



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Bachelor's Thesis

A Recommender Framework for Skills Management

In Cooperation With SinnerSchrader

Torben Reetz

3reetz@informatik.uni-hamburg.de
BSc Software Systems Development
Matr.-Nr. 6524064
Semester 07

Primary Supervisor: Prof. Dr. Walid Maalej
Secondary Supervisor: Prof. Dr. Eva Bittner

Date: 03.2017

Abstract

TODO rewrite

Contents

Abstract	i
1 Introduction	1
1.1 Motivation	1
1.1.1 SinnerSchrader	1
1.2 Leading Goals	2
2 Related Work	3
3 Functional Objectives and Existing Solutions	5
3.1 Usage Scenarios	5
3.1.1 Asking for Help	5
3.1.2 Finding Potential Team Members	5
3.1.3 Collecting Information	5
3.1.4 Notifications by Supervisors	6
3.2 Requirements	7
3.2.1 Functional Requirements	7
3.2.2 Non Functional Requirements	7
3.3 Commercial Solutions	8
3.3.1 Skills Base	8
3.3.2 Talent Management (engage!)	8
3.3.3 SkillsDB Pro	9
3.3.4 Conclusion	9
4 Concept	11
4.1 Person Search	11
4.1.1 Recommender Systems	11
4.1.2 Techniques of Content Filtering	12
4.1.3 Search Algorithm	13
4.1.4 Scoring Algorithm	14
4.1.5 Example	17
4.2 Search Suggestions	18
4.2.1 Available Data	18
4.2.2 Chosen Approach	19
4.2.3 Concept	19

4.2.4	Pseudo-Code	21
4.2.5	Example	22
4.3	Visual Concept & Wireframes	23
5	Implementation	25
5.1	Application Structure	25
5.2	MongoDB	25
5.2.1	BSON	25
5.2.2	Data Structure	26
5.2.3	Queries	26
5.3	LDAP	27
5.4	Reverse Proxy	27
5.5	API	27
5.6	Backend	27
5.7	Architecture	28
5.7.1	Spring Boot	28
6	Evaluation	29
6.1	Algorithm vs. Leute (aka interviews)	29
6.2	Scores gleichverteilt?	29
6.3	Implementation vs. Algorithm	29
6.4	Meeting the Requirements	29
6.4.1	FUntional Requirements	29
6.4.2	Nonfuntional Requirements	29
6.5	Misc	29
6.6	Conclusion	29
	Bibliography	31
	Eidesstattliche Versicherung	35

1 Introduction

1.1 Motivation

Project driven organisations have to face the problem of constantly needing to put teams together based on the members' skills, experience and preferences. In many businesses, there is no sophisticated source of information about those data which makes finding the right person with a specific ability even more complicated. A popular approach to this problem is using computer programs to find the right person for a given set of tasks from all available employees.

SinnerSchrader, a hamburg based web agency that will serve as the practical context for this thesis, decided to launch an internal application for skills management that is meant to solve the aforesaid problems. This thesis will deal with the design and implementation of this software.

1.1.1 SinnerSchrader

The SinnerSchrader Group is a full service web agency based in Hamburg aggregating the subcompanies SinnerSchrader Deutschland, SinnerSchrader Content, SinnerSchrader Commerce and SinnerSchrader Swipe. The broad spectrum of expertise, including, but not limited to, digital communication strategies, visual and interaction design, technical architecture, full stack development, editorial services, content production, E-Commerce, mobile app development, hosting, and maintenance, allows SinnerSchrader to serve all needs regarding their customer's digital transformation. The combination of all said competencies under one single roof reduces organisational friction between the discipline-specific teams because they all share the same vision of the big picture they are creating. This does not only lead to faster development cycles, but also to a more coherent and unified product.

Project-Driven Business

As a web agency, it is clear that SinnerSchrader has to operate in a project-driven way. This means there is no continuous stream of recurring work repeating constantly, but many different projects for different clients, each one dealing with varying challenges and questions. From a technical point of view, the diversity of know-how needed for each project is extremely huge since every application uses its own dedicated stack of technologies. As a consequence, the developers' skill sets are based on the combination of projects they

have worked on and their general field of interest. This results in one problem: Managers frequently have to put teams together based on the members' skills with respect to the individual requirements of the project.

This thesis will cover the creation of a tool helping managers with that problem by providing a centralized source of data about each employee's professional abilities.

Matrix Organisation

The personnel of SinnerSchrader are divided into two different kinds of teams: functional teams of employees sharing the same specialization, i.E. backend development, frontend development, design, or concept, and project teams of people from different functional teams working collaboratively on the same project. This structural model is called a matrix organisation. [Ber16, P. 75] The organisational head of functional teams will further be called the 'supervisor'; the pendant for project team will be mentioned as 'team manager'.

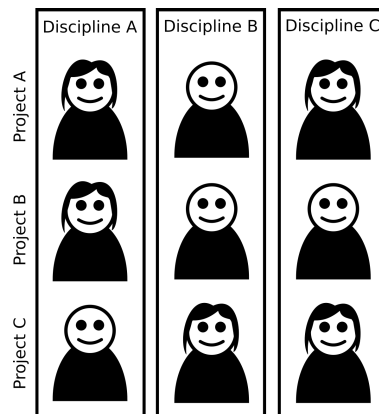


Figure 1.1: Illustration of a matrix organisation

1.2 Leading Goals

This thesis will discuss the requirements a skills management will have to fulfill in order to be advantageous to SinnerSchrader. Existing solutions will be examined and evaluated regarding those requirements. The outcome of this analysis will lay the foundations for the design and concept of a skills management web application custom-tailored to SinnerSchrader's individual needs. The Backend part of said application will be implemented and evaluated.

2 Related Work

TODO PApers und so

3 Functional Objectives and Existing Solutions

3.1 Usage Scenarios

3.1.1 Asking for Help

Having the possibility to ask for help is a vital part of knowledge driven businesses, so collaboration and the sharing of ideas are a major factor in the company guidelines of SinnerSchrader. The application should act as a central repository for knowledge and contacts, enabling employees to find someone who can help in order to solve questions and find solutions faster.

