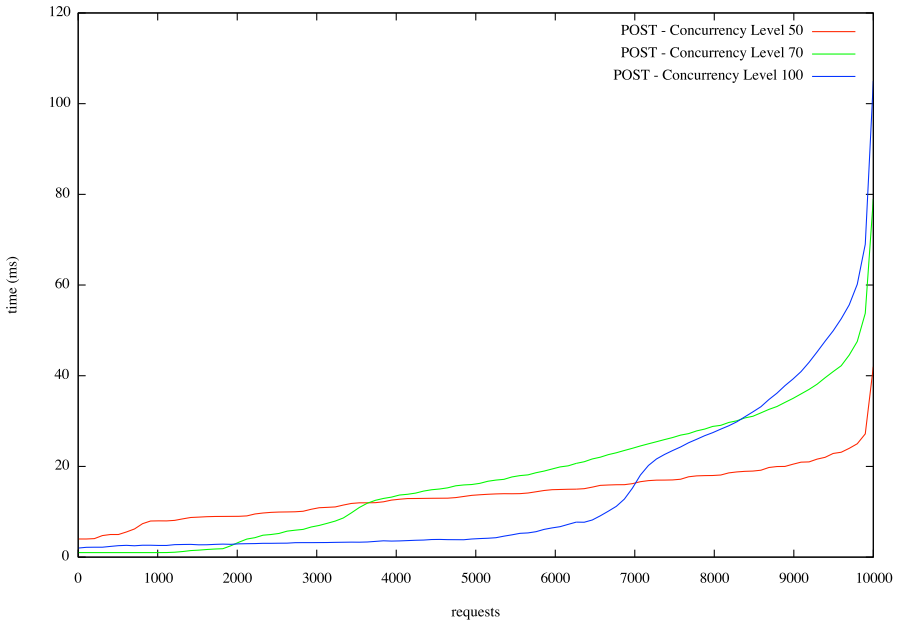

Figure 6.5: POST tests without load balancer

and 250 were less than 100 requests and within the concurrency level 300 the failed requests were 213.

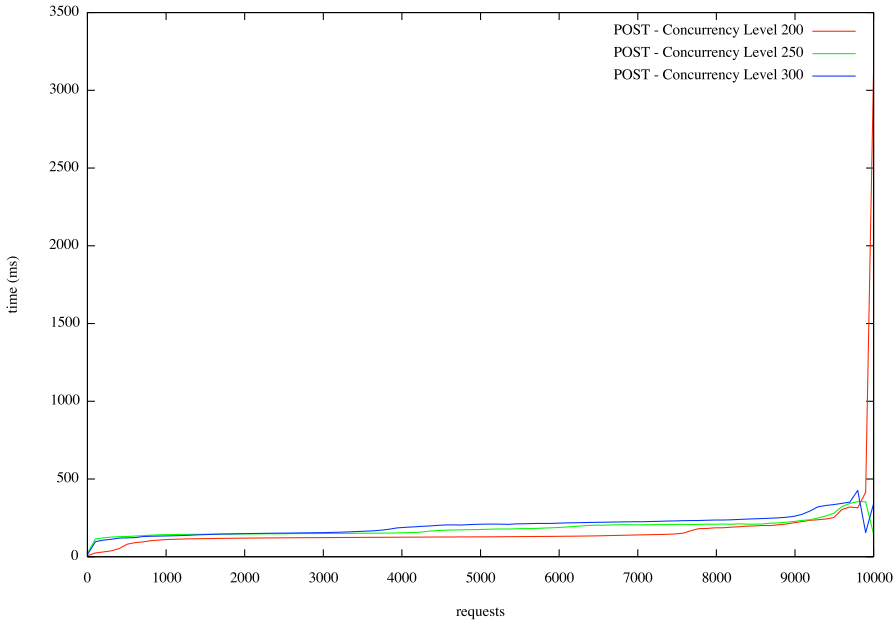

Figure 6.6: POST tests with load balancer

7 Conclusion

The objective of this thesis is to develop a contextual content management framework that supports the correlation and synchronization of context information and multimedia content and also provides an efficient and scaleable discovery and distribution mechanism. The framework facilitates the development of applications that need transcoding of contents, an easy scaled and distributed repository and also a powerful, ready for the cloud search engine in order to discover the data.

