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
B.Sc. Software Systems Development
Matr.-Nr. 6524064
Semester 07

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

February 2017

Abstract

Project driven organizations have to face the problem of constantly needing to put teams together based on the members' skills, experience, motivation and preferences. At SinnerSchrader, like in many businesses, there is no central source of information about this data. Commercial solutions only take into account the employee's knowledge whereas motivation and preferences will be ignored. This thesis covers the design and partial implementation of a web application that provides a search function to find the most suitable employee for a searched skill set based not only on their skills but also on their motivation. Usage scenarios and requirements that should be fulfilled by the system will be outlined. The design and realization of the application's backend based on those requirements will be drafted and evaluated. To validate that the scoring algorithm created for this application is capable of fulfilling the users' needs, a survey will be conducted and interpreted.

Contents

1 Introduction

1.1 Motivation

Project driven organizations 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 elaborate 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 an employee skilled in a given set of tasks in all availble 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 said application.

1.1.1 SinnerSchrader

The SinnerSchrader Group is a full-service web agency based in Hamburg aggregating the sub companies 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 customers' digital transformation. The combination of all said competencies under one single roof reduces organizational 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.

Matrix organization

The personnel of SinnerSchrader is divided into two different types of teams: functional teams of employees sharing the same specialization, e.g. 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 organization [Ber16, P. 75]. The organizational head of functional teams will further be called the *supervisor* the pendant for project team will be mentioned as *team manager*.

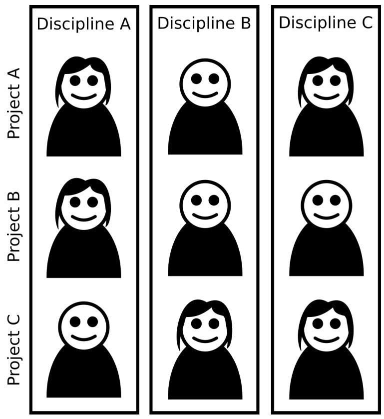

Figure 1.1: Illustration of a matrix organization

1.2 Leading Goals

This thesis will discuss the requirements a skills management software 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

Skills management is a trending topic in today's management world, as it is a vital part of the success of a modern business. Darvish et al. highlight the importance of knowledge management in various forms of institutions and compare different knowledge management strategies. Furthermore, the authors show how enterprises can introduce tools and mindsets to use the positive effects of knowledge management in their organizations [DAQ13], but do not discuss concrete software tools. A market analysis by Lehner, however, compares multiple commercially available software systems for skills management and provides information adapted in 3.3 [Leh04].
Beck outlines a case study that deals with the introduction of a skills management system at *Putzmeister GmbH* and reveals important success factors of such a software, e.g. ways to motivate employees to provide a sufficient amount of data, legal concerns and cooperation with the industrial council (*Betriebsrat*), obstacles that occur in the maintenance phase of the system's lifecycle, and usability requirements [Bec03].
In contrast to the systems analyzed by Lehner and Beck, the application that this thesis will deal with does not only show the information saved in its database, but also provides a powerful search function and recommends employees based on the combination of multiple personal factors including their motivation.
The concept of approaching team building and management challenges with algorithms has been evaluated and successfully implemented multiple times. In 2013, Ivanovksa et al. compared various data mining algorithms for the automatic composition of teams and propose the usage of *Bayesian Networks* for this kind of problems [IIK13]. Unfortunately, the authors do not take into account factors like the employees' motiviation and satisfaction. Those aspects have been examined by Canós-Darós who introduced an algorithm to measure employees' motivation [Can13] and highlights aspects the algorithm used by the application should include. Spoonamore et al. created an algorithm that deals with the matching of personnel to open positions in the *United States Navy* [SSH07]. An adaption of this algorithm lays the foundation for the scoring algorithm used in this application (see 4.1.4). Furthermore, the authors describe non-technical requirements an algorithm which ranks personal abilities has to meet in order to be accepted by the target audience that will be scored by it.
This thesis does not cover the visual concept and implementation of the applications graphical user interface, since Strecker's bachelor's thesis addresses this field of functionality [Str17].

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 the working culture in many knowledge driven businesses, so collaboration and the sharing of ideas are a major factor in the company guidelines of SinnerSchrader. The application is supposed to act as a central repository for knowledge and contacts, enabling employees to find someone who can help in order to answer questions and find solutions to domain-specific problems.

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 at SinnerSchrader, managers often do not find the most 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 their skills. 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 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 to accommodate the human factors of self-perception. A case study performed by Beck indicates that this approach to collect information about the employees' skills generates a sufficient amount of data to run such a system [Bec03].

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

In contrast to the supervisor, the application cannot be able to find the best situations to notify 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 the *Direct Marketing Association (UK)*[1], only about one in five automatically generated e-mails will be opened [DMA14], which reflects SinnerSchraders experiences with email reminders. Those facts justify the decision, that the system will not send any notification to its users.

[1] https://dma.org.uk/

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 is 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

- Accessible to all Employees
  Every employee must be able to use the application regardless of their equipment or preferred 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 skill level and a will level, both on four step scale.

- 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 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 specialization in the searched skills.
  - Sorting
    The user should be able to sort the search results not only by said score but also 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 users' profiles is maintained at all times.

3.2.2 Non-Functional Requirements

- Device Types
  Even tough smartphones are gaining more and more share in terms of total internet traffic [Sta], the application will be optimized for desktop use. Every employee has permanent access to their computer and uses it for their work, so it is assumed, that the very same machines will be used to access the skill management application.

- Browsers
  Every employee is allowed and encouraged to install their favorite software on their personal computer, so nearly every web browser can be found. The application should run on *Google Chrome*[2], *Mozilla Firefox*[3] and *Safari*[4] in their latest versions. *Internet Explorer*[5] and *Edge*[6] will not be supported.

- Scalability
  SinnerSchrader has 459 full-time employees [Sin17], so the application will be designed for approximately 500 users. In the event of a rapid growth in the userbase, e.g. due to the opening of another office, the system will have to scale up to handle the larger number of users.

- Load/Response Times
  According to *Google's* RAIL model, a website needs to respond to the user's input within 100ms to offer a fluent user experience; the triggered action should be finished within one second after the user's interaction [Kea17]. In the context of the application's search function, this means that the system will show the user that their input has been acknowledged within 100ms. Within one second after the submission of the search request, the result list will be rendered completely.

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 Lehner shows that most tools provide a spectrum of features and limitations very similar to the examined solutions [Leh04], so that the selection is assumed to be representative of the market.

[2] https://www.google.com/chrome/browser/desktop/index.html
[3] https://www.mozilla.org/en-US/firefox/products/
[4] http://www.apple.com/lae/safari/
[5] https://www.microsoft.com/en-us/download/internet-explorer.aspx
[6] https://www.microsoft.com/en-us/windows/microsoft-edge

3.3.1 Skills Base

Skills Base[7] offers most of 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 cannot be customized and does not include the employees' motivation, which is a central point of the requirements. A central asprect of the application are dashboards displaying information about the most popular skills in the organization and long term statistics.

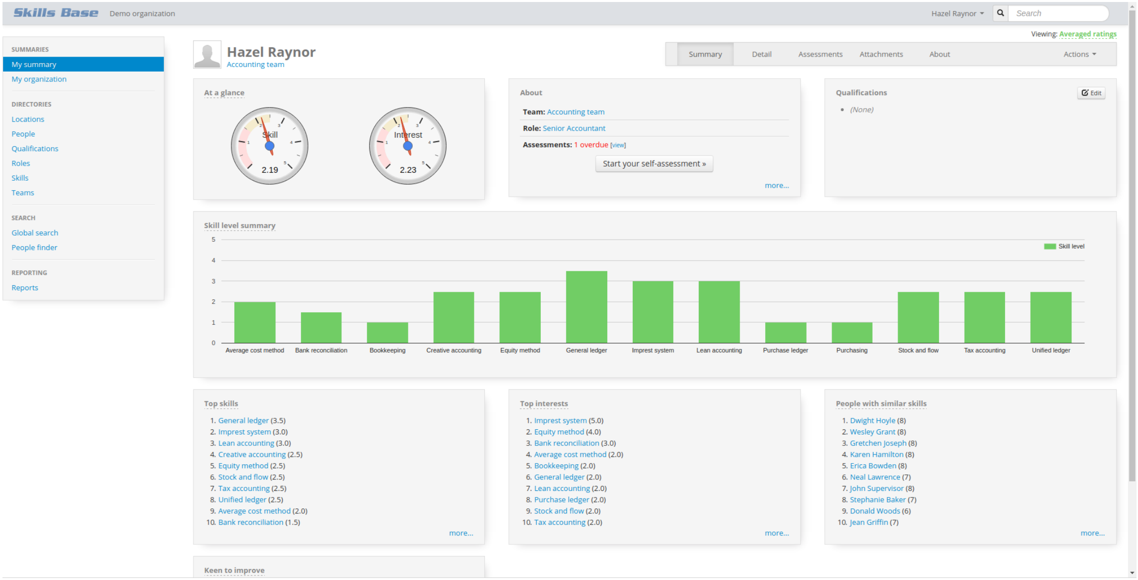

Figure 3.1: SkillsBase Dashboard

3.3.2 Talent Management (engage!)

Talent Management[8] is a module for *Infoniqa's* management software engage![9]. 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, but lacks the possibility to register motivation. It can only be used in combination with engage!, a complete human resources management solution including features like time tracking, e-learning, applicant management and payroll accounting.

[7]http://www.skills-base.com/
[8]http://www.infoniqa.com/hr-software/skill-management
[9]http://www.infoniqa.com/hr-software/personalmanagement

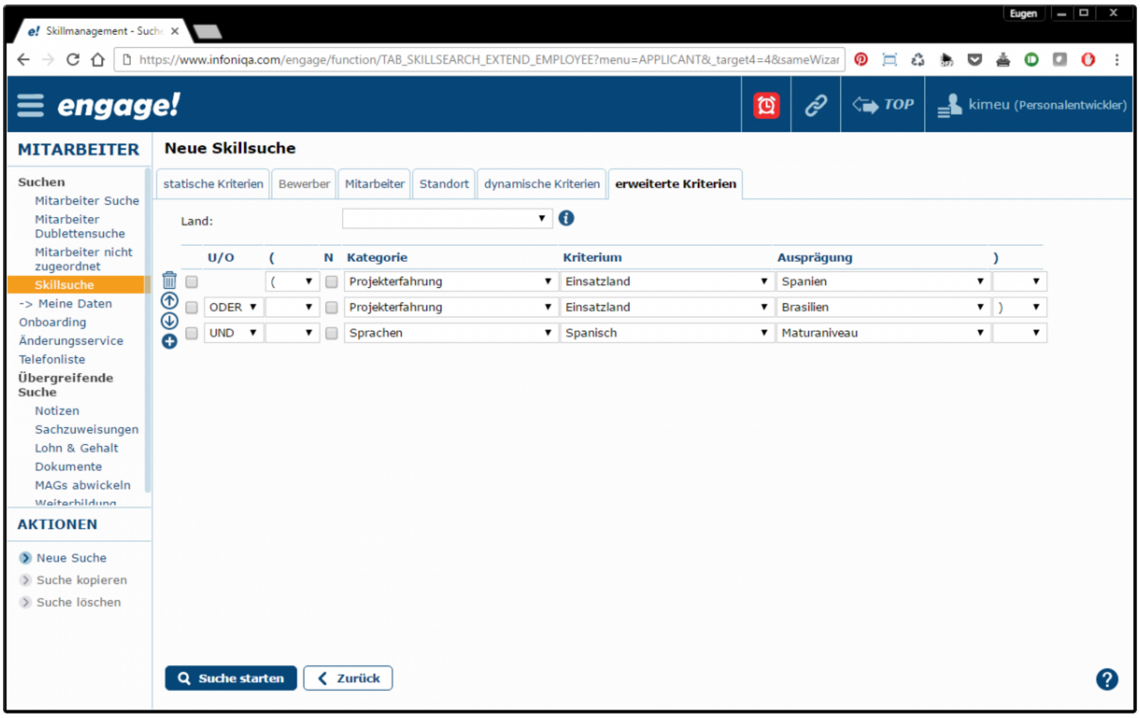

Figure 3.2: Talent Management Search

3.3.3 SkillsDB Pro

SkillsDB Pro[10] is an application designed to serve as a database in an organization, providing an overview of every person's own skills and training only to themselves and their supervisor. 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 supervisors can also do this with the limitation that no employee can see their supervisor's rating about themselves. Information about motivation, like in the other examined systems, cannot be captured. Furthermore, only privileged like supervisors 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 Pro also offers features SinnerSchrader does not intend to use including the automatic generation of project reports based on plan succession and demands for assessments.

