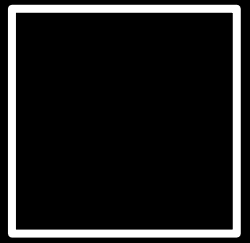
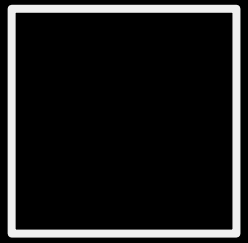
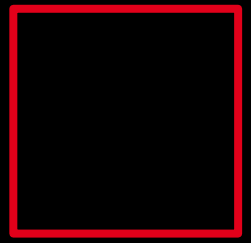
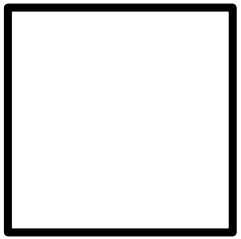
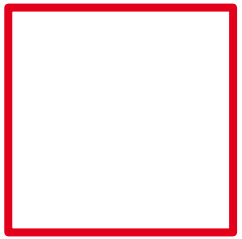




White Text in Red Box





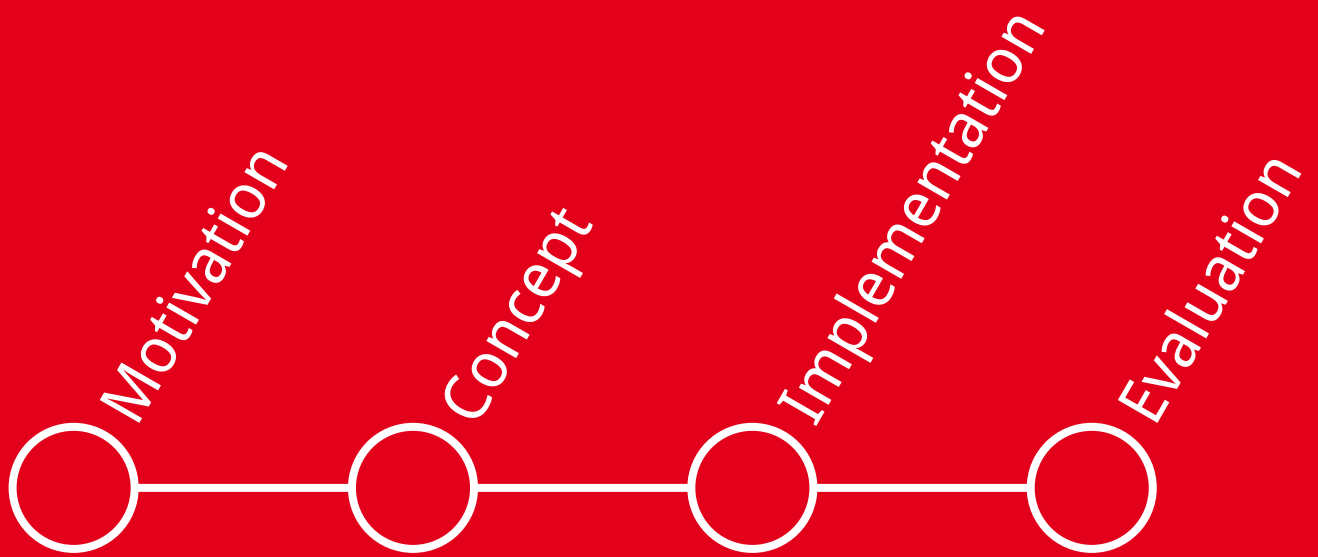
# Hello

Torben Reetz
7th Semester SSD

# A Recommender Framework for Skills Management

@SinnerSchrader

# Agenda



# Motivation

# Context SinnerSchrader

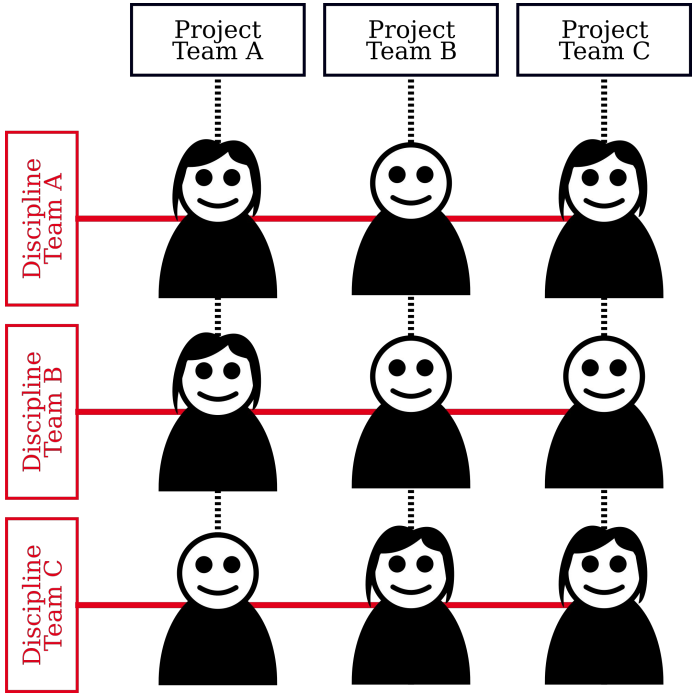


- Hamburg based
- Full service web agency
- 459 full-time employees
- Revenue > 51M Euro (15/16)

# Matrix Organization



- Domain specific teams
- Project teams
- Definitions:
  - Project Manager → Project team
  - Supervisor → Domain specific team



# Struggles (Project Manager)



- Employees leave their teams
- Workload changes
- Shift in disciplines

Project managers frequently look for new team members.

# Struggles (Looking for Help)



- Different experience and knowledge
- Different disciplines
- Different project setups

Employees search for people that can help to solve a specific problem.

# Challenges to Tackle