In order to develop the desired framework, the general background of the different related technologies that were or might have been involved in this thesis was discussed in chapter 2. In the subsequent chapter 3, the requirements for the framework to be developed, were defined.

The framework prototype was implemented in chapter 5 for the purpose of evaluating the design architecture presented in chapter 4. The design is characterized mainly by its high scalability. Thereby each component can be scaled independently, i.e. one can have two or more instances of the app management component that can be managed by a load balancer for distributing the incoming traffics across these instances, or one can have more instances of the repository, the search engine or the content distribution component for providing applications that are highly available and protected against single-instance failure.

Chapter 6 provided two test scenarios, a usability and performance test, so that the implemented framework prototype could be evaluated.

One of the task of this thesis was to implement an application on a mobile device that evaluates the implemented framework, however, this task could not be done due to lack of time.

7.1 Problems encountered

The development of frameworks is not trivial. Compared to concrete conventional applications, frameworks are much more complex. The objective of developing a framework is to identify abstractions with a bottom-up approach, meaning that firstly one needs to provide some concrete applications, which will be used as use cases. These use cases can be used to illustrate better the functional assets of the framework and facilitate a deduction of its requirements.

Another problem faced in this thesis, was that the original approach was to use

a ready open source content management system and try to build the framework upon it. This task was not at all easy, as most of these systems have a fixed-schema data model. In order to provide a generic framework that enables the user to easily update or extend the data model, there was no other option apart from starting to develop a new framework from ground level, based on schema-less databases.

7.2 Outlook

It was not the purpose of this thesis to implement a market-ready product. Nevertheless, most of the components within the prototype implementation of this framework are based on technologies, i.e. Spring framework, MongoDB, Elasticsearch or Nginx, that are widely used in many production environments. The component which needs to be improved or even completely replaced is the content adaptation. This component was developed only to provide a proof of concept of a one use case scenario which needs a content transcoding service. The problem with this component is that it runs FFmpeg as a terminal command and it is not easy to determine if the command has been successfully processed. A better solution would be to use the C API of the FFmpeg platform for transcoding and segmenting videos.

Furthermore, the REST interfaces for the repository and the search engine should be extended to support their wide features.

In this thesis, Ontologies or RDF was not used intentionally so that the solution would not be unnecessarily more complex. However, in works that will be based on the results of this thesis in the future, it would be interesting to examine how the use of Ontologies or RDF could enhance the efficiency of the solution.

List of Acronyms

Bibliography

[1] Web services architecture. URL http://www.w3.org/TR/ws-arch/.

[2] Amqp concepts, 2013. URL http://www.rabbitmq.com/tutorials/amqp-concepts.html. [Online; accessed 15.03.2013].

[3] Introduction to bson, 2013. URL http://bsonspec.org/. [Online; accessed 13.03.2013].

[4] Compile ffmpeg on ubuntu, 2013. URL https://ffmpeg.org/trac/ffmpeg/wiki/UbuntuCompilationGuide. [Online; accessed 20.03.2013].

[5] Http live video stream segmenter and distributor, 2013. URL https://github.com/carsonmcdonald/HTTP-Live-Video-Stream-Segmenter-and-Distributor. [Online; accessed 18.01.2013].

[6] Spring framework reference, 2013. URL http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/pdf/spring-framework-reference.pdf. [Online; accessed 18.01.2013].

[7] Apache tomcat ssl configuration how-to, 2013. URL http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html. [Online; accessed 20.03.2013].

[8] Welcome to solr, 2013. URL https://svn.apache.org/repos/asf/lucene/dev/branches/lucene3622/solr/site/index.pdf. [Online; accessed 13.01.2013].

[9] Inc. 10gen. Install mongodb on ubuntu, 2013. URL http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/. [Online; accessed 12.03.2013].

