Master's Thesis

Design and Implementation of a File Recommendation System Using Collaborative Filtering and Content-Based Recommendation for the Nextcloud Platform

Author:
Doğan Can Uçar

Frankfurt University of Applied Sciences
Frankfurt am Main, March 22, 2018

Matriculation number:
Degree program: Barrierefreie Systeme - Intelligente Systeme (M.Sc.)
1st Supervisor: Prof. Dr. Jörg Schäfer
2nd Supervisor: Prof. Dr. Eicke Godehardt

Ji bo her kesên kû dixwestin cîhanê bikin cîhekî başţir ...

For those who wanted to make the world a better place ...

Declaration of Authorship

I hereby declare that the paper presented is my own work and that I have not called upon the help of a third party. In addition, I affirm that neither I nor anybody else has submitted this paper or parts of it to obtain credits elsewhere before. I have clearly marked and acknowledged all quotations or references that have been taken from the works of others. All secondary literature and other sources are marked and listed in the bibliography. The same applies to all charts, diagrams and illustrations as well as to all Internet resources. Moreover, I consent to my paper being electronically stored and sent anonymously in order to be checked for plagiarism. I am aware that the paper cannot be evaluated and may be graded "failed" ("nicht ausreichend") if the declaration is not made.

Frankfurt, March 22, 2018 .....

Acknowledgement

Submitting this masters thesis finishes a chapter in my life. From the very beginning of my life at university, one of my goals was to contribute to the community. The following work is a chance to achieve this goal by contributing to an open source project.

I would like to thank my thesis advisors Prof. Dr. Jörg Schäfer and Prof. Dr. Eicke Godehardt at Frankfurt University of Applied Sciences, who have provided an excellent support during this work. Prof. Dr. Schäfer and Prof. Dr. Eicke Godehardt always had the right answer whenever a question about my research occured to me.

My special thanks are addressed to Mr. Frank Karlitschek, Managing Director at Nextcloud GmbH, Joas Schilling, Björn Schießle, Ivan Sein, Julius Härtl, Morris Jobke and Roeland Douma, Software Engineers at Nextcloud GmbH and Mr. Jan-Christoph Borchardt, Design Lead at Nextcloud GmbH who gave me the chance and confidence to complete this thesis under practical circumstances. Mr. Schilling, Mr. Schießle and Mr. Sein made it easy for me to get started with the Nextcloud framework by always having the right answers. Mr. Härtl helped me to get familiar with the Nextcloud UI development and Mr. Jobke and Mr. Douma reviewed and helped me to increase source code quality. Mr. Borchardt had the right eye for design and helped me to improve the UI from a simplicity and ease of use point of view.

Another supporter was Dr. Eng. Thomas Hildmann from "Technische Universität Berlin". Mr. Hildmann and his entire team have supported this work by contributing information and statistics for a better assessment about the real life usage of Nextcloud.

I would also like to express my gratitude to Isabell Meyer, who has reviewed this thesis continously. The thesis would not be in this quality without her professional help.

Finally I would like to thank my family and friends who have supported me not only during the master's thesis, but also during the bachelors and masters programme. Without their help my studies would not have been possible this way.

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Abstract

Design and Implementation of a File Recommendation System Using Collaborative Filtering and Content-Based Recommendation for the Nextcloud Platform

Faculty of Barrierefreie Systeme - Intelligente Systeme

Master of Science

Doğan Can Uçar

In this document I will examine the idea behind the software architecture of a recommendation system for Nextcloud. Nextcloud is an open source file sync and sharing and collaboration system, which provides data security by self-hosting. As a fork of ownCloud, Nextcloud was founded in 2016 and provides enterprise support for customers like TU Berlin, Wikimedia Deutschland, Atlassian Confluence, University of Minnesota, Bundeszentrale für politische Bildung and Bundesministerium für Arbeit und Soziales. The goal of the resulting recommendation system is a better user experience in daily work with Nextcloud.

The recommendation system is based on "Collaborative Filtering", a technique which helps to find (common) interests of users. In addition to that the system uses "Content-Based Recommendation" to analyze the content of items and use it as a similarity metric for recommendations. The combination of Memory-Based Collaborative Filtering and Content-Based Recommendation is called "hybridization" and represents the final step of the recommendation system.

Several challenges were to overcome in the development of such a recommendation system. First, the system had to be implemented as a Nextcloud App and therefore had to work within the boundaries provided by Nextcloud. In addition, there were known limitations in processing large amounts of data with PHP. Moreover, there are no official APIs or libraries for Collaborative Filtering and Content-Based Recommendation which is also a challenge to overcome. And finally, it is still not possible to centrally train the recommendation system, as it is usual for Machine Learning systems, due to the philosophy of Nextcloud being "a safe home for all your data".

The result of this thesis is a Nextcloud app that represents a recommendation system. The app is released on GitHub under the AGPLv3 and it is planned to release the app on the Nextcloud App Store.
The App ist structured in two parts: the first part, that declines the main part of the recommendation system, is a background job. The second part is the integration in the Nextcloud UI that reads and displays the results of the first part.

This document is primarily intended for the open source community, people who want to make the world a better place and to all those who are interested in machine learning topics. Therefore, I have tried to use open source tools wherever I had the chance to.

Contents

Chapter 1

Introduction

1.1 Background

Artificial Intelligence (AI) is hyped nowadays. It is a broadly arranged field, allowing applications to be created for different purposes. AI finds its way to laptops, smartphones and even to the "smart home", where each device is connected to the Internet and can make decisions autonomously. This hype arouses a lot of questions from the people who are confronted with it. What does "intelligence" mean and how can a computer be enabled to "learn"? What are the advantages in relation to "conventional" software? How does it affect everyday life, when each "object" in our environment is equipped with these techniques?

AI enables new fields of application which help people, companies and even devices to handle everyday life, reduce costs and optimize repetetive tasks. While conventional systems - systems that work according to an algorithm - are developed for a defined use-case, AI-based systems allow the adaption to a situation and learn over time. Even if commercial software is designed to be flexible and provides a wide use case, it is not able to adapt to the situation or the user without AI. This leads to the fact that users often adapt their goals to the software or create workarounds to reach their goals. Another consequence is, that the user gets used to a sequence of actions and does not elaborate new ones. AI based systems can learn the behaviour of a user and simply make suggestions to make the work easier.

In an analog world people trust in their innate likes and tastes or follow those of a person who is trusted in. So when people go to buy new things they base their choices on those tastes or recommendations. The challenge begins here: are there similar items related to this one? Could I have a better or cheaper one? Does other shops offer similar or better items?
Even though it is possible to browse all the stores and base decisions on the information that is gathered from all the possibilities, in general it does not happen, because for most people the cost-benefit ratio of these activities does not make it worth to go through all the trouble.

In the age of Internet where the selection seems to be endless, it is not only common but also impossible to inspect each item and select the best. Trying this would literally mean searching for a needle in a haystack. But what should the discovery of new, yet unknown, but also taste matching items look like? [Lev14]

Recommendation Systems

Luckily, this problem has been faced in early stages of Web 2.0. Online shops for instance have elaborated algorithms that either learn our tastes or measure the similarity between an item that we have rated earlier and is unknown to us. Amazon.com was one of the first platforms that has provided a "recommendation system". We are all familiar with the "customer who bought X have also bought Y" when we want to buy something at Amazon.com [Lev14].

A common way to create such a recommendation system is "Collaborative Filtering". There are several approaches and classifications of Collaborative Filtering[1]. The main idea of these techniques is to suggest items to a specific user that are unrated and unknown by him/her, but correspond to his tastes. Moreover, a user can get "similar" and not yet rated items recommended when the user has a common rated item base with other users.

In addition to Collaborative Filtering, there is also "Content-Based Recommendation" that can be used to create a recommendation system. Content-Based Recommendation may use the properties of an item to measure similarity. The main difference to Collaborative Filtering is that Content-Based Recommendation does not require a relation to other users [Kla09].

Both approaches have their advantages and disadvantages, which make a combination of both useful, which is known as hybridization [Kla09].

Recommendation systems can be traced back to the 1980s. One of the first Content-Based Recommendation approaches, "The Information Lense", was described in [MGT86]. Lense was developed as a "intelligent system that helps people share and filter information communicated by computer-based messaging systems". GroupLens was one of the earliest Collaborative Filtering projects describing "a system to help people find articles they will like" [PR94].
Today almost every online shop or community has a recommendation system integrated, which aims to learn the user's taste and provides them the best item as a recommendation.
But recommendation systems are not limited to shops. There are several services which already integrate recommendation systems to recommend news, music, books, posts, persons etc. to create a personal profile [Lev14].

[1] Details are described in State of the Art.

1.2 Motivation

The increasing amount of "intelligent agents" which make it easier to complete tasks or discover new items are an omnipresent reality in recent history. While computers made it easy to "compute" algorithms, the newer technologies in the area of AI will help organizing everyday life, find out customer's interests and predict possibilities of events[2].

During the masters programme I got familiar with various topics in the area of AI in a number of projects. In the "Wissen 1: Grundlagen adaptiver Wissenssysteme" course I have seen a software agent that learned to play the game "Schlag den Raab" autonomously. In "Wissen 2: Adaptive Wissenssysteme mit Gedächtnis und Symbolgebrauch" the students got to know a software system that was able to measure the similarity of two items using "Cased Based Reasoning" and in "Spracherkennung und -synthese" I had the chance to see how the music identifying service "Shazam" compares music frequencies in order to measure similarity.
We have addressed these topics in the context of accessibility (Barrierefreiheit). But is there a way that these techniques and methods may help physically or cognitively handicapped people? Is it possible to facilitate their lifes or help them do things, which were inconceivable in their past?

The recommendation system for Nextcloud, which is the subject of this thesis, is designed to help people sense fewer impulses, such as notifications, emails or entries in an activity feed. This motivation matches that of open source to be helpful to other people.
In that point the goals of an open source software like Nextcloud and the goals of my masters programme meet each other: making a contribution to the community.

1.3 Objectives of this Work

The goal of this thesis is to create a Nextcloud app that works as a recommendation system using Collaborative Filtering and Content-Based Recommendation. The app should work as an assistant, that recommends files if they are appropriate for the user.
The recommendation system aims a better overview about uploaded files. Thinking about a company's Nextcloud, where files are uploaded or deleted nearly every hour, users are overrun with new notifications. But in most of the cases, people are not interested in every file and therefore, the notification is nothing more then disturbance. For instance, a software developer does not usually care about files from the sales department and the recommendation system may help to reduce the notifications so that the users do not feel overrun.
The recommendation system consists of Collaborative Filtering and Content-Based Recommendation techniques. The results of these techniques are combined in a final "hybridization" step. This step can be reached with different methods as described in Hybridization.

[2] Example: doing X will led to Y.

1.4 Scientific Context

Recommendation systems are part of Information Retrieval particularly of Information Filtering and are developed for a better orientation within the exponential growth of data. Recommendation systems are also a part of Machine Learning since they "learn" user preferences and behave accordingly.
Recommendation systems can also benefit from Natural Language Processing to analyze the content of items [Lev14].

1.5 About Nextcloud

Nextcloud is an open source file hosting software that has functionality similar to Google Drive or Dropbox. The software provides the usage of file sharing and editing on a self-hosted server. Nextcloud also offers contacts and calendar management within the platform. Several apps on the Nextcloud App Store are available from third party developers which extend the core functionality.
Nextcloud also offers synchronization via CardDAV, CalDAV and WebDAV in order to provide contacts, calendar and files on end devices.

Nextcloud GmbH, the company behind the software, was founded by many core developers from ownCloud. As a fork of ownCloud, Nextcloud differs from its predecessor in that it is completly open source. Nextcloud provides just one software, intended for the community and as an enterprise edition. Both versions have almost the same functionality. Nextcloud also does not require a Contributor License Agreement, which means, that the source code remains the property of the developers.
Frank Karlitschek, founder and managing director of Nextcloud, has described the differences for contributers, users, customers and partners and employees in an announcement on his blog[3].

1.6 Accessibility

The masters programme "Barrierefreie Systeme / Intelligente Systeme" at Frankfurt University of Applied Sciences focuses on accessible systems from the perspective of computer science. Moreover, the interdisciplinary structure of the masters programme gave me the chance to work with students from the faculties of "Architecture" and "Case Management" in order to create accessible systems.
This thesis will focus on accessibilty in chapter Relation to Accessibility. The chapter explains the advantages of a recommendation system regarding accessibility.

[3] http://karlitschek.de/2016/06/nextcloud/

1.7 Outline

Chapter 1: Introduction

A background to recommendation systems, the motivation and objectives behind this work and the scientific context is provided.

Chapter 2: Requirements Analysis

Project organization, requirements analysis, general framework and a overview of the software design and architecture.

Chapter 3: State of the Art

This chapter contrasts the different approaches of recommendation systems, how they are used as a method to predict user preferences and make recommendations and provides an overview of the current state of research.

Chapter 4: Implementation

This chapter describes the implementation of the recommendation system described above as a Nextcloud App.

Chapter 5: Evaluation

This chapter evaluates the results of the implemented recommendation system Nextcloud app. The results of the app are compared with a predefined test data set.

Chapter 6: Relation to Accessibility

Chances and benefits of the recommendation system for Nextcloud in the context of accessibility are described shortly in this chapter.

Chapter 7: Conclusion

The results of the thesis are summarised in this chapter and the further work is explained.

Chapter 2

Requirements Analysis

This chapter gives an introduction to the requirements for Nextcloud's recommendation systems in order to get an overview over the project management andthe resulting software product. In the first section Problem Definition, the problem is explained in detail. General Conditions as defined by the environment, stackholders and tools used are explained in General Conditions. As part of the software project and in context of this work, milestones are defined and explained in section Milestones. The subsequent sections Functional Requirements and Non-Functional Requirements discuss the functional and non-functional requirements before an overview over the requirements is provided in section Requirements Matrix. The last section Software Architecture and Design introduces the global Nextcloud software architecture and the Nextcloud app architecture to have a better understanding about the resulting software of this work.

2.1 Problem Definition

Nextcloud is an open source file sharing and synchronisation software that provides only one software as the enterprise and community version. As a free and open source competitor to products like Dropbox, Google Drive or iCloud, Nextcloud has a high demand.
The demand for cloud software is obvious: people want the same version of a document on all devices, everytime and everywhere. They also do not want to care about different file versions for each device. Another big issue is the usability. People just want to "take a quick look into a document" and usually do not want to boot computers for that. This may be due to laziness, but also to practical usability.
This shall be illustrated with an example of a photographer who just bought a new camera. During a shooting he notices that he needs the camera manual because he is unable to find some of the new functions of the camera immediately. Since manuals usually are not really handy, the photographer would like to have a digital version of the manual. Booting up his laptop in the middle of a photo shoot is inconvenient and he would therefore prefer to use a cloud product to store the manual as document. Cloud products like Nextcloud require only the upload of the file into a cloud. Simply installing the corresponding mobile app will solve many problems: The photographer is not supposed to carry several manuals in his camera bag

and does not have to boot the laptop. Additionally he is no longer responsible to keep the files up to date on each device he uses.
A recommendation system is not yet relevant up to this point. But the situation changes rapidly when the photographer described above works in a company that have many employees with multiple cameras.
In case of Nextcloud, users would get a notification about new or changed files or it would be shown on the Activity App. This applies also to files that are irrelevant for the user, such as documents from sales or human resources departments.
At this point it becomes apparent that effective filtering in large Nextcloud environments with a high amount of users is necessary. Assuming that every user has access to all files, users will have to face a range of files they are not able to organize anymore. Therefore a machine learning approach is needed to make filtering possible.
The following chapters will define requirements for a recommendation system that is used for Nextcloud. The filtering techniques are based on Collaborative Filtering und Content-Based Recommendation as discussed in State of the Art.

2.2 General Conditions

This section describes the general project framework such as basic conditions and system requirements. Subsection System Requirements describes technical requirements such as hard and software required for running Nextcloud, their minimum required versions and the used tools throghout the development process. The following subsection Basic Conditions describes basic conditions such as programming language/Nextcloud related boundaries, available development tools and decisions made by the stakeholders as well as provided by the company philosophy.

System Requirements

During the time this thesis was written, Nextcloud 13 was in development. The final version was planned for December 2017 or January 2018. But there are no requirements for Nextcloud 13 defined yet. The app is therefore based on the requirements of version 12, but is intended for version 13. The requirements for Nextcloud 12 are defined in [Nex] as:

- minimum 128MB RAM,
- Linux Server,
- Apache or Nginx webserver,
- MySQL, MariaDB or PostgreSQL database,
- PHP 5.6 or 7.0.

As explained in the officiall PHP documentation, PHP in version 7 or above got a massive performance improvement [PHPc]. For running machine learning related tasks this is essential

due to the nature of processing large amount of data. Therefore, it is highly recommended to use PHP 7.0 or above.

Throughout the entire development process, a MacBook Pro with MacOS High Sierra, 16GB RAM and a SSD hard disk and a MacBook with MacOS High Sierra, 8GB RAM and a SSD hard are used. Software testing is based on PHPUnit which allows unit tests of single classes and modules. The evaluation is mainly made on local machines (the MacBooks) as well as Nextcloud GmbH internal Nextcloud instances.

Basic Conditions

There are some basic conditions that have to be considered for this app. First of all it has to be mentioned that Nextcloud is a web application that loads its apps at runtime. This situation is associated with several possible problems: if the request runs against a timeout, the webserver will stop the request and throw an exception. If this is not the case, the app will possibly need too long to respond so that the user gets frustrated. This will inevitably cause users to delete the app.
Therefore, it is a good approach to implement the apps core functionality as a background job and share the results with other apps. A different approach is to split the recommendation system in two parts: the backend which is responsible for the recommendations itself and a frontend that displays the results.

A general approach to machine learning related tasks is the collection of training and test data centrally. This way the app could better learn about all user preferences and make better predictions. In practice, this would result in sending training data to Nextcloud servers through the app.
However, this is not possible with Nextcloud. Nextcloud promises privacy as a "safe home for all your data". The transfer of data from the "home" to a central database would lead to an inconsistency of their corporate philosophy. This results in the app only being trained locally, meaning that there is only local stored data available to train.

The first idea at the early stages of the discussions about the recommendation system for Nextcloud was the implementation of software that is nearly independent of the Nextcloud framework. The reason was that PHP, in which is Nextcloud is written, is not a good choice for machine learning related tasks. Better options are Python or Java programming languages. Not only because of their ability to process big data sets and run parallel threads but also because there are a lot of frameworks, which would be a better choice. The goal was a background job that processes all necessary data and stores the results in a database where Nextcloud can simply read out.
However, the stakeholders insisted that it has to be a Nextcloud App within Nextcloud boundaries. This circumstances require PHP as the programming language for at least the core implementation and the restrictions described in System Requirements.

2.3 Milestones

The time schedule is provided in form of milestones in order to have a better overview:

Milestone	Description	Planned End
Initializitaion	assembling tools, final discussions and registration	01.11.2017
Scientific Elaboration	scientific elaborating and reasoning	01.01.2018
Implementation	implementation and test	01.03.2018
Conclusion	conclusion	01.04.2018

2.4 Functional Requirements

Functional requirements were defined in several meetings with Nextcloud and Frankfurt University of Applied Sciences. The focus was on a intelligent agent for Nextcloud, better usability for end users and a new approach to keep the overview of data.

2.4.1 Relation to Machine Learning

From the very beginning of the discussions of this app Machine Learning was the central topic. The common agreement was that only Machine Learning related techniques are able to provide a powerful way to learn user preferences and a better feeling with the product. The resulting app should act as an intelligent agent that is able to make decisions autonomously, provides a better overview about relevant data and should help users in working with Nextcloud.

2.4.2 Input & Output

The main data source during this thesis are document files like Office, PDF or plain text files. The user is not required to provide additionally data. Using Nextcloud continiously is enough. The results have to be stored in a database or provided by an interface to other apps.
The app must assemble all required information such as ratings, co-rated users and the content independently. In case of the file content, the app must be able to parse different file types and read out the content.
The app stores its results in a database or provides it to other apps via an interface.

2.4.3 Data Processing

The app should work as a background job and therefore independent from the user interface. However, it is up to the user to activate or deactivate the app via the Nextcloud app settings page. The core processing of the app should be extendable for other types of input[1].

[1] more Information in subsection Extensibility.