3.1.2 Finding Potential Team Members

Team managers constantly face the problem of reassembling parts of their teams, forming new teams for new projects, and disbanding teams whose projects have ended. As there is no unified source of information about all employees, managers often do not find the best suitable team member to fill an open position because they simply do not know each other yet. The tool will give managers the opportunity to search the entirety of employees at SinnerSchrader and find one meeting all requirements of the open position, thus making collocating teams easier and more efficient.

3.1.3 Collecting Information

The application will give employees the possibility to provide information about their personal knowledge regarding the skills known by the application. Furthermore, they can assign a will value for every skill that describes if they prefer doing the implicitly linked activity or working with the tool described by said skill. That is, people can define what they want to do and what tasks they would like to refuse.

As employees continually enlarge their knowledge while their fields of interest shift towards new technologies, tools or even functional divisions, providing data about their skills and preference once will lead to the system being filled with obsolete information. The quality of the applications search results and suggestions heavily relies on the fidelity and volume of the underlying data about the employees, hence keeping said information up to date is crucial to the performance of the application.

Biannual Feedback Meetings

Every employee has biannual meetings with their respective supervisor to interchange bidirectional feedback, define personal goals, and negotiate possible changes of salary. Part of the feedback given by the employee are subjects they learned or enhanced their knowledge about, and newly developed interests. These insights are documented and registered in the employee's personnel file. This meeting will be the regular occasion for supervisors and employees to refine the data saved in the application and to add newly gained skills to it. The supervisor is advised to address discrepancies between the employee's and their own estimations of skills.

3.1.4 Notifications by Supervisors

Employees and supervisors are encouraged to be in rich contact with each other in order to deliver continuous feedback about the individual person's needs, impediments, and the status of their current projects. As a result, supervisors can identify appropriate moments for reevaluating the skills and preferences saved in the application and notify the employee.

For example, according to Tuckman's team development model, the so called 'adjourning' phase of a project is an occasion for 'recognition of individual achievements and reflection on how far the team has come' [Wil10, P. 3] and thus is a convenient chance to add new skills acquired during the project and to refine the existing data.

Automatic Notifications

Other than the supervisor, the application cannot be able to find the best situations to remember employees to maintain their skill profile, so sending automatic notifications will not be nearly as effective as being reminded by the supervisor. Furthermore, according to [DMA14], only about one in five automatically generated eMails will be opened, which reflects SinnerSchraders experiences with email reminders. Those facts justify the decision, that the system will not send any notification to its users.

Intrinsic Motivations

In addition to the mentioned reasons for employees to provide data about their skills which are all extrinsic motivations, there also exist intrinsic motivations to do so. Being motivated intrinsically can be defined as 'doing something because it is inherently interesting or enjoyable' [RD00]. As people are motivated to focus on tasks they fancy, they are also motivated to voluntarily keep an eye on the quality of the data that is used to determine the tasks they will have to perform.

3.2 Requirements

3.2.1 Functional Requirements

- Accessable to all Employees Every employee must be able to use the application regardless of their equipment or used operating system.
- User Profiles
Anyone can see another user's profile consisting of basic information about the user such as Name, Location, E-Mail and personal skills. Personal skills are composed of a name, a knowledge level and a will level, both on a scale from one to four.
- Provide/Edit skills
Users can add new skills from a pool of known skills to their own profile. Already added skills can be edited and removed from the profile.
- Search
A search function can be used to find people who have added one or more specific skills to their profile. When searching for multiple skills, only persons matching all of them will will be displayed.
 - Ranking
By default, the search results order should be defined by a score aggregating the individual employee's skill level, will level and grade of spezialization in the searched skills. Employees willing to enhance their knowledge about a searched skill should be preferred to others having the same knowledge but a lower will.
 - Sorting
The user should be able to sort the search results not only by said score, but also solely by knowledge and will level.
- Management of known skills
New skills can be added to the set of known skills in the application. Existing skills can be edited and removed. Users personal skills are automatically updated when a skill has been edited so that the integrity of the user profiles is maintained at all times.

3.2.2 Non Functional Requirements

TODO: Klären und formulieren

- Desktop/Devices
 - Browsers
 - Scalability
-

- Load/Response Times

3.3 Commercial Solutions

Three commercial skills management applications have been picked randomly for further examination: 'Skills Base', 'engage! Talent Management' and 'SkillsDB Pro'. A more detailed analysis by [Leh04] shows that most tools, provide a spectrum of features and limitations very similar to the examined solutions, so that the selection is assumed to be representative to the market.

3.3.1 Skills Base

Skills Base¹ offers the required features, but also includes a large number of functionality SinnerSchrader does not need and is not willing to use. This includes assessments, the categorization of skills and a role model for advanced access rights configuration. The search function does not provide searching for multiple skills. Furthermore, the sorting of results found cannot be customized. A central point of the application are dashboards displaying information about the most popular skills in the organisation and long term statistics.

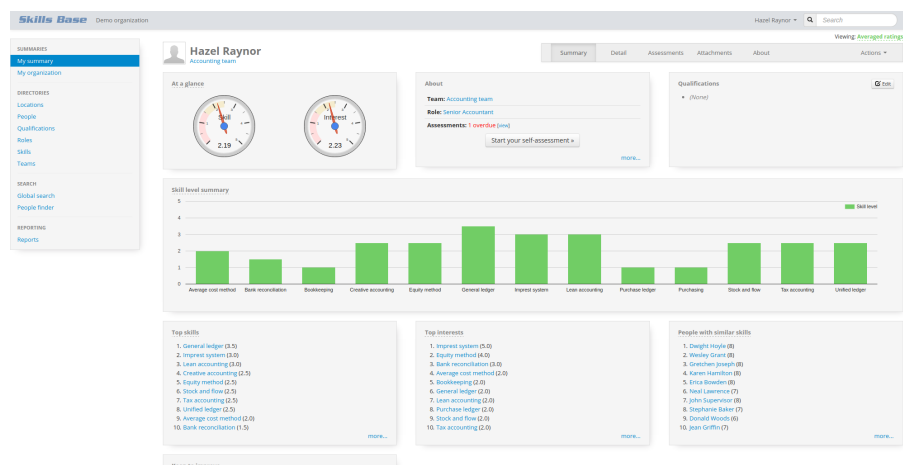


Figure 3.1: SkillsBase Dashboard

3.3.2 Talent Management (engage!)

Talent Management² is a module for Infoniqa's management software engage!. It offers advanced features for managers such as a powerful search function controlled via a special query language. It also includes data about the employees' salaries, feedback protocols and certificates. It can only be used in combination with engage!, an complete

¹<http://www.skills-base.com/>

²<http://www.infoniqa.com/hr-software/skill-management>

human resources management solution including features like time tracking, e-learning, applicant management and payroll accounting.