[10]http://www.skillsdbpro.com

3.3.4 Conclusion

None of the analyzed applications offers all required features, 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 both knowledge and motivation is not offered by any of the commercial solutions. Furthermore, all those systems differentiate between employees and their supervisors and thus restrain transparency. The application is not supposed to be used for monitoring and rating employees, but should give employees the possibility to find each other; categorizing them would clearly defeat this purpose.

Pain Point Fitness Scoring

As shown by Canós-Darós, motivation is a vital factor regarding any employee's performance and quality of work [Can13]. Although motivation is a complex construct of many highly diverse dimensions, the overlaping 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. Consequently, when searching for persons having specific skills, the application should not only take into account the employees' skills but also 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 the users' 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 smartphones 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 a positive factor regarding the overall usability of the system and assures bugs and security issues are eliminated the moment a new 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 choice.

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] and 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 fancy. In this application's context, the recommender system will filter the set of all employees and recommend better matching results by showing them first.

4.1.2 Techniques of Content Filtering

As described by Isinkaye et al., filtering techniques used in recommender systems are divided in three classes: content based, collaborative, and hybrid. Each of this classes relies on a different approach for gaining data by which the information is filtered [IFO15].

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 deductible from other users' former interactions.

- Model Based Filtering
  Model based filtering applies methods of machine learning and data mining to learn a precomputed 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 examined 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 the 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 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' interactions with search results, e.g. 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. This fitness score takes into account the user's skill/will levels and their specializations.
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

```
function search(searchItems, searchLocation) {
  var results = getAllEmployees()

  for (Employee e in results) {
    if (e.skills does not contain all elements of searchItems) {
      results.remove(e)
    }
  }

  for (Employee e in results) {
    if (e.location is not searchLocation) {
      results.remove(e)
    }
  }

  for (Employee e in results) {
    e.assignFitnessScore()
  }

  results = results.sortByFitnessScore()

  return results
}
```

Figure 4.1: Pseudo-Implementation of the search algorithm

4.1.4 Scoring Algorithm

The application will sort all found persons by their fitness into the searched skill set; this fitness will be scored on a scale from zero (worst) to one (best). The requirements of the algorithm calculating this fitness and its design will be explained in this section.

Requirements

According to Spoonamore et al., an algorithm that matches persons to positions based on their skills has to meet more demands than solely the functional ones [SSH07, P. 14]. 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 NECs[1] and future taxonomies characterizing required knowledge, skills and abilities

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 deduced: it may not be too complex as employees must 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.

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. The latter can be described as the skill and will levels that are higher than the person's average level. 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 from 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.

$$S = \{Java, Ruby, C++, ...\}$$
$$E = \{x \in S \mid \text{employee has skill x}\}$$
$$Q = \{x \in S \mid \text{user searches for skill x}\}$$

[1] Navy Enlisted Classifications

The function v_s assigns a value of skill to any item in E; the function v_w assigns the respective level of will to any item in E.

$$v_s : E \mapsto V$$
$$v_w : E \mapsto V$$

Those values map to defined terms that describe the person's knowledge or interest:

Value	v_s	v_w
0	novice	uninterested
1	basic knowledge	indifferent
2	advanced knowledge	somewhat interested
3	expert	highly interested

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 and are 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 the lowest ratings in their other items would have the greatest specialization possible and thus get assigned a value of one.

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

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

The resulting fitness score f is a weighted mean of the introduced factors. The weights $w_{as}, w_{aw}, w_{ss}, w_{sw}$ are positive real numbers and sum up to one.[2]

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

$$f = \frac{w_{as} \cdot a_s}{max(V)} + \frac{w_{aw} \cdot a_w}{max(V)} + w_{ss} \cdot s_s + w_{sw} \cdot s_w$$
$$w_{as} + w_{aw} + w_{ss} + w_{sw} = 1$$
$$w_{as}, w_{aw}, w_{ss}, w_{sw} \in \mathbb{R}^+ \cup \{0\}$$

Example Calculation

Let there be three example employees, *Alice*, *Bob* and *Charlie*, having 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 search for the skills *Java* and *Ruby* results in the following values[3]:

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

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]

[3] Values have been rounded off to two significant digits.

[4] The need to adjust the weights should not be considered a flaw since the algorithm has been intendionally designed to be customizable to the users' needs as defined in 4.1.4.

4.1.5 Example Search

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

Let the weights used in the fitness score be $w_{as} = w_{aw} = w_{ss} = w_{sw} = 0.25$. Applying the algorithm and searching for employees knowing *Java* and *Ruby* in *Hamburg*:

- Create a list of all employees
  $\Rightarrow$ Alice, Bob, Charlie, Donald, Erika

- Filter by skills: Donald gets eliminated because he does not have the skill *Ruby*
  $\Rightarrow$ Alice, Bob, Charlie, Erika

- Filter by location: Erika works in Frankfurt and thus will be excluded
  $\Rightarrow$ Alice, Bob, Charlie

- Assign fitness scores[5]
  $\Rightarrow$ Alice (0.44), Bob (0.71), Charlie (0.69)

- Sort by fitness score
  $\Rightarrow$ Bob (0.71), Charlie (0.69), Alice (0.44)

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 uses a different filtering approach.

[5]The calculation of the fitness scores can be found in 4.1.4

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 and recommend their next one.
As the system is designed to be a web application, a cookie holding a 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 because multiple devices will not share the same ID. Furthermore, data collected about a user will be discarded if they delete their devices' cookies or switch 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). The user has to give their approval and must be able to refuse the saving of their data (BDSG, Section 4a).
There also are 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 hardware components of the device. All this data combined can be used to form an almost unique fingerprint suitable to recognize a device [WGGK16].
Another possible method would be to recognize users by examining their very own behavior such as typing patterns or mouse strokes. This approach called *user fingerprinting* does not depend on the user's 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, it needs a multitude of samples of each user in order to be able to recognize them, and it is very failure-prone [AJL+05]. Although device and user fingerprinting are 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 metainformation, 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 create clusters of related skills. Unfortunately, most of the skills' names are arbitrarily chosen 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 creates 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

Assuming 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 available, 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 repository of all previous searches. Having this information, the system will use a markov chain to predict the next item that the user is likely to enter and recommend it to them.

Markov Chains

Markov chains are a relatively simple tool for predicting future states of systems based on the current state. In fact, makrov chains rely on the fact, that the next state of the system is exclusively dependent on the current state, which is called the *markov property* [Sie]. In the context of the skill management application, this is assumed 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 the skill set of the current search plus the item the user is going to enter next. The basic concept is to store all possible states of the system and the respective probability of switching from any state to any other one. Knowing the current state one can easily deduce the most probable future state. When a state transistion occurs, the outgoing probabilities of the origin states can be adjusted accordingly in order to factor the transition into the prospective projections.

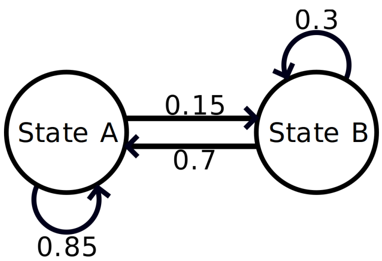

Figure 4.2: A simple markov chain displayed as a graph. The states are represented as vertices. All possible transitions between states are denoted as edges. The edge weights define the probability of the transition relative to all other outgoing edges of a node.

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 downside, this would result in a lot of different origin states that contain very few possible future states due to being overly 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 engine 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 afterward. 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

```
var knownSkills = [
  {
    name: "java",
    similar: [
      {
        name: "php",
        count: 3
      }, {
        name: "ruby",
        count: 2
      }
    ]
  }, {
    name: "php",
    similar: [
      {
        name: "java",
        count: 5
      }, {
        name: "ruby",
        count: 2
      }
    ]
    name: "ruby",
    similar: [
      {
        name: "java",
        count: 0
      }, {
        name: "php",
        count: 5
      }
    ]
  }
]
```

Figure 4.3: Pseudo-Implementation of the known skills' data structure

```
function suggest(searched) {
  var accumulated = {};

  for (s in searched) {
    for (t in knownSkills.getByName(t).getSkillsSearchedWith()) {
      if (accumulated.getByName(t) exists) {
        accumulated.getByName(t).count += t.count
      } else {
        accumulated.getByName(t) = t.clone()
      }
    }
  }

  for (t in accumulated) {
    if (searched contains t) {
      remove t from accumulated
    }
  }

  return accumulated.getHighestCount();
}
```

Figure 4.4: Pseudo-Implementation of the suggestion of a known skill based on a set of already searched items.

4.2.5 Example

Let there be the following 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 works 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

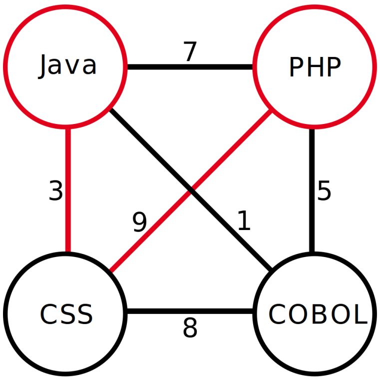

Figure 4.5: Markov Model used in the example. The two origin states and the transitions that form the end result are highlighted red.

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 search function 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 e-mail or *Google Hangouts*[6]. Unlike the considered commercial solutions (see 3.3), 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 role model to differentiate between "regular" employees and managers since this application is meant to be a tool enhancing collaboration, not supervision. More information about the concept behind the visual design and the frontend's implementation is provided by Strecker [Str17].