2.4.4 Recommendations

The app should recommend other files that are shared with a user and could potentially interest him. The recommendations should be a help, not a disturbing factor. The user should have the chance to dismiss or deactivate the recommendations.

2.5 Non-Functional Requirements

Non-Functional requirements explain properties that are not in relation with specific functions to the system. The topics of software quality, data protection and legal aspects are relevant for the recommendation system.

2.5.1 Software Quality

Software Quality is a central requirement. Since PHP is not intended for processing big data sets, source code must be as small and effective as possible.
All tools and the programming language itself has to be latest version possible. In case of PHP 7 this is essential due to the performance improvements.
Furthermore, Object Oriented Programming is also a measure of quality due to reusable source code. Coding guidelines are also available in the Nextcloud developer documentations [NC2].

2.5.2 Privacy and Legal Aspects

As mentioned before, privacy is very important to Nextcloud philosophy. Therefore, the app should use as little data as possible. This might be a legal problem as data transfer to and especially from some countries is not allowed. In addition, the data is not allowed to leave the local instance at any time.

2.5.3 Performance

The app is a software product around a web application. Since there are several aspects to consider, like server timeouts and network transfer, all decisions about algorithms and source code should take performance as a key point into account. That means that the most performant algorithm has be used to achieve the functionality [NC3].

2.5.4 Usability

Usability refers to the Nextcloud guidelines and are defined in [NC4]. Thus, software should work out of the way. Instead of configuring options, the app should automate the workflow. Furthermore, only the most important elements should be visible on a page. Secondary elements should rather be placed on hover or via "Advanced" pages.
With the final version of the app, Nextcloud should also provide a better usability and user

experience. Users should be able to use Nextcloud more easily. The app should support the user in finding an orientation within a growing number of files.

2.5.5 Portability

The app should work on any system that runs Nextcloud. Therefore, no operating system or compile version related functions may be used. In Software Quality "the newest version of all tools and programming language" is defined as a requirement. However, the app should also work with previous versions as long as it is possible.

2.5.6 Supportability

Due to the nature of open source, the source code is open for everyone for inspection, modification and extension. This requires that the source code is maintainable and that the complexity level is as low as possible. Furthermore, source code that needs explanation has to be commented.

2.5.7 Extensibility

Data files are the main data source during this thesis. However, the system should be designed to be extensible.
There are a lot of Nextcloud apps in the Nextcloud App Store. Popular apps are the Calendar, Contacts or the Mail app. These apps should be able to use the recommendation system and provide their own data as input or get the results as the output in further versions of the app.

2.5.8 Transparency

The app should also provide transparency. The users should be informed about actions that are performed by the app. In case of recommendations, the user should know why a specific item is recommended to him/her. The app should also provide information about files that are processed and user preferences that are derived from them. However, this requirement is optional since it is not a key feature of the resulting app.

2.6 Requirements Matrix

The requirements described above are summarized in this table in order to have a better overview. All requirements are mandatory, except for the "Transparency" requirement.

Name	Description	Functional/Non-Functional
Machine Learning Area	implementing a machine learning related topic	Functional
File Metadata Access	reading metadata of files in order to process them	Functional
File Content	reading the content of a file in order to process it	Functional
Different File Types	processing word, PDF and plain text files	Functional
No additional actions	no additional actions by users required	Non-Functional
Providing results to other apps	providing results in database or via an interface	Functional
Parallel Data Processing	background job that does not block the main UI	Non-Functional
Recommend Files	recommending files that are relevant for the user	Functional
newest development and production tools	using newest development tools and programming language for better performance	Non-Functional
taking Nextcloud coding, privacy, usability guidelines into account	taking the appropriate guidelines (coding, security, usability) into account	Non-Functional
Independence from the environment	being independent from the operating system and compile version of any software	Non-Functional
Maintainability	being maintainable for everyone who wants to work with it	Non-Functional
Extendability	being extendable for other kinds of input or output data	Non-Functional
Transparency	providing information about recommendations ("why" and "how")	Non-Functional

Table 2.1: Requirements Matrix

2.7 Software Architecture and Design

This section should introduce Nextcloud's software architecture and app architecture. This should help to have a better overview about the software product and the way how apps are developed.
The following description is based on Nextcloud 12 hosted on GitHub. The core code is public and available for inspection. Because the core source code of Nextcloud is not well documented, the following subsections are results of code inspection in the context of this thesis. Much information and insight was gained by asking the core developers.

2.7.1 Nextcloud Architecture

This section introduces the Nextcloud architecture shortly. The main goal is the familiarization with the framework in order to achieve a better understanding.

index.php

The entry point of Nextcloud is the $index.php$ source code file. After some initial checks this file loads $base.php$ that contains the OC class[2]. During inclusion, the static function $init()$ is called which loads the Nextcloud core, checks sessions and environment variables, loads the autoloader[3] and registers the hooks[4].

base.php

$index.php$ calls the $handleRequest()$ static method of the OC class in $base.php$. This method checks whether Nextcloud is installed or in maintenance mode and searches for a logged in user. All enabled apps and their classpathes are also loaded here. Then, the app differs the actual "use mode". If the request is web UI based, the method matches URLs to controllers using routers. If the request ist a WebDAV request, the appropriate modules are loaded and responded. If none of these modes match, the user is redirected to the default page or login page, respectively.

Routing

Nextcloud uses the routing mechanism of Symfony. A router lets developers define URLs that can be mapped to to different areas of the application. It allows a flexible, extendable and easy way to read and process URLs [NC6][SYM].
Each Nextcloud app defines its own routes. Nextcloud can then decide at runtime which URL corresponds to the which app or class. The routes are registered by the $routes.php$ file that is mandatory for each app.

[2] Because Nextcloud is a fork of ownCloud, all classes, files, databases etc. are prefixed with or named OC.
[3] Normally, a developer is responsible to include his/her source code manually. This means that each file has to be included separatly. The autoloader automates this task and searches for source code files at defined places. More informations are available at [PHPa].
[4] hooks are used to execute code before and after events. More informations are available at [NC5].

Dependency Injection and Container

Dependency Injection is a software design pattern that defines a simple rule: do not instantiate objects within a class but pass them through the constructor. The problem of acquiring dependencies is a different problem by disallowing instantiating objects. Thus, creating or configurating different objects for specific classes is delegated to one or more central "injectors". Moreover, Dependency Injection allows easy source code testing [NC7].
But Dependency Injection has one central disadvantage: All lines of code have to be refactored if the number of arguments passed to an object gets changed. The solution of this problem is to limit the instantiation of objects to one container. Instantiating all objects in the container allows an easy handling of the objects passed through with Dependency Injection.
However, in case of Nextcloud there is "Automatic Dependency Assembly" implemented and is the recommended way using Dependency Injection. That means developers are forced to use type hints[5] for their constructor arguments. Doing this Nextcloud determines the needed objects and passes them through the constructors automatically [NC7].

Background Jobs

Nextcloud defines three types of Background Jobs: AJAX, Webcron and Cron are possible types that are introduced in this subsection:

- AJAX: Ajax is a asynchronous web request technique usually combined with JavaScript. Nextcloud uses AJAX requests each time a page is refreshed to check whether jobs are registered or not. AJAX is the default but not the recommended mode by Nextcloud.
- Webcron: Webcron uses external cron services to execute tasks within Nextcloud.
- Cron: Cron is a system feature of unix-like operating systems. This mode enables flexible execution of jobs that would fail through webserver limitations. Cron is the recommended background job type by Nextcloud.

Registering Background Jobs is also done within a Nextcloud App [NC8].

Nextcloud Request/Response Flow

Nextcloud provides three main flows in order to interact with the underlying system. The first flow is the access to the API and core through Apps[6]. Different Sync-Clients, such as mobile apps or desktop clients, can request or submit data via HTTP.
The second flow is the command line interface CLI. The Nextcloud CLI, named *occ*, allows users to interact with Nextcloud via a shell for mass changes, for example.

[5] Because PHP is a "weakly typed" programing language, meaning that the types of variables and objects are determined at runtime, type hinting is not required by default.

[6] Nextcloud is organized with apps which extend the core functionality. More information in section Nextcloud App Architecture.

The last flow is the web application itself, where the user can interact through an HTML interface.

Figure 2.1 shows possible interaction types with Nextcloud:

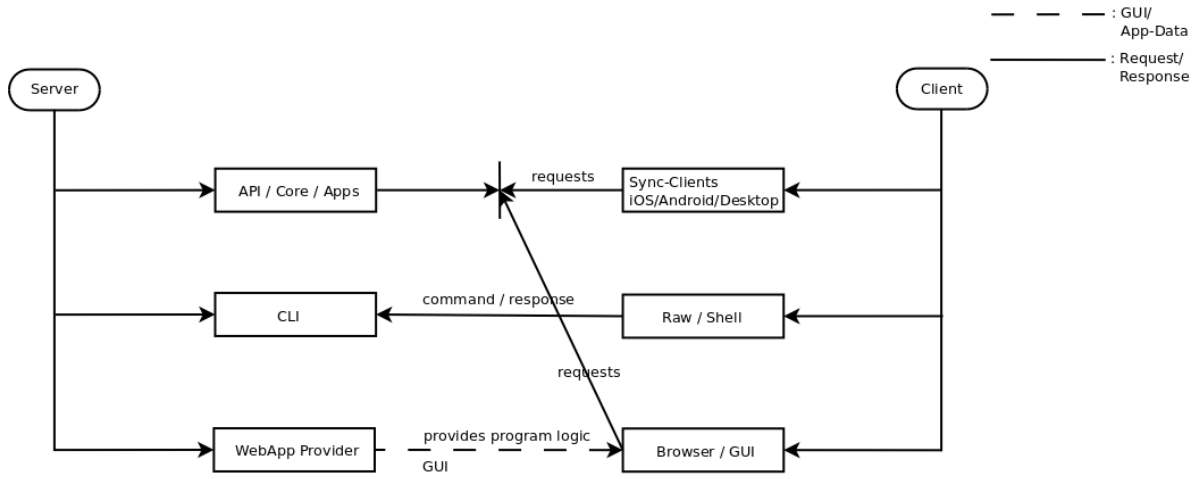

Figure 2.1: Request and Responses in Nextcloud

2.7.2 Nextcloud App Architecture

The main entry point to Nextcloud is *index.php*. The file loads all apps and configures all necessary options like hooks, for example.
When loading an app, there are three files that are relevant before executing any app related code. The first one is *info.xml* and contains app descriptive information, such as minimum and maximum version numbers.
app.php configures basic options, such as including composer autoloaders and registering the *Application* class, which then registers services to the dependency injection container, for example.
routes.php registers all app related routes (URLs). The file returns an array with the appropriate values. The following listing shows the route for the recommendations UI. The "routes" array contains three keys: "name" defines the class name and method ("Recommendation" and "index"), "url" defines the URL that should match the name and verb defines the HTTP RESTful verb.

```
1 return [
2 'routes' => [
3   ['name' => 'recommendation#index', 'url' => '/
        recommendation_assistant_for_files', 'verb' => 'GET'],
4 ]
5 ];
```

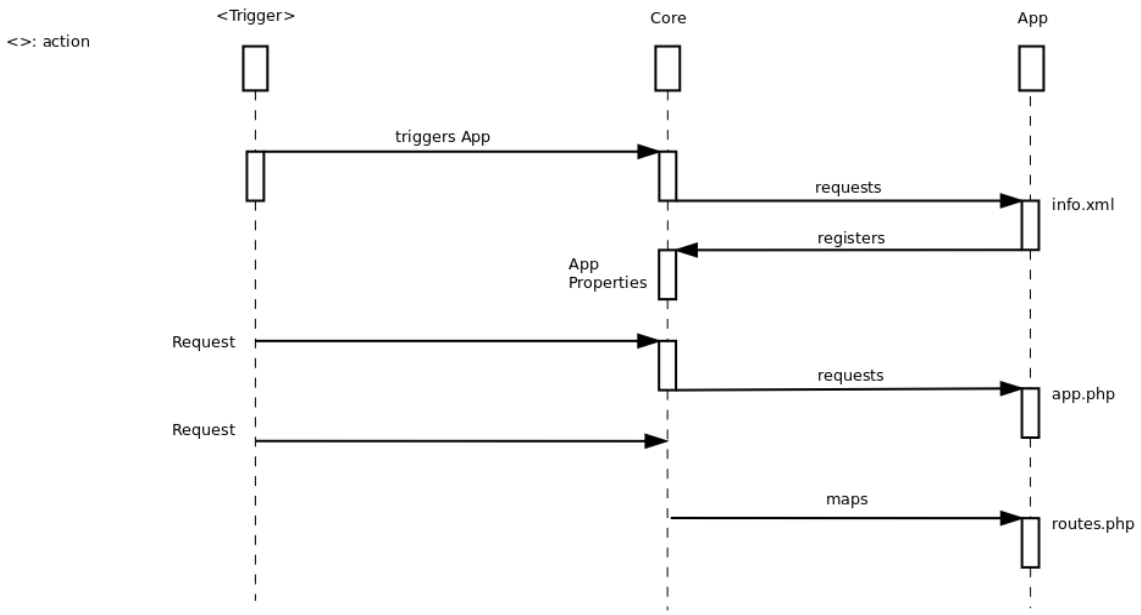

Figure 2.2: Sequence Diagramm of an Loading Apps

Chapter 3

State of the Art

3.1 Introduction

People recommend various things on a daily basis. Doing this they rely on the knowledge, experience or preferences of themselves or other people. This can be seen as a social process since recommendations depend on many factors. At the same time these recommendations are utterly individual depending on the situation. A system that is able to make recommendations should idealy work the same way. All these factors, which can vary over time, should be taken into account when making a recommendation [BHC98].

The demand for recommendation systems is growing parallel to the huge amount of data and the wide range of selection. As discussed in subsection Recommendation Systems it is nearly impossible to browse all available options within this selection.
A good recommendation system that leads users to interesting items in a large selection is essential in order to keep the overview of the selection. Ideally the recommendation system is personalized, meaning that it measures similarities that are appropriate for a user. It can track the change of user preferences over time so that the recommendations are not outdated.

There are several approaches to classify recommendation systems. In [Bur02] and [Bur07] the systems are classified by their data sources on which recommendation is based. The first one is "Content-Based Recommendation System" that uses properties of an item as a similarity metric. "Collaborative Filtering" uses items or users and "Demographic Recommendation System" uses demographic information like age, sex and occupation to measure similarity. "Utility-Based Recommendation System" applies a utility function, usually from the area of Machine Learning, over an item to determine a similarity rank and the "Knowledge-Based Recommendation System" attempts a recommendation based on inferences about a user's preferences.

[Gor16] follows another approach and classifies in "Neighborhood-Based Recommendation Systems", "Personalized Recommendation Systems" and "Model-Based Recommendation Systems". "Neighborhood-Based Recommendation System" searches for items similar to the items

rated by a user. "Personalized Recommendation System" considers the content or context, like location, date or time, of an item. "Model-Based Recommendation System" calculates weights of items preferences, along with Machine Learning methods, and uses these weights for recommendation[1].

[NP13] has stated Collaborative Filtering, Content-Based Recommendation and Knowledge-Based Recommendation as the most "important and widely used" recommendation systems and [Kla09] discusses Collaborative Filtering and Content-Based Recommendation for its recommendation system.

Considering that each of these techniques has its own advantages and disadvantages, it is a good approach to combine them to a "hybrid recommendation system" to get the best results and bypass the disadvantages. In [Bur02], [LS13] and [Kla09] this approach is discussed.

In order to create the recommendation system for Nextcloud, appropriate techniques, such as Content-Based Recommendation, Collaborative Filtering as well as their hybridization are needed. The following chapter introduces these techniques, explains advantages and disadvantages and describes the way they are hybridized and reasons why they were used.

3.2 Collaborative Filtering

3.2.1 Introduction

The basic idea behind Collaborative Filtering is to find users with common interests by assuming that people with similar tastes in the past will likely have similar tastes in the future. The main property to work with Collaborative Filtering are ratings.
Ratings by users for items are used to measure the "similarity" and may be captured explicitly with continous request for user feedback or implicitly by observing his/her behaviour. The rating may be binary such as like/dislike or a numeric value in a given scale.
A rating matrix is defined as "the set of ratings given to different items" and is also known as the user profile [NP13][LH16][CCFF11].

As stated above, Collaborative Filtering depends on the overlap in ratings accross the rating matrix. Once the ratings are sparse, meaning that a couple of users have rated a limited number of items, measuring similarity gets more difficult. Correlation between users can only be computed on the base of co-rated items. This problem is also known in domains where the selection of items is large and - assuming there is no large user base as well - users have not rated many items. The larger the item base, the less likely is the overlap of co-ratings. This leads to the fact, that most recommendations are based on just a part of the available items and possibly represent no real similarities. This difficulty is known as the "Sparsity Problem".

[1] In Machine Learning, inputs are weighted. The constant adjustment of these weights to obtain the best possible results is essential in order to "learn".

Collaborative Filtering is also limited when a user is recently added to the user base since he/she has no ratings on which predictions can be made. Due to the limited number of ratings by the new user the predictions will be inaccurate. This problem, known as the "Early Rater Problem", also applies in environments that even have a large user base[2].
Usually in small user environments there are individuals with opinions that do not consistently match with other users of the user base. These users would not benefit from Collaborative Filtering since their interests do not match the interests of others. This problem, known as the "Gray Sheep Problem", would also apply even if the users have successfully bypassed the "Early Rater Problem".
Memory-based approaches use one global model for all items when making recommendations. This means the model does not consider special circumstances. This may result in inaccurate or wrong recommendations [Bur02][LH16][BP98].

It soon becomes apparent that Collaborative Filtering works best, if the user interests are highly distributed across a small selection of items. If the article base changes rapidly, old ratings become less relevant. Furthermore, similarilty will be impossible if the item base is too large and the user interests manageable.
Collaborative Filtering works best for users that clearly fit into one or more "group of interest". The "group" need enough members with similar (but not identical) tastes since Collaborative Filtering relies completly on user ratings and do not need any descriptive data [Bur02][LH16].

Another strength of Collaborative Filtering is the independence of the item type and works well even for complex objects like movies, music or books since it relies on object properties [Bur02]. This becomes clear in [LH16], where a "Personalized Sports News Recommendation System" using Collaborative Filtering is discussed. The paper explains the advantage of their system over Content-Based Recommendation as the "possibility to provide event or trend based recommendations, such as news about the World Cup". Assuming that a user is not interested in dart league but in the championship, a simple Content-Based Recommendation cannot be able make recommendations because the user has never read articles about dart before. Another advantage is the ability to recommend cross over items with different properties, such as books, movies or articles [Bur02][LH16].

The following chapter will introduce the different Collaborative Filtering approaches, illustrate advantage and disadvantages and explain the choice for the recommendation system for Nextcloud.

3.2.2 Formal Definition

Let C be the set of all users in the user base and S the set of all items in the item base. Moreover, let u be the utility function that measures the utility of s to c. The following function maximizes the user's utility results in order to choose such items $s \in S$ for each user

[2]This problem is also known as the Cold Start Problem.

$c \in C$:

$$\forall\ c \in C, s'_c = arg\max\ u(c,s)_{s \in S} \tag{3.1}$$

where $u(c,s)$ is defined as:

$$u(c,s) = \sum u(c_j, s) \tag{3.2}$$

where c_j is defined as "all users similar to c" [JZ09].

3.2.3 Memory-Based Collaborative Filtering

Memory-Based Collaborative Filtering uses the whole rating matrix to compare users or items by using a measurement method to make predictions [CCFF11]. [JZ09] calculates an unknown rating $r_{c,s}$ for item s by user c with the following aggregation:

$$r_{c,s} = aggr\ r_{c',s} \tag{3.3}$$

where c' is defined as "a set of N users that are most similar to c and have rated item s". Three aggregation functions are introduced in [JZ09]. The first one is a simple average function, defined as:

$$r_{c,s} = \frac{1}{N} \sum r_{c',s} \tag{3.4}$$

The second one considers a similarity factor $sim(c',c)$ and a weight k in order to calculate the rating:

$$r_{c,s} = k \sum sim(c,c') \times r_{c',s} \tag{3.5}$$

where k is defined as: $k = 1/\sum |sim(c,c')|$.