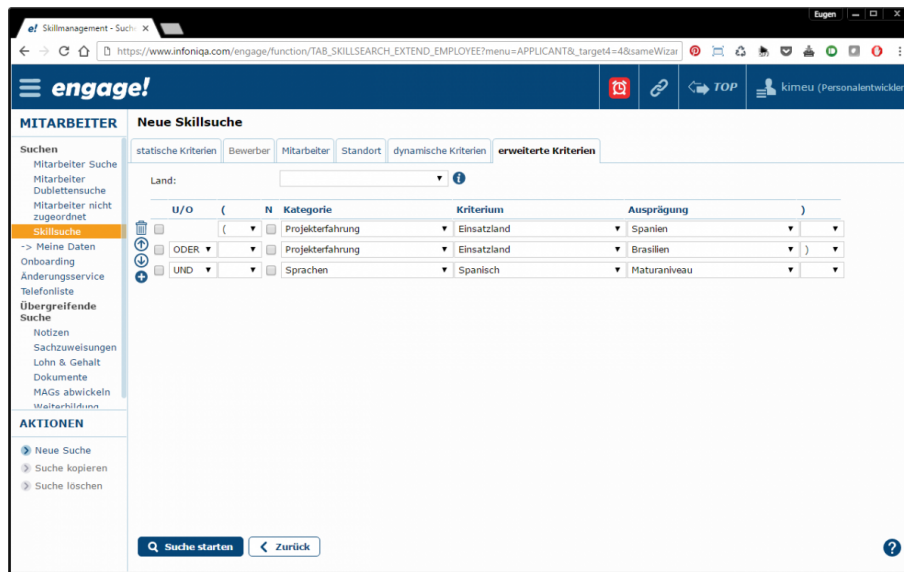


Figure 3.2: Talent Management Search

3.3.3 SkillsDB Pro

SkillsDB Pro³ is an application designed to serve as a database in an organization providing an overview about every person's own skills and and trainings only to themselves and their manager. The search function is capable of searching for multiple skills combined with different logical operators which enables users to enter very sophisticated queries. Not only can users provide information about their skills, but managers can also do this with the limitation that no employee can see their manager's rating about themselves. Furthermore, only managers can search for persons. Taking into consideration that SinnerSchrader needs a tool to enable everyone to find someone with a specific skillset, this is a serious disadvantage. SkillsDB also offers features SinnerSchrader does not intent to use including, but not limited to the automatic generation of project reports based on plan succession and demands for assessments.

3.3.4 Conclusion

None of the analyzed softwares offers all features required, but all of them include various functions SinnerSchrader does not intend to use, which brings undesired complexity into the applications. One of the most critical features, sorting the search results by best match, is not offered by any of the commercial solutions. Furthermore, all those systems differentiate between employees and their supervisors and thus restrain transparency.

³<http://www.skillsdbpro.com>

Instead of a solution for monitoring employees and rating them, we want a tool for everyone to find another person who offers the skills needed to solve a concrete problem.

Pain Point Fitness Scoring

As shown by [Can13], motivation is a vital factor regarding any employee's performance and quality of work. Although motivation is a complex construct of many highly diverse dimensions, the overlap of a person's interests and their duties is a key aspect to it. Assuming that every member of the company has some skills they prefer to employ over others, matching people to tasks that require the exact same abilities they are interested in employing will lead to more motivated employees and thus have a positive impact on the overall productivity of SinnerSchrader. Consequently, when searching for persons having specific skills, the application should not only take into account the employees' skills but only their preferences in order not to find the most skilled, but the best fitting one. Unfortunately, none of the examined applications does provide a way to aggregate both, skills and preferences, into a single score indicating the overall grade suitability of a person relative to the searched skills.

4 Concept

The application should be accessible to all employees of SinnerSchrader. Due to the heterogeneity of the people's computer setups running Windows, macOS and Linux, creating a native application supported by everyone's system is a rather complicated task. A web application using standard technologies does not only solve this problem, but can also be used from mobile devices such as smart phones and tablets. Furthermore, there is no need to manually install and update the software so that it can be assumed that all users use the latest version of the application. This is not only a positive factor regarding the overall usability of the system, but also assures bugs and security issues are eliminated the moment a fixed version of the software is deployed. All those advantages compared to native clients and the fact that SinnerSchrader's expertise lies in the development of web applications lead to the decision, that such an application would be the appropriate solution.

4.1 Person Search

The central feature of the application is the search function that returns a list of all persons matching the entered set of skills. By default, the results are ordered by a 'fitness score' which describes how well a person matches into the set of searched skills. As a consequence of this sorting, the application implicitly recommends the best matching person to the user, thus it falls within the group of 'recommender systems'.

4.1.1 Recommender Systems

Recommender systems 'are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of [...] information' [IFO15] that are commonly used to recommend an item to the user based on their previous interactions with other items. For example, recommender systems are used to predict products a customer might want to buy based on the ones they already bought in order to present those items more prominently than articles the customer is unlikely to buy. In the context of the skill management application, a recommender system will be used to predict skills a user might want to search for based on the ones they already entered and a the knowledge which skills are more often searched for together. When entering an item into the search bar, the user will be offered to select items the recommender system predicts they might want to combine with the already entered ones.

4.1.2 Techniques of Content Filtering

As described by [IFO15], there are five basic techniques for filtering items to find the ones to recommend to the user:

Content Based Filtering

The content is filtered by examining its attributes in order to find items that are contentually similar to the one the user is currently or has previously been interacting with.