[6]https://hangouts.google.com

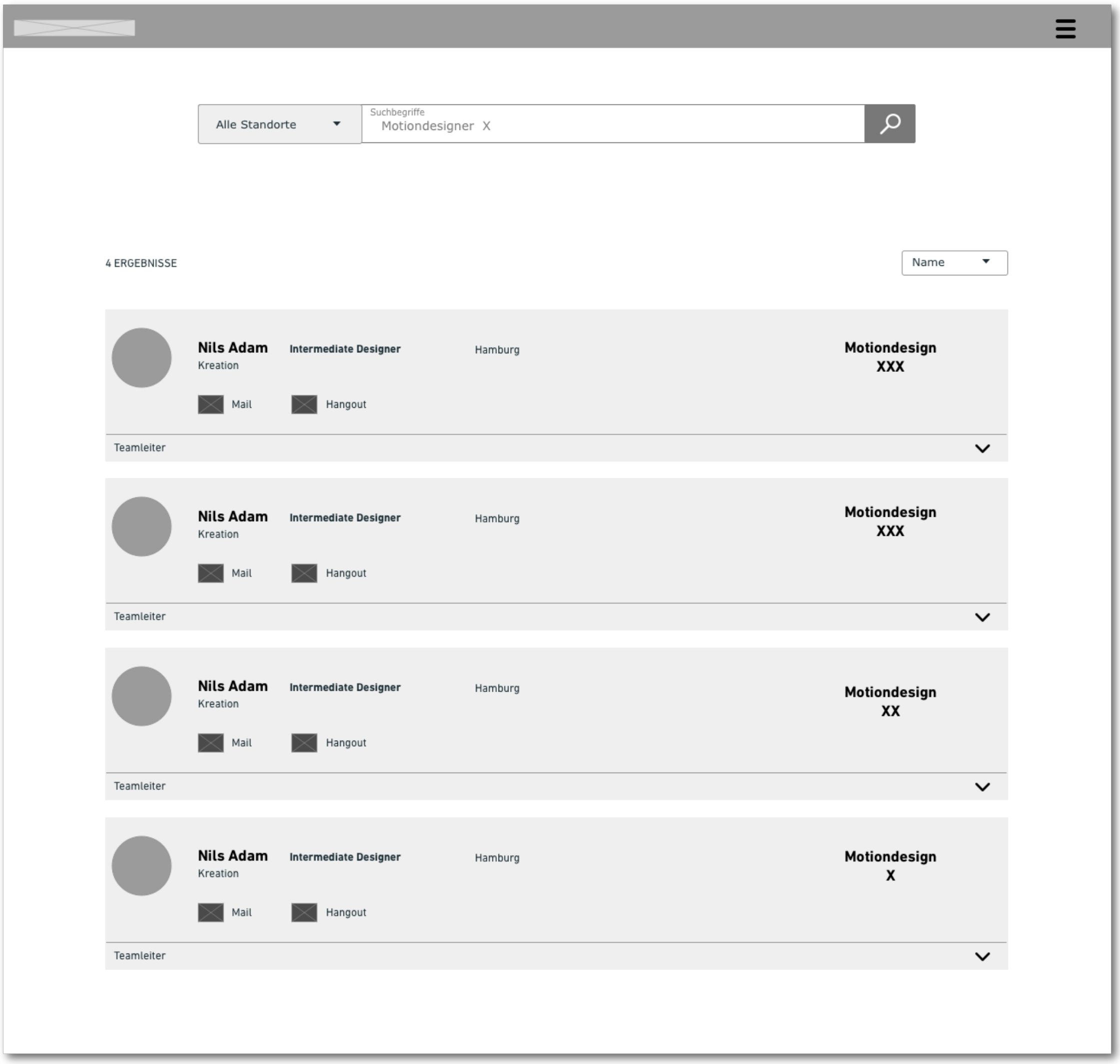

Figure 4.6: Wireframe of the search result view

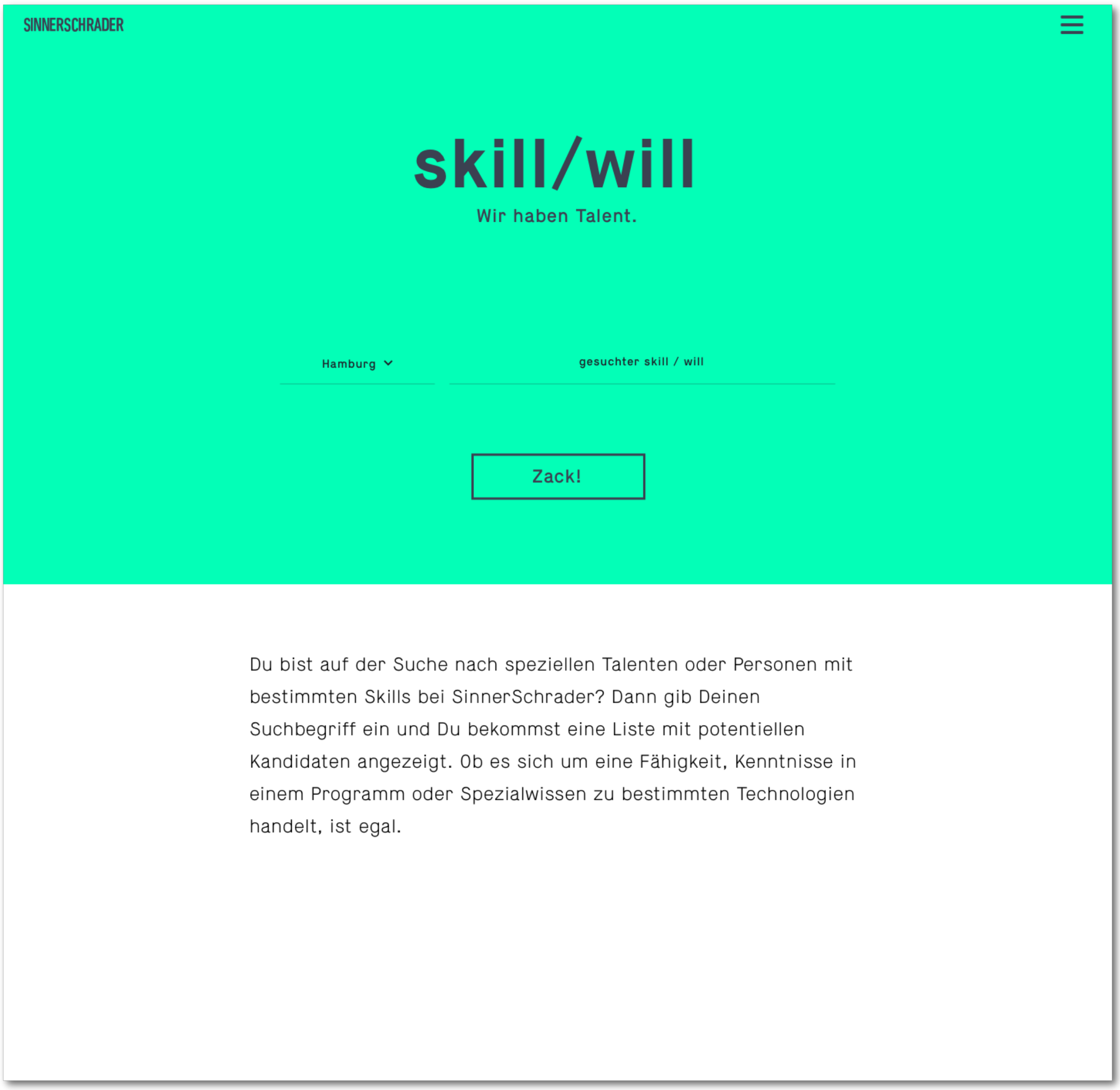

Figure 4.7: An early prototypical design for the search view

4.4 Legal Concerns

Nearly all of data fall into the category of personal data (*Personenbezogene Daten*), which is defined as "any information concerning the personal or material circumstances of an identified or identifiable individual (the data subject)" (BDSG, 3(1)). The personal data will be collected (BDSG, 3(3)), processed (BDSG, 3(4)) and transferred (BDSG, 3(3)) to other employees of SinnerSchrader. Generally, this does not violate any law, since the "collection, storage, modification or transfer of personal data or their use as a means of fulfilling one's own business purposes shall be admissible" (BDSG, 28(1)), but some restrictions apply: the data subjects have to be informed about the processing of their personal data, they must be able to deny their consent (BDSG, Section 4a), and the personal data shall not be made public. To ensure the latter, the application must not be accessible to persons that do not work for or on behalf of SinnerSchrader. Technically, this will be arranged by making the application attainable from SinnerSchrader's internal network only, which can exclusively be used by employees and authorized persons.

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 not only provides static resources like HTML and CSS files which resemble the frontend, but also acts as an 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 MongoDB Database. User details are synced from the existing LDAP server which acts a central repository for personal information of all employees.

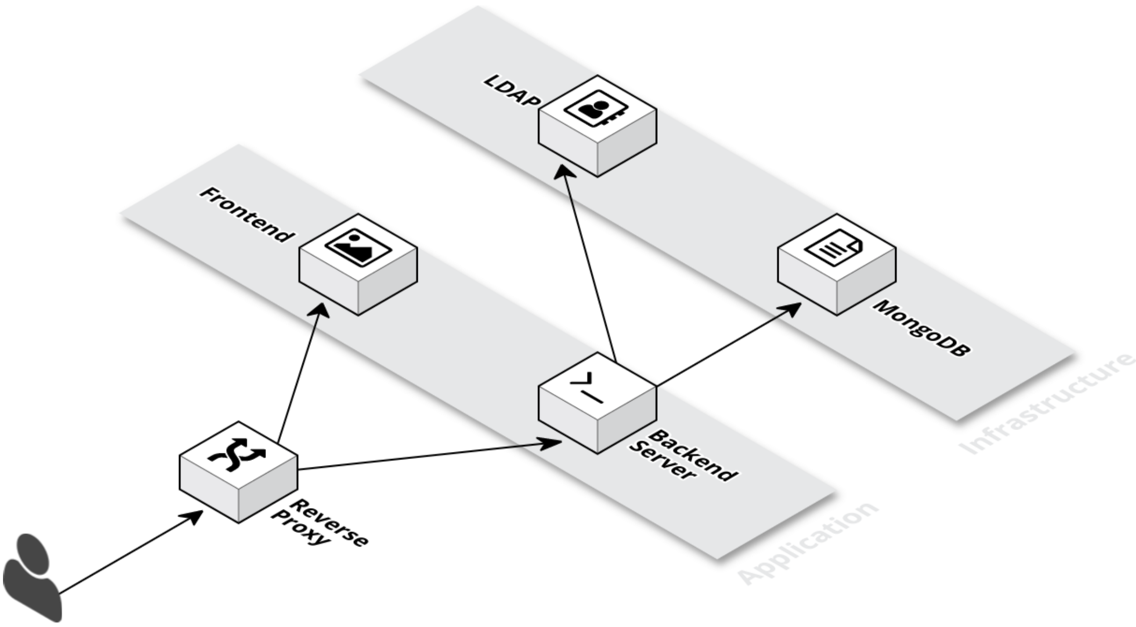

Figure 5.1: The system's architecture. Created with *https://cloudcraft.co*.