[10] Inc. 10gen. Introduction to mongodb, 2013. URL http://www.mongodb.org/about/introduction/. [Online; accessed 13.03.2013].

[11] Inc. 10gen. Deploy a replica set, 2013. URL http://docs.mongodb.org/manual/tutorial/deploy-replica-set/. [Online; accessed 15.03.2013].

[12] Inc. 10gen. Mongodb replication, 2013. URL `http://wiki.10gen.com/pages/viewpage.action?pageId=21270635`. [Online; accessed 13.03.2013].

[13] J. Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB - The Definitive Guide: Time to Relax.* O'Reilly, 2010. ISBN 978-0-596-15589-6.

[14] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, jun 2007. ISSN 1743-8225. doi: 10.1504/IJAHUC.2007.014070.

[15] Inc. Basho Technologies. Riak buckets, 2013. URL `http://docs.basho.com/riak/latest/references/appendices/concepts/Buckets/`. [Online; accessed 07.02.2013].

[16] M. Biba and F. Xhafa. *Learning Structure and Schemas from Documents.* Studies in Computational Intelligence. Springer, 2011. ISBN 9783642229121. URL `http://books.google.de/books?id=7Zh0oFDFBvEC`.

[17] P.J. Brown, J.D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *Personal Communications, IEEE*, 4(5):58–64, oct 1997. ISSN 1070-9916. doi: 10.1109/98.626984.

[18] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1267308.1267323`.

[19] Cisco. Cisco visual networking index: Forecast and methodology, 2010-2015. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[20] Nginx Community. Httpsecurelinkmodule, 2013. URL `http://wiki.nginx.org/HttpSecureLinkModule`. [Online; accessed 12.03.2013].

[21] The Nielsen Company. State of the media: The mobile media report. http://www.nielsen.com/content/dam/corporate/us/en/reports-downloads/2011-Reports/state-of-mobile-Q3-2011.pdf.

[22] Oracle Corporation. Web services and the sun one developer platform, 2013. URL `http://dsc.sun.com/appserver/reference/techart/webservice.html`. [Online; accessed 12.03.2013].

[23] Anind K. Dey. Context-aware computing: The cyberdesk project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, 1998. AAAI Press. URL `http://www.cc.gatech.edu/fce/cyberdesk/pubs/AAAI98/AAAI98.html`.

[24] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness (CHI 2000)*, The Hague, The Netherlands, April 2000. URL `http://www.cc.gatech.edu/fce/contexttoolkit/`.

[25] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[26] Apache Software Foundation. Eventual consistency, 2013. URL `http://guide.couchdb.org/draft/consistency.html`. [Online; accessed 13.03.2013].

[27] EC FIArch Group. Fundamental limitations of current internet and the path to future internet. http://ec.europa.eu/information_society/activities/foi/docs/current_internet_limitations_v9.pdf.

[28] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993. URL `http://tomgruber.org/writing/ontolingua-kaj-1993.pdf`.

[29] Michael Havey. *Essential business process modeling.* O'Reilly, 2005. ISBN 978-0-596-00843-7.

[30] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *Proceedings of the 2011 International Conference on Cloud and Service Computing*, CSC '11, pages 336–341, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4577-1635-5. doi: 10.1109/CSC.2011.6138544. URL `http://dx.doi.org/10.1109/CSC.2011.6138544`.

[31] Elasticsearch Inc. Elasticsearch mapping, 2013. URL `http://www.elasticsearch.org/guide/reference/mapping/`. [Online; accessed 20.01.2013].

[32] Elasticsearch Inc. Installation of elasticsearch, 2013. URL `http://www.elasticsearch.org/guide/reference/setup/installation.html`. [Online; accessed 20.03.2013].

[33] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010. ISSN 0163-5980. doi: 10.1145/1773912.1773922. URL `http://doi.acm.org/10.1145/1773912.1773922`.

[34] John Mccarthy. Notes on formalizing context. In *IJCAI*, pages 555–562, San Mateo, California, 1993. Morgan Kaumann. URL `http://dblp.uni-trier.de/rec/bibtex/conf/ijcai/McCarthy93`.