Collaborative Filtering

Collaborative filtering techniques rely on the assumption that users can be divided into groups of 'neighbors' that behave similarly, so that recommendations are decutable from other users' former interactions.

- **Model Based Filtering**

Model based filtering applies methods of machine learning and data mining to learn a pre-computed model which predicts the users' interactions.

- **Memory Based Filtering**

Memory based filtering techniques employ the saved interaction history and generate recommendations based on it. In contrast to model based filtering, memory based filtering does not learn a given model but operates directly on the known data.

- **User Based Filtering**

The user's interactions with items are examied in order to find neighbors that share a similar activity history. Once neighbors are found, the system combines their interaction histories in order to find items to user is likely to appreciate getting recommended.

- **Item Based Filtering**

Item based filtering combines all users' interactions and creates a model describing which items are similar to another. This model is then used to recommend items similar to the ones the user has given positive feedback for.

Hybrid Filtering

Hybrid filtering combines two or more filtering methods either by aggregating their respective results into a single set of recommendations preferring the items multiple methods recommend, or by bringing content based aspects into the approach of collaborative filtering and/or vice versa.

4.1.3 Search Algorithm

In the context of the search function, all employees are searchable items. Their attributes include name, location and their respective skills structured as pairs of skill level and will level. This data will be used in a content based filtering approach that not only finds suitable employees, but also ranks them by their fitting into the searched skill set. Other users's interactions with search results, for example the opening of a found person's profile, will not be taken into account, since there is no direct connection between these actions and the person's fitness. Furthermore, a system based on the user's former selections would be inadequate, because the application is meant to give managers the ability to find employees they did not already have contact with; recommending persons the searching user had interacted with would thus be counterproductive.

Outline

The basic structure of the search function will be:

1. Create a list of all employees
2. Filter by Skills
Remove all employees from the list that do not have all skills the user searched for. At this point, only the presence of the skill in the employees' profiles is taken into account; skill/will levels are ignored.
3. Filter by Location
If the user specified a location to search for, remove all employees from the list that do not match it.
4. Assign Fitness Scores
Assign a fitness score to all remaining employees.
5. Sort by fitness score
The results will be sorted by fitness score. The employee with the best fitness score will be shown first in the list of results, so an implicit recommendation is made by the system.
6. Return Results

Pseudo-Implementation

```
1 function search(searchItems, searchLocation) {  
2   var results = getAllEmployees()  
3  
4   for (Employee e in results) {  
5     if (e.skills does not contain all elements of searchItems) {
```

```
6     results.remove(e)
7 }
8 }
9
10 for (Employee e in results) {
11     if (e.location is not searchLocation) {
12         results.remove(e)
13     }
14 }
15
16 for (Employee e in results) {
17     e.assignFitnessScore()
18 }
19
20 results = results.sortByFitnessScore()
21
22 return results
23 }
```

4.1.4 Scoring Algorithm

The application should sort all found persons by their fitness into the searched skill set. This implies there has to be an algorithm that can assign a score describing said fitness to every person found.

Requirements

According to [JHS07], an algorithm that matches persons to positions based on their skills has to meet more demands than solely the functional ones. They define the specific requirements such an algorithm assigning naval personnel to positions on a ship as follows:

- Easy to implement and maintain
- Fast to execute, so as not to become a computational bottleneck
- Takes into account factors: rating, pay grade and NECs¹ and future taxonomies characterizing required knowledge, skills and abilities

[JHS07, P. 14]

These qualities include factors very specific to the US Navy and thus will have to be evaluated and translated into SinnerSchraders field of operation, but general requirements such an algorithm has to meet can be deducted: It may not be too complex as employees should be able to understand the system they are rated by, it should take into account different groups of factors and must be easy to adjust in order to keep the system maintainable.

¹Navy Enlisted Classifications

Factors to Include

An Estimation of a concrete person's fitness into a position described by the searched skill set needs not only to take into account the matching of offered and required skills, but also the employees motivation to apply said skills derived from their preferences and their personal specialities and expertise. Latter can be described as the skill and will levels that stick out of their overall capabilities. So the important factors to be included in the algorithm are:

- Average level of knowledge regarding the searched skills.
- Average level of will regarding the searched skills.
- Specialization in the searched skills, including:
 - Specialization in knowledge about the searched skills.
 - Preference of the searched skills over others.

Proposed Fitness Score Algorithm

Skill and will levels are described as integer values on a scale form zero to three. This scale, called V , can be expressed as

$$V = \{x \in \mathbb{N}_0^+ \mid 0 \leq x \leq 3\}$$

All existing skills are accumulated in the set S . The employee's skills are represented by E which is a subset of S . The search items are defined as Q .

$$\begin{aligned} S &= \{java, ruby, \dots\} \\ E &= \{x \in S \mid \text{employee has skill } x\} \\ Q &= \{x \in S \mid \text{user searches for skill } x\} \end{aligned}$$

The functions v_s and v_w assign a level of skill/will to each value in E .

$$v_s : E \mapsto V$$

$$v_w : E \mapsto V$$

The averages of the employees' skill/will values of the searched skills are defined as a_s and a_w . The variables s_s and s_w describe the employee's specialization in the searched items defined as the difference of the average skill/will level of the searched items and the average level of all other items. A person with maximal interest and knowledge in

all searched items and an infinite number of skills with the minimal value of zero would have the greatest specialization possible and thus get assigned a value of one.

$$a_s = \left(\sum_{x \in E \cap Q} v_s(x) \right) \cdot \frac{1}{|E \cap Q|}$$

$$a_w = \left(\sum_{x \in E \cap Q} v_w(x) \right) \cdot \frac{1}{|E \cap Q|}$$

$$s_s = \frac{|V| + a_s - \left(\left(\sum_{x \in E \setminus Q} v_s(x) \right) \cdot \frac{1}{|E \setminus Q|} \right)}{2 |V|}$$

$$s_w = \frac{|V| + a_w - \left(\left(\sum_{x \in E \setminus Q} v_w(x) \right) \cdot \frac{1}{|E \setminus Q|} \right)}{2 |V|}$$