5.2 MongoDB

MongoDB[1] is a popular non-relational NoSQL database that aims to be fast and easy to use [HMPH15, p. 10]. To increase performance, like many NoSQL databases, it does not provide ACID[2] 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 [HMPH15, 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 JSON[3] (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 an RDBMS, as the latter needs to join the tables storing the object's nested sub-objects and compose the requested element whereas MongoDB has it stored in the exact same form it is requested [HMPH15, p. 10].

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 personal skills, and sessions used to authenticate users that wish to modify their profiles. In order to instantiate the elements as java objects, Spring Data[4], the framework used for database access, also stores the class name the object needs to be mapped to as a field inside of it. Furthermore, every item has the field *version* which holds a version number used to resolve writing conflicts that may occur when multiple threads access the same object simultaneously.

[1] https://www.mongodb.com
[2] Atomicity, Consistency, Isolation, Durability
[3] Javascript Object Notation
[4] http://projects.spring.io/spring-data/

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. This list will be used to predict the next item a user is likely to enter as described in 4.2.

```
{
  "_id" : "Java",
  "_class" : "[...].skills.KnownSkill",
  "suggestions" : [
    {
      "name" : "AEM",
      "count" : 1
    }, {
      "name" : "jquery",
      "count" : 1
    }
  ],
  "version" : NumberLong(3)
}
```

Figure 5.2: Data structure of a skill stored in the database

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.

```
{
    "_id" : "87163f310f124830bac677fe31484262",
    "_class" : "com.sinnerschrader.skillwill.session.Session",
    "username" : "foobar",
    "expireDate" : ISODate("2017-01-09T08:36:40.128Z"),
    "version" : NumberLong(1)
}
```

Figure 5.3: Data structure of a session stored in the DB

Persons

The documents that represent persons contain the respective person's id[5], their personal data like first and last name, telephone number, e-mail address, office location, job title[6], 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.

```
{
  "_id" : "foobar",
  "_class" : "com.sinnerschrader.skillwill.domain.person.Person",
  "skills" : [
    {
      "_id" : ".NET",
      "skillLevel" : 1,
      "willLevel" : 2
    },{
      "_id" : "Scrum",
      "skillLevel" : 3,
      "willLevel" : 1
    }
  ],
  "version" : NumberLong(1),
  "ldapDetails" : {
    "firstName" : "Fooberius",
    "lastName" : "Bartels",
    "mail" : "foobar@mail.org",
    "phone" : "+49 12 345678 901",
    "location" : "Hamburg",
    "title" : "Development"
  }
}
```

Figure 5.4: Data structure of a person stored in the DB

[5]Each employee gets assigned an internal ID (*Benutzerkürzel*) that is globally used to uniquely identify a person.

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

5.2.3 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 straightforward query:

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

Figure 5.5: MongoDB query to retrieve all users with the skill *Ruby*

5.3 LDAP

SinnerSchrader runs an LDAP server which acts as a centralized source of personal information of all employees to provide all internal tools with data. 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 [NGI]. 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 one uniform service to the outside.

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 fronted 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. All API endpoints are listed in table 5.1.

URL	HTTP Method	Non-URL Parameters	Return Statuses	Comment
/login	POST	username, password	200, 401, 500	Try to login a user; returns session key
/logout	POST	session	200, 401, 500	Logout a session
/skills	GET	search	200, 500	Search for autocompletion; returns all skills if search is empty
/skills	POST	name	200, 400, 401, 500	Add new skill with the given name
/skills/next	GET	search	200, 400, 500	Suggest a skill based on the comma separated list of skills (parameter: search)
/skills/{skill}	DELETE		200, 400, 401, 404, 500	Remove the skill with the given name
/skills/{skill}	PUT	name	200, 400, 401, 404, 500	Rename the skill
/users	GET	skills, location	200, 400, 500	Get all users matching the searched skills in the given location (parameters may be empty)
/users/{user}	GET		200, 404, 500	Return the specified user
/users/{user}/skills	POST	session, skill, skill_level, will_level	200, 400, 401, 404, 500	Create new skill/modifiy existing personal skill
/users/{user}/skills	DELETE	session, skill	200, 400, 401, 404, 501	Remove the skill from the users profile

Table 5.1: All API endpoints provided by the backend

5.5.1 API Responses

The API returns data in the JSON format, which is one of the two de-facto standards for data exchange on the web[7] because it is part of the Javascript (JS) language [Smi15, p. 37]. Approximately 94% of all websites use JS [QS]; since JSON directly represents JS objects and is both easy to parse and human-readable, it became the leading data format for web applications. For every HTTP request, the backend will return a JSON response, notwithstanding the request may not demand data to be returned. In this case, the response will contain status information about the success of the requested action.

```
{
    "id" : "foobar",
    "firstName" : "Fooberius",
    "lastName" : "Bartels",
    "mail" : "foobar@mail.org",
    "phone" : "+49 12 345678 901",
    "location" : "Hamburg",
    "title" : "Development",
    "skills" : [
        {
            "name" : ".NET",
            "skillLevel" : 1,
            "willLevel" : 2
        }, {
            "name" : "Scrum",
            "skillLevel" : 3,
            "willLevel" : 1
        }
    ]
}
```

Figure 5.6: Example JSON response by the API for the request */users/foobar*. For comparison, the corresponding database entry is shown in 5.2.2.

```
{
    "message" : "logout successful"
}
```

Figure 5.7: Example JSON response for a request that does not demand any data.

[7]The other one is *XML* used by the *Simple Object Access Protocol*

5.6 Backend

The backend component is implemented in Java 8[8] using the Spring Boot framework[9]. Maven[10] is employed to manage the build process and run unit and integration tests.

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.

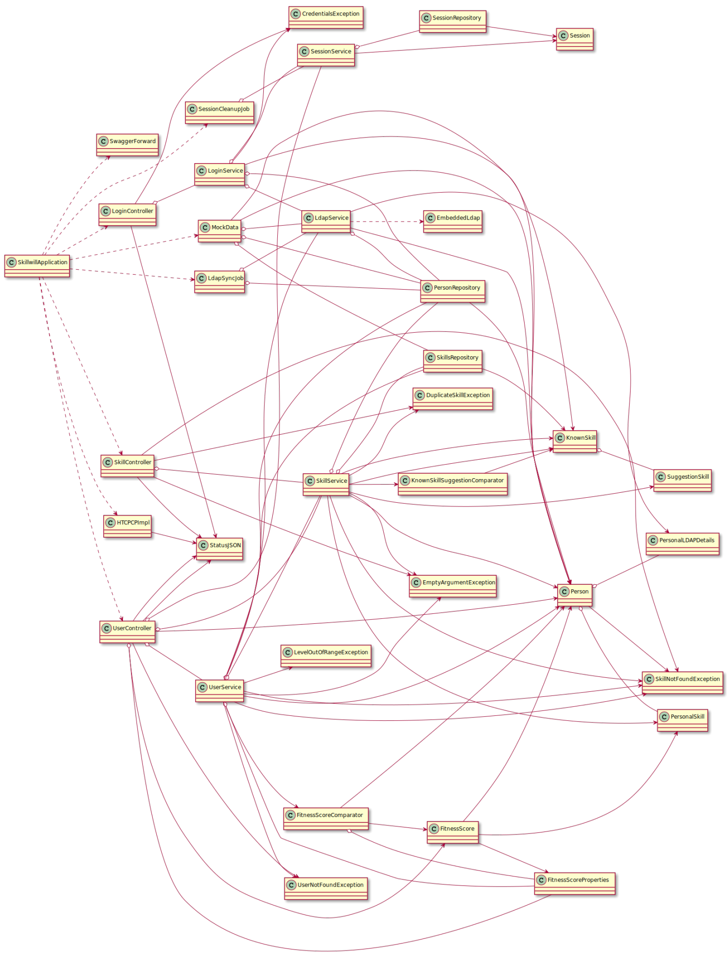

Figure 5.8: UML class diagram of the backend to illustrate its dimensions.

[8]https://go.java/
[9]https://projects.spring.io/spring-boot/
[10]https://maven.apache.org/what-is-maven.html

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, a modular design to extend its features, and dependency injection. It comes to use 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.

```java
@Controller
public class HTCPCPImpl {

    @RequestMapping(path = "/coffee", method = RequestMethod.GET)
    public ResponseEntity<String> coffee() {
        StatusJSON json = new StatusJSON("I'm a teapot \u2615");
        return new ResponseEntity<String>(
      json.toString(),
      HttpStatus.I_AM_A_TEAPOT
    );
    }

}
```

Figure 5.9: Example controller using Spring Boot

5.7.2 Spring Data Repositories

Spring Data[11] is a module for Spring Boot that streamlines the way elements can be accessed from a database. The components mainly used in this application are CRUD[12] repository objects that enclose the database connections and serve simple java methods as an interface. To create such a repository, a java interface defining the stored data type and custom database queries has to be constructed. No actual implementation of the interface has to be created since it will be generated automatically by Spring Data. The parameters needed to connect to the database have to be configured in any source of properties known to Spring Boot, e.g. in *src/main/resources/application.properties*.

```java
public interface PersonRepository
        extends MongoRepository<Person, String> {

    Person findById(String id);

    @Query("{ skills._id : ?0 }")
    List<Person> findBySkill(String skillName);

}
```

Figure 5.10: Example for a repository interface managing person objects stored in the database.

```
spring.data.mongodb.host=127.0.0.1
spring.data.mongodb.port=27017
spring.datasource.driverClassName=com.mongodb.Mongo
```

Figure 5.11: All configuration parameters needed to run Spring Data

5.7.3 LDAP Connection

To connect to the LDAP server, the *unboundid* library[13] is used in the class *LdapService*, It provides methods to open a TCP connection to the server, make requests, and parse the server's response. The connection to the LDAP server will be kept alive and is reused for all operations, so that the effort to open a new connection is eliminated.

[11] http://projects.spring.io/spring-data/
[12] Create, Read, Update, Delete
[13] https://www.ldap.com/unboundid-ldap-sdk-for-java

```java
@Service
@Scope("singleton")
@EnableRetry
public class LdapService {

    private static Logger logger =
        LoggerFactory.getLogger(LdapService.class);

    // [fields not used in this example]

    private static LDAPConnection ldapConnection;

    @Autowired
    private PersonRepository personRepo;

    // [methods for user sync and connection handling]

    public boolean canAuthenticate(String username,
            String password) {
        try {
            BindRequest bindRequest = new SimpleBindRequest(
                "uid=" + username + "," + ldapBaseDN, password);
            BindResult bindResult =
                ldapConnection.bind(bindRequest);
            if (bindResult.getResultCode()
                    .equals(ResultCode.SUCCESS)) {
                return true;
            }
            return false;
        } catch (LDAPBindException e) {
            return false;
        } catch (LDAPException e) {
            logger.error("Failed to authenticate: LDAP error");
        }
        return false;
    }

}
```

Figure 5.12: LDAP user authentication using the unboundid library. Note: parts of the code have been removed to simplify this example.

5.7.4 Swagger

Swagger[14] is an open source framework for creating documentations of REST APIs. Its annotation based java integration is heavily used to generate an interactive overview of the API endpoints provided by the backend. This overview is automatically served by spring boot and contains a list of all URLs to make requests to, HTTP response codes to expect, the content type of the response and a built in form to make example requests. The main advantage of this approach is that the code and its documentation are located at the very same place and that parts of the documentation are automatically generated, so that both are maintained synchronously, thus avoiding the documentation differing from the implementation.