The last equation takes into account that users make their ratings in a different way and introduces an "adjusted weighted sum". While some user rate rather low, others rate comparebly high. Taking this into account while calculating a rating, the algorithm subtracts the average rating of a user from each rating on both items.
A more significant problem in this approach is the fact, that if there is only one common user between two items, the similarity for those items will have the highest value due to the subtraction of the average rating. To avoid this a minimum number of users that two articles need to have in common has to be specified.

The algorithm is defined as [LH16]:

$$r_{c,s} = \overline{r_c}\ k \sum sim(c,c') \times (r_{c',s}, \overline{-r_c}) \tag{3.6}$$

where adjustion $\overline{r_c}$ is defined as:

$$\overline{r_c} = \frac{1}{|S_c|} \sum s \in S_c r_{c,s} \tag{3.7}$$

and S_c is the set of all items s that have at least one rating by user c with the formal definiton:

$$S_c = \{s \in S | r_{c,s} \neq \emptyset\} \tag{3.8}$$

[JZ09]. [LH16] uses 3.5 for their "Personalized Sport News Recommendation System".

Similarity Function

Equation 3.5 and 3.6 refer to a similarity function $sim(c, c')$. Many approaches exist in order to compute this similarity. [JZ09] and [LH16] explain the most common ones:

1. *Nearest Neighbor:* this method is divided in two subcategories.

   - *Pearson Correlation Coefficient:*

   $$sim(x,y) = \sum_{s \in S_{xy}} (r_{x,s} \overline{-r_x})(r_{y,s} \overline{-r_y}) / \sqrt{\sum_{s \in S_{xy}} (r_{x,s} \overline{-r_x})^2 \sum_{s \in S_{xy}} (r_{y,s} \overline{-r_y})^2} \tag{3.9}$$

   where S_{xy} is the set of all items that have been rated by two users x and y. The algorithm identifies a set of users C that contains all corated items s by users x and y. Note that only these co-rated items are appropriate for the algorithm. The basic goal is to identify how much a user rating deviates from the average rating for a item. $R_{x,s}$ is the rating of the user on the item and $\overline{r_x}$ is the average rating of the item.

   - *Cosine-based:* treating users as vectors in m-dimensional space, where $m = |S_{xy}|$ containing all corated items. $sim(x, y)$ is measured by computing the cosine of the angle between them:

   $$sim(x,y) = cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}||_2 \times ||\vec{y}||_2} = \sum_{s \in S_{xy}} \frac{r_{x,s} r_{y,s}}{\sqrt{\sum_{s \in S_{xy}} (r_{x,s})^2} \sqrt{\sum_{s \in S_{xy}} (r_{y,s})^2}} \tag{3.10}$$

   where $\vec{x} \cdot \vec{y}$ is the dot product between $\vec{x}$ and $\vec{y}$. Note that this approach does not consider the average rating of an item.

2. *Graph-Based:* The basic idea behind this approach is to utilize graph theory in order to build relations between users. Doing this, a graph between users is maintained over time whose nodes are the users and edges the relationship between them. This approach enables the "on demand" recommendation without computing through the entire rating matrix since the graph is constantly being updated. This approach is described in detail in [AWWY99].

3. *Mean Squared Difference:* measures the degree of dissimiliarity between users by mean squared difference. Recommendations may be made by considering all users with a dissimilarity to the user which is less than a defined threshold [SM95].

User-Based and Item-Based approaches

Memory-Based Collaborative Filtering is applicable for User-Based and Item-Based approaches. User-Based approaches find a set of users who share co-rated items. Then, the algorithm aggregates items of similar users, eliminates already rated items and recommends the remaining items.
[LSY03] introduces a different approach and focuses on finding similar items, not users. For each rated item the introduced algorithm finds similar items and recommends them to the user. The similarity can be computed in various ways. However, the *Cosine-Based* function is mentioned by [LSY03].
After calculating the similarity between the items, a prediction for the user has to be made. [SKKR01] lists different prediction approaches, such as weighted sum and regression.
Weighted sum uses of the ratings given by the user on the items similar a specific item. Each rating is weighted by the corresponding similarity between two items. More formally:

$$P_{u,i} = \frac{\sum_{k<0}^{k<N} s_{i,k} * r_{u,k}}{\sum_{k<0}^{k<N} |s_{i,k}|} \tag{3.11}$$

The regression approach is similar to weighted average. Instead of using the ratings directly, it uses an approximation of the ratings based on a regression model.

3.2.4 Model-Based Collaborative Filtering

Model-Based Collaborative Filtering uses the entire rating matrix to learn a model that is used to derive a model to make predictions. This can be seen as the calculation of the expected rating for an item by an user.
The "Sparsity Problem" explained in Introduction can be reduced in model-based approaches thanks to their ability to learn data characteristics underlying for a user. In spite of the fact that model-based approaches take a certain amount of time to learn a model, they tend to be faster then memory-based approaches.
The "Early Rater Problem" and "Gray Sheep Problem" also apply for model-based approaches since time is needed to learn a model and in model-based approaches can even be available users whose tastes do not consistently agree with any other group of people.
However, model-based approaches still have several problems. Many models are too complex and several parameters rely on estimations. Moreover, data changes are a problem as model-based algorithms are sensitive to them. The system will recommend wrong items if one of the models does not fit the rating matrix. Finally, when modifying the model or adding new data, the model needs some time to adjust [Bur02][JZ09][CCFF11].

[BHK98] defines Model-Based Collaborative Filtering as:

$$p_{c,s} = E(r_{c,s}) = \sum_{i=0}^{n} Pr(r_{c,s} = i | r_{c,s'}, s' \in S_c) \times i \tag{3.12}$$

where

- $p_{c,s}$ is the rating prediction for item s by user c,
- 0 and n are the ranges for rating values,
- $Pr(r_{c,s} = i | r_{c,s'}, s' \in S_c)$ is the probability expression for the probability that c will rate s with a particular rating taking the previously observed ratings into account.

[Bur02], [BHK98], [CCFF11], [JZ09] and [AT05] introduce several learning methods, such as Neural Networks, Bayesian Networks, Linear Regression and Support Vector Machines in order to build the model above. [JZ09] explains the approach to define the recommendation process as a decision problem and suggests Markov Decision Processes for making recommendations. [BP98] compared different algorithms of model and memory-based approaches and concludes that model-based approaches perform better under some circumstances. But the results are empirical and have no scientific foundation.

3.3 Content-Based Recommendation

3.3.1 Introduction

While Collaborative Filtering focuses on ratings and ignores content, Content-Based Recommendation works with properties of items and uses them to make recommendations. In this context "property" can be defined broadly. It can be a type of item, several buzzwords or the content itself such as text, images or audio.

In case of Content-Based Recommendation, a "user profile" is smiliar as it is defined in Collaborative Filtering and is the "set of item properties that are owned by a user". [CGM+99] describes a content-based and collaborative filter for an "online newspaper" and divides their properties into sections that contain implicit keywords identified by the system itself. Moreover, users can specify additional keywords in order to let the system know about further preferences.

Content-Based Recommendation is less affected by the problems stated in Collaborative Filtering because they do not rely (only) on items that are rated by users. For example, if a newly added item has a given property that is defined in the user profile, the item will be recommended to the user before anyone has rated it [LH16].

However, Content-Based Recommendation has a number of limitations. First, it is only focused on content and cannot differentiate between "good" and "bad" content. This circumstance applies also for two different items that are described by the same set of properties. In this case they are indistinguishable. Second, the number of "similar items" will increase with the growth of the item base and thus, Content-Based Recommendation becomes increasingly ineffective. Third, content-based approaches will work fine for machine readable content such as text or URLs. In case of multimedia content such as images or audio it requires a step of preprocessing

which can be challenging. As an alternative, users may assign the properties manually which is often not a good solution [CGM+99][CCFF11][AT05].
Content-Based Recommendation will also return poor results for items or user profiles with fewer keywords. The reason for this is simple: If, for example, the item/user profile does not contain enough keywords, the overlap coefficient will compute a slight similarity.

As described above, Content-Based Recommendation is most effective for textual items. The text item, hereinafter reffered as document, is parsed to get the most important information out of it. The result is a list of keywords. These keywords are then used to calculate a degree of match in order to measure the similarity of two or more documents.

The following section will introduce a few common techniques for content-based recommendation that are relevant for this work.

3.3.2 Formal Definition

Content-Based Recommendation Systems computes the similarity between the properties of an item and the properties from the user profile in order to make a recommendation. More formally, a utility function $u(c, s)$ that estimates the utility of s for c based on past utilities $u(c, s')$ to items $s_i \in S_c$ that are assigned by c and "similar" to s. The formal definition of the utility of item s to user c would be:

$$u(c, s) = score(UserProfile(c), ItemContent(s)) \tag{3.13}$$

Where $UserProfile(c)$ contains a set of keywords associated to c and $ItemContent(s)$ contains the keywords from an item s. There are many methods available in order to calculate the $score$. They will be introduced in subsection Degree of Match.

3.3.3 Stopword Removal and Stemming

Stopword Removal and Stemming is part of preprocessing of unstructured text in order to optimize the comparision between documents. Stopwords are words that do not represent the content and are often found in a document. Examples include "and","or","in", etc. [Fox89].
Stemming is defined as the process to summarize words that are similar in their meaning in their root forms. For example, term that reflects the common meaning behind "computation", "computes", "computes" or "computers" could be "compute" [PB07].
The following subsections will shortly introduce both methods.

Stopword Removal

There are several approaches for removing stopwords. [Fox89] has generated a stop list based on the Brown corpus[3]. The list consists of 421 words that is "maximally efficient in filtering

[3]The Brown corpus is a lexical and grammatical analysis of one million words of American English assembled at the Brown University, Providence, USA in 1963.

the most frequently occurring neutral words" according to the authors. Several stopword lists are available at different web platforms such as GitHub.

However, stopwords are highly tied to the language used and therefore, a lot of maintanance effort is required. The TF-IDF measure addresses this problem and can be used to work more language independent. TF-IDF weights all keywords within a document and the entire item base where these weights represent the importance of the keyword. TF-IDF is based on two main assumptions:

1. words that occure often within a document are relavant (Term Frequency, TF)
2. words that occure often in the item base are not relevant (Inverse Document Frequency, IDF)

TF-IDF positively rewards words that occur frequently within a document, whereas frequent occurrences in the item base are rewarded negatively.
More formally, let $t \in T$ be a keyword, $d \in D$ a document and $i \in I$ the item base. Furthermore, let n be the number of keywords in a document, n_i the number of keyword i within a document and m the number of documents that contain t at least once. Then, the weight w for t in TF-IDF is calculated as [Kla09][AT05][PB07][Paz99]:

$$\begin{aligned} w(t) &= TF \times IDF \\ TF &= \frac{n_i}{n} \\ IDF &= log_{10}\frac{|I|}{m} \end{aligned} \tag{3.14}$$

TF-IDF will return the value 0 when the number of total items in the item base and the number of items containing a keywords at least once are equal. In this case, $|I|$ and m equal to 1, and log_{10} of 1 will result in 0. The multplication of TF and 0 equals to 0.

3.3.4 Degree of Match

After extracting descriptive keywords out of the document, a measure that defines the matching degree between documents has to be specified. There are a lot of approaches to achieve this. Many of them are specialized versions of classification learners that aims to learn a function to predict the class of a document to make recommendations. There are also regression based approaches that try to predict a rating for the document. The following subsection will introduce some of them [Paz99].

Overlap Coefficient

[CGM+99] uses the "Overlap Coefficient" to measure the degree of match between keywords. Formally:

$$M = \frac{2|D \cap P|}{min(|D|, |P|)} \tag{3.15}$$

where D is the set of keywords of the article and P is the set of keywords from the user profile.

Sport1.de Algorithm

[LH16] introduces a separate algorithm in order to measure similarity[4]:

$$sim(g,h) = \frac{\sum_{i \in W}(g_i \times h_i)}{\sqrt{\sum_{i \in W} {g_i}^2 \times \sum_{i \in W} {h_i}^2}} \tag{3.16}$$

where g, h are vectors with keywords and their weights, W is the set of keywords, i is a keywords and g_i, h_i are weights of i in g and h.

Latent Semantic Indexing

[Kla09] introduces "Latent Semantic Indexing" to make recommendations. First, a matrix of keywords and documents is created. The matrix is large due to the property that the keywords denote rows and the documents denote the columns of the matrix.
The size of the matrix is reduced by using "Single Value Composition". Keywords, that often occure together within different documents are summarized to one "virtual keyword". This approach can be used to identify similar documents.

3.4 Other Filtering Techniques

Recommendation systems are not limited to Collaborative Filtering and Content-Based Recommendation. There are several different approaches that are shortly introduced in this section for reasons of completeness.

3.4.1 Kowledge-Based Filtering

Knowledge-Based Filtering relies on data provided by the user and uses them as constraints in order to get the most similar results. For example, if the constraint has the rules "price $<$ 150 USD", all items that are cheaper then 150 USD should be recommended to the user. Knowledge-Based Filtering does not build long-term generalization about the users, but bases their recommendations on users need and the items available [Bur02][PG14].

3.4.2 Demographic Filtering

Demographic recommenders classifies the users in demographic classes and tries to recommend based on them. "Demographic" information can be the age, sex, religion or ethnicity, for example [Bur02].

[4] [LH16] does not define a name for this algorithm. Therefore, it is called "Sport1. de algorithm" in the following.

3.5 Hybridization

After defining the preferred methods for recommendation, the combination of these methods has to be considered in more detail. The hybridization can help to overcome the disadvantages of the specific methods. For example, the content-based recommendations are useful in case of missing ratings in the Collaborative Filtering approaches. Thus, it can help to bypass the "Sparsity" and "Cold Start" problems. There are several methods for the "Hybridization" available that are introduced in the following section [CCFF11].

3.5.1 Weighted Average

The goal in weighted average is to find the weights that come up with results that have the most accurate prediction. [CGM+99] defines weights for the output of Content-Based Recommendation and Collaborative Filtering methods and adjusts them over time. The weights are initialized equally and are adjusted when the user provides feedback. The feedback is obtained by confirmation or disconfirmation, for example.
The weighted average is also used in [LH16]. The combination is called "Weighted Hybrid Recommender" and weights the recommendations equally. When both methods recommend the same item, the item is recommended. Notice that [LH16] does not adjust the weights.
The advantage of weighting the recommendations of each method lies in the simplicity of combining all results in a final recommendation. It is also easy to influence the final result if one of the methods used is not accurate enough.
However, this approach assumes that each method used has the same recommendation accuracy for an item. From the discussion in section Content-Based Recommendation it is known that this is not always the case. For example, a Content-Based Recommendation method will be weak if there are too little keywords in the user profile [Bur02].

3.5.2 Switching

In this approach the decision about the final recommendation is based on the switching between the used recommendation methods. In [BPC00] a switching hybridization is used, where the system tries to make a content-based recommendation first. If the result of the Content-Based recommendation is inaccurate, the system makes a Collaborative Filtering based recommendation.
Since both methods have the "Cold Start" problem, this approach also has its limitations. Moreover, switching requires additional complexity because the switching criteria has to be defined and evaluated continously [Bur02].

3.5.3 Mixing

Mixing may be useful if the result should consist of more than one recommendation. In that case, the final recommendations may consist of the results of several recommendation methods [Bur02].

3.5.4 Feature Combination

In Feature Combination, the results of the Collaborative Filtering method is treated as feature data for a Content-Based Recommendation method. This approach considers Collaborative Filtering results without relying on it exclusively. Doing this, the "cold starter" problem may be mitigated [Bur02].

3.5.5 Cascading

Cascading uses recommendation methods for a staged process. The first step is to find candidates that may be recommended. In a second step, these candidates are refined to a set of final recommendations [Bur02].

3.5.6 Model Using

The main idea is the creation of model that uses Content-Based Recommendation and Collaborative Filtering methods. For instance, [AT05] discusses different approaches with rule-based classifiers, probabilistic methods, Bayesian classifiers using Monte Carlo methods or Case-Based Reasoning.

3.5.7 Monolithic

[PG14] introduces a "monolithic" approach where a final recommendation bundles a number of all features of specific methods.

3.5.8 Pipelining

A pipelined recommendation process consists of multiple recommendation methods where the output of one recommendation method is the input of another. The last recommendation method makes the final recommendation [PG14].

3.6 Reasons for Methods Chosen

This section concludes by explaining the reasons for the chosen methods and reflecting the described approaches above.
The process of making recommendations is structured as follows: First, methods of Collaborative Filtering and Content-Based Recommendation are applied to those items that are not recommended. The results are stored separately in the system. Then, the hybridization process will make a final recommendation out of these results and store them in the system. And finally, the calculated results are provided to other parts of the system on demand.

3.6.1 Collaborative Filtering

This thesis makes use of Memory-Based Collaborative Filtering for the recommendation system for Nextcloud for performance reasons. As discussed in subsection System Requirements the system only has only limited computing power. Moreover, the limitations of the used tools and programming language as explained in subsection Basic Conditions forces us to use methods that are fast in computing and resource saving. In addition, Model-Based Collaborative Filtering requires training data to learn a model and apply it to new data. Since no training data is available for this thesis, an evaluation of an implemented Model-Based Collaborative Filtering is not possible.

As described in User-Based and Item-Based approaches, Memory-Based Collaborative Filtering is aplicable for items or users, respectively. The main difference is how recommendations are made: the user-based approach finds "similar" users followed by "similar" items. This results in $O(U * I)$ time complexity. Item-based approaches depend only on the number of items and independent from users. Therefore, its complexity is limited to $O(I)$. Since computation power is restricted, the user-based approach is not an option for Memory-Based Collaborative Filtering. The *Cosine-Based* approach as described in Similarity Function is used to measure similarity between two items.
After measuring similarity between items the Collaborative Filtering part will use weighted average[5] as described in Similarity Function in order to make a prediction for a user.

3.6.2 Content-Based Recommendation

Content-Based Recommendation covers stopword removal and content similarity by a degree of match. The items are provided by the Nextcloud framework and thus, the recommendation process can focus on content extraction and not about file system operations, for example.

Stopword Removal

The Content-Based method will care about stopwords but not about stemming. Stopwords are "naturally" excluded with TF-IDF as explained in Stopword Removal. It is not necessary to care about used language or need to import stopword lists externally.
In case of stemming the situation differs: since the linguistic constitution of words are highly tied to the language, it is not possible to provide support for different languages within this work. Early thoughts were the support for only Englisch language. But this approach would be problematic since our test data is in German language.

Degree of Match

The "Degree of Match" between a document and keywords associated to a user profile is calculated with the $Overlap\ Coefficient$ with a minor deviation as defined in [CGM+99].

[5] not to be confused with weighted average in section Hybridization.

The paper defines:

$$M = \frac{2|D \cap P|}{min(|D|, |P|)} \tag{3.17}$$

As a result, the following applies to M: $0 \leq M \leq 2$.
Since a similarity value between 0 and 1 is required, the equation is modified as follows:

$$M = \frac{|D \cap P|}{min(|D|, |P|)} \tag{3.18}$$

In Evaluation is a second evaluation approach based on modification time stamps explained. Since the rating range is spread over 0 to 5, the $Overlap\ Coefficient$ is modified as:

$$M = \frac{5|D \cap P|}{min(|D|, |P|)} \tag{3.19}$$

The $Overlap\ Coefficient$ has a time complexity of $O(1)$ since it does not iterate over all keywords of D or P and bases on a simple equation.

Similarity and TF-IDF

A first attempt was made to calculate the degree of correspondence with the Sport1.de algorithm. However, this failed because the measure TF-IDF does not define a constant weight for a keyword over all documents. This should be demonstrated with a simple example:

$$D = \begin{pmatrix} This \\ is \\ an \\ example \end{pmatrix} \qquad P = \begin{pmatrix} The \\ weather \\ is \\ rainy \\ this \\ week \end{pmatrix} Q = \begin{pmatrix} Hello \\ world \end{pmatrix}.$$

The keyword "this" has different TF-IDF results since the calculation considers the number of keywords of a single document:

TFIDF("this" in D) $= \frac{\frac{1}{4}}{log(\frac{3}{2})} = \frac{0.25}{0.17609125905568} = 1.4197183968169$
TFIDF("this" in P) $= \frac{\frac{1}{6}}{log(\frac{3}{2})} = \frac{0.16666667}{0.17609125905568} = 0.94647895014085$

Instead of using TF-IDF as a part of similarity measurement, it servers only for excluding keywords while generating a user profile and the documents keywords. Keywords with a TF-IDF value less then a defined threshold[6] are considered as "stopword".