The resulting fitness score f is a weighted mean of the introduced factors. The weights w_{as} , w_{aw} , w_{ss} , w_{sw} sum up to one.²

$$f = \frac{w_{as} \cdot a_s}{|V|} + \frac{w_{aw} \cdot a_w}{|V|} + w_{ss} \cdot s_s + w_{sw} \cdot s_w$$

$$w_{as} + w_{aw} + w_{ss} + w_{sw} = 1$$

Example Calculation

Given are three example employees, Alice, Bob and Charlie, with the same three skills each. (Notation: skill level/will level)

Person	Java	Ruby	C++
Alice	2/1	2/2	3/3
Bob	2/3	0/3	0/1
Charlie	3/3	2/1	1/2

Applying the algorithm with $w_{as} = w_{aw} = w_{ss} = w_{sw} = 0.25$ to a search for the skills 'Java' and 'Ruby' results in the following values³:

Person	a_s	a_w	s_s	s_w	f
Alice	2	1.5	0.33	0.25	0.44
Bob	1	3	0.67	0.83	0.71
Charlie	2.5	2	0.75	0.5	0.69

²Mathematically, this is not necessary, but it results in much more human readable values between zero and one.

³Values have been rounded off to two significant digits.

Ranking the employees only by the average value of skill regarding the two searched items would result in Charlie being preferred to Alice and Alice being preferred to Bob. Sorting them using the proposed fitness score, however, would result in Bob being recommended as the best match, because his relatively high interest in the searched skills and his specialization in them compensates his low average skill. Interestingly, Alice has the best average skill level but nonetheless gets scored the worst due to her obvious specialization in C++. In real life usage, the weighting constants w_{as} , w_{aw} , w_{ss} and w_{sw} might need to be adjusted so that the average skill plays a bigger role in the resulting score.⁴

4.1.5 Example

For this example, let the set of employees be Alice, Bob, Charlie, Donald, and Erika, and the set of all known skills be Java, Ruby, and C++. The assignment of skill/will levels and the respective locations are:

Person	Location	Java	Ruby	C++
Alice	Hamburg	2/1	2/2	3/3
Bob	Hamburg	2/3	0/3	0/1
Charlie	Hamburg	3/3	2/1	1/2
Donald	Hamburg	3/3	-	2/2
Erika	Frankfurt	1/1	2/3	3/1

Applying the algorithm and searching for employees knowing Java and Ruby in Hamburg:

(Let the weights used in the fitness score be $w_{as} = w_{aw} = w_{ss} = w_{sw} = 0.25$)

- Create a list of all employees
⇒ Alice, Bob, Charlie, Donald, Erika
- Filter by skills
⇒ Alice, Bob, Charlie, Erika
- Filter by location
⇒ Alice, Bob, Charlie
- Assign fitness scores⁵
⇒ Alice (0.44), Bob (0.71), Charlie(0.69)
- Sort by fitness score
⇒ Bob (0.71), Charlie(0.69), Alice (0.44)

⁴The need to adjust the weights should not be considered a design flaw, since the algorithm has been intentionally designed to be customizable to the users' needs.

⁵The calculation of the fitness scores can be seen in 4.1.4

4.2 Search Suggestions

After entering an item into the search bar, the user will be presented other items they are likely to enter next. This minor feature can be seen as another recommender system, because it recommends the next item from the set of all available search items and thus matches the definition given in 4.1.1. This recommender system has to deal with other limitations than the one used for the main search function. It filters a completely different set of data, namely all skills instead of all persons, and has to deal with other limitations.

4.2.1 Available Data

Distinguishable Users

Since it is not planned that users will have to log in before performing a search, there is no user context that can be used to examine a person's former interactions in order to predict their next ones and recommending it.

As the system is designed to be a web application, a cookie holding an unique identifier could be stored on the client device. The application would then use this ID to aggregate interactions made by the same person. Unfortunately, this method cannot identify a known person using another device as multiple devices will not share the same ID. Furthermore data collected about a user will be unassignable to them if they delete their devices cookies or change browsers. This approach would also need the application to inform the user that data will be stored on their devices as stated by Article 15(3) of the Telemediengesetz (TMG).

There also exist various methods to identify users without the need to store any data on their devices by examining and recognizing their devices attributes. The collected data can include factors like language settings, the used browser and its version, and the devices hardware components. All this data combined can be used to form an almost unique fingerprint which can be used to recognize a device. [WGGK16]

Another possible method would be to recognize users by examining their very own usage behaviour such as typing patterns or mouse strokes. This approach called 'user fingerprinting' does not depend on the users device and thus can be used to identify people across multiple devices and browsers. On the downside, this method can only differentiate between users typing the same word and needs a multitude of samples of each user in order to be able to recognize them. [AJL⁺05] Although device and user fingerprinting is not prohibited by law, the 'Opinion 9/2014 on the application of Directive 2002/58/EC to device fingerprinting' by the EU's Article 29 data protection working party states that a user must be informed about the fingerprinting process and be able to deny this. To sum up, the described methods to identify unique persons create an exorbitant computational effort and/or have high error rates for not being able to determine a single user across multiple devices. For the further design of the skill search recommender system, it will be assumed that there is no data about unique users, but about their entirety.

Skill Attributes

The skills are planned to be saved as simple names not enriched with any meta-information. so using content based filtering is not a trivial task. One possible approach would be to use linguistic methods to find similarities in names of skills in order to form clusters of related skills. Unfortunately, most of the skills are arbitrarily given names or acronyms, so that this form of analysis will fail to detect any meaningful attributes.

Regarding the concept of the suggestion engine, the assumption is made, that there is no context to the skills and that the only information about any skill is its name.

Aggregated Search History

Tracking which skills have been searched together can be implemented easily and generates a fair amount of data to generate potentially useful suggestions. Legally, this is not problematic if the application does neither save personal data about the users (Article 15 Telemediengesetz), nor stores information that could potentially be used to create personal usage profiles that can be matched to specific persons (Article 15(3) Telemediengesetz). Grouping skills that have been searched jointly does not stand in conflict with those regulations and requires no information about distinguishable user, so the application will store the search history.