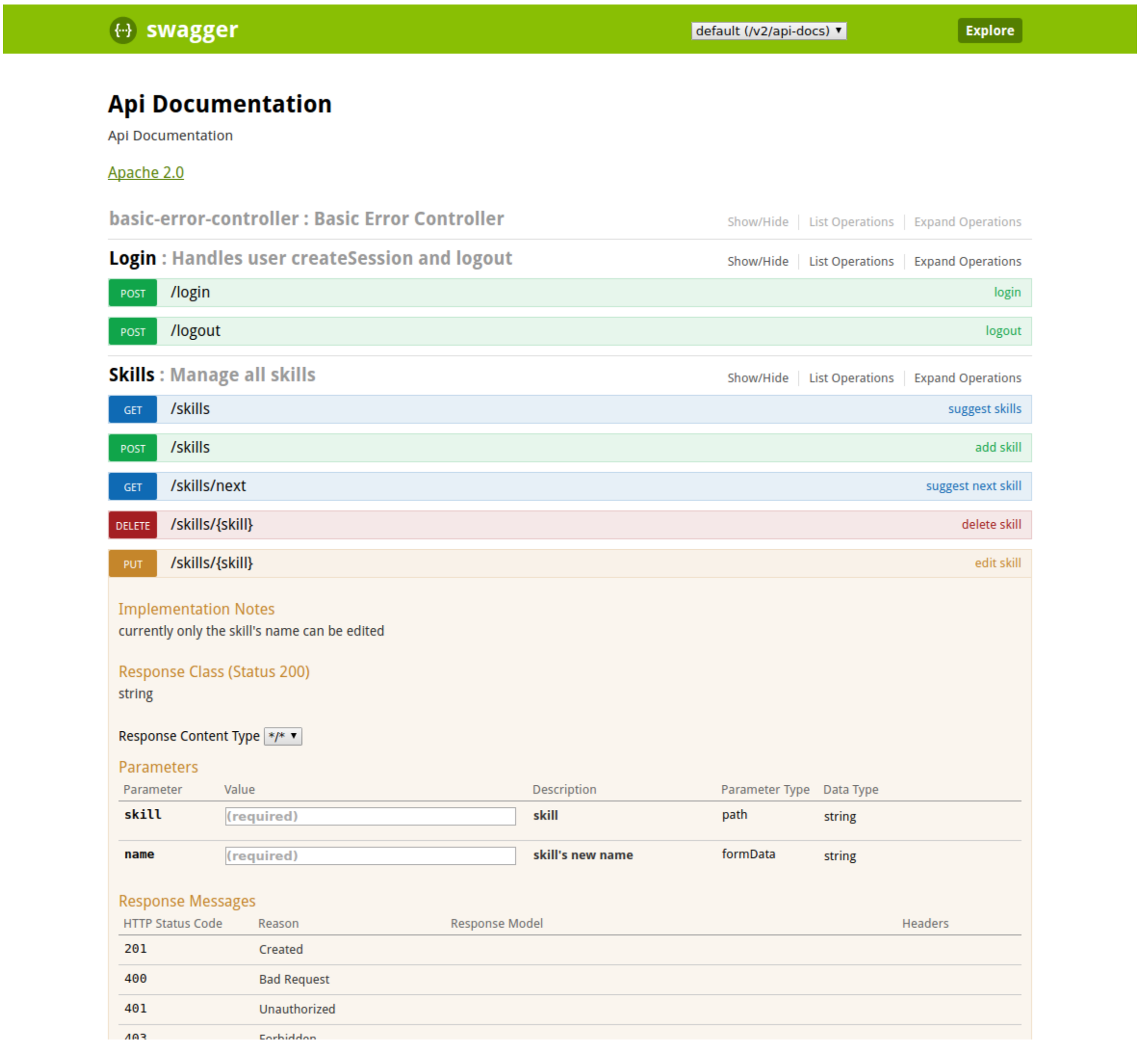

Figure 5.13: Interactive API documentation generated by Swagger

[14] http://swagger.io/

5.7.5 Testing

As a part of the build process, automatic tests are run using the *JUnit*[15] framework. Two types of tests are employed to ensure the proper working of the software: unit tests that validate isolated segments (java classes), and integration tests that simulate calls to the controllers and test the interplay of the individual components.

```java
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class KnownSkillSuggestionComparatorTest {

    @Test
    public void testNoneStarts() {
        KnownSkill a = new KnownSkill("Wurstwasser");
        KnownSkill b = new KnownSkill("foo");

        List<KnownSkill> toSort = new ArrayList<KnownSkill>();
        toSort.add(a);
        toSort.add(b);
        toSort.sort(new KnownSkillSuggestionComparator("42"));

        assertEquals(a, toSort.get(0));
        assertEquals(b, toSort.get(1));
    }

    @Test
    public void bothStart() {
        KnownSkill a = new KnownSkill("foobar");
        KnownSkill b = new KnownSkill("foowurst");

        List<KnownSkill> toSort = new ArrayList<KnownSkill>();
        toSort.add(a);
        toSort.add(b);
        toSort.sort(new KnownSkillSuggestionComparator("foo"));

        assertEquals(a, toSort.get(0));
        assertEquals(b, toSort.get(1));
    }

}
```

Figure 5.14: Example unit test using JUnit

[15] http://junit.org

Embedded Services

During the integration test phase, external services like LDAP and a database have to be accessed in order to ensure the proper working of the interfaces connecting to them. Using the real services, however, is not an option as it cannot be assumed that the machine that runs the tests has a connection to them, and because the tests have to take control over the state of the services. To solve this, an LDAP server and a MongoDB are embedded into the application and will be used during testing. The embedded database is the MongoDB implementation by *flapdoodle*[16], which has the advantage of being effortlessly deployed by importing it; all further configuration and setup happen automatically. To embed an LDAP service, the *unboundid* library is used.

5.8 License

The software is licensed under the MIT license [Mas88] which is considered one of the most popular open source licenses, mainly because it grants a high level of freedom to modify and use the software under the sole condition that a copy of the original license is distributed algonside the software.

> Copyright 2017 SinnerSchrader Deutschland GmbH
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

[Mas88]

[16] https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo

6 Evaluation

6.1 Fitness Score Algorithm

6.1.1 Uniform Distribution of Score Values

The scores calculated by the fitness score algorithm (see 4.1.4) should be distributed uniformly in the complete range of possible values because this does not only generate the best search results, but also is an indicator for the algorithm's fairness. To test this, one hundred persons have been fed into the system. Each test person has been assigned a random number of skills between zero and 17, with random skill and will levels each. A search for any skill will return a list of up to one hundred persons sorted by their fitness score. The search results for the skills *Atomic Design*, *Datenbanken*, *Funkspots*, *hybris*, *Kommunikation*, *MySQL*, *Sktech* and *Text* have been analyzed because those have the highest number of results (33 each); all results can be found in table 6.1. Ideally, the scores in each result list decline linearily from one to zero. The ideal value for the n-th result value is:

$$f_{Ideal}(n) = 1 - \frac{n}{33}$$

The distribution of the score values for every respective search is shown in figure 6.1. Figure 6.2 illustrates the average of all eight scores for each postion in the search result lists, the maximum value found at this position, and the respective minimum value. It shows that the average score declines linearily, which means that the score values are distributed uniformly thorough the result lists. In figure 6.1, every result function shows a variance from the ideal value; the average value in figure 6.2, however, deviates significantly less from the ideal than any of the isolated data rows. This leads to the conclusion that the drift from the ideal that occurs when observering one single set of data for one specific search is based on the small amount of examined values (33 data points). The average fitness score shows an average error of approx. 6% (see figure 6.3). The maximum error in the given set of data is 27%. This leads to the conclusion that the fitness score algorithm generates uniformly distributed score values with an acceptable error.
Interestingly, the maximum error seems to have a declining trend; the small amount of examined data does not provide a solid basis for assumptions about whether there is a systematic reason for this, and, if so, whether it could have a negative impact on the proper working of the system. Unfortunately, there is no authentic usage data yet, so that this question will remain unanswered and might be subject to further research.

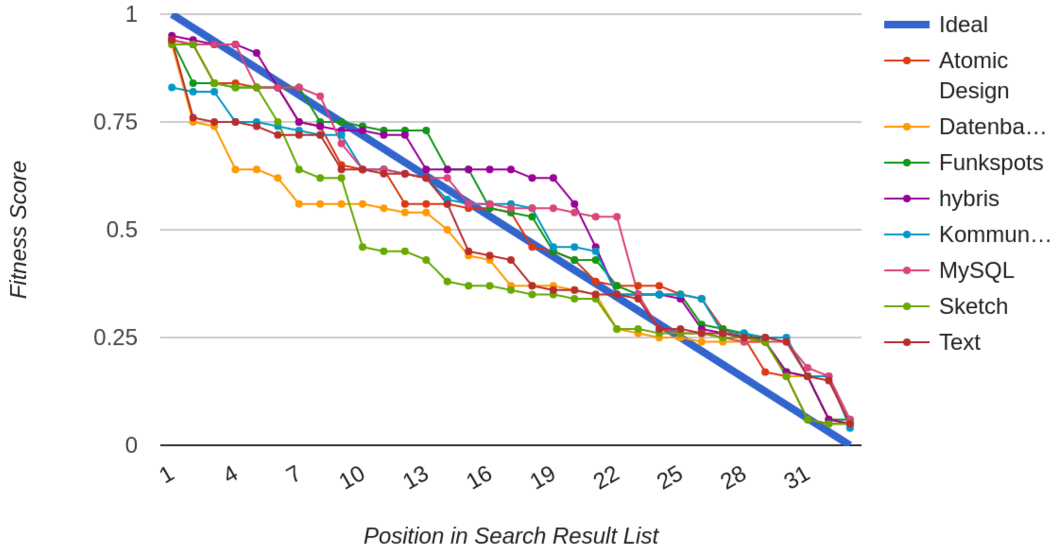

Figure 6.1: All search results and the ideal value.

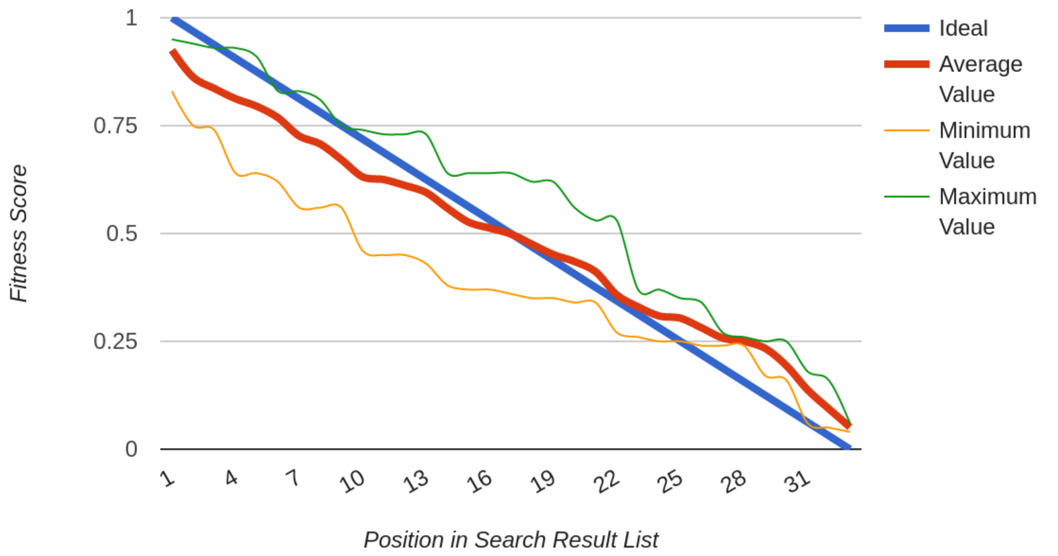

Figure 6.2: Average, maximum and minimum score for earch position in the respective search result list.