3.6.3 Hybridization

The weighted average approach described in Weighted Average is suitable for this thesis since it provides extendibility. This is very useful, since the resulting app can be extended in further versions with additional data sources that may require other filtering techniques.
In addition, the weighted average provides an easy way to change weights if one of the techniques dominates the result.
Hybridization has a constant time complexity of $O(1)$ since it first multiplies the weights with the similarity results and builds the average. For n items, the hybridization complexity is $O(n)$.

[6] The threshold is not defined yet since it should be evaluated later.

Chapter 4

Implementation

4.1 Introduction

The previous chapter has described the theory behind a hybrid recommendation system using Collaborative Filtering and Content-Based Recommendation. Based on these results, this chapter will elaborate the implementation of a recommendation system within the Nextcloud framework.
"Recommendation Assistant" will mainly work as a background job. The results are stored in a database table[1] that can be used to query the data and visualize them, for example. The reason for a background job is simple: Processing X files of Y users "on demand" would result in bad response times. "On demand" is defined as a (re)load of the page in case of Nextcloud.

The familiarization with the Nextcloud framework can be done using the official documentation [NC9]. However, Nextcloud is in ongoing development, which means that there are constant changes that are not part of the documentation yet.
As mentioned in further chapters Nextcloud 13 was in development when working on this masters thesis. The main new feature of this version was the "end-to-end encryption" as anounced in [NC1a]. In previous versions, the way of working on registering background jobs, menu entries and database tables has also changed.
Since the official documentation of Nextcloud 13 is not released yet, the only way to access necessary information was to ask the (core) developers. A further way getting familiar with the Nextcloud framework is the ownCloud documentation. As mentioned in previous chapters, Nextcloud is a fork of ownCloud and therfore, it almost consists of the same software architecture. Considering this, it is possible to use ownCloud documentation or forums in order to access informations about the app development.

The resulting recommendation system uses five external libraries which are loaded via composer and the autoloader mechanism. *rtf-html-php*[2] is a framework to parse RTF documents

[1] Database access is described in section Database Storage.
[2] released on GitHub: https://github.com/henck/rtf-html-php

and licensed under the AGPLv2 license. *PHPSpreadsheet*, *PHPWord* and *PHPPresentation* are libraries to parse office documents by *PHPOffice*[3] and are licensed under LGPLv3. The last library, *PdfToText*[4], is used to parse PDF files and licensed under *GPLv3*.

4.2 General App Architecture

Before the app architecture and implementation is described, this sections starts off by describing general app architecture and the design patterns behind Nextcloud.

4.2.1 PSR-4 Autoloading

Nextcloud supports the PSR-4 autoloading mechanism since version 10. The basic idea behind this mechanism is to map class namespaces to file paths.

For example, the namespace:

```
1 \Acme\Log\Writer\File_Writer
```

is mapped to the file:

```
1 ./acme-log-writer/lib/File_Writer.php
```

where ./acme-log-writer/lib/ is the base directory of the project.

In case of Nextcloud, the *OCA\MyApp* is mapped to the */apps/myapp/lib/* where "myapp" is the name of the app.
In addition to that, the autoload mechanism of package managers such as composer can be used. If this is the case, the *application/app.php* file needs to include the *vendor/autoload.php* file of composer [NC1b][PHPb].

4.2.2 Dependency Injection

Nextcloud uses Dependency Injection software pattern in order to assemble necessary object instances and provides them to apps if necessary. Dependency Injection means that classes should not instantiate any other class but get them passed in. There are different ways to pass the instances such as through the constructor and getter-setter methods.
Dependency Injection simplifies source code for maintenance and testing. The class that gets an instance injected must not care about it anymore. It "just uses" it. If the object passed to a class requires any changes, for example in that way how it is created, this is done outside of the

[3] released on GitHub: https://github.com/PHPOffice

[4] The library is available via GitHub but was not set up for composer. Therefore, I have forked the project and added composer support. The library is available here: https://github.com/doganoo/PdfToText

class. Moreover, in case of such changes, the class that uses does not need retested because the preparation of the object is out of its responsibility.
Dependency Injection has also several disadvantages. If the class that gets instances injected needs further objects, all occurrences of the class need to be changed. This issue can be solved using so called "containers".

A container is a central location where the classes that gets instances injected are created. Within the container, the classes are created and the necessary instances are injected. If there are new instances required by a class, the only place where the changes have to be applied is inside the container.
The container is queried wherever the class is needed. The container creates the class und passes it to the file that requests the class[5].
Nextcloud provides an "Automatic Dependency Assembly" that provides the passthrough of core objects to the classes which is the recommended way by Nextcloud. Doing this makes the code more readable and maintable according to the official documentation [NC1c][MWD][JSD].

4.3 Nextcloud App Architecture

In several discussions with the Nextcloud core developers as well as the managing director, one of the strategic goals of Nextcloud was described as being modular and extensible. That is why all features, including the file sharing feature, are organized as "apps" that are downloadable from the Nextcloud App Store as well as from GitHub.
An app is defined as a bundle of source code that extends the Nextcloud core with a specific functionality. The app can easily be enabled or disabled on the settings page[6]. According to [NC1d], the folder structure of an app is as follows:

- *appinfo*: apps metadata and configuration
- *css*: (additional) CSS files
- *js*: (additional) JavaScript files
- *lib*: apps PHP files
- *templates*: (HTML) templates
- *tests*: (unit) tests of the app

The first file regarding to an app that is processed by Nextcloud is the $appinfo/app.php$ file. Upon the purpose and version of the app, this file registers menu entries, background jobs or

[5] In this work, the container is used to get the necessary service classes for the background jobs as shown in TimedJob.

[6] Some features are architectural outlined as apps but it is not possible to deactivate or delete them. They are therefore part of the Nextcloud core although they are located under the "apps" folder.

hooks.
$appinfo/info.xml$ contains metadata about the app such as the unique app id, name, description, license or version. $info.xml$ defines also the minimum version of Nextcloud that is required by the app. And since Nextcloud 10 it is also possible to register background jobs via this file. The appropriate section of the $info.xml$ of this app looks like:

```
1    <dependencies>
2    <nextcloud min-version="13" max-version="13"/>
3  </dependencies>
4
5  <background-jobs>
6    <job>OCA\RecommendationAssistant\BackgroundJob\RecommenderJob</job>
7    <job>OCA\RecommendationAssistant\BackgroundJob\UserProfileJob</job>
8  </background-jobs>
9
```

The $background - jobs$ XML-node is responsible for registering background jobs. In this app are two background jobs defined: the $RecommenderJob$, which performs the recommendation process, and $UserProfileJob$, that assembles the keywords for a user profile. Both classes are in the namespace $OCA\RecommendationAssistant\BackgroundJob$.

4.4 RecommenderJob

As mentioned in section Nextcloud App Architecture, the $RecommenderJob$ class is the entry point for making recommendations. The autoloader described in PSR-4 Autoloading ensures that $RecommenderJob.php$ located at $apps/recommendation\_assistant/lib/backgroundjobs$ is included successfully. The $RecommenderJob$ class is registered as an background job via the $info.xml$ as shown above.

4.4.1 TimedJob

The $RecommenderJob$ class must extend the abstract class $OC\BackgroundJob\TimedJob$. $TimedJob$ requires the implementation of the $run()$ method. Moreover, $RecommenderJob$ needs to specifiy an interval that defines when the background job should be executed. Thanks to Dependency Injection, the service class $RecommenderService$ can be injected through the constructor. Nextcloud will then pass the right instance to the class.

The $run()$ method executes the service class that implements the business logic:

```
1 class RecommenderJob extends TimedJob {
2
3   public function __construct(RecommenderService $recommenderService) {
4     $this->setInterval(RecommenderJob::INTERVAL);
5     $this->recommenderService = $recommenderService;
6   }
7
8   protected function run($argument) {
9     $this->recommenderService->run();
10  }
11 }
```

4.4.2 RecommenderService

The $RecommenderService$ class is responsible for executing the Collaborative Filtering and Content-Based Recommendation algorithms described in chapter State of the Art. This subsection will explain the structure of this class in detail.

$RecommenderService$ gets instances of $IRootFolder$, $IUserManager$, $ITagManager$, $IGroupManager$ and several database access instances injected. The $IRootFolder$ instance provides a view to the users files folder in order to access the files.
The $IUserManager$ instance provides all users that are registered to the Nextcloud instance. This object provides a callback $callForSeenUsers()$ for all users that have ever logged in into Nextcloud. This callback allows to consider only that kind of users who uses Nextcloud continuously. Moreover, considering that the Nextcloud instance has several hundred users, it would be a performance problem if the app loops over all users every time.
The $ITagManager$ instance provides access to the tags that a user gives to a file. The $ITagManager$ also provides access to the information about whether one file is tagged as favorite or not. Technically, tagging a file as favorite is nothing more than tagging a flag with the string "favorite".
The $IGroupManager$ instance provides access to the the groups in which a user is a member.

The $callForSeenUsers()$ is called within the $run()$ method that is called by $RecommenderJob$ as shown above. The callback function which must be provided to $callForSeenUsers()$ gets an instance of $IUser$ interface and represents the actual user that is processed.
The callback function first needs to initialize all mount points of the user. This initialization is critical since without initializing the user profile will not contain shared files and files that are on a remote storage. Under certain circumstances the files that are owned and uploaded by the user are also not visible.
The $IRootFolder$ provides user folders using the $getUserFolder()$ method which returns the

user's root folder. The method can be executed after initializing the mount points.
The root folder may contain files or folders again. Therefore it is necessary to check the type of the returned instance in order to ensure that all files in all folders are processed.

The $handleFolder()$ method first checks the type of the returned instance. If the object is an instance of $OCP\backslash Files\backslash Folder$, the method calls itself recursively. If the object is an instance of $OCP\backslash Files\backslash File$, the $handleFile()$ method gets called.
The recursive call to $handleFolder()$ returns a list of items and has to be merged with the already existing list. The $handleFile()$ method returns a single item which needs to be added to the existing list.
Before the (recursive) calls are executed the method ensures that the node is valid, meaning that it is not encrypted and readable by the app. It is critical to validate an instance of $OCP\backslash Files\backslash Node$ and not $OCP\backslash Files\backslash File$, as Nextcloud 13 introduces end-to-end-encryption whose "encrypted" flag is set to folders and not files.

```
1  private function handleFolder(Folder $folder, IUser $currentUser):
       ItemList {
2    $itemList = new ItemList();
3    try {
4      foreach ($folder->getDirectoryListing() as $node) {
5        $valid = NodeUtil::validNode($node);
6        if ($valid) {
7          if ($node instanceof Folder) {
8            $return = $this->handleFolder($node, $currentUser);
9            $itemList->merge($return);
10         } else if ($node instanceof File) {
11           $return = $this->handleFile($node, $currentUser);
12           $itemList->add($return);
13         }
14       }
15     }
16   } catch (NotFoundException $exception) {
17     Logger::warn($exception->getMessage());
18   }
19   return $itemList;
20 }
```

The $handleFile()$ method checks the files mime type first. If it is not a supported mime type, the method returns "false". Then, the method calls three more methods in order to prepare the files for Collaborative Filtering and Content-Based Recommendation: First, an instance of $OCA\backslash RecommendationAssistant\backslash Objects\backslash Item$ is created by calling the $createItem()$ method. Then, the $addRater()$ method is called which checks if the file is tagged as favorite or not. The last method $addKeywords()$ reads the files keywords and appends them to the *Item* object.

The way how the content is parsed and the determination of favorites are described below. The result of the $handleFile()$ method is an instance of $Item$ which contains the necessary information for Content-Based Recommendation and Collaborative Filtering. The list is then used in two further classes: $OverlapCoefficientComputer$, that is the implementation of the Degree of Match using the Overlap Coefficient as described in Overlap Coefficient and $CosineComputer()$, which is the implementation of Cosine-Based similarity as described in Similarity Function. Both implementations are explained in Content-Based Recommendation Implementation and Collaborative Filtering Implementation in detail.

4.4.3 Reading File Content

One of the key properties of Content-Based Recommendation is the ability to make recommendations based on the content of an item (a document in this case). Therefore, it is necessary to parse the content of a file that is in the user profile. There are several file types that are supported actually:

- Microsoft Word (docx)
- Microsoft Excel (xlsx)
- Microsoft PowerPoint (pptx)
- OpenDocument Spreadsheet (ods)
- OpenDocument Text (odt)
- PDF
- Plain Text (txt)
- HTML
- JSON
- RTF
- TXT
- XML

Each of these types has its own class which implements the $IContentReader$ interface. This interface requires only one function $read()$ that takes a $OCP\backslash Files\backslash File$ instance as an argument. The function requires a string as the return type. Detailed information about the implementation is available in subsection Reading File Content.

4.4.4 Favorites

As mentioned above, the $ITagManager$ instance can be used to achieve tags that are given to a file. To determine whether a file is tagged as "favorite" or not, the tags for a user must be loaded with the $load()$ provided by $ITagManager$. The method returns an object of $OCP\backslash ITags$ which provides the method $getFavorites()$. This method returns an array with the file ids that are tagged as favorite. Requesting the array for an appropriate file id determines whether this file is a favorite or not.

In chapter Evaluation, the evaluation is made with two kinds of rating types. In Evaluation Based on Modifications Timestamps, the favorite tagging and last modification timestamps are converted to a rating. The timestamps are stored in a database which is filled whenever a user decides to tag a file as a favorite.

4.4.5 TU Berlin Statistics

The Technische Hochschule Berlin uses Nextcloud within their own infrastructure in order to provide webspace for their students and staff. While the TU is one of the biggest customers of Nextcloud, the company had the chance to ask them for some statistics in order to asses the situation.
The Managing Director of Nextcloud, Mr. Frank Karlitschek, has requested statistics about the usage of the "favorites" functionality from the TU Berlin on 11/17/2017.
Dr. Thomas Hildemann, Head of Infrastructure Department at TU Berlin, has responsed to our request on 11/20/2017 with the following statistics:

Name	Description
Number of Persons that uses the "Favorites" function	1982
Number of items that are tagged as "Favorite"	2727
User with the most favorites	72
User with the least favorites	1

The "TU Berlin" Nextcloud instance has approximately 22.000 active users. The statistics show that approximately 2.000 users use the "tag as favorite" functionality. Therefore, the function can be used for a first evaluation but is not reliable for a constant success of Collaborative Fitering[7].

[7] In a second evaluation attempt, the evaluation is based on different kind of "ratings". More information is available at chapter Evaluation.

4.4.6 Item

The representation of items (documents) is necessary for both Collaborative Filtering and Content-Based Recommendation. Therefore, a class $Item$ is created and structured as follows:

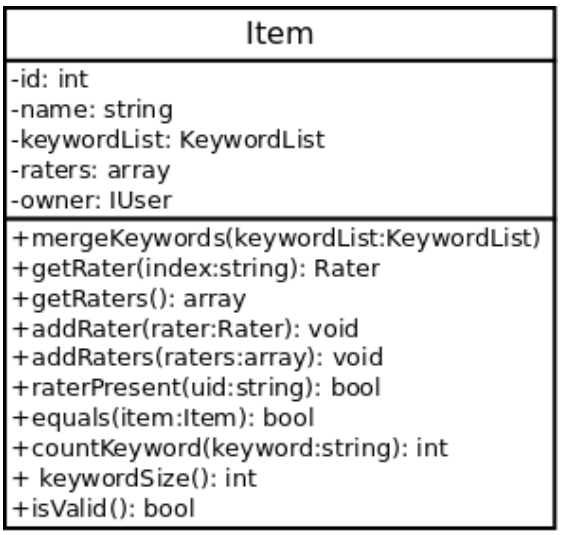

Figure 4.1: UML Class Diagram of class Item

4.4.7 ItemList

The above explained $Item$ class has to be managed within a list in order to access the instances easier when calculating the similarity. Therefore, the class $ItemList$ is created. This class implements the $IteratorAggregate$ interface in order to be iterable. The interface requires a single function $getIterator()$ that returns an instance of $ArrayIterator$ with the item classes as an array. The structure of $ItemList$ is as follows:

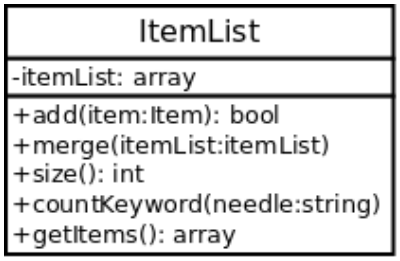

Figure 4.2: UML Class Diagram of class ItemList

A special attention should be paid to the $add()$ method which adds an instance of $Item$ to the list. This method has to ensure that there are no duplicated items in the list. The list uses the item id as the unique index and first checks if the list contains an item with this id. If this is

the case, the item keywords and raters are merged with the available item. If not, the item is added to the list.
This characteristic lets the $ItemList$ class act like a Set, where each element can be represented only once.

4.4.8 Keyword

The keywords are represented by the $Keyword$ class. Each $Keyword$ instance has the keyword itself as a string, the corresponding TF-IDF value and an integer value to count the occurrences within a list. The $count$ attribute enables the following $KeywordList$ class to contain each keyword only once and increment the attribute whenever the keyword is added multiple times.

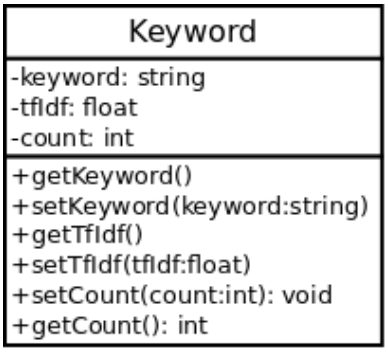

Figure 4.3: UML Class Diagram of class Keyword

4.4.9 KeywordList

The $Keyword$ instances are managed in a $KeywordList$ that is constructed similarily to the $ItemList$ class. $KeywordList$ implements the $IteratorAggregate$ interface in order to be iterable.

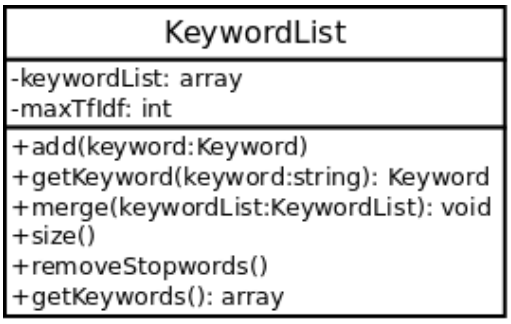

Figure 4.4: UML Class Diagram of class KeywordList

$Keyword$ instances are accepted by the $add()$ method of the $KeywordList$ class. When adding an instance to the list, the method first searches for the keyword in the list. If the keyword is

not in the list, it is added and no other actions are required. If the keyword is already in the list, the available instance is retrieved and the instance counter is increased by one. This approach is known as Bag-of-Words [BoW]. Doing this it is ensured that the list contains each keyword only once. The keyword counter, which is a simple integer value, represents the number of occurrences within in the list. This number can then be used for CosineComputer, for example.

4.4.10 HybridItem

Instances of $HybridItem$ class represent the results of Content-Based Recommendation and Collaborative Filtering measurements and contain the corresponding values. Moreover, it implements a method $getWeightedAverage()$ which performs the Weighted Average as described in Weighted Average. The $isRecommendable()$ method compares the hybrid recommendation result to a defined threshold in order to decide whether the item is recommendable.

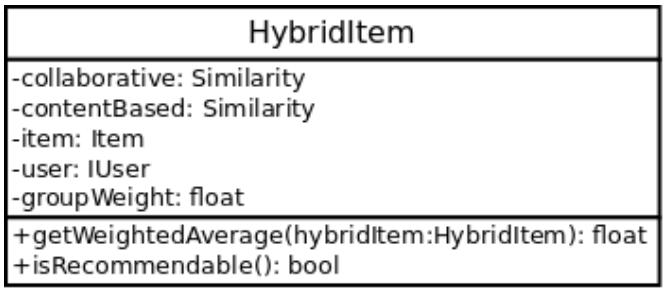

Figure 4.5: UML Class Diagram of class HybridItem

4.4.11 HybridList

The $HybridList$ class serves as a collection for $HybridItem$ instances and is similar constructed as the $List$ classes described above.