[35] pSkills consortium. Object-oriented modeling, 2012. URL `http://pskills.ced.tuc.gr/Restricted/WP3/Scripts_for_training_material_development/17._Modelling_and_Data_Management/02._Object-oriented_modelling`. [Online; accessed 01.02.2013].

[36] Oriana Riva. A conceptual model for structuring context-aware applications. In *Fourth Berkeley-Helsinki Ph.D. Student Workshop on Telecommunication Software Architectures*, 2004.

[37] N Ryan. Mobile computing in a fieldwork environment: Metadata elements. Project working document, version 0.2, 19997.

[38] N. Ryan, J. Pascoe, and D. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications and Quantitative Methods in Archaeology (CAA 97), Oxford*, 1997. URL `http://www.cs.ukc.ac.uk/research/infosys/mobicomp/Fieldwork/Papers/CAA97/ERFldwk.html`.

[39] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 85 –90, dec 1994. doi: 10.1109/MCSA.1994.512740.

[40] B.N. Schilit and M.M. Theimer. Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22 –32, sept.-oct. 1994. ISSN 0890-8044. doi: 10.1109/65.313011.

[41] Quan Z. Sheng and Boualem Benatallah. Contextuml: A uml-based modeling language for model-driven development of context-aware web services development. In *Proceedings of the International Conference on Mobile Business*, pages 206–212, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2367-6. doi: 10.1109/ICMB.2005.33. URL `http://portal.acm.org/citation.cfm?id=1084013.1084215`.

[42] Thomas Strang and Claudia L. Popien. A context modeling survey. In *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, Nottingham, September 2004. URL `http://citeseer.ist.psu.edu/712164.html`.

[43] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992. URL `http://dblp.uni-trier.de/db/journals/tois/tois10.html#WantHFG92`.

[44] Youtube. press statistics. http://www.youtube.com/t/press_statistics.

[45] Pinar Öztürk and Agnar Aamodt. Towards a model of context for case-based diagnostic problem solving. In *IN CONTEXT-97; PROCEEDINGS OF THE*, pages 198–208, 1997.

Annex

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
              http://maven.apache.org/maven-v4_0_0.xsd">
  <artifactId>cccd</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <inceptionYear>2012</inceptionYear>
  <packaging>war</packaging>
  <groupId>de.fhg.fokus.ngni</groupId>
  <version>1.0</version>
  <properties>
    <releaseCandidate>1</releaseCandidate>
    <spring.version>3.1.3.RELEASE</spring.version>
    <spring.amqp.version>1.1.3.RELEASE</spring.amqp.version>
    <java.version>1.6</java.version>
    <jackson.mapper.version>1.5.6</jackson.mapper.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <maven.javadoc.reporting.version>2.7</maven.javadoc.reporting.version>
    <commons.logging.version>1.1.1</commons.logging.version>
    <log4j.version>1.2.16</log4j.version>
    <context.path>cccd</context.path>
    <jackson.mapper.version>1.5.6</jackson.mapper.version>
  </properties>

  <build>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
      </resource>
    </resources>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>tomcat-maven-plugin</artifactId>
        <configuration>
          <warFile>target/cccd.war</warFile>
          <server>tomcat7</server>
          <url>http://localhost:8080/manager/text</url>
```

```
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
          <warName>${context.path}</warName>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>${log4j.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aspects</artifactId>
      <version>${spring.version}</version>
```

```
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-asm</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-expression</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.codehaus.jackson</groupId>
      <artifactId>jackson-mapper-asl</artifactId>
      <version>${jackson.mapper.version}</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.4</version>
    </dependency>
    <!-- Abdul -->
    <dependency>
      <groupId>org.springframework.data</groupId>
      <artifactId>spring-data-mongodb</artifactId>
      <version>1.2.0.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.2.2</version>
    </dependency>
    <dependency>
      <groupId>commons-io</groupId>
      <artifactId>commons-io</artifactId>
      <version>2.4</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
```