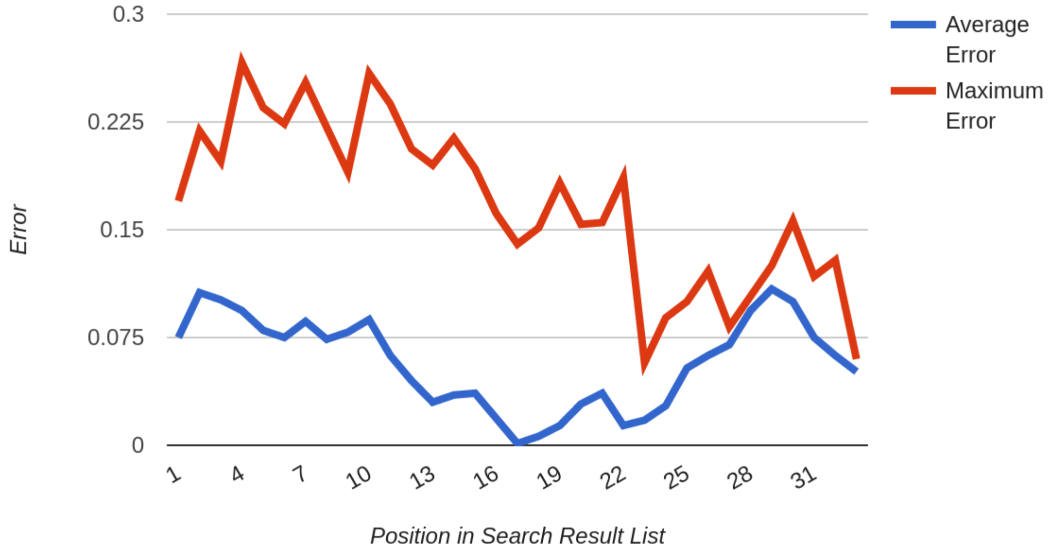

Figure 6.3: Average and maximum error (Difference between respective score value and the ideal value).

No.	v_{Ideal}	A.D.	DBs	Funk.	hybris	Kom.	MySQL	Sketch	Text	Mean Value	Min. Val.	Max. Val.	Mean Error	Max. Error
1.00	1.00	0.94	0.93	0.94	0.95	0.83	0.94	0.93	0.94	0.925	0.83	0.95	0.075	0.17
2	0.96875	0.93	0.75	0.84	0.94	0.82	0.93	0.93	0.76	0.8625	0.75	0.94	0.10625	0.21875
3	0.9375	0.84	0.74	0.84	0.93	0.82	0.93	0.84	0.75	0.83625	0.74	0.93	0.10125	0.1975
4	0.90625	0.84	0.64	0.83	0.93	0.75	0.93	0.83	0.75	0.8125	0.64	0.93	0.09375	0.26625
5	0.875	0.83	0.64	0.83	0.91	0.75	0.83	0.83	0.74	0.795	0.64	0.91	0.08	0.235
6	0.84375	0.83	0.62	0.83	0.83	0.74	0.83	0.75	0.72	0.76875	0.62	0.83	0.075	0.22375
7	0.8125	0.75	0.56	0.83	0.75	0.73	0.83	0.64	0.72	0.72625	0.56	0.83	0.08625	0.2525
8	0.78125	0.74	0.56	0.75	0.74	0.72	0.81	0.62	0.72	0.7075	0.56	0.81	0.07375	0.22125
9	0.75	0.65	0.56	0.75	0.73	0.72	0.7	0.62	0.64	0.67125	0.56	0.75	0.07875	0.19
10	0.71875	0.64	0.56	0.74	0.73	0.64	0.64	0.46	0.64	0.63125	0.46	0.74	0.0875	0.25875
11	0.6875	0.64	0.55	0.73	0.72	0.64	0.64	0.45	0.63	0.625	0.45	0.73	0.0625	0.2375
12	0.65625	0.56	0.54	0.73	0.72	0.63	0.63	0.45	0.63	0.61125	0.45	0.73	0.045	0.20625
13	0.625	0.56	0.54	0.73	0.64	0.62	0.62	0.43	0.62	0.595	0.43	0.73	0.03	0.195
14	0.59375	0.56	0.5	0.64	0.64	0.57	0.62	0.38	0.56	0.55875	0.38	0.64	0.035	0.21375
15	0.5625	0.55	0.44	0.64	0.64	0.56	0.56	0.37	0.45	0.52625	0.37	0.64	0.03625	0.1925
16	0.53125	0.55	0.43	0.55	0.64	0.56	0.56	0.37	0.44	0.5125	0.37	0.64	0.01875	0.16125
17	0.5	0.54	0.37	0.54	0.64	0.56	0.55	0.36	0.43	0.49875	0.36	0.64	0.00125	0.14
18	0.46875	0.46	0.37	0.53	0.62	0.55	0.55	0.35	0.37	0.475	0.35	0.62	0.00625	0.15125
19	0.4375	0.45	0.37	0.45	0.62	0.46	0.55	0.35	0.36	0.45125	0.35	0.62	0.01375	0.1825
20	0.40625	0.43	0.36	0.43	0.56	0.46	0.54	0.34	0.36	0.435	0.34	0.56	0.02875	0.15375
21	0.375	0.38	0.35	0.43	0.46	0.45	0.53	0.34	0.35	0.41125	0.34	0.53	0.03625	0.155
22	0.34375	0.37	0.27	0.37	0.35	0.35	0.53	0.27	0.35	0.3575	0.27	0.53	0.01375	0.18625
23	0.3125	0.37	0.26	0.35	0.35	0.35	0.35	0.27	0.34	0.33	0.26	0.37	0.0175	0.0575
24	0.28125	0.37	0.25	0.35	0.35	0.35	0.27	0.26	0.27	0.30875	0.25	0.37	0.0275	0.08875
25	0.25	0.35	0.25	0.35	0.34	0.35	0.26	0.26	0.27	0.30375	0.25	0.35	0.05375	0.1
26	0.21875	0.34	0.24	0.28	0.27	0.34	0.26	0.26	0.26	0.28125	0.24	0.34	0.0625	0.12125
27	0.1875	0.27	0.24	0.27	0.26	0.26	0.25	0.25	0.26	0.2575	0.24	0.27	0.07	0.0825
28	0.15625	0.25	0.24	0.26	0.25	0.26	0.24	0.25	0.25	0.25	0.24	0.26	0.09375	0.10375
29	0.125	0.17	0.24	0.24	0.24	0.25	0.24	0.24	0.25	0.23375	0.17	0.25	0.10875	0.125
30	0.09375	0.16	0.16	0.17	0.17	0.25	0.24	0.16	0.24	0.19375	0.16	0.25	0.1	0.15625
31	0.0625	0.06	0.16	0.16	0.16	0.16	0.18	0.06	0.16	0.1375	0.06	0.18	0.075	0.1175
32	0.03125	0.05	0.06	0.06	0.06	0.16	0.16	0.05	0.15	0.09375	0.05	0.16	0.0625	0.12875
33	0	0.05	0.05	0.06	0.05	0.04	0.06	0.05	0.05	0.05125	0.04	0.06	0.05125	0.06

Table 6.1: Fitness score values that have been selected for this test.

6.1.2 Fitness Score Algorithm vs. Human Estimations

The algorithm is supposed to calculate a person's fitness so that its results match the ratings estimated by other employees. To validate that the chosen approach is capable of this, a set of fictional employees has been rated by both the algorithm and human beings. The respective estimations have been compared to analyze if there is a configuration of weigthing parameters w_{as}, w_{aw}, w_{ss} and w_{sw} that make the algorithm produce scores that are congruent to the estimations made by the rating persons.

Examined Test Records

For this test, five test persons will be examined: *Alice*, *Bob*, *Charlie*, *Donald* and *Erika*. Each of this persons has different skill and will levels for the abilities *Java*, *AEM*, *Ruby* and *.NET*. The fitness scores that will be collected and examined focus on the scenario that a potential user searches for the skills *Java* and *AEM*.

Test Record	Java	AEM	Ruby	.NET
Alice	3/3	2/3	0/1	2/2
Bob	2/1	3/0	2/0	3/3
Charlie	1/3	0/2	1/2	2/3
Donald	1/2	2/1	1/2	2/1
Erika	1/0	0/1	3/2	3/1

Table 6.2: Skill and will levels of the persons presented in the survey. Notation: [skill level]/[will level]

Survey

A random group of 161 employees (35% of SinnerSchrader's staff) have been presented a survey using *Google Forms*[1]. In total, 41 persons have given their responses in a timeframe of 72 hours. The survey consisted of two sections: the estimation of the test persons' fitness scores and the evaluation of the weighting of the factors included in the algorithm. To collect the personal estimations regarding the test records' fitness, the test subjects have been presented *Likert Items* using a scale form one ("does not fit at all") to five ("perfect match"). As table 6.3 shows, values on this scale can easily be translated into the corresponding fitness score. The results of the survey are shown in table 6.4[2] and figure 6.4.

Survey Rating	1	2	3	4	5
Fitness Score	0	0.25	0.5	0.75	1

Table 6.3: Conversion from survey rating to fitness score.

Test Record	1	2	3	4	5	-	Mean	f
Alice	0	0	0	15	26	0	4.63	0.91
Bob	5	26	9	1	0	0	2.15	0.29
Charlie	1	14	23	3	0	0	2.68	0.42
Donald	0	10	27	3	0	1	2.83	0.44
Erika	32	70	0	2	0	0	1.31	0.08

Table 6.4: Fitness scores estimated by 41 test subjects

Furthermore, the participants were asked to rate the importance of the four factors included in the algorithm (see 4.1.4), namely the person's average level of skill and will in the searched items, and their respective specialization in the items, on a scale from one ("not important") to five ("very important"). The results illustrated in table 6.5 show that all factors are valued nearly equally important.

Factor	1	2	3	4	5	-	Mean
Avgerage Skill Level in Searched Items	1	3	10	16	11	0	3.80
Avgerage Will Level in Searched Items	0	10	8	16	16	0	4.15
Specialization in Searched Items (Skill Levels)	3	5	17	12	4	0	3.22
Specialization in Searched Items (Will Levels)	1	0	11	22	5	2	3.77

Table 6.5: The importance of the four factors included in the fitness score algorithm as estimated by 41 test subjects.

[1] https://forms.google.com
[2] Values have been rounded off to two significant digits.