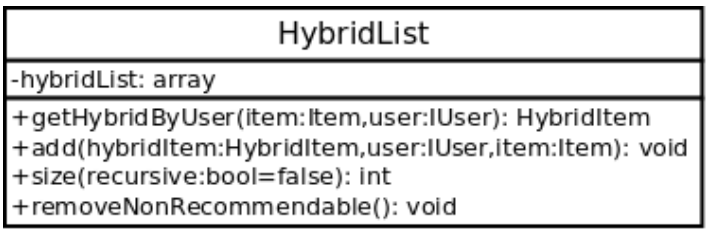

Figure 4.6: UML Class Diagram of class HybridList

4.4.12 Interface IComputable

The $IComputable$ interface has to be implemented for each object that calculates similarity between two or more objects. The interface requires only one method $compute()$. The UML representation is as follows:

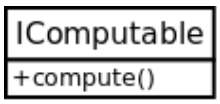

Figure 4.7: UML Class Diagram of interface IComputable

4.4.13 Interface IContentReader

The *IContentReader* interface has to be implemented for each object that reads file content of a given mime type. The interface requires only one method *read()*, that gets an instance of $\backslash OCP\backslash Files\backslash File$ as an argument passed. The UML representation is as follows:

Figure 4.8: UML Class Diagram of interface IContentReader

4.5 Collaborative Filtering Implementation

This section describes the implementation of the Collaborative Filtering component as described in Collaborative Filtering. The Collaborative Filtering component is implemented using a Memory Based algorithm. Model Based approaches require a training phase and data set in order to train the model. But in the actual stage of this thesis, these requirements can not be met. Moreover, Memory Based algorithms require much more computation time than Model Based aproaches.
There are several algorithms within Memory Based approaches available as explained in Similarity Function. For this implementation, we are going to use the $CosineSimilarity$ in order to calculate the similarity between two items.

4.5.1 User Ratings

Collaborative Filtering relies on user ratings as explained in detail in chapter Collaborative Filtering. While ratings between 1 and 5 are possible on platforms such as Amazon.com or Netflix, this is not possible on Nextcloud. There is neither a functionality nor a use case for such a kind of rating.
Instead, we the "tag as favorite" functionality of Nextcloud is used where users can tag files as favorites. Doing this the files will appear on top of the list.
This means, that the $CosineSimilarity$ measure will use a binary rating. If the user has tagged a file as favorite, the rating will be 1. Otherwise the rating equals to 0.

In a second evaluation attempt, the way how user ratings are considered has been modified in order to evaluate it in comparison to a second model. In this attempt, user ratings are

assembled as a weighted average of two main attributes: the file modification and favorite tagging time stamps by a user. The advantage of this approach is that the time stamps can be spread and thus, enable rating ranges that are normally not possible on Nextcloud, as mentioned above. The rating range is defined from 0 to 5 inspired by platforms like Amazon.com or Netflix. Therefore, a rating with value 0 means that the user has not reacted to the item yet. A rating value 1 is the lowest and 5 the most highest possible rating and are defined as:

min days since last change	**max change since last change**	**rating**
0	3	5
3	5	4
5	10	3
10	15	2
15	20	1
20	∞	0

The results of the first attempt are described in Evaluation Based on Static Tags whereas the results of the second attempt are described in Evaluation Based on Modifications Timestamps.

4.5.2 CosineComputer

The $CosineComputer$ class is responsible for the $Cosine\ Similarity$ algorithm and implements the $IComputable$ interface. The class gets two instances of $Item$ through the constructor injected and calculates the similarity of them. Since the class implements $IComputable$, it is required to implement the $compute()$ method, which is structured as follows:

```
1  public function compute(): Similarity {
2    $similarity = new Similarity();
3    if ($this->sourceItem->equals($this->targetItem)) {
4      $similarity = Util::createSimilarity(1.0, Similarity::
           SAME_COSINE_ITEMS, "the items are the same");
5      return $similarity;
6    }
7    $denominatorA = 0;
8    $denominatorB = 0;
9    $numerator = 0;
10   $denominator = 0;
11   /** @var Rater $rater */
12   foreach ($this->sourceItem->getRaters() as $rater) {
13     $yValid = $this->targetItem->getRater(
14       $rater->getUser()->getUID()
15     )->isValid();
16     if (!$yValid) {
17       continue;
18     }
```

```
19      $sourceRating = $rater->getRating();
20      $targetRating = $this->targetItem->getRater($rater->getUser()->getUID
            ())->getRating();
21      $numerator += $sourceRating * $targetRating;
22      $powX = pow($sourceRating, 2);
23      $powY = pow($targetRating, 2);
24      $denominatorA += $powX;
25      $denominatorB += $powY;
26    }
27    $denominator = sqrt($denominatorA) * sqrt($denominatorB);
28    if ($denominator == 0) {
29      $similarity = Util::createSimilarity(0.0, Similarity::
            NO_COSINE_SQUARE_POSSIBLE, "multiplication returned 0");
30    } else {
31      $similarity = Util::createSimilarity($numerator / $denominator,
            Similarity::VALID, "ok");
32    }
33    return $similarity;
34  }
```

The method checks first if the items are equal. If this is the case, the method returns a instance of *Similarity* with a similarity value of 1. If the items are not the same, the method calculates *Cosine Similarity* as described in Similarity Function. If the denominator equals to 0, the method returns an instance of *Similarity* with a value of 0 in order to prevent a "division by zero" exception. Otherwise, the division required by *Cosine Similarity* is executed and the *Similarity* instance is returned.

It has to be mentioned that the *CosineComputer* class only calculates the similarity between items and does not predict a user's rating. For this purpose, there is a second class *RatingPredictor*:

```
1   public function predict(): Similarity {
2     $similarity = new Similarity();
3     $numerator = 0;
4     $denominator = 0;
5
6     $itemArray = $this->itemList;
7     if (!Application::DEBUG) {
8       /*
9        * k-nearest neighbor approach: remove all items that have a
             similarity less than a threshold
10       */
11      $itemArray = array_filter($this->itemList->getItems(), function (Item
            $item, string $key) {
12        $sim = $this->matrix->get($this->item, $item);
13        return $sim->getValue() > Application::
              K_NEAREST_NEIGHBOR_SIMILARITY_THRESHOLD;
```

```
14      }, ARRAY_FILTER_USE_BOTH);
15    }
16
17    /** @var Item $item1 */
18    foreach ($itemArray as $item1) {
19      if ($this->item->equals($item1)) {
20        continue;
21      }
22      $sim = $this->matrix->get($this->item, $item1);
23      $rating = $item1->getRater($this->user->getUID())->getRating();
24      $numerator += $sim->getValue() * $rating;
25      $denominator += $sim->getValue();
26    }
27    if ($denominator == 0) {
28      $similarity = Util::createSimilarity(0.0, Similarity::
            NO_SIMILARITY_AVAILABLE, "denominator is 0");
29    } else {
30      $similarity = Util::createSimilarity($numerator / $denominator,
            Similarity::VALID, "ok");
31    }
32    return $similarity;
33  }
```

The *predict* method first removes all items whose similarity value is less than a defined threshold. The $K\text{-}Nearest\text{-}Neighbor$ method ensures that only items that are similar enough are used for prediction. Then, the weighted average as described in Similarity Function are used to calculate a prediction for a given item and user.

4.6 Content-Based Recommendation Implementation

This section explains the Content-Based Recommendation component of the recommendation system as defined in Content-Based Recommendation. This component implements the $Overlap\ Coefficient$ as defined in Degree of Match in order to measure the degree of match between two sets of keywords.
The "Sport1.de Algorithm" approach, that was originally the preferred algorithm for Content-Based Recommendation, uses keyword weights in order to calculate the similarity. However, this will not work with TF-IDF values. The reason is simple: one keyword may have a different weight within two different documents. An example is described in Similarity and TF-IDF.
"Latent Semantic Indexing" requires a lot of computation time which has to be avoided according to the requirements in Requirements Analysis. For that reason, this approach is inappropriate, considering the limitations of the programming language and in order to avoid timeouts due to long computation time.

4.6.1 User Profile

Many Content-Based Recommendation approaches create a user profile that consists of keywords which describe the user's tastes. This profile has to be maintained over time because user tastes can (and will) change over time. This requires at least one additional process that assembles new keywords and removes those that no longer describe the user's tastes.
Forcing users for additional actions such as inserting keywords can be frustrating and time-consuming. Therefore, this thesis will follow a different approach. "Dynamic" keywords will be considered and associated to a user profile. This means, that all keywords of all files will serve as the set of keywords that describe the user's tastes. Doing this, the user is not requested to do any additional steps. Assuming that the user maintains his/her files and folders he/she will also maintain the set of keywords that describe his tastes.

The user profile is created once a day out of all documents that are in a user's Nextcloud. All keywords that are older than a defined day are going to be deleted first. Then, all files are processed in order to create a new user profile. Each keyword is addded only once to a user profile.
The user profile creation is implemented as a second background job. The $UserProfileJob$ class is implemented similarly as the $RecommenderJob$ class. First, the background job needs to registered in the $info.xml$ file:

```
1 <background-jobs>
2 <job>OCA\RecommendationAssistant\BackgroundJob\UserProfileJob</job>
3 </background-jobs>
```

Then, the $TimedJob$ must be implemented, similar to the job described in RecommenderJob:

```
1  class UserProfileJob extends TimedJob {
2
3  public function __construct(UserProfileService $userProfileService) {
4    $this->setInterval(UserProfileJob::INTERVAL);
5    $this->userProfileService = $userProfileService;
6  }
7  protected function run($argument) {
8    $this->userProfileService->run();
9  }
10 }
```

The $UserProfileService$ class is similar to the $RecommenderService$ class. First, the class iterates over all users that have logged in at least once. Then, a list of keywords using the $KeywordList$ class is created for all users. The list scores all keywords using TF-IDF scoring and removes all stopwords. Finally, the relevant keywords are stored into the database.

4.6.2 Reading File Content

The $IContentReader$ interface is responsible for parsing file content of file types (mime types) as described in Reading File Content. Therefore, the following classes are created. Each class gets an instance of $OCP\backslash Files\backslash File$:

- DocxReader

  This class works with the absolute path of the files and not with the $getContent()$ method of the $File$ instance. The absolute path is composed of the user's data directory [8] and the $getPath()$ method of the $File$ instance, which defines the relative path within the data directory.
  Parsing DOCX documents is realized with the $PHPOffice/PHPWord$ library. The library gets the absoulte file path and returns an array of sections, that contains $Text$ instances. This instances contain the plain text, which can be accessed by the $getText()$ method:

```php
public function read(File $file): string {
  $dataDir = Application::getDataDirectory();
  $filePath = $dataDir . "/" . $file->getPath();
  $word = new Word2007();

  ...

  $reader = $word->load($filePath);
  $sections = $reader->getSections();

  $string = "";
  foreach ($sections as $section) {
    foreach ($section->getElements() as $element) {
      if ($element instanceof Text) {
        if (null !== $element->getText()) {
          $string .= $element->getText();
        }
      }
    }
  }
  return $string;
}
```

[8] the data directory is usually the folder named "data" within the Nextcloud root folder if not defined otherwise.

- EmptyReader

  This class simply returns an empty string and is intended for files (mime types) that are not supported yet.

- HTMLReader

  This class returns the file content through the $getContent()$ method of the $File$ instance and removes all HTML tags with the PHP core function $strip\_tags()$.

- JSONReader

  This class encodes the file content, a JSON string, and converts the resulting array into an string. The resulting string is then returned.

- ODSReader

  This class works similar to the $DOCXReader$. The only difference is that an array of $Sheet$ instances is returned by the $load()$ method, which contains all sheets of the document. A $Sheet$ instance contains rows that represents the rows within a sheet.

- ODTReader

  This class works exactly like the $ODSReader$ class.

- PDFReader

  The PDF content is parsed by the $PdfToText$ library. The library gets the absolute file path and the plain text is accessible by the $Text$ attribute. In a final step, all non-ASCII characters are removed by an regular expression[9]:

```
1  public function read(File $file): string {
2    $dataDir = Application::getDataDirectory();
3    $filePath = $dataDir . "/" . $file->getPath();
4    $pdf = new PdfToText($filePath);
5    $text = $pdf->Text;
6    $text = preg_replace('/[^(\x20-\x7F)]*/', '', $text);
7    return $text;
8  }
```

[9] the *preg_replace()* command strips all non-ASCII characters from a string and is originally from: http://hawkee.com/snippet/4224/

- PPTXReader

  The $PPTXReader$ class works similarly to the $DOCXReader$ class. The difference is that an $Reader$ instance contains slides instead of sections. The slides contain shapes that contain the plain text.

- TextFileReader

  This class simply returns the file content that is provided by the $getContent()$ method of the $File$ instance.

- XLSXReader

  This class works exactly as the $ODSReader$ class.

- XMLReader

  The $XMLReader$ class removes all XML tags with the PHP core function $strip\_tags()$ function and returns the resulting string.

4.6.3 Term Frequency / Inverse Document Frequency

The TF-IDF measures the relevance of a single keyword within a document in which it is present and within the whole item base, respectively. As described in Stopword Removal and Stemming the TF-IDF algorithm helps to find stopwords and is highly language independent. Stopword removal enables a better description of the document and makes the Content-Based Recommendation component more effective.

The TF-IDF algorithm is implemented as a PHP class that gets the item and the entire item base injected. The $compute()$ method of the $TFIDFComputer$ class iterates over all keywords of an item. First the $Term\ Frequency$ is calculated by dividing the number of keywords in the document by the total number of keywords. Then, the $Inverse\ Document\ Frequency$ is calculated by the logarithm of the division of the number of documents in the item base by the number of documents that contain the keyword at least once. The TF-IDF value is calculated by the multiplication of the Term Frequency value with the Inverse Document Frequency value:

```php
public function compute() {
  $result = new KeywordList();
  $itemBaseSize = $this->itemList->size();

  /** @var Keyword $keyword */
  foreach ($this->item->getKeywordList() as $keyword) {
    if (trim($keyword->getKeyword()) == "") {
      continue;
    }
    $termFrequency = $this->item->countKeyword($keyword->getKeyword()) /
        $this->item->keywordSize();
    $count = $this->itemList->countKeyword($keyword->getKeyword());

    if ($count == 0) {
      $count = 1;
    }

    $inverseDocumentFrequency = log10($itemBaseSize / $count);
    $tfIdf = $termFrequency * $inverseDocumentFrequency;
    if ($tfIdf < 0) {
      $tfIdf = 0;
    }
    $keyword->setTfIdf($tfIdf);
    $result->add($keyword);
  }
  return $result;
}
```

The method returns an new instance of $KeywordList$ that contains the keywords and the corresponding TF-IDF values. The $KeywordList$ class provides the method $removeStopwords()$ that removes all keywords with a TF-IDF value of 0 or is less than a defined threshold. Removing stopwords is implemented with the PHP core function $array\_filter()$ that accepts a callback function which defines the rule for elements that remain in the array:

```php
public function removeStopwords() {
  $this->keywordList = array_filter($this->keywordList,
    function (Keyword $item, string $keyword) {
      $precision = 10;
      $floatVal = floatval($item->getTfIdf());
      $zero = round(0, $precision, PHP_ROUND_HALF_EVEN);
      $tfidf = round($floatVal, $precision, PHP_ROUND_HALF_EVEN);
      $maxTfIdfThreshold = round($this->maxTfIdf * Application::
          STOPWORD_REMOVAL_PERCENTAGE, $precision, PHP_ROUND_HALF_EVEN);
      return ($tfidf > $zero) || ($tfidf > $maxTfIdfThreshold);
    }, ARRAY_FILTER_USE_BOTH);
}
```

4.6.4 OverlapCoefficientComputer

The $OverlapCoefficientComputer$ uses the Overlap Coefficient algorithm and implements the $IComputable$ interface. The class gets two instances of $Item$ and a instance of $KeywordList$ (the keywords in a user profile) through the constructor injected.
Since the class implements $IComputable$, it is required to implement the $compute()$ method. The method first removes all stopwords as described in Term Frequency / Inverse Document Frequency. Then, the number of common and total keywords of the item and the user profile are determined. The smaller number of total keywords is then used below as the denominator. The similarity depends on the numerator and denominators value:

- numerator equals to 0: there are no overlapping keywords found and the similarity equals to 0.
- denominator equals to 0: the user profile or document has no keywords and the similarity is 0.
- numerator and denominator are greater than 1: similarity value is the result of dividing numerator by denominator[10].

```
1  public function compute(): Similarity {
2    $similarity = new Similarity();
3    $tfIdf = new TFIDFComputer($this->item, $this->itemList);
4
5    /** @var KeywordList $itemKeywords */
6    $itemKeywords = $tfIdf->compute();
7    $itemKeywords->removeStopwords();
8
9    $arr = array_intersect($itemKeywords->getKeywords(), $this->
         keywordList->getKeywords());
10   $arr = array_unique($arr);
11   $numerator = count($arr);
12
13   $denominator = $itemKeywords->size() > $this->keywordList->size() ? $
         this->keywordList->size() : $itemKeywords->size();
14
15   if ($numerator == 0) {
16     $similarity = Util::createSimilarity(0.0, Similarity::
           NO_OVERLAPPING_KEYWORDS, "no overlapping keywords found");
17   }
18
19   if ($denominator == 0) {
20     $similarity = Util::createSimilarity(0.0, Similarity::
           ITEM_OR_USER_PROFILE_EMPTY, "no keywords in item / user profile")
           ;
21   }
22
```

[10] the factor variable is necessary for rating ranges greater than 0. More information in chapter Evaluation.

```
23    if ($numerator > 0&& $denominator > 0) {
24      $factor = Rater::RATING_UPPER_LIMIT;
25      $similarity = Util::createSimilarity(($numerator / $denominator) * $
            factor, Similarity::VALID, "valid calculation");
26    }
27    return $similarity;
28  }
```

4.7 Hybridization

After calculating the similarity values using Collaborative Filtering and Content-Based Recommendation algorithms, it is required to combine these measures in order to have a "final" similarity value. This value decides whether the file should be recommended or not. Different approaches were introduced in Hybridization.
The main challenge in dealing with the hybridization is the fact that the components are calculated independently from each other. More technically, all items are first used in a foreach loop to calculate the Collaborative Filtering component. A second foreach loop is then used again to iterate over all items and calculate the Content-Based Recommendation component.

4.7.1 HybridItem

The $HybridItem$ is a simple getter and setter class as described in HybridItem. The class implements a static method that calculates the weighted average:

```
1  public static function weightedAverage(HybridItem $hybrid): float {
2    $contentBasedResult = self::$contentBased * $hybrid->getContentBased()
         ->getValue();
3    $collaborativeResult = self::$collaborative * $hybrid->
         getCollaborative()->getValue();
4    $weightedAverage = $hybrid->getGroupWeight() * ($contentBasedResult +
         $collaborativeResult);
5    return $weightedAverage;
6
7  }
```

The calculation of the weighted average, which is a key part of the hybridization, is so simple, that the decision was made on implementing it as a static class method. The function first defines the weights of each component and returns the weighted average afterwards.

Weighting per Group

Nextcloud users are organized in groups. For example, a company's Nextcloud instance can structure groups according to their departments. There can be a Research and Development, Sales, Marketing and Engineering departments.
A Sales user is not necessarily interested in documents from users who are membered to Engineering, for example. Therefore, a simple matrix is implemented that defines group related weights. The matrix defines weights for each group pair. Considering that a user can be a member of several groups, the "final" weight has to be a average of all group pairs:

	G1	**G2**	**G3**
G1	1	0.2	0.7
G2	0.2	1	0.4
G3	0.7	0.4	1

For example, if a file that is owned by user is in G1 should be recommended to a user who is in G2 and G3, the group weight would be calculated as:

$$\frac{0.2 + 0.7 + 0.4 + 1}{4} = 0.575 \tag{4.1}$$

This weight can be multiplied with the hybrid recommendation result described in the previous chapter.

Recommendation Transparency

In most recommendation systems, users get recommendations without knowing why. Even if it is not easy to define the "why", since there are many calculation steps to do, a basic explanation should be provided to the user. Therefore, the app defines three basic reasons why the user gets an item recommended:

1. Collaborative Filtering similarity value is greater than Content-Based Recommendation value,
2. Content-Based Recommendation value is greater than Collaborative Filtering value,
3. Collaborative Filtering value is equal to Content-Based Recommendation value.