4.2.2 Chosen Approach

Given that no user profiles exist, user based filtering and model based filtering cannot be applied. Due to the lack of metadata about the skills, content based filtering can also be eliminated as a possible approach. Item based filtering, however, does not require any data that is not given, so this approach will be used for the recommender system.

4.2.3 Concept

The application has access to the list of skills the user already entered and to a store of all previous searches. Having this information, the system will use a markov chain to predict the next item that will be entered and recommended it.

Markov Chains

Markov chains are a relatively simple tool for predicting future states of systems based on the current state. In fact, markov chains rely on the fact, that the next state of the system is only dependent on the current state, which is called the 'markov property'. In the context of the skill management application, we assume this to be true because only two states will be examined: the current state is represented by the set of all items entered in the search field; the future state is skill set of the current search plus one more item. The basic concept is to store all possible states of the system and the respective probability

of switching between from any state to any other one. Knowing the current state one can easily deduct the most probable future state. When a state transition occurs, the outgoing probabilities of the origin states can be adjusted accordingly in order to factor the transition into the prospective projections.

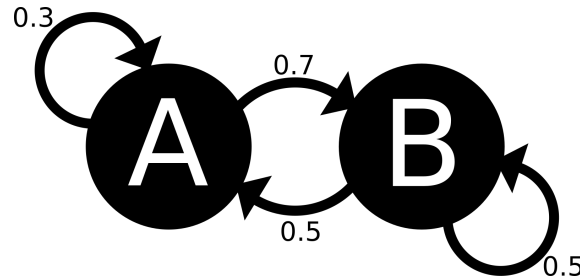


Figure 4.1: A simple markov chain displayed as graph. The states are represented as vertecies. All possible transitions between states are denoted as edges. The edge weights define the probability of the transition relative to all other outgoing edges of a node.

Data Structure and Algorithm

In order to get the best results, the recommender system should save all combinations of items that have previously been searched for together and then recommend the next one based on this exact starting point. On the other downside, this would result in a lot of different origin states that contain very few possible future states due to being too specific to a single search. The stored data can only be used if the exact same combination of skills is entered again.

Another way to implement such a recommendation system would be to only inspect the last entered search item and ignoring all other ones. Given n known skill items, all probabilities for the future state could be saved in one single $n \times n$ matrix saving only n^2 values. This solution would disregard the whole context of the search item.

As a tradeoff, the system will generate predictions for each skill in the search query independently and aggregate these predictions afterwards. For each skill, a list of possible recommendations paired with the total count of searches for both will be stored. The recommendation lists of all skills combined represent the transition matrix. Instead of transition probabilities, the total number of searches is saved in order to simplify the aggregation of multiple suggestions.

The recommender system will concatenate the suggestion lists of all items in the current search query and add up the counts of skills appearing in multiple suggestion lists. Then, all elements of the combined list that are part of the search query will be removed, the result is a list of suggestions for the whole search query.

4.2.4 Pseudo-Code

```
1 var knownSkills = [  
2   {  
3     name: "java",  
4     similar: [  
5       {  
6         name: "php",  
7         count: 3  
8       }, {  
9         name: "ruby",  
10        count: 2  
11      }  
12    ]  
13  }, {  
14    name: "php",  
15    similar: [  
16      {  
17        name: "java",  
18        count: 5  
19      }, {  
20        name: "ruby",  
21        count: 2  
22      }  
23    ]  
24    name: "ruby",  
25    similar: [  
26      {  
27        name: "java",  
28        count: 0  
29      }, {  
30        name: "php",  
31        count: 5  
32      }  
33    ]  
34  }  
35 ]
```

```

1 function suggest(searched) {
2   var accumulated = {};
3
4   for (s in searched) {
5     for (t in knownSkills.getByname(t).getSimilar()) {
6       if (accumulated.getByname(t) exists) {
7         accumulated.getByname(t).count += t.count
8       } else {
9         accumulated.getByname(t) = t.clone()
10      }
11    }
12  }
13
14  for (t in accumulated) {
15    if (searched contains t) {
16      remove t accumulated
17    }
18  }
19
20  return accumulated.getHighestCount();
21 }

```

4.2.5 Example

Transition Matrix:

	Java	PHP	CSS	COBOL
Java	-	7	3	1
PHP	7	-	9	5
CSS	3	9	-	8
COBOL	1	5	8	-

Using the given transition matrix and the search query 'Java, PHP', the algorithm would work like this:

1. Retrieve suggestion lists for each search item
 \Rightarrow (PHP (7), CSS (3), COBOL (1)), (Java (7), CSS (9), COBOL (5))
2. Aggregate lists (combine counts)
 \Rightarrow (PHP (7), Java (7), CSS (12), COBOL (6))
3. Remove suggestions that are part of the search query
 \Rightarrow (CSS (12), COBOL (6))
4. Suggest item with the highest total count
 \Rightarrow CSS

4.3 Visual Concept & Wireframes

The application should be as simple as possible and usable for everyone, in order to provide an efficient and fast tool. Thus, it will be designed as a single page application based around a people search that provides a way to input the skills needed and returns all persons offering said skills. After entering a search, the user can select any of the found colleagues and view their personal profile showing extended information like contact details, more skills the user did not search for, and the employee's location. This profile will also include links to directly contact the inspected person via Email or Google Hangouts⁶. Unlike the considered commercial solutions, this tool will not include features like creating statistics, assessments, applicant management, or any dashboard other than the basic search view. Furthermore, there will not be any different roles with different access rights for employees and their managers, since this application is meant to be a tool enhancing collaboration, not supervision.