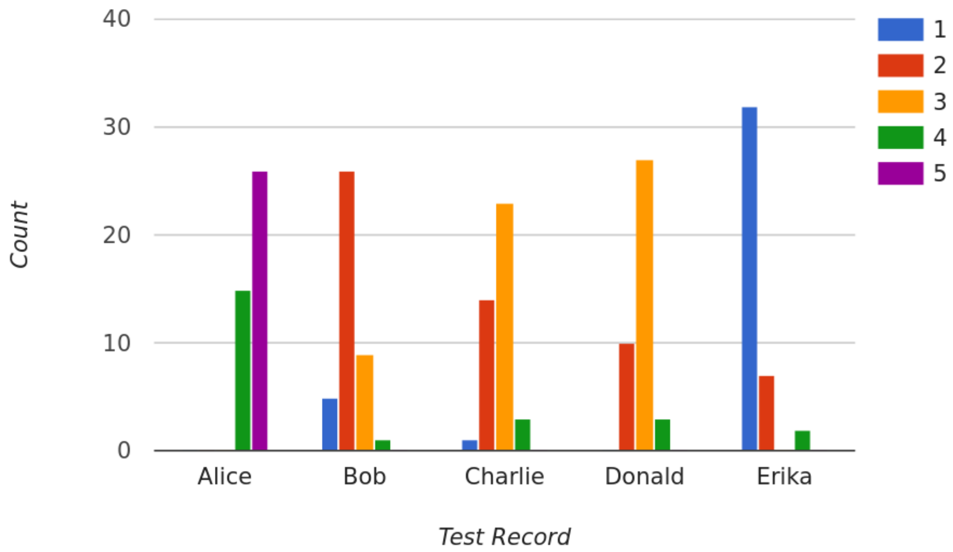

Figure 6.4: Data collected in the survey.

Calculating the weighting parameters w_{as}, w_{aw}, w_{ss} and w_{sw} based on the average estimations of importance of the four factors results in the following values:

Parameter	w_{as}	w_{aw}	w_{ss}	w_{sw}
Weight	25.44%	27.78%	21.55%	25.23%

Table 6.6: Weighting parameters based on the collected data.

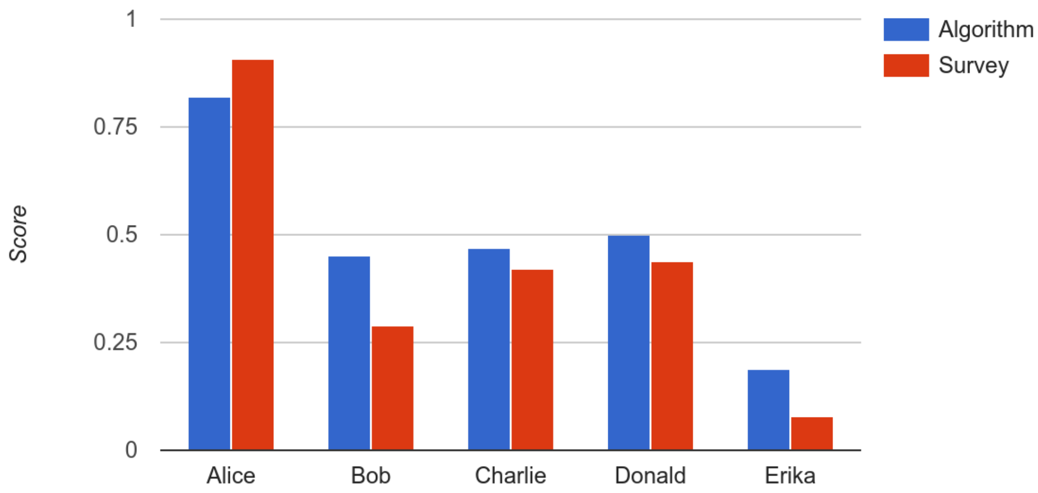

Figure 6.5: Comparison of estimated scores collected in the survey and generated by the algorithm.

Comparison

The fitness score algorithm has been configured to use the aforesaid parameters. Its results for the five test records have been compared to the test subjects' estimations using a two-tailed heteroscedastic T-Test with a significance level of 0.1 in order to show if the persons' estimations deviate significantly from the results calculated using the algorithm. Table 6.8 lists the algorithm's result f_a, the average fitness score f_s estimated by the test subjects, the standard deviation in the collected data, and the values of the T-Test for each test row. As shown, the algorithm's results do not deviate significantly from the values collected in the survey. Using the more common significance level of 0.05, however, would show significant deviations for *Charlie* and *Donald* but the low resolution of the scale used in the survey (five possible values) and the small sample size do not justify such a precise analysis.

Test Record	f_a	f_s	Standard Dev.	p	$p \geq 0.1$
Alice	0.82	0.91	0.12	0.0000358436701	No
Bob	0.45	0.29	0.16	0.0000001306616949	No
Charlie	0.47	0.42	0.16	0.05912471945	No
Donald	0.5	0.44	0.17	0.05091395	No
Erika	0.19	0.08	0.18	0.0003324460113	No

Table 6.7: Comparison of the algorithms results and the scores collected in the survey.

Test Subject	Alice	Bob	Charlie	Donald	Erika
Subject 1	0.75	0.25	0.00	0.25	0.00
Subject 2	1.00	0.50	0.50	0.50	0.00
Subject 3	1.00	0.00	0.50	-	0.00
Subject 4	1.00	0.25	0.50	0.50	0.25
Subject 5	0.75	0.50	0.25	0.25	0.25
Subject 6	0.75	0.25	0.25	0.75	0.75
Subject 7	1.00	0.25	0.25	0.50	0.00
Subject 8	0.75	0.25	0.50	0.25	0.25
Subject 9	1.00	0.25	0.50	0.50	0.00
Subject 10	1.00	0.50	0.50	0.50	0.00
Subject 11	1.00	0.25	0.50	0.50	0.00
Subject 12	0.75	0.25	0.25	0.25	0.00
Subject 13	1.00	0.50	0.25	0.50	0.00
Subject 14	1.00	0.25	0.50	0.50	0.00
Subject 15	0.75	0.00	0.25	0.50	0.00
Subject 16	1.00	0.00	0.50	0.25	0.25
Subject 17	1.00	0.25	0.50	0.50	0.25
Subject 18	1.00	0.25	0.50	0.25	0.00
Subject 19	1.00	0.25	0.75	0.50	0.00
Subject 20	1.00	0.50	0.50	0.50	0.25
Subject 21	1.00	0.50	0.50	0.50	0.00
Subject 22	0.75	0.00	0.25	0.50	0.00
Subject 23	1.00	0.25	0.25	0.75	0.75
Subject 24	0.75	0.25	0.25	0.25	0.00
Subject 25	1.00	0.25	0.25	0.50	0.00
Subject 26	1.00	0.50	0.50	0.75	0.00
Subject 27	1.00	0.25	0.50	0.50	0.00
Subject 28	0.75	0.50	0.50	0.50	0.00
Subject 29	0.75	0.25	0.50	0.50	0.00
Subject 30	0.75	0.25	0.25	0.50	0.00
Subject 31	1.00	0.25	0.50	0.50	0.00
Subject 32	0.75	0.25	0.25	0.50	0.00
Subject 33	1.00	0.25	0.75	0.50	0.00
Subject 34	1.00	0.25	0.50	0.50	0.00
Subject 35	1.00	0.25	0.50	0.50	0.00
Subject 36	0.75	0.50	0.25	0.25	0.00
Subject 37	0.75	0.00	0.50	0.50	0.00
Subject 38	1.00	0.25	0.75	0.50	0.00
Subject 39	0.75	0.25	0.50	0.25	0.00
Subject 40	1.00	0.75	0.25	0.50	0.25
Subject 41	1.00	0.25	0.50	0.25	0.00

Table 6.8: All test subjects' responses in the survey.

Refining the Fitness Score Algorithm

The weighting parameters generated from the survey data are all in the order of 25%; in fact, setting all parameters to 0.25 will not cause any significant change in the algorithms error rate[3], but it will result in all factors being considered equally important and thus reduce the algorithm's complexity. Furthermore, setting the factors to $w_{as} = w_{aw} = w_{ss} = w_{sw} = 0.25$ means they could be eliminated in the fitness score function[4]:

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

$$\Rightarrow f = \frac{a_s + a_w}{4max(V)} + \frac{s_s + s_w}{4}$$

Conclusion

Comparing the algorithm's results with the values collected in the survey has shown that the algorithmic approach can generally be used to generate suitable ratings of an employee's fitness into a specific set of searched skills. The The analysis of the collected data has also demonstrated that the algorithm does not need to include weighting parameters since the test subjects perceive all factors to be equally important. Nonetheless, the factors will not be excluded from the algorithm's implemenation since having the possibility to tweak its working might come in handy if the future day to day use of the application reveals other requirements for the weighting.

[3] It would reduce the average difference between the algorithm and the test subjects' estimations from 9.46% to 9.43% and the maximum deviation from 11.07% to 10.7%.

[4] Definitions can be found in 4.1.4.

6.2 Implementation

6.2.1 Scalability

A software system has to be able to scale according to the number of its users in order to be future-proof, as the current trend to dynamically scalable cloud solutions and server-less web architectures highlights [Mü15]. There are two concepts of preparing an application for a higher workload: vertical scaling and horizontal scaling. Vertical scaling is done by providing more resources, e.g. memory and CPU power, to the machines running the application. Horziontal scaling, however, means setting up more machines providing the same service, so that the workload can be distributed between them [Bea]. In contrast to vertical scaling, horizontal scaling has vital advantages: the application will be more robust since the crashing of one machine can be compensated by others [Fed16], the capacity of the system can, theoretically, be unlimited, and it is cheaper because virutal machines running the service can be created dynamically if needed and then be destroyed in times of low workload, whereas the resources given to a machine that has been scaled vertically will remain unused.

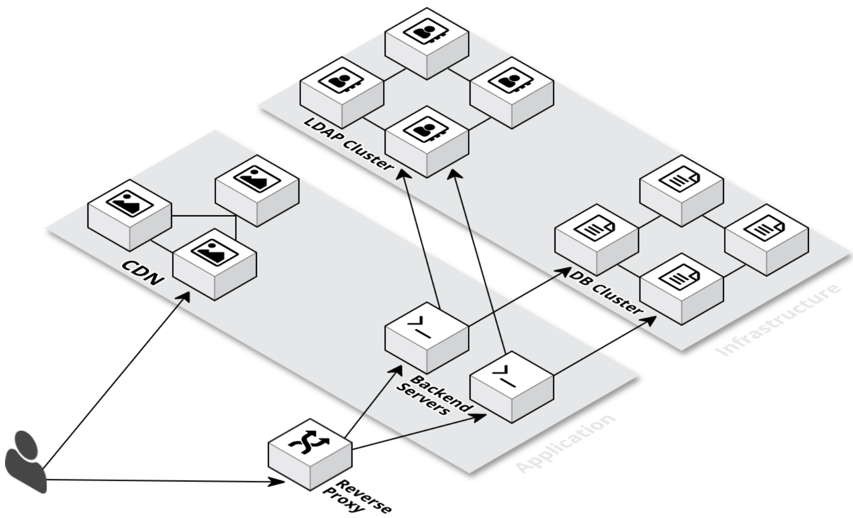

Figure 6.6: A possible approach to scale the system using multiple backend servers, a CDN, and multiple clustered database and ldap servers. Created with *https://cloudcraft.co*.