The app inserts a transparency code into the appropriate database. This code is then retrieved by the frontend in order to display the corresponding explanation.

```
1   public function getTransparencyCode(): int {
2   if (NumberUtil::compareFloat($this->collaborative->getValue(), $this->
        contentBased->getValue())) {
3     return self::TRANSPARENCY_BOTH;
4   }
5   if (NumberUtil::floatGreaterThan($this->collaborative->getValue(), $
        this->contentBased->getValue())) {
6     return self::TRANSPARENCY_COLLABORATIVE;
7   }
8   if (NumberUtil::floatGreaterThan($this->contentBased->getValue(), $
        this->collaborative->getValue())) {
9     return self::TRANSPARENCY_CONTENT_BASED;
10  }
11  return self::TRANSPARENCY_BOTH;
12 }
```

4.7.2 HybridList

The $HybridList$ class is similar to $ItemList$ or $KeywordList$ with a small extension. The class implements the $IteratorAggregate$ interface provided by the PHP core in order to make the class iterable. The $add()$ method adds a $HybridItem$ to the list. The list is a two dimensional array that has the user id and the item id as their indices.

```
1   public function add(HybridItem $hybrid, IUser $user, Item $item) {
2   $this->hybridList[$user->getUID()][$item->getId()] = $hybrid;
3  }
```

The $getHybridByUser()$ method searches for a $HybridItem$ in a list for a given user and item. In any case the method returns an instance of $HybridItem$. The instance is returned from the list if it is available. If not, the method creates a new instance without any values and returns it.

```
1   public function getHybridByUser(Item $item, IUser $user) {
2   if (isset($this->hybridList[$user->getUID()][$item->getId()])) {
3     return $this->hybridList[$user->getUID()][$item->getId()];
4   } else {
5     return new HybridItem();
6   }
7  }
```

Doing this allows avoiding the problem mentioned above that both components are calculated in different loops. Each loop requests a $HybridItem$ from the list. The loop does not have

to consider the right instance or value association. Simply adding to the list after setting the necessary values is enough. The next loop may then work with the instances that are already present in the list:

```
1  ...
2  foreach($itemList as $item){
3    $hybrid = $hybridList->getHybridByUser($item, $user);
4    ...
5    $hybrid->setContentBased($sim);
6    $hybrid->setItem($item);
7    $hybrid->setUser($user);
8    $hybridList->add($hybrid, $user, $item);
9  }
```

4.8 Database Storage

As mentioned several times in this chapter a database is used to store some data. Since Nextcloud supports multiple database types like MySQL, MariaDB, PostgreSQL, SQLite and even Oracle Database, it is required to use an abstraction layer in order to access or store the data. This section should shortly introduce the way how Nextcloud abstracts the database access before the essential storage and data are describe in detail.

4.8.1 Doctrine Framework

Because every database storage works differently, it is required to have an abstraction layer that makes the business code independent from the way how and where data is stored. This is what the Doctrine Framework provides: a persistence layer and related functionality in order to support different database engines. One of the key functions of Doctrine is DQL (Doctrine Query Language). DQL is a object oriented dialect of SQL (Structured Query Language)[DOC].

4.8.2 Table Creation and Access

To avoid multiple analysis of a file, it is necessary to store the information that the file is already processed. Thus, the "files_processed" database table is created. The table stores the file id, the keyword and the corresponding TF-IDF.
The database creation is completly abstracted from the SQL layer and is realized with PHP code. Nextcloud goes one step further and has a "Migration Wizard" that helps in creation of databases/tables for different Nextcloud versions.
To change a database table or create a new one it is necessary to create a class that implements the $IMigrationStep$ interface, which provides the $changeSchema()$ method. The class name has to contain a version number and a timestamp of the actual version. This timestamp is then

used to check whether an upgrade is available or not. To perform the database upgrade it is enough to increment the apps version number in $info.xml$. Nextcloud will do the remaining work.

```
1   public function changeSchema(IOutput $output, \Closure $schemaClosure,
        array $options) {
2   /** @var Schema $schema */
3   $schema = $schemaClosure();
4
5   if (!$schema->hasTable(DBConstants::TABLE_NAME_FILES_PROCESSED)) {
6     $table = $schema->createTable(DBConstants::TABLE_NAME_FILES_PROCESSED
          );
7     $table->addColumn(DBConstants::ID, Type::BIGINT, [
8       'autoincrement' => true,
9       'notnull' => true,
10      'length' => 20,
11    ]);
12    $table->addColumn(DBConstants::CREATION_TS, Type::INTEGER, [
13      'notnull' => true,
14      'length' => 4,
15      'default' => 0,
16    ]);
17    $table->addColumn(DBConstants::FILE_ID, Type::INTEGER, [
18      'notnull' => true,
19      'length' => 4,
20      'default' => 0
21    ]);
22    $table->setPrimaryKey([DBConstants::ID]);
23  }
24  return $schema;
25 }
```

Database Access

After creating the database table, which is a one time job, an access layer is necessary in order to read/write into the database table. This layer is created by classes that get a $IDBConnection$ instance injected.
The $IDBConnection$ instance provides the $getQueryBuilder()$ method that returns an instance of $QueryBuilder$. This instance allows the creation of a database query using DQL as shown in the following listing:

```php
private function insertKeyword(Keyword $keyword, IUser $user): bool {
  if ($this->isPresentable($keyword, $user)) {
    return true;
  }
  $query = $this->dbConnection->getQueryBuilder();
  $query->insert(DbConstants::TABLE_NAME_USER_PROFILE)->values(
    [
      DbConstants::TB_UP_USER_ID => $query->createNamedParameter($user->
          getUID()),
      DbConstants::TB_UP_KEYWORD => $query->createNamedParameter(($
          keyword->getKeyword())),
      DbConstants::TB_UP_CREATION_TS => $query->createNamedParameter((
          time())),
      DbConstants::TB_UP_TFIDF_VALUE => $query->createNamedParameter($
          keyword->getTfIdf())
    ]
  );
  try {
    $query->execute();
  } catch (\Exception $exception) {
    ConsoleLogger::debug($exception->getMessage());
    return false;
  }
  $lastInsertId = $query->getLastInsertId();
  return is_int($lastInsertId);
}
```

The $QueryBuilder$ instance provides the $insert()$ and $values()$ methods in order to define the table name and the field names and values. The $createNamedParamater()$ method is responsible to avoid SQL injections and ensure that all values are escaped with quotation marks. Finally, the *execute* method executes the query.
The $getLastInsertId()$ method returns the last inserted auto increment value that is the primary key of the table. If the query was not successfull the method returns $null$. This way it is simple to prove by verifying that the $lastInsertId$ variable is an integer.

The $QueryBuilder$ also provides methods to delete and select values. To delete a row in a database it is necessary to create an expression that defines the conditions about affected rows:

```
1  public function deleteOldKeywords(int $days, IUser $user) {
2    $dateTime = new \DateTime();
3    $dateTime->modify("-$days day");
4    $query = $this->dbConnection->getQueryBuilder();
5    $query->delete(DbConstants::TABLE_NAME_USER_PROFILE)
6      ->where($query->expr()->lte(DbConstants::TB_UP_CREATION_TS, $query->
           createNamedParameter($dateTime->getTimestamp())))
7      ->andWhere($query->expr()->eq(DbConstants::TB_UP_USER_ID, $query->
           createNamedParameter($user->getUID())))
8      ->execute();
9  }
```

The $delete()$ method accepts the table name and the $where()$ method accepts an expression that is defined as a $equals$ condition in the sample above. The $eq()$ method acceppts the field name and the fields value that should be deleted from the table.

The $select()$ method works similar to $delete()$. The only difference is that $select()$ returns a set of rows that have to be fetched and represented in a preferred way:

```
1  public function getKeywordListByUser(IUser $user) {
2    $keywordList = new KeywordList();
3    $query = $this->dbConnection->getQueryBuilder();
4    $query->select(DbConstants::TB_UP_KEYWORD, DbConstants::
         TB_UP_TFIDF_VALUE)
5      ->from(DbConstants::TABLE_NAME_USER_PROFILE)
6      ->where($query->expr()->eq(DbConstants::TB_UP_USER_ID, $query->
           createNamedParameter($user->getUID())));
7
8    $result = $query->execute();
9    while (false !== $row = $result->fetch()) {
10     $keyword = new Keyword();
11     $keyword->setKeyword($row[DbConstants::TB_UP_KEYWORD]);
12     $keyword->setTfIdf($row[DbConstants::TB_UP_TFIDF_VALUE]);
13     $keywordList->add($keyword);
14   }
15   $result->closeCursor();
16   return $keywordList;
17 }
```

Other Tables

There are several other tables that are necessary for this app. The way how these tables are created and accessed are the same as described above. This is why it is not regarded necessary to explain them in detail as well.

Chapter 5

Evaluation

5.1 Introduction

Wherears the preceding chapter Implementation discussed the implementation of the proposed recommendation system, this chapter evaluates the expected results for users. Since there is a limited access to test data, it is not possible to evaluate every test case in detail. For this purpose a minimalistic test with a set of files and users was created. The test is set up with fictional users and randomly chosen real life files.

Starting with Test Environment, the used tools, Nextcloud version and test environment in general is described. Section Evaluation Based on Static Tags discusses results of static file tags. These tags are "favorites" that are given to a file by a user. This approach is more static because it relies only on the fact that a file is tagged and does not consider when tagging happend, for example.

This approach is developed on a git branch named "recommendation_based_on_file_ratings". Section Evaluation Based on Modifications Timestamps is developed on the master branch and is more dynamic. The user ratings are assembled by modification and tagging timestamps and are spread over a range between 0 and 5.

In chapter State of the Art are different known problems like the "Sparsity Problem", "Cold-Start Problem", "Early Rater Problem" and the "Gray Sheep Problem" discussed. Unfortunately, chapter Evaluation has not addressed all these problems due to the lack of data. The "Sparsity Problem" addresses a subset of rated items within a large item base. The "Cold Start Problem" is defined as users or items which are newly added to the system and have no ratings yet. "Early Rater Problem" applies also for users or items which are newly added but have a few ratings where Collaborative Filtering results in inaccurate recommendations and the "Gray Sheep Problem" addresses people with opinions that do not consistently match with other users. These problems require a large user and item base and the consideration of special circumstances. For example, "Gray Sheep Problem" requires at least one user within a user

base with ratings that slightly overlap with the ratings of other users.
Content-Based Recommendation has also known limitations which are discussed in chapter State of the Art. Content-Based Recommendation does not distinguish between "good" and "bad" content, which also applies to this work. Furthermore, two different items can be described by (nearly) the same keywords after stopword removal and TF-IDF weighting which leads to inaccuracy. The content-based recommendation will be ineffective when the item base contains too much items and thus, the overlap between the user profile and items is to much. Multimedia content, such as audio or video, were not addressed within this work and represent a challenge in their own. The evaluation has also shown that the content-based recommendation is inaccurate if the number of article and user profile keywords differs greatly from each other. The sections below describe the approaches stated above in detail and show the results. These results are compared to those of the app. This will provide valid information about the apps reliability.

5.1.1 Test Environment

The test environment is set up on a local machine. The evaluation is based on Nextcloud 13 Beta 4 which has been cloned from GitHub on 5 January 2018.
The instance has five registered users where each user has five files that contain news from several online newspapers and belong to different categories. It can be assumed that the users are using Nextcloud regularly, meaning that they maintain their files and have logged in at least once. The files - more precisely their content - represent their profile and keywords for Content-Based Recommendation. Tagging a file as a favorite is necessary for Collaboaritve Filtering since it represents the rating of a file from a user.
The users are Brian (interested in computer science), John (soccer), Luke (politics), Robert (economy) and Tom (health). The following structure results:

Brian	John	Luke	Robert	Tom
Apple buys app development service Buddybuild[CBRa]	Thibaut Courtois close to signing new Chelsea deal[CBRb]	Trump lawyer seeks to block insider book on White House[CBRc]	Apple leads race to become world's first $1tn company[CBRd]	Patients in Africa twice as likely to die after an operation than global average, report shows [CBRe]
Twitter ended the year on a fascinating run[CBRf]	The FA Cup is where teams are afforded the chance to dream[CBRg]	What in the world was Stephen Bannon thinking? 3 theories.[CBRh]	Bankers work around the clock to iron out EU finance reforms[CBRi]	The Guardian view on the NHS winter crisis: not such a happy birthday [CBRj]
Instagram tests letting users post Stories directly to WhatsApp[CBRk]	Arsène Wenger's referee paranoia will chip away at Arsenal's conviction[CBRl]	Ukraine: killing of lawyer sparks protests against 'criminal system'[CBRm]	Good for factories, bad for shoppers: a Brexit pattern is emerging[CBRn]	US drug firm offers cure for blindness – at $425,000 an eye[CBRo]
Apple readies Siri for the HomePod by adding a podcast-powered news brief[CBRp]	Jürgen Klopp says Philippe Coutinho will return for Manchester City game[CBRq]	What we've learned about Trump's campaign and Russia since Trump first denied collusion[CBRr]	Trump tax cut to dent BP profits by $1.5bn, company warns[CBRs]	Alcohol can cause irreversible genetic damage to stem cells, says study [CBRt]
Apple Developer Program fee waivers are now available for nonprofits, schools and government[CBRu]	Juventus confident of signing Emre Can for free after offering five-year contract[CBRv]	Trump administration seeks $18bn from Congress for Mexico border wall[CBRw]	Hammond relying on household debt to hit targets, says McDonnell [CBRx]	Doctors and patients accuse government of failing to stop NHS crisis[CBRy]

Table 5.1: User-Files Matrix

For simplicity, the files are sequentially numbered and prefixed with a "D" and are mentioned by their numbers from top to down. The evaluation uses only D1 to D5 for a better overview.

5.2 Evaluation Based on Static Tags

This section explains the results of Content-Based Recommendation and Collaborative Filtering based on "static favorite tags". As explained in User Ratings, Nextcloud provides file tagging functionality. Special kind of file tags are "favorites" that are used by users to express a special interest in this file. The user ratings are binary, meaning that tagging a file as favorite corresponds to 1, otherwise to 0.

Content-Based Recommendation

The user profile keywords of each user are calculated out of all files that the user has in his profile. It can be assumed that there no other files than listed in figure User-Files Matrix. After removing stopwords, Brian has 829, John 674, Luke 1106, Robert 774 and Tom 919 keywords in their user profile.
The files D1 to D5 have the following number of keywords: D1 = 63, D2 = 418, D3 = 461, D4 = 179 and D5 = 410.

The intersection of both sets (user profile and document keywords) according to the $Overlap\ Coefficient$ described in OverlapCoefficientComputer calcuation results in the following:

	John	Luke	Robert	Tom
D1	11	11	14	10
D2	75	84	73	69
D3	79	78	62	72
D4	38	34	28	31
D5	58	70	64	55

Table 5.2: Number of Overlapping Keywords

The calculation steps are as follows:

D4 and Luke:

$$M = \frac{|D4 \cap P_{Luke}|}{min(|D4|, |P_{Luke}|)} = \frac{|179 \cap 1106|}{min(|179|, |1106|)} = \frac{34}{179} = 0,190 \tag{5.1}$$

D4 and Robert:

$$M = \frac{|D4 \cap P_{Robert}|}{min(|D4|, |P_{Robert}|)} = \frac{|179 \cap 774|}{min(|179|, |774|)} = \frac{28}{179} = 0,156 \tag{5.2}$$

D5 and Tom:

$$M = \frac{|D5 \cap P_{Robert}|}{min(|D5|, |P_{Robert}|)} = \frac{|410 \cap 919|}{min(|410|, |919|)} = \frac{55}{410} = 0,134 \tag{5.3}$$

After applying the $Overlap\ Coefficient$ to all files and users, the following table results[1]:

	John	Luke	Robert	Tom
D1	0,17	0,17	0,22	0,16
D2	0,18	0,20	0,17	0,17
D3	0,17	0,17	0,13	0,16
D4	0,21	0,19	0,16	0,17
D5	0,14	0,17	0,16	0,13

Table 5.3: Overlap Coefficient Results

As shown in table Overlap Coefficient Results, the calculation results in poor similarity for each item and user. This is due to the fact that the user profiles have significantly more keywords than the document and therefore the overlap is very small.

Collaborative Filtering

The Collaborative Filtering part of the recommendation system assumes that the "tag as favorite" function of Nextcloud is used constantly. Therefore, the following table is assumed:

	D1	D2	D3	D4	D5
Brian	0	0	1	0	1
John	1	1	0	0	0
Luke	0	1	0	?	1
Robert	0	1	1	?	0
Tom	1	1	1	0	?

Table 5.4: User Ratings for D1 to D5

In User Ratings for D1 to D5, 1 corresponds to "tagged as favorite" and 0 to "not tagged as favorite". ? means "is the file recommendable to the user".

First, it is necessary to calculate the similarity of the files using $Cosine\ Similarity$ described as in Similarity Function. The following will demonstrate this calculation:

[1]Brian is not involved in the calculation because he is the owner of the files D1 to D5 and should not get his own files recommended.

The similarity of D1 and D2 is calculated as follows:

$$sim(D1, D2) = \frac{(0*0)+(1*1)+(0*1)+(0*1)+(1*1)}{\sqrt{(0^2+1^2+0^2+0^2+1^2)} * \sqrt{(0^2+1^2+1^2+1^2+1^2)}} = \frac{2}{\sqrt{2}*\sqrt{4}} = \frac{2}{1.41*2} = 0,709 \tag{5.4}$$

For D1 and D3:

$$sim(D1, D3) = \frac{(0*1)+(1*0)+(0*0)+(0*1)+(1*1)}{\sqrt{(0^2+1^2+0^2+0^2+1^2)} * \sqrt{(1^2+0^2+0^2+1^2+1^2)}} = \frac{1}{\sqrt{2}*\sqrt{3}} = \frac{0}{1,41*1,73} = \frac{1}{2,44} = 0,41 \tag{5.5}$$

For D1 and D4:

$$sim(D1, D4) = \frac{(0*0)+(1*0)+(0*0)+(0*0)+(1*0)}{\sqrt{(0^2+1^2+0^2+0^2+1^2)} * \sqrt{(0^2+0^2+0^2+0^2+0^2)}} = \frac{0}{\sqrt{2}*\sqrt{0}} = 0 \tag{5.6}$$

For D1 and D5:

$$sim(D1, D5) = \frac{(0*1)+(1*0)+(0*1)+(0*0)+(1*0)}{\sqrt{(0^2+1^2+0^2+0^2+1^2)} * \sqrt{(1^2+0^2+1^2+0^2+0^2)}} = \frac{0}{\sqrt{2}*\sqrt{2}} = \frac{0}{2*2} = \frac{0}{4} = 0 \tag{5.7}$$

This calculation is applied to all files. After applying *Cosine Similarity* to D1 to D5, the following table results:

	D1	**D2**	**D3**	**D4**	**D5**
D1	1	0,71	0,41	0	0
D2	0,71	1	0,58	0	0,35
D3	0,41	0,58	1	0	0,41
D4	0	0	0	1	0
D5	0	0,35	0,41	0	1

Table 5.5: User Ratings for D1 to D5

Next, the prediction for Luke and Robert for D4 and Tom for D5 can be calculated using the weighted average as described in Similarity Function. The following results:

Prediction for D4 to Luke:

$$P_{D_4,Luke} = \frac{(0*0)+(0*1)+(0*0)+(0*1)}{|0+0+0+0|} = \frac{0}{0} = 0 \tag{5.8}$$

Prediction for D4 to Robert:

$$P_{D_4,Robert} = \frac{(0*0)+(0*1)+(0*1)+(0*0)}{|0+0+0+0|} = \frac{0}{0} = 0 \tag{5.9}$$

Prediction for D5 to Tom:

$$P_{D_5,Tom} = \frac{(0*1)+(0,35*1)+(0,41*1)+(0*0)}{|0+0,35+0,41+0|} = \frac{0,76}{0,76} = 1 \tag{5.10}$$

Hybridization

After computing the Content-Based Recommendation and Collaborative Filtering components, the hybridization step has to be made. Both values are combined to hybrid recommendation using the weighted average as described in Weighted Average. It can be assumed that both, Content-Based Recommendation and Collaborative Filtering, are weighted with 0.5 each. In addition to that a final recommendation is only made if the result exceeds a threshold. It can also be assumed that the group weights described in Weighting per Group are all equal to 1 in order to keep a better overview. Because multiplying with 1 does not change the result, the calculation is ignored in the following examples.