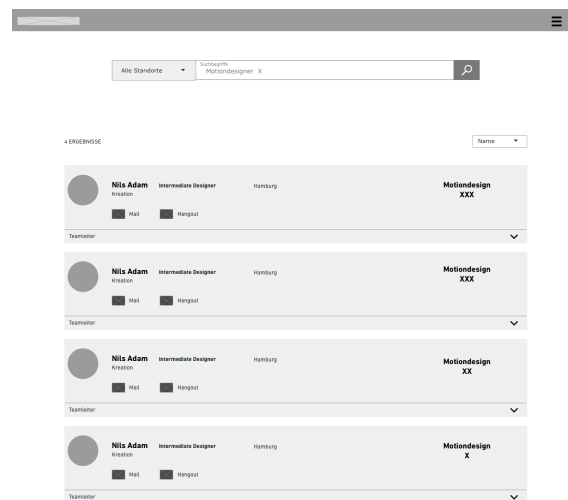
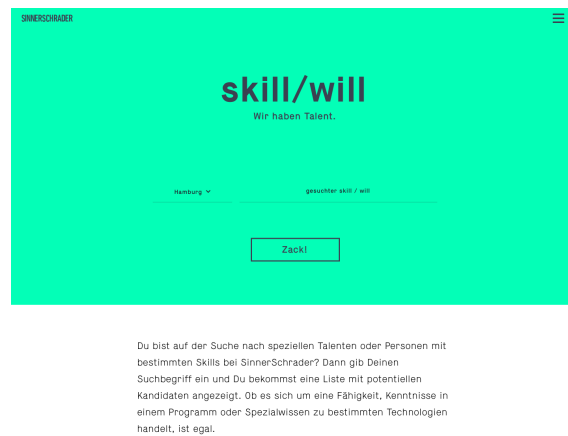


Figure 4.2: Wireframe

⁶<https://support.google.com/hangouts/answer/2944865>



The image shows a web form for SinnerSchrader. The header is dark blue with the company name 'SINNER Schrader' in white on the left and a hamburger menu icon on the right. The main content area has a light blue background. It features the text 'skill/will' in large, bold, black letters, followed by the tagline 'Wir haben Talent.' in a smaller font. Below this is a search bar with a light blue border and a light blue background. The search bar contains the text 'Suchbegriff' and a dropdown arrow. To the right of the search bar is a label 'gesuchter skill / will'. Below the search bar is a red button with the text 'Zack!' in white. At the bottom of the form, there is a paragraph of text explaining the search function.

Du bist auf der Suche nach speziellen Talenten oder Personen mit bestimmten Skills bei SinnerSchrader? Dann gib Deinen Suchbegriff ein und Du bekommst eine Liste mit potentiellen Kandidaten angezeigt. Ob es sich um eine Fähigkeit, Kenntnisse in einem Programm oder Spezialwissen zu bestimmten Technologien handelt, ist egal.

Figure 4.3: Design of the Search View

5 Implementation

5.1 Application Structure

The application consists of two main components: the frontend that presents the user with a graphical interface and the backend that provides data and actions on it to the frontend. The user's browser connects to a web server that acts as a reverse proxy and that not only provides static resources like HTML and CSS files which resemble the frontend, but also acts as a SSL endpoint. Requests for dynamic data and actions that are handled via the REST API provided by the backend are passed on to it, its response will then be directed through the reverse proxy to the client. To store and read data, the backend connects to a Mongo DB Database. User details are synced from the existing LDAP that acts a central repository for personal information of all employees. TODO: Bild einfügen

5.2 MongoDB

MongoDB¹ is a popular non-relational NoSQL Database that aims to be fast and easy to use [Mon, p. 10]. To increase performance, like many NoSQL databases, it does not provide acid transactions which are a well-known feature of relational database management systems (RDBMS). This, however, simplifies horizontal scaling since new machines can easily be inserted into an existing cluster of database servers without the need to be in sync. [Mon, p. 3]

5.2.1 BSON

In contrast to relational databases that store all data in tables, MongoDB uses a document-orient data structure saving every element in the Binary JSON² (BSON) format. This approach allows complex data to be stored as one object rather than having to dissect its elements and storing them in separate tables. As a consequence, retrieving an object from the database is much more efficient than it would be using a RDBMS, as the latter needs to join the tables storing the objects nested sub-objects and compose the requested element whereas MongoDB has it stored in the exact same form it is requested. [Mon, p. 10]

¹<https://www.mongodb.com>

²Javascript Object Notation

5.2.2 Data Structure

The application stores three different object classes in the Database: Skills that are known to the system, persons with their individual contact data and skills, and sessions used to authenticate users that wish to modify their profiles. In order to instantiate the elements as java objects, Spring Data³, the framework used for database access, also stores the class name the object needs to be mapped to as a field inside of it.

Known Skills

Skills known to the system consist of a unique name and a list of suggestions that themselves are expressed by a name and a total count of searches of the respective suggestion together with the skill.

```
1 TODO paste
```

Persons

The documents that represent persons contain the respective person's id⁴, their personal data like first and last name, telephone number, e-mail address, office location, and job title⁵, and a list of the person's skills. Each of those skills consists of a name, a level of skill and a level of will.

```
1 TODO paste
```

Sessions

Sessions are used to authenticate users that wish to modify their personal profile. The client has to authenticate the user with their credentials; if this is successful, a new session holding a unique id, the point of time it will expire, and the id of the authenticated user, will be created and stored in the Database.

```
1 TODO paste
```

5.2.3 Queries

As shown in 5.2.1, the document based data structure of MongoDB allows the database to efficiently perform complex requests. Furthermore, it provides simple and straightforward search queries to retrieve objects based on their attributes. For example, getting all users who offer the skill 'Ruby' from the collection 'person' can be done with this query:

```
1 db.person.find({ "skills._id" : "Ruby" })
```

³<http://projects.spring.io/spring-data/>

⁴Each employee gets assigned an internal id ('Benutzerkürzel') that is globally used to uniquely identify a person.

⁵The job title data is not maintained consistently in the LDAP, so that, unfortunately, it is not suitable to be used in the person search.

5.3 LDAP

SinnerSchrader runs a LDAP server which acts as a centralized source of personal information of all employees. The application connects to this server in order to retrieve contact information to display in users' profiles. In comparison with having the users to enter their data manually, this method has the benefit that the users' data will be kept in sync across all internal services, and that it reduces the effort a user has to spend to create their profile.