MongoDB

MongoDB is meant to be scaled horizontally and supports the adding of new instances to a running cluster of databases out of the box [HMPH15, p. 19]. So, new machines running the database as a cluster will be created, if needed. As shown in 6.6, the backend servers can be connected to any of the database servers in order to request data. If the demanded document is not found on the instance the backend is connected to, MongoDB will handle the lookup in the cluster. To the application, the cluster is completely transparent and appears as if it was one machine.

LDAP

The LDAP servers[5] can also be run as a cluster in order to improve response times and prevent data loss by replicating the stored information [Fou]. In fact, the LDAP is currently being provided by six servers that represent the service. As illustrated in figure 6.6, the backend servers can connect to any of the LDAP servers; the data replication and synchronization is handled transparently.

Static Content

The static content like HTML, CSS and JS files, that altogether represent the frontend, are served by the reverse proxy web server. In the event of an increasing number of requests that cannot be handled by the single server, a *Content Delivery Network* (CDN) could be deployed. A CDN is a network of webservers that provide static content and large files. The reverse proxy would redirect the URLs for those files, so that the users' browsers will connect directly to said network in order to retrieve the assets.

Backend

The backend application itself does not save any data on the machine it is running on, but connects a database server (see 5.6). As a result, any number of backend instances can be set up. In contrast to the other services, the backend servers do not have to synchronize. In order to receive HTTP requests, the reverse proxy must be configured so that it redirects API calls to the backend servers. This is called *load balancing* and is supported by many modern web servers such as *nginx*[6], *Apache*[7] and *Tomcat*[8].

[5] SinnerSchrader is running *OpenLDAP* (http://www.openldap.org)
[6] https://www.nginx.com/resources/wiki/
[7] https://httpd.apache.org/
[8] http://tomcat.apache.org/

Conclusion

In theory, the application should be able to scale according to the number of its users. Practically, only the running of multiple backend and LDAP servers has been tested successfully. Running multiple database instances has not been tested; since MongoDB has been designed to be horizontally scalable and comes to use in various companies like *Github*[9], *eBay*[10], and *Otto*[11], it can be assumed that this can be done successfully for this application, too. Deploying a CDN that serves the static content has not been evaluated, as the implementation of the frontend was not part of this thesis, but has been worked on by Strecker [Str17].

6.2.2 Response Times

As defined in 3.2.2, the application should need less than one second of response time between the user pressing the search button and the displaying of the search results. The response times of the corresponding API endpoint have been measured and can be found in table 6.9. The average response time of the backend is 28ms, the maximum in the test data is 44ms.

Request Parameters	Response Times (in ms)	Mean
No parameters	26, 28, 27, 40, 27, 33, 28, 29, 26, 29	29
Specific Skill	32, 40, 30, 32, 26, 24, 28, 44, 33, 32	32
Specific Location	27, 26, 26, 25, 26, 36, 26, 22, 21, 25	26
Specific Skill and Location	36, 24, 23, 21, 22, 24, 33, 20, 30, 21	25

Table 6.9: Measured response times of the api endpoint for the search function.

Measurements of a prototypical stage of the frontend using *Google Chrome's* built-in profiling tools showed a total response time, that is sending the HTTP request to the API, waiting for the response, parsing the response, and rendering the results, of approximately 90ms on average. The maximum response time was 106ms.

Those results show that the API is capable of serving the requests quickly enough to reach the goal of a response time under one second. The outcome of the profiling of the prototypical frontend suggest that it might even be possible to attain a total loading time of under 100ms; according to Kearney, this is would result in the users percieving the interaction with the system as immediate [Kea17], which enhances the overall user experience.

[9] https://www.mongodb.com/presentations/mongosv-2012/mongodb-analytics-github
[10] https://www.mongodb.com/presentations/mongodb-ebay
[11] https://www.mongodb.com/industries/retail

6.3 Meeting the Requirements

In 3.2, a set of functional and non functional requirements has been defined. The backend application that has been designed and implemented has to meet those requirements, and it does: the API supports the creation of user profiles that then can be retrieved, the skills in those profiles can be edited by the profile's owner only, everybody can search for profiles of persons that offer specific skills (see table 5.1). The application was designed to be accessible to all employees, thus it is implemented as a web application that can be opened by any internet enabled device. New skills can be fed into the system so that people can add them to their profiles.

The non functional requirements included scalability, low response times, the supported devices and browsers. As shown in 6.2.1, scalability has been partially evaluated, whereas services that the application relies on such as MongoDB and LDAP were assumed to be scalable in this architecture as comparable systems have already shown. The response times have been evaluated in 6.2.2; the results and tests with and prototypical stage of the frontend show that the application is capable of delivering the requested inforamtion in significantely less time than defined. The requirements for supported devices and browsers, however, could not be evaluated since the determinative factor for those is the implementation of the frontend which has not been part of this thesis and will be evaluated by Strecker [Str17].

7 Résumé and Outlook

In this thesis, the creation of a skills management application custom-tailored to SinnerSchrader has been drafted from its underlying concept to its partial implementation. The motivation why SinnerSchrader needs such an application as well as the challenges specific to this company have been outlined. Based on this information, requirements the tool will have to fulfill have been compiled; they provide the basis for the analysis of available skill management tools. As shown, two mayor factors required by SinnerSchrader cannot be served by any of the commercial solutions: the tool is supposed to put emphasis on collaboration, not supervision, and users should be able to search for persons that not only have knowledge about a certain topic but should also take into account their interests and personal preferences. These two aspects form the backbone of the application's design including an algorithm that represents them in the search function and thus is an essential part the tool's proper working. Furthermore its technical structure has been laid out and the backend component dealing with its business logic has been implemented. Both the implementation and the underlying concept have been evaluated regarding possible concerns including technical issues and the fulfilling of the end users' needs. To examine the latter, a survey has been conducted and analyzed.

Altough the application has been tested using a fair amount of generated data, real data entered by real users could invalidate the results found in the evaluation phase and reveal obstacles not considered in its design. Only a long-run test phase exposing the application to the users will provide sufficient information about those factors. Unfortunately, the frontend of the application is not finished yet, so that that a user test cannot be run. The constructed tests suggest that the basic principles behind the fitness score algorithm provide a solid foundation for a new type of skills management tools that incorporate not only employees' skills, but also their motivation and preferences thus leading to an increasing employee satisfaction and accelerating personnel development. Integratig the search algorithm or an approach based on its principles into a more sophisticated and feature-rich application, like the ones available on the market, would be a logical step towards such a management tool.

The concrete application created for SinnerSchrader has purposely been designed to be a simple search tool as the focus lays on a streamlined user experience. The implementation fullfils all requirements defined for this special use case and once the frontend component will be integrated into the application, it has the potential to become a deeply useful gadget for the daily work at SinnerSchrader.

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

[Bea] BEAUMONT, David: *How to explain vertical and horizontal scaling in the cloud*. https://www.ibm.com/blogs/cloud-computing/2014/04/explain-vertical-horizontal-scaling-cloud/

[Bec03] BECK, Simon: Skill and Competence Management as a Base of an Integrated Personnel Development (IPD) - A Pilot Project in the Putzmeister, Inc./Germany. In: *Journal of Universal Computer Science* 9 (2003), Nr. 12, S. 1381 – 1287

[Ber16] BERGMANN, Rainer: *organization 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

[DAQ13] DARVISH, Hassan ; AHMADNIA, Hadi ; QRYSHYAN, Seyed A.: Studying the Personal Knowledge Management Profile: a Case Study at Payame Noor University. In: *Economic Insights – Trends and Challenges* 4/2013 (2013), Nr. 1, S. 1 – 12

[DMA14] DMA: *National email benchmarking report 2013*. November 2014 Edition. 2014

[Fed16] FEDERRATH, Hannes: *Foliensatz Grundlagen der Systemsoftware SoSe 2016*. 2016

[Fou] FOUNDATION, The O.: *Replication - OpenLDAP Administrator's Guide*. http://www.openldap.org/doc/admin24/replication.html

[Gut16] GUTIERREZ, Felipe: *Pro Spring Boot - A no-nonsense guide containing case studies and best practices for Spring Boot*. Apress; Springer Science+Business Media, 2016. – ISBN 978–1–4842–1431–2

[HMPH15] HOWS, David ; MEMBREY, Peter ; PLUGGE, Eelco ; HAWKINS, Tim: *The Definitive Guide to MongoDB*. 3. Apress; Springer Science+Business Media, 2015. – ISBN 978–1–4842–1183–0

[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

[IIK13] IVANOVSKA, Sashka ; IVANOVSKA, Ilinka ; KALAJDZISKI, Slobodan: Algorithms for Effective Team Building, 2013 (The 10th Conference for Informatics and Information Technology (CIIT 2013))

[Kea17] KEARNEY, Meggin: *Measure Peformance with the RAIL Model*. `https://developers.google.com/web/fundamentals/performance/rail`. Version: 2017

[Leh04] LEHNER, Franz: *Marktanalyse zum Angebot an Skill-Management-Systemen*. 2004

[Mas88] MASSACHUSETTS INSTITUTE OF TECHNOLOGY: *MIT License*. `https://opensource.org/licenses/MIT`. Version: 1988

[Mü15] MÜNZL, Gerald: *Cloud Computing als neue Herausforderung für Management und IT*. Berlin : Springer Vieweg, 2015 (essentials)

[NGI] NGINX INC.: *WHAT IS A REVERSE PROXY SERVER?* `https://www.nginx.com/resources/glossary/reverse-proxy-server/`

[QS] Q-SUCCESS SOFTWARE QUALITY MANAGEMENT: *Usage of JavaScript for websites*. `https://w3techs.com/technologies/details/cp-javascript/all/all`

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

[Sie] SIEGRIST, Kyle: *Markov Chains*. `http://www.math.uah.edu/stat/markov/Introduction.html`. – Department of Mathematical Sciences; University of Alabama in Huntsville

[Sin17] SINNERSCHRADER: *Quartalsbericht 1 2016/2017*. `https://sinnerschrader.ag/de/berichte/`. Version: 2017

[Smi15] SMITH, Ben: *Beginning JSON*. Berkeley, Calif. : Apress, 2015 (The expert's voice in web development). – 324 S.

[SSH07] SPOONAMORE, Janet H. ; SIMIEN, H. J. ; HALL, Ricky D.: *Person-to-Position Matching*. 2007

[Sta] STATISTA: *Share of users with mobile or stationary internet access in Germany in 2014*. `https://www.statista.com/statistics/432146/internet-access-devices-used-germany/`

[Str17] STRECKER, Katharina: *Front-End Konzeption und Umsetzung eines unternehmensinternen Webtools für Skillmanagement*. 2017

[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 Tukman's forming, storming, norming & perfoming team development model*. 2010

List of Figures

List of Tables

Affidavit (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: _________________________

Digital Storage Device

On this device, you will find both this thesis and the codebase of the application that has been implemented as part of it. Instructions on how to build and run the application can be found in */code/README.md*.