Content-Based Recommendation has calculated a similarity value of 0,19 for Lukes user profile and D4. Collaborative Filtering predicts that Luke won't rate D4. Therefore, the calcuation is as follows:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,19) + (0.5 * 0) = 0,095 \tag{5.11}$$

The same calculation is made for D4 and Robert:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,16) + (0.5 * 0) = 0,080 \tag{5.12}$$

D5 and Tom:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,13) + (0.5 * 1) = 0,565 \tag{5.13}$$

The hybridization step has to decide whether the file is going to be recommended to a user or not. Therefore, a threshold has to be chosen in order to make this decision. Assuming that the threshold is 0.5, only D5 would be recommended to Tom.

5.3 Evaluation Based on Modifications Timestamps

This section explains the results of Content-Based Recommendation and Collaborative Filtering based on modification timestamps. The user ratings are assembled by last modification and tagging timestamps and spread over a range between 0 and 5 as described in User Ratings. The following evaluation works with the same files as described in Evaluation Based on Static Tags.

5.3.1 Content Based Recommendation

Content Based Recommendation works similar as described in Evaluation Based on Static Tags except one deviation. Since the $Overlap\ Coefficient$ measures the similarity of two items in the range of 0 and 1, but $Cosine\ Computer$ as demonstrated below can compute over greater ranges (in this case 0 and 5), it is required to adjust the range in order to have the same scale. Therefore, a $factor$ is introduced and set to 5, which is the highest possible rating for an item in this evaluation approach.

Content Based Recommendation for D2 and Tom is calculated as follows:

$$M = \frac{|D2 \cap P_{Tom}|}{min(|D2|, |P_{Tom}|)} * 5 = 0,17 * 5 = 0,85 \tag{5.14}$$

D2 and Robert:

$$M = \frac{|D2 \cap P_{Robert}|}{min(|D2|, |P_{Robert}|)} * 5 = 0,17 * 5 = 0,85 \tag{5.15}$$

D3 and Luke:

$$M = \frac{|D3 \cap P_{Luke}|}{min(|D2|, |P_{Robert}|)} * 5 = 0,17 * 5 = 0,85 \tag{5.16}$$

D3 and Robert:

$$M = \frac{|D3 \cap P_{Robert}|}{min(|D2|, |P_{Robert}|)} * 5 = 0,13 * 5 = 0,65 \tag{5.17}$$

D4 and Luke:

$$M = \frac{|D4 \cap P_{Robert}|}{min(|D2|, |P_{Robert}|)} * 5 = 0,19 * 5 = 0,95 \tag{5.18}$$

D4 and Robert:

$$M = \frac{|D4 \cap P_{Robert}|}{min(|D2|, |P_{Robert}|)} * 5 = 0,16 * 5 = 0,80 \tag{5.19}$$

D4 and Tom:

$$M = \frac{|D4 \cap P_{Robert}|}{min(|D2|, |P_{Robert}|)} * 5 = 0,17 * 5 = 0,85 \tag{5.20}$$

D5 and Tom:

$$M = \frac{|D5 \cap P_{Robert}|}{min(|D2|, |P_{Robert}|)} * 5 = 0,13 * 5 = 0,65 \tag{5.21}$$

After applying the *Overlap Coefficient* to all files and users, the following results[2]:

	John	**Luke**	**Robert**	**Tom**
D1	0,85	0,85	1,10	0,80
D2	0,90	1,00	0,85	0,85
D3	0,85	0,85	0,65	0,80
D4	1,05	0,95	0,80	0,85
D5	0,70	0,85	0,80	0,65

Table 5.6: Overlap Coefficient Computer Results

As shown in Evaluation Based on Static Tags, the Content-Based Recommendation calculation for this approach results in poor similarity for each item and user as well. This is also due to the fact that the keyword ratio of user profile and document is disproportionately.

5.3.2 Collaborative Filtering

The Collaborative Filtering part of the recommendation system uses modification and favorite tagging timestamps in order to define ratings. The rating range is between 0 and 5 as described in User Ratings. The ratings are as follows:

	D1	**D2**	**D3**	**D4**	**D5**
Brian	1	5	4	3	2
John	2	4	4	3	0
Luke	1	3	0	2	1
Robert	5	0	0	2	5
Tom	4	0	2	0	3

Table 5.7: User Ratings for D1 to D5 (2)

The user ratings are spread over a range of 0 to 5 where 0 correspondents to "no rating" and 5 to the highest possible rating. The Collaborative Filtering part predicts ratings for user-item

[2]notice that Brian is not involved in the calculation because he is the owner of the files D1 to D5

tupels that have no ratings. These tuples are highlighted in User Ratings for D1 to D5 (2) with red color.

The similarity for D1/D2, D1/D3, D1/D4 and D1/D5 using $Cosine\ Similarity$ is calculated as follows:

$$sim(D1, D2) = \frac{(1*5)+(2*4)+(1*3)+(5*0)+(4*0)}{\sqrt{(1^2+2^2+1^2+5^2+4^2)} * \sqrt{(5^2+4^2+3^2+0^2+0^2)}} = \frac{16}{\sqrt{47} * \sqrt{50}} = \frac{16}{6,86 * 7,07} = \frac{16}{48,50} = 0,33 \tag{5.22}$$

$$sim(D1, D3) = \frac{(1*4)+(2*4)+(1*0)+(5*0)+(4*2)}{\sqrt{(1^2+2^2+1^2+5^2+4^2)} * \sqrt{(4^2+4^2+0^2+0^2+2^2)}} = \frac{20}{\sqrt{47} * \sqrt{36}} = \frac{20}{6,86 * 6} = \frac{20}{41,16} = 0,49 \tag{5.23}$$

$$sim(D1, D4) = \frac{(1*3)+(2*3)+(1*2)+(5*2)+(4*0)}{\sqrt{(1^2+2^2+1^2+5^2+4^2)} * \sqrt{(3^2+3^2+2^2+2^2+0^2)}} = \frac{21}{\sqrt{47} * \sqrt{26}} = \frac{21}{6,86 * 5,10} = \frac{21}{34,94} = 0,60 \tag{5.24}$$

$$sim(D1, D5) = \frac{(1*2)+(2*0)+(1*1)+(5*5)+(4*3)}{\sqrt{(1^2+2^2+1^2+5^2+4^2)} * \sqrt{(2^2+0^2+1^2+5^2+3^2)}} = \frac{40}{\sqrt{47} * \sqrt{39}} = \frac{40}{6,86 * 6,24} = \frac{40}{42,81} = 0,93 \tag{5.25}$$

The calculation is applied to all files and the resulting similarities are as follows:

	D1	**D2**	**D3**	**D4**	**D5**
D1	1	0,33	0,49	0,60	0,93
D2	0,33	1	0,85	0,92	0,29
D3	0,49	0,85	1	0,78	0,37
D4	0,60	0,92	0,78	1	0,57
D5	0,93	0,29	0,37	0,57	1

Table 5.8: User Ratings for D1 to D5 (2)

Next, the prediction for Luke and Robert for D4 and Tom for D5 can be calculated using the weighted average as described in Similarity Function. The following results:

Prediction for D2 to Tom:

$$P_{D_2,Tom} = \frac{(0,33 * 4) + (0,85 * 2) + (0,92 * 0) + (0,29 * 3)}{|0,33 + 0,85 + 0,92 + 0,29|} = \frac{3,89}{2,39} = 1,63 \tag{5.26}$$

Prediction for D2 to Robert:

$$P_{D_2,Robert} = \frac{(0,33 * 5) + (0,85 * 0) + (0,92 * 2) + (0,29 * 5)}{|0,35 + 0,84 + 0,91 + 0,29|} = \frac{4,94}{2,39} = 2,07 \tag{5.27}$$

Prediction for D3 to Luke:

$$P_{D_3,Luke} = \frac{(0,49 * 1) + (0,84 * 3) + (0,78 * 2) + (0,37 * 1)}{|0,51 + 0,84 + 0,78 + 0,37|} = \frac{4,96}{2,5} = 1,98 \tag{5.28}$$

Prediction for D3 to Robert:

$$P_{D_3,Robert} = \frac{(0,51 * 5) + (0,85 * 0) + (0,78 * 2) + (0,37 * 5)}{|0,51 + 0,84 + 0,78 + 0,37|} = \frac{5,96}{2,5} = 2,38 \tag{5.29}$$

Prediction for D4 to Luke:

$$P_{D_4,Luke} = \frac{(0,60 * 1) + (0,92 * 3) + (0,78 * 0) + (0,55 * 1)}{|0,59 + 0,91 + 0,78 + 0,55|} = \frac{3,87}{2,83} = 1,36 \tag{5.30}$$

Prediction for D4 to Robert:

$$P_{D_4,Robert} = \frac{(0,59 * 5) + (0,91 * 0) + (0,78 * 0) + (0,57 * 5)}{|0,59 + 0,91 + 0,78 + 0,57|} = \frac{5,80}{2,85} = 2,04 \tag{5.31}$$

Prediction for D4 to Tom:

$$P_{D_4,Tom} = \frac{(0,59 * 4) + (0,91 * 0) + (0,78 * 2) + (0,57 * 3)}{|0,51 + 0,84 + 0,78 + 0,37|} = \frac{5,63}{2,85} = 1,98 \tag{5.32}$$

Prediction for D5 to Tom:

$$P_{D_5,Tom} = \frac{(0,93 * 4) + (0,29 * 0) + (0,37 * 2) + (0,57 * 3)}{|0,93 + 0,29 + 0,37 + 0,55|} = \frac{6,17}{2,14} = 2,87 \tag{5.33}$$

Hybridization

Hybrid recommendation using the weighted average as described in Weighted Average is applied to all items and users in order to have a final recommendation. Content-Based Recommendation and Collaborative Filtering are weighted equally with 0.5. It is assumed that the recommendation is made if the result exceeds a given threshold and the group weights are all equal to 1 in order to keep a better overview. Because multiplying with 1 does not change the result, the calculation

is ignored in the following examples.
Content-Based Recommendation has calculated a similarity value of 0,85 for Toms user profile and D2. Collaborative Filtering predicts that Tom would rate D2 with 1,63. Therefore, the calcuation is as follows:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,85) + (0.5 * 1,63) = 0,425 + 0,815 = 1,24 \quad (5.34)$$

D2 and Robert:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,85) + (0.5 * 2,07) = 0,425 + 1,035 = 1,46 \quad (5.35)$$

D3 and Luke:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,85) + (0.5 * 1,98) = 0,425 + 0,99 = 1,415 \quad (5.36)$$

D3 and Robert:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,65) + (0.5 * 2,38) = 0,325 + 1,19 = 1,515 \quad (5.37)$$

D4 and Luke:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,95) + (0.5 * 1,36) = 0,475 + 0,68 = 1,155 \quad (5.38)$$

D4 and Robert:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,80) + (0.5 * 2,04) = 0,40 + 1,02 = 1,42 \quad (5.39)$$

D4 and Tom:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,85) + (0.5 * 1,98) = 0,425 + 0,99 = 1,415 \quad (5.40)$$

D5 and Tom:

$$(w_{cbr} * value_{cbr}) + (w_{cf} * value_{cf}) = (0,5 * 0,65) + (0.5 * 2,87) = 0,325 + 1,435 = 1,76 \quad (5.41)$$

The hybridization result depends on a threshold that has to be exceeded in order to get recommended to an user. For example, if the threshold would be 1 for the examples above, all items would be recommended to the users except of example 5.39. The items 5.37, 5.38 and 5.41 would be recommended if the threshold would be set to 1,5.

5.4 Graphical User Interface

Creating recommendations and storing them into a data storage was the main part of the resulting app. The most important part from the perspective of usability is the way how providing the information to the user.

This question was discussed in several meetings with Nextcloud core developers as well as the managing director. The decision was made on a grid view[3] in order to show the recommendations. The grids are only visible in the root folder and are limited to three recommendations. The single items are clickable which opens the documents.

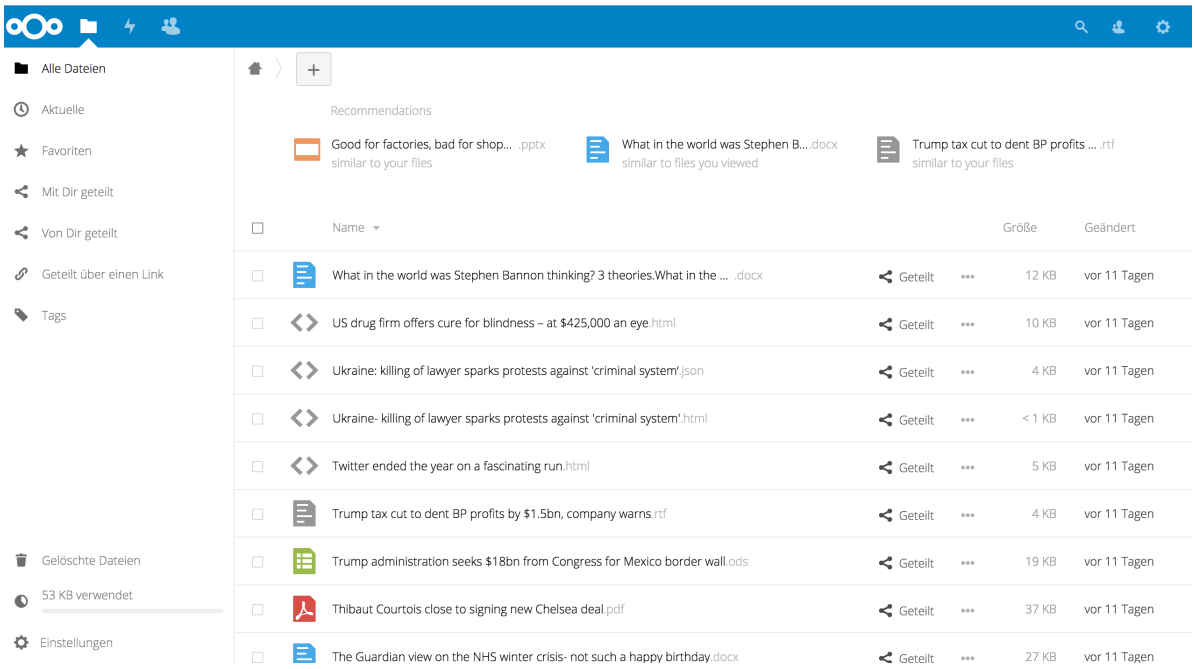

Figure 5.1: Nextcloud Root Folder with three recommendations

5.5 Defining Weights and Thresholds

In chapter CosineComputer, Term Frequency / Inverse Document Frequency and before inserting recommendations into the appropriate database there are three different thresholds to exceed. The first threshold removes all items from the list which are not regarded as similar. This way only the most similar items are used to make a prediction to a user. The second threshold removes keywords whose TF-IDF value do not exceed a given threshold (stopword removal) and the last one is a threshold that has to be exceeded by an item in order to get considered as a "recommendation".
Moreover, in the hybridization step both, Collaborative Filtering and Content Based Recommendation, are weighted and have to be defined in advance. In Evaluation Based on Modifications Timestamps - one of two evaluation attemps - are modification and tagging time stamps also gathered to an rating using weights.

[3]Grid views are not supported in Nextcloud 13. Therefore, it was required to create a separate stylesheet file in order to define the CSS classes.

5.5.1 Threshold

It is not easy to define a threshold as they depend on different factors. In the previous section Content-Based Recommendation and Collaborative Filtering approaches are evaluated in detail. Taking a closer look at the evaluation, one will find that the Content-Based Recommendation results are always not sufficient. This is due to the fact that in all cases the number of keywords in a user profile is higher than the number of keywords in a document. Therefore, the number of overlapping keywords is not high and the $Overlap\ Coefficient$ will always divide a small number by a big number. Therefore, the threshold value should be a small value.
The thresholds for $Cosine\ Computer$ and keyword removal have to be defined intuitive. Ideally, those threshold values are calculated as an average out of test data. However, because this is not possible yet, the threshold for the *Cosine Computer* is set to 0.5 and for keyword removal to the $1/3$ value of the highest TF-IDF value in the list.

5.5.2 Weights

Based on the results of the Content-Based Recommendation described in Threshold, the weights for hybridization are set to 0.75 for Collaborative Filtering and 0.25 for Content-Based Recommendation.
In case of ratings that are gathered out of modification time stamps, the weights for editing a file is set to 0.75 and tagging to 0.25. This approach assumes that editing a file is more often and therefore more expressive than tagging as favorite.

Chapter 6

Relation to Accessibility

6.1 Introduction

The masters programme "Barrierefreie Systeme / Intelligente Systeme" is structured in a special way. The course consists of three faculties: the first one is "Planen und Bauen" (Planning and Building) and is addresses architects. The second faculty is "Intelligente Systeme" (intelligent systems) and addresses computer scientists. The last one is "Case Management" (case management) and adresses students, who studied a human health and social work activities related bachelor course.

Each faculty has its own schedule of module and teaches in the specific area. In addition, there is one course per semester in which all students of all faculties work on mutual projects. These courses count more than the faculty specific courses. The main goal of these projects is focused on working interdisciplinary on accessible (barrier-free) systems.

6.2 Recommender Systems in the Context of Accessiblity

Barrier-free systems should enable people to have an independent and and self-determined life, despite age and health or functional restrictions of any sort. This chapter should reflect the recommendation system, which was described in chapter State of the Art and Implementation, under these aspects.

Many people suffer from visual impairment or have other kinds of cognitive restrictions such as perception, attention, memory, action planning, judgement, problem solving and communication. Recommendation systems can help people with visual impairments. Considering a Nextcloud instance where many files are up and downloaded and shared with users, the recommendation system can help to filter the most interesting files and arrange files further above or in a separate section. In doing so, the user gets less files listed in his profile. Unfortunately, the proposed recommendation system cannot help blind people since the recommender system does not output speech yet.

Hearing-impaired people do not necesseraily benefit from a recommendation system. There is no speech output from the recommendation system and thus, the users cannot benefit from the whole functionality. Since the system filters interesting files and may arrange them further above in a list, people with hearing impairments benefit from this arrangement.

In case of cognitive limitations, where people have problems with learning and understanding, can also benefit of a recommendation system.
An example of this is figure Nextcloud Root Folder with three recommendations. The figure shows the recommendations UI. The UI contains three recommended files for the user. This can help users not feel overrun and find files they like easier.

Chapter 7

Conclusion

7.1 Summary

In chapter Introduction the intention and motivation for this thesis was explained. The subsequent chapter Requirements Analysis describes the requirements from the perspective of software engineering. Moreover, the section introduces the Nextcloud and Nextcloud app software architecture. Section State of the Art profoundly describes the scientific state of the art and arguments the usage scientific method. In chapter Implementation are general conditions and the structure of the recommender system introduced. The implementation is based on the evaluation of the previous chapter. Chapter Evaluation appraises the results in an local environment with fictional users and files and chapter Relation to Accessibility establishes a relation of a recommender system to accessibility.

The resulting software will be released on GitHub on a date after submitting the thesis. The probability of further development is very high and thus the progress can be seen on my GitHub account @doganoo.
All necessary documents and files of the thesis are submitted on a CD which includes the project as well as the Git history.