```
      <artifactId>spring-security-core</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-web</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-config</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.elasticsearch</groupId>
      <artifactId>elasticsearch</artifactId>
      <version>0.20.6</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.amqp</groupId>
      <artifactId>spring-amqp</artifactId>
      <version>${spring.amqp.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.amqp</groupId>
      <artifactId>spring-rabbit</artifactId>
      <version>${spring.amqp.version}</version>
    </dependency>
  </dependencies>
</project>
```

Listing 20: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">

  <display-name>Contextual and Conditional Content Distribution</display-name>

  <!-- Main configuration file for this Spring web application. -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath:cccd-config.xml
    </param-value>
  </context-param>

  <!-- Loads the Spring web application context using the config file defined
    above. -->
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <!-- Define the Spring Dispatcher Servlet for the REST services.
    The 'contextConfiguration' param with an empty value means
    that the Spring Context won't try to load a default file
    called cccd-servlet.xml -->
  <servlet>
    <servlet-name>cccd</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value></param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!-- This Servlet mapping means that this Servlet will handle all incoming
    requests -->
  <servlet-mapping>
    <servlet-name>cccd</servlet-name>
    <url-pattern>/</url-pattern>
```

```
  </servlet-mapping>

  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
      org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

Listing 21: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <!-- Appenders -->
  <appender name="console" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.out" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
        value="%-5p: [%d{MMM-dd HH:mm:ss,SSS}] %c{3} - %m%n" />
    </layout>
  </appender>

  <!-- File Appender -->
  <appender name="file" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="rest-sample.log" />
    <param name="MaxFileSize" value="10000KB" />
    <param name="MaxBackupIndex" value="20" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern"
        value="%-5p: [%d{MMM-dd HH:mm:ss,SSS}] %c{3} - %m%n" />
    </layout>
  </appender>

  <logger name="org.springframework.beans">
    <level value="debug" />
  </logger>

  <logger name="org.springframework.web">
    <level value="debug" />
  </logger>

  <logger name="org.codehaus.jackson">
    <level value="debug" />
  </logger>

  <logger name="de.fhg.fokus.ngni.cccd">
    <level value="debug" />
  </logger>
  <!-- Root Logger -->
  <root>
    <priority value="info" />
    <appender-ref ref="console" />
  </root>
</log4j:configuration>
```

Listing 22: log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oxm="http://www.springframework.org/schema/oxm"
  xmlns:util="http://www.springframework.org/schema/util"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:sec="http://www.springframework.org/schema/security"
  xmlns:rabbit="http://www.springframework.org/schema/rabbit"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.1.xsd
    http://www.springframework.org/schema/oxm
    http://www.springframework.org/schema/oxm/spring-oxm-3.1.xsd
    http://www.springframework.org/schema/util
      http://www.springframework.org/schema/util/spring-util-3.1.xsd
      http://www.springframework.org/schema/security
      http://www.springframework.org/schema/security/spring-security-3.1.xsd
      http://www.springframework.org/schema/rabbit
      http://www.springframework.org/schema/rabbit/spring-rabbit-1.1.xsd">


  <!-- Enables automatic mapping of fund objects to and from JSON -->
  <mvc:annotation-driven />

  <!-- Setup spring to pull in @Controller, @RequestMapping, etc Configuration
    scans specified packages for classes configured as Spring managed beans and
    automatically sets up objects annotated with @Controller, @Service etc. -->
  <context:component-scan base-package="de.fhg.fokus.ngni.cccd.rest" />
  <context:component-scan base-package="de.fhg.fokus.ngni.cccd.services" />

  <sec:http create-session="stateless">
    <sec:intercept-url pattern="/app/**" access="ROLE_USER" />
    <sec:intercept-url pattern="/users/**" access="ROLE_ADMIN" />
    <sec:http-basic />
  </sec:http>

  <bean id="customUserDetailsService"
    class="de.fhg.fokus.ngni.cccd.services.CustomUserDetailsService" />

  <bean id="saltSource"
    class="org.springframework.security.authentication.dao.ReflectionSaltSource">
    <property name="userPropertyToUse" value="username" />
  </bean>
```

```
<bean id="passwordEncoder"
  class="org.springframework.security.authentication.encoding.Md5PasswordEncoder" />