5.4 Reverse Proxy

Between the client and the backend, an intermediary web server that acts as a reverse proxy is switched in. Its main purpose is the distinguishing between requests for static files, like HTML and CSS content that will be directly delivered by said server, and API calls that are redirected to the backend. This increases the system's security by protecting the backend server's identity and presenting an additional defense layer. [Inc]. Furthermore, this server can handle SSL encryption between the application and the client, and, if multiple backend servers are needed, balance the workload between them while presenting them as one uniform service.

5.5 API

To exchange data between the backend and the frontend, a Representational State Transfer (REST) API is provided by the backend. Its endpoints are called by the frontend code to either request data or to command the backend to perform modifying operations on it. The used HTTP method is the main indicator of the action to perform: GET is used to retrieve data, POST to insert new elements, PUT to modify existing ones and DELETE to remove them. The URLs of the individual action express the entity on which the action will be performed.

TODO: Swagger Table paste und in text verweisen

5.6 Backend

The backend component is implemented in Java ⁶ using the Spring Boot framework⁷. Maven⁸ is employed to manage the build process and run unit and integration tests.

⁶<https://go.java/>

⁷<https://projects.spring.io/spring-boot/>

⁸<https://maven.apache.org/what-is-maven.html>

5.7 Architecture

The software architecture consists of three main categories of classes: services handling data manipulation and filtering that hold the business logic, repository objects that wrap the database operations into easy-to-use handlers, and domain specific data types. Additionally, numerous helper classes like custom exception types, comparators and general utilities are implemented.

// TODO UML und paste

5.7.1 Spring Boot

Spring Boot is a highly sophisticated web framework that provides numerous features to create web applications including, but not limited to, annotations to expose java methods as HTTP request endpoints, an embedded webserver to run the application on, and a modular design to extend its features. It is used because its credo to provide default configurations where possible and thus reduce the need to write infrastructure code simplifies the applications structure.[Gut16, p. 6] For example, a controller that returns a static response can be created using two annotations: '@Controller' to make spring boot identify the class as a resource that will listen to HTTP calls, and '@Request' to specify the URL and HTTP method to use. Unlike most other web frameworks, spring boot does not require any more configuration or dispatching classes.

```
1 @Controller
2 public class HTCPImpl {
3
4     @RequestMapping(path = "/coffee", method = RequestMethod.GET)
5     public ResponseEntity<String> coffee() {
6         StatusJSON json = new StatusJSON("I'm_a_teapot_\u2615");
7         return new ResponseEntity<String>(json.toString(), HttpStatus.I_AM_A_TEAPOT);
8     }
9
10 }
```

flapdoodle, unboundid embedded, junit,

6 Evaluation

TODO

6.1 Algorithm vs. Leute (aka interviews)

TODO

6.2 Scores gleichverteilt?

TODO

6.3 Implementation vs. Algorithm

TODO

6.4 Meeting the Requirements

6.4.1 FUntional Requirements

6.4.2 Nonfuntional Requirements

6.5 Misc

TODO

6.6 Conclusion

War das jetzt was?

Bibliography

- [AJL⁺05] ARAÚJO, MLívia C. F. ; JR., Luiz H. R. S. ; LIZÁRRAGA, Miguel G. ; LING, Lee L. ; YABU-UTI, João B. T.: *User Authentication Through Typing Biometrics Features*. 2005
- [Ber16] BERGMANN, Rainer: *Organisation und Projektmanagement*. 2. Aufl. 2016. Springer Gabler, 2016 (BA KOMPAKT). <https://beluga.sub.uni-hamburg.de/vufind/Record/866785566>
- [Can13] CANÓS-DARÓS, Lourdes: An algorithm to identify the most motivated employees. In: *Management Decision* 51 (2013), Nr. 4, 813-823. <http://dx.doi.org/10.1108/00251741311326581>. – DOI 10.1108/00251741311326581
- [DMA14] DMA: *National email benchmarking report 2013*. November 2014 Edition. 2014
- [Gut16] GUTIERREZ, Felipe: *Pro Spring Boot - A no-nonsense guide containing case studies and best practices for Spring Boot*. Springer Science+Business Media, 2016. – ISBN 978-1-4842-1431-2
- [IFO15] ISINKAYE, F.O. ; FOLAJIMI, Y.O. ; OJOKOH, B.A.: Recommendation systems: Principles, methods and evaluation. In: *Egyptian Informatics Journal* 16 (2015), Nr. 3, 261 - 273. <http://dx.doi.org/http://dx.doi.org/10.1016/j.eij.2015.06.005>. – DOI <http://dx.doi.org/10.1016/j.eij.2015.06.005>. – ISSN 1110-8665
- [Inc] INC., NGINX: *WHAT IS A REVERSE PROXY SERVER?* <https://www.nginx.com/resources/glossary/reverse-proxy-server/>
- [JHS07] JANET H. SPOONAMORE, Ricky D. H. H. James Simien S. H. James Simien: *Person-to-Position Matching*. 2007
- [Leh04] LEHNER, Franz: *Marktanalyse zum Angebot an Skill-Management-Systemen*. 2004
- [Mon]
- [RD00] RYAN, Richard M. ; DECI, Edward L.: Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. In: *Contemporary Educational Psychology* 25 (2000), Nr. 1, 54 - 67. [http://dx.doi.org/10.1016/S0190-5602\(00\)00015-1](http://dx.doi.org/10.1016/S0190-5602(00)00015-1)

doi.org/http://dx.doi.org/10.1006/ceps.1999.1020. – DOI
http://dx.doi.org/10.1006/ceps.1999.1020. – ISSN 0361–476X

- [WGGK16] WANG, Cliff ; GERDES, Ryan M. ; GUAN, Yong ; KASERA, Sneha K.: *Digital Fingerprinting*. New York, NY s.l. : Springer New York, 2016. – 189 S. <https://beluga.sub.uni-hamburg.de/vufind/Record/871476827>
- [Wil10] WILSON, Carol: *Bruce Tushman's forming, storming, norming & performing team development model*. 2010
-

List of Figures

1.1	Matrix Organisation	2
3.1	SkillsBase Dashboard	8
3.2	Talent Management Search	9
4.1	Markov Chain	20
4.2	Wireframe	23
4.3	Home View Design	24

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____