7.2 Results

For various reasons writing this thesis was quite a challenge for me. I joined the Nextcloud GmbH company only to profoundly finish the masters programme. The company offered a completely different way of working, in which I was fully on my own and was able to work from home. Creating an app as a master's thesis was also something new for Nextcloud since they had never supervised a thesis before. This has not always been easy, as the views sometimes diverged but in general, it was a great cooperation and also a professional and personal experience.
The thesis is also the first attempt in research and implementation of a machine learning project (or at least, a related topic). Realising the implementation with PHP, a programming language

not known for performance, has also been a challenge. But at least for smaller amounts of data this went well.
For the methods used, there are no officialy known APIs available. This means that the recommendation system is completely implemented by myself after finishing the scientific research. This went very well because object orientation is supported by recent versions of PHP and has helped me to structure the code.
In addition to the challenges of the programming language, the app was implemented as a Nextcloud app. The challenge in doing this was my lack of knowledge of this framework as I have never implemented a Nextcloud app before. Due to the supportive Nextcloud team and core developers this implementation also went well. They helped me wherever it was stagnating and therefore the process of getting familiar with the framework was quite simple.
Unfortunately the effectiveness of the resulting recommendation system was not measurable due to a lack of data and users. The effectiveness of such a system is rather subjective. A longer testing phase for a reliable case study would thus be needed. Within the time of research and writing this thesis, a longer testing phase was impossible. The evaluation is therefore limited to a smaller test case in order to compare the results of the app with the calculations by hand. The evaluation was based on two main methods: static file tags (favorites) and modification time stamps of files. Where the first method had a binary rating range (0 or 1), the second method was based on time stamp ranges which were classified in a range from 0 to 5. The second method has proven to be more suitable due to two reasons: First, the user has not to maintain his favorite tags. The daily work (opening/modifying) with files is the only action required for recommendations. Second, the range from 0 to 5 provides more accuracy as thresholds can be chosen flexible.
Another result that is more personal than scientific is the experience with GitHub, where I had the chance to discuss features in public[1], create pull requests[2] and fork an entire project[3].

7.3 Future Work

Among other factors, the success of recommendation systems depend on the selection of the correct weight and threshold values. Further research for binary scales must be done. This is necessary to find out whether it is sufficient to define the threshold value as 0.5. It is also important to find out whether the component of Content-Based Recommendation or Collaborative Filtering has more accuracy.
The recommendation system explained in the previous chapters can be extended with learning algorithms in order to define thresholds and weights. Doing this, the system can scale over time and adjust the thresholds to the proper value. In doing so, the user can be asked for feedback (a question like "Did this recommendation help you?"). Other metrics, such as measuring user behaviour, can also be used as a feedback tool. One of these metrics exemplarily is the amount of time in which a recommended file is looked at or kept in a user's profile.

[1] https://github.com/nextcloud-gmbh/recommendation_assistant/issues/1
[2] https://github.com/smalot/pdfparser/pull/184
[3] https://github.com/doganoo/PdfToText

The recommendation system which is further explained in this thesis assembles keywords dynamically from files that are owned or shared with a user. These keywords are considered as "describing keywords" of a user's profile. However, there is no opportunity to influence the keyword assembly yet. The user can neither black nor whitelist keywords nor define new ones by himself. This procedure would help understanding the user's interests more and thus increases the quality of recommendations. User defined keywords in this case could be given more importance.
Furthermore, the evaluation has shown that if user profiles have too many keywords, the recommendation results for Content-Based Recommendation are poor. Therefore, the number of keywords have to be limited to a certain amount of keywords. This can be achieved during the Overlap Coefficient calculation by limiting the number of user profile keywords to the number of item keywords or vice versa.

Nextcloud offers many other features, such as (automated) file tagging, comments, sharing and activities. Tags and comments can be considered as some kind of "content" and can be used for Content-Based Recommendation. Since tags are also some kind of "content", they are very suitable for this purpose. Comments represent additional descriptive "content" and are also very suitable for Content-Based Recommendation.

Test data was not available during the writing and research process of this thesis. It was therefore not possible to test the system with a large amount of data. A further challenging project will be the evaluation and optimization of the proposed system under this aspect.

Glossary

Activity App	Nextcloud's Activity App shows recent activities on the server, such as file creation or deletions.
AGPLv3	Affero General Public License version 3 is a free software license.
AI	Artificial Intelligence is intelligence demonstrated by machines.
AJAX	Asynchronous JavaScript And XMLHttpRequest is a way creating asynchronous web requests.
Amazon.com	Amazon.com is an electronic commerce and cloud computing company which uses Collaborative Filtering heavily for recommendations.
API	Application Programming Interface defines a set of protocols and services in order to build software.
Bag-of-Words	In Bag-of-Words, a text is represented as the bag and each word together with the number of occurance is represented uniquely within the bag.
Bayesian Networks	Bayesian Networks represents a probabilistic model to represent a set of variables and their dependencies.
CalDAV	Calendaring Extension to WebDAV is a protocol to access scheduling (calendar) information on a server.
CardDAV	vCard Extension to WebDAV is a protocol to access address book information on a server.
Command Line Interface	Command Line Interface accepts program related commands via a shell.
Composer	Composer is a package manager for PHP which provides format for PHP dependencies and libraries.

Contributor License Agreement	The Contributor License Agreement defines the conditions under which contributions are made to a project.
Cron	Cron is a time-based job scheduler on a local server.
Design Pattern	A design pattern is a re-usable form of a solution to a problem.
Dropbox	Dropbox is a file hosting and synchronisation service provided by the company Dropbox.
End-To-End Encryption	End-to-end encryption is way of communication where only the communicating participants can read the messages.
Fork	Creating a new project from an existing project.
Git	Git is version control system for tracking changes in software projects.
Git Branch	Branches in version control systems diverges from the main line (usually called master) and continue to work without messing the master.
GitHub	GitHub is a web-based hosting service for source code version control using git.
Google Drive	Google Drive is a file storage and synchronisation service provided by Google.
Grid View	Grid views display items in a multiple-dimensional grid which usually provides clicking, dragging or replacing.
GroupLens	GroupLens was one first study about recommendation systems.
HTML	Hypertext Markup Language is a markup language to create web applications and pages.
HTTP	Hypertext Transfer Protocol is the foundation of data communication for the internet.
JavaScript	a programming language usually combined with HTML and CSS in order to create a web frontend.

Linear Regression	Linear Regression is a linear approach for modelling the relationship between two variables X and Y.
Machine Learning	Machine Learning gives computers the ability to learn without being explicitly programmed.
MariaDB	MariaDB is a fork of MySQL intended to remain under a free license.
Markov Decision Process	Provides a mathematical framework for modeling decision making.
MySQL	MySQL is an open source relational database management system.
Natural Language Processing	Natural Language Processing is a research field related to the interaction between machines and humans using human language.
Netflix	Netflix is an entertainment company and provides video-on-demand media online.
Neural Networks	Artificial Neural Networks are computing models inspired by biological neural networks.
Nextcloud App Store	Nextcloud's app store where apps are available to download and install.
Object Oriented Programming	Object Oriented Programming is a paradigm based on the concept of represent everything as an "object".
Oracle Database	Oracle Database is a commercial multi-model database management system by Oracle Inc.
ownCloud	ownCloud is an open source file sharing and synchronization software and the predecessor of Nextcloud.
ownCloud Console	ownCloud Console is ownCloud's/Nextcloud's command line interface.
PHP	PHP is a server-side scripting programming language designed for web-development.
PostgreSQL	PostgreSQL is an object relational database management system.
Pull Request	When contributing to an project managed with a version control system, users make usually a pull request in order to merge the changes to the main project.

RESTful	Representational State Transfer allows requesting and manipulating textual representations of web resources.
Schlag den Raab	Schlag den Raab was a live german show where a candidate had to beat the moderator Stefan Raab in a number of disciplines.
Set	A Set is defined as objects that allow easy determination whether an object is already present or not. Several ways, such as using arrays with hashed keys, are available.
Shazam	Shazam application can identify music among others based on a short sample.
Shell	The Shell, typically for Unix-like operation systems, is the command line user interface to interact with the operating system.
Single Value Composition	Single Value Composition is the factorization of a real or complex matrix.
SQLite	SQLite is a relational database management system and works usually embedded into the application.
Support Vector Machines	Support Vector Machines are supervides learning models for classification and regression analysis.
Symfony	Symfony is a PHP based library for creating web applications.
Sync-Client	A client to synchronize remote content with a device and vice versa.
UI	User Interface (UI) in context of web applications is usually referred to HTML that provides input and shows output to the underlying system.
UML	Unified Modeling Language is a modeling language to visualize the design of a system.
URL	Uniform Resource Locator is a reference to a web resource within a computer network and is usually known as web address.
Web 2.0	In Web 2.0, users not only consume web content but also generate it. Web 2.0 is also known as social media.
Webcron	Webcron is a time-based job scheduler on a remote server.

WebDAV	WebDAV is an extension to HTTP that allows remote clients to access/change web content.

Bibliography

[AT05] Adomavicius, Gediminas ; Tuzhilin, Alexander: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. In: *IEEE Trans. on Knowl. and Data Eng.* 17 (2005), Juni, Nr. 6, 734–749. http://dx.doi.org/10.1109/TKDE.2005.99. – DOI 10.1109/TKDE.2005.99. – ISSN 1041–4347

[AWWY99] Aggarwal, Charu C. ; Wolf, Joel L. ; Wu, Kun-Lung ; Yu, Philip S.: Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering. In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA : ACM, 1999 (KDD '99). – ISBN 1–58113–143–7, 201–212

[BHC98] Basu, Chumki ; Hirsh, Haym ; Cohen, William: Recommendation as Classification: Using Social and Content-Based Information in Recommendation. In: *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, AAAI Press, 1998, S. 714–720

[BHK98] Breese, John S. ; Heckerman, David ; Kadie, Carl: Empirical Analysis of Predictive Algorithm for Collaborative Filtering. In: *Proceedings of the 14 th Conference on Uncertainty in Artificial Intelligence*, 1998, S. 43–52

[BoW] https://en.wikipedia.org/wiki/Bag-of-words_model

[BP98] Billsus, Daniel ; Pazzani, Michael J.: Learning Collaborative Information Filters. In: *Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1998 (ICML '98). – ISBN 1–55860–556–8, 46–54

[BPC00] Billsus, Daniel ; Pazzani, Michael J. ; Chen, James: A Learning Agent for Wireless News Access. In: *Proceedings of the 5th International Conference on Intelligent User Interfaces*. New York, NY, USA : ACM, 2000 (IUI '00). – ISBN 1–58113–134–8, 33–36

[Bur02] Burke, Robin: Hybrid Recommender Systems: Survey and Experiments. In: *User Modeling and User-Adapted Interaction* 12 (2002), Nov, Nr. 4,

331–370. http://dx.doi.org/10.1023/A:1021240730564. – DOI 10.1023/A:1021240730564. – ISSN 1573–1391

[Bur07] Burke, Robin: The Adaptive Web. Version: 2007. http://dl.acm.org/citation.cfm?id=1768197.1768211. Berlin, Heidelberg : Springer-Verlag, 2007. – ISBN 978–3–540–72078–2, Kapitel Hybrid Web Recommender Systems, 377–408

[CBRa] https://techcrunch.com/2018/01/02/apple-buys-app-development-service-buddybuild/

[CBRb] https://www.theguardian.com/football/2018/jan/04/thibaut-courtois-close-to-signing-new-chelsea-deal

[CBRc] https://www.washingtonpost.com/politics/trump-slams-bannon-when-he-was-fired-he-not-only-lost-his-job-he-lost-his-mind/2018/01/03/21fb158a-f0aa-11e7-b3bf-ab90a706e175_story.html?utm_term=.cd5d63f8d13c

[CBRd] https://www.theguardian.com/business/2018/jan/03/apple-leads-race-to-become-world-first-1tn-dollar-company

[CBRe] https://www.theguardian.com/society/2018/jan/03/patients-in-africa-twice-as-likely-to-die-after-an-operation-than-global-average-report-shows

[CBRf] https://techcrunch.com/2017/12/31/twitter-ended-the-year-on-a-fascinating-run/

[CBRg] https://www.theguardian.com/football/blog/2018/jan/04/liam-rosenior-fa-cup-romance-as-teams-have-chance-to-dream

[CBRh] https://www.washingtonpost.com/news/the-fix/wp/2018/01/04/what-in-the-world-was-steve-bannon-thinking-3-theories/?tid=pm_politics_pop&utm_term=.4a18d47927db

[CBRi] https://www.theguardian.com/business/2018/jan/02/eu-bankers-work-around-the-clock-ahead-of-launch-of-mifid-ii-reforms

[CBRj] https://www.theguardian.com/commentisfree/2018/jan/03/the-guardian-view-on-the-nhs-winter-crisis-not-such-a-happy-birthday

[CBRk] https://techcrunch.com/2018/01/03/instaapp/

[CBRl] https://www.theguardian.com/football/blog/2018/jan/04/arsene-wenger-referee-paranoia-arsenal-mindset-chelsea

[CBRm] https://www.theguardian.com/world/2018/jan/04/ukraine-killing-of-rights-lawyer-sparks-protests-against-criminal-system

[CBRn] https://www.theguardian.com/business/2018/jan/02/good-factories-bad-shoppers-brexit-pattern-emerging

[CBRo] https://www.theguardian.com/business/2018/jan/03/us-drug-firm-offers-cure-for-blindness-at-425000-an-eye

[CBRp] https://techcrunch.com/2018/01/03/apple-readies-siri-for-the-homepod-by-adding-a-podcast-powered-news-brief/

[CBRq] https://www.theguardian.com/football/2018/jan/04/jurgen-klopp-philippe-coutinho-liverpool-manchester-city-barcelona-everton

[CBRr] https://www.washingtonpost.com/news/politics/wp/2018/01/04/what-weve-learned-about-trumps-campaign-and-russia-since-trump-first-denied-collusion/?utm_term=.581f11307330

[CBRs] https://www.theguardian.com/business/2018/jan/02/trump-tax-cut-to-dent-bp-profits-by-15bn

[CBRt] https://www.theguardian.com/science/2018/jan/03/alcohol-can-cause-irreversible-genetic-damage-to-stem-cells-says-study

[CBRu] https://techcrunch.com/2018/01/03/apple-developer-program-fee-waivers-are-now-available-for-nonprofits-schools-and-government/

[CBRv] https://www.theguardian.com/football/2018/jan/04/juventus-confident-signing-emre-can-liverpool

[CBRw] https://www.theguardian.com/us-news/2018/jan/05/donald-trump-mexico-border-wall-congress-18-billion

[CBRx] https://www.theguardian.com/money/2017/dec/31/hammond-relying-on-household-debt-to-hit-targets-says-mcdonnell

[CBRy] https://www.theguardian.com/society/2018/jan/03/doctors-patients-government-failing-stop-nhs-crisis

[CCFF11] Cacheda, Fidel ; Carneiro, Víctor ; Fernández, Diego ; Formoso, Vreixo: Comparison of Collaborative Filtering Algorithms: Limitations of Current Techniques and Proposals for Scalable, High-performance Recommender Systems. In: *ACM Trans. Web* 5 (2011), Februar, Nr. 1, 2:1–2:33. http://dx.doi.org/10.1145/1921591.1921593. – DOI 10.1145/1921591.1921593. – ISSN 1559–1131

[CGM+99] Claypool, Mark ; Gokhale, Anuja ; Miranda, Tim ; Murnikov, Pavel ; Netes, Dmitry ; Sartin, Matthew: *Combining Content-Based and Collaborative Filters in an Online Newspaper*. 1999

[DOC] http://www.doctrine-project.org/about.html

[Fox89] Fox, Christopher: A Stop List for General Text. In: *SIGIR Forum* 24 (1989), September, Nr. 1-2, 19–21. http://dx.doi.org/10.1145/378881.378888. – DOI 10.1145/378881.378888. – ISSN 0163–5840

[Gor16] Gorakala, Suresh K. ; Sinha, Manisha (Hrsg.): *Building Recommendation Engines*. Packt Publishing, 2016

[JSD] http://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html

[JZ09] Jia Zhou, Tiejian L.: Towards an Introduction to Collaborative Filtering. In: *International Conference on Computational Science and Engineering* (2009)

[Kla09] Klahold, André: *Empfehlungssysteme*. Vieweg+Teubner, 2009

[Lev14] Levinas, Claudio A.: *An Analysis of Memory Based Collaborative Filtering Recommender Systems with Improvement Proposals*, Universitat Politecnica de Catalunya Barcelonatech, Diplomarbeit, 2014

[LH16] Lenhart, Philip ; Herzog, Daniel: Combining Content-based and Collaborative Filtering for Personalized Sports News Recommendations. In: *CBRecSys@RecSys*, 2016

[LS13] Laila Safoury, Akram S.: Exploiting User Demographic Attributes for Solving Cold-Start Problem in Recommender System. In: *Lecture Notes on Software Engineering* 1 (2013), Aug, Nr. 3, S. 303–307

[LSY03] Linden, Greg ; Smith, Brent ; York, Jeremy: Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. In: *IEEE Internet Computing* 7 (2003), Januar, Nr. 1, 76–80. http://dx.doi.org/10.1109/MIC.2003.1167344. – DOI 10.1109/MIC.2003.1167344. – ISSN 1089–7801

[MGT86] Malone, T. W. ; Grant, K. R. ; Turbak, F. A.: The Information Lens: An Intelligent System for Information Sharing in Organizations. In: *SIGCHI Bull.* 17 (1986), April, Nr. 4, 1–8. http://dx.doi.org/10.1145/22339.22340. – DOI 10.1145/22339.22340. – ISSN 0736–6906

[MWD] https://martinfowler.com/articles/injection.html

[NC1a] https://nextcloud.com/blog/nextcloud-introducing-native-integrated-end-to-end-encryption/

[NC1b] https://docs.nextcloud.com/server/12/developer_manual/app/classloader.html

[NC1c] https://docs.nextcloud.com/server/9/developer_manual/app/container.html

[NC1d] https://docs.nextcloud.com/server/10/NextcloudDeveloperManual.pdf

[NC2] https://docs.nextcloud.com/server/12/developer_manual/general/codingguidelines.html#coding

[NC3] https://docs.nextcloud.com/server/12/developer_manual/general/codingguidelines.html#general

[NC4] https://docs.nextcloud.com/server/12/developer_manual/general/codingguidelines.html#user-interface

[NC5] https://docs.nextcloud.com/server/9/developer_manual/app/hooks.html

[NC6] https://docs.nextcloud.com/server/9/developer_manual/app/routes.html

[NC7] https://docs.nextcloud.com/server/12/developer_manual/app/container.html

[NC8] https://docs.nextcloud.com/server/12/admin_manual/configuration_server/background_jobs_configuration.html

[NC9] https://docs.nextcloud.com

[Nex] Nextcloud: *Nextcloud 12 Server Administration Manual - System Requirements¶*. https://docs.nextcloud.com/server/12/admin_manual/installation/system_requirements.html, Abruf: 16.10.2017

[NP13] Nikolaos Polatidis, Christos K. G.: Mobile recommender systems: An overview of technologies and challenges. In: *Second International Conference on Informatics & Applications (ICIA)* (2013), Sept

[Paz99] Pazzani, Michael J.: A Framework for Collaborative, Content-Based and Demographic Filtering. In: *Artif. Intell. Rev.* 13 (1999), Dezember, Nr. 5-6, 393–408. http://dx.doi.org/10.1023/A:1006544522159. – DOI 10.1023/A:1006544522159. – ISSN 0269–2821

[PB07] Pazzani, Michael J. ; Billsus, Daniel: Content-based recommendation systems. In: *THE ADAPTIVE WEB: METHODS AND STRATEGIES OF WEB PERSONALIZATION. VOLUME 4321 OF LECTURE NOTES IN COMPUTER SCIENCE*, Springer-Verlag, 2007, S. 325–341

[PG14] Polatidis, Nikolaos ; Georgiadis, Christos K.: Mobile recommender systems: An overview of technologies and challenges. In: *CoRR* abs/1408.6930 (2014). http://arxiv.org/abs/1408.6930

[PHPa] http://php.net/manual/en/language.oop5.autoload.php

[PHPb] http://www.php-fig.org/psr/psr-4/

[PHPc] PHP: *PHP 7.0.0 Release Announcement*. http://php.net/releases/7_0_0.php, Abruf: 16.10.2017

[PR94] Paul Resnick, Mitesh Suchak Peter Bergstrom John R. Neophytos Iacovou I. Neophytos Iacovou: GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In: *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work, Chapel Hill* (1994)

[SKKR01] Sarwar, Badrul ; Karypis, George ; Konstan, Joseph ; Riedl, John: Item-based Collaborative Filtering Recommendation Algorithms. In: *Proceedings of the 10th International Conference on World Wide Web*. New York, NY, USA : ACM, 2001 (WWW '01). – ISBN 1–58113–348–0, 285–295

[SM95] Shardanand, Upendra ; Maes, Pattie: Social Information Filtering: Algorithms for Automating &Ldquo;Word of Mouth&Rdquo;. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1995 (CHI '95). – ISBN 0–201–84705–1, 210–217

[SYM] https://symfony.com/doc/current/routing.html