<sec:authentication-manager alias="authenticationManager"
  erase-credentials="false">
  <sec:authentication-provider
    user-service-ref="customUserDetailsService">
    <sec:password-encoder ref="passwordEncoder">
      <sec:salt-source ref="saltSource" />
    </sec:password-encoder>
  </sec:authentication-provider>
</sec:authentication-manager>

<bean
  class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />

<!-- Configures view for returning JSON to the client -->
<bean
  class="org.springframework.web.servlet.view.json.MappingJacksonJsonView">
  <property name="contentType" value="application/json" />
</bean>

<bean id="multipartResolver"
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>

<!-- TODO -->
<bean
  class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
  <property name="messageConverters">
    <util:list id="beanList">
      <ref bean="jsonMessageConverter" />
    </util:list>
  </property>
</bean>

<!-- Converts JSON to POJO and vice versa -->
<bean id="jsonMessageConverter"
  class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" />

<!-- Abdul -->
<bean id="mongoDb" class="com.mongodb.Mongo">
  <constructor-arg index="0">
    <list value-type="com.mongodb.ServerAddress">
      <bean class="com.mongodb.ServerAddress">
        <constructor-arg index="0" value="10.0.0.136" />
        <constructor-arg index="1" value="27017" />
      </bean>
```

```
        <!--bean class="com.mongodb.ServerAddress">
          <constructor-arg index="0" value="10.0.0.170"/>
          <constructor-arg index="1" value="27017"/>
        </bean>
        <bean class="com.mongodb.ServerAddress">
          <constructor-arg index="0" value="10.0.0.208"/>
          <constructor-arg index="1" value="27017"/>
        </bean -->
      </list>
    </constructor-arg>
  </bean>

  <bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
    <constructor-arg ref="mongoDb" />
    <constructor-arg name="databaseName" value="users" />
  </bean>

  <bean id="debugResponse" class="java.lang.Boolean">
    <constructor-arg index="0" value="true" />
  </bean>

  <bean id="esClient" class="org.elasticsearch.client.transport.TransportClient" />

  <bean
    class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
    <property name="targetObject">
      <ref local="esClient" />
    </property>
    <property name="targetMethod">
      <value>addTransportAddresses</value>
    </property>
    <property name="arguments">
      <list value-type="org.elasticsearch.common.transport.InetSocketTransportAddress">
        <bean
          class="org.elasticsearch.common.transport.InetSocketTransportAddress">
          <constructor-arg index="0" value="10.0.0.249" />
          <constructor-arg index="1" value="9300" />
        </bean>
        <!-- bean class="org.elasticsearch.common.transport.InetSocketTransportAddress">
          <constructor-arg index="0" value="10.0.0.187"/>
          <constructor-arg index="1" value="9300"/>
        </bean>
        <bean class="org.elasticsearch.common.transport.InetSocketTransportAddress">
          <constructor-arg index="0" value="10.0.0.188"/>
          <constructor-arg index="1" value="9300"/>
        </bean -->
      </list>
    </property>
  </bean>
```

```
  <bean id="connectionFactory"
    class="org.springframework.amqp.rabbit.connection.CachingConnectionFactory">
    <constructor-arg value="10.0.0.235" />
    <property name="username" value="guest" />
    <property name="password" value="guest" />
  </bean>

  <rabbit:template id="amqpTemplate" connection-factory="connectionFactory" />

  <rabbit:admin connection-factory="connectionFactory" />

  <rabbit:queue name="cccdApp" durable="true" />
  <rabbit:queue name="cccdNewDocument" durable="true" />

  <bean id="baseCtrl" class="de.fhg.fokus.ngni.cccd.rest.BaseCtrl">
    <property name="mongoDb" ref="mongoDb" />
    <property name="esClient" ref="esClient" />
    <property name="debugResponse" ref="debugResponse" />
    <property name="customUserDetailsService" ref="customUserDetailsService" />
    <property name="amqpTemplate" ref="amqpTemplate" />
  </bean>

</beans>
```

Listing 23: cccd-config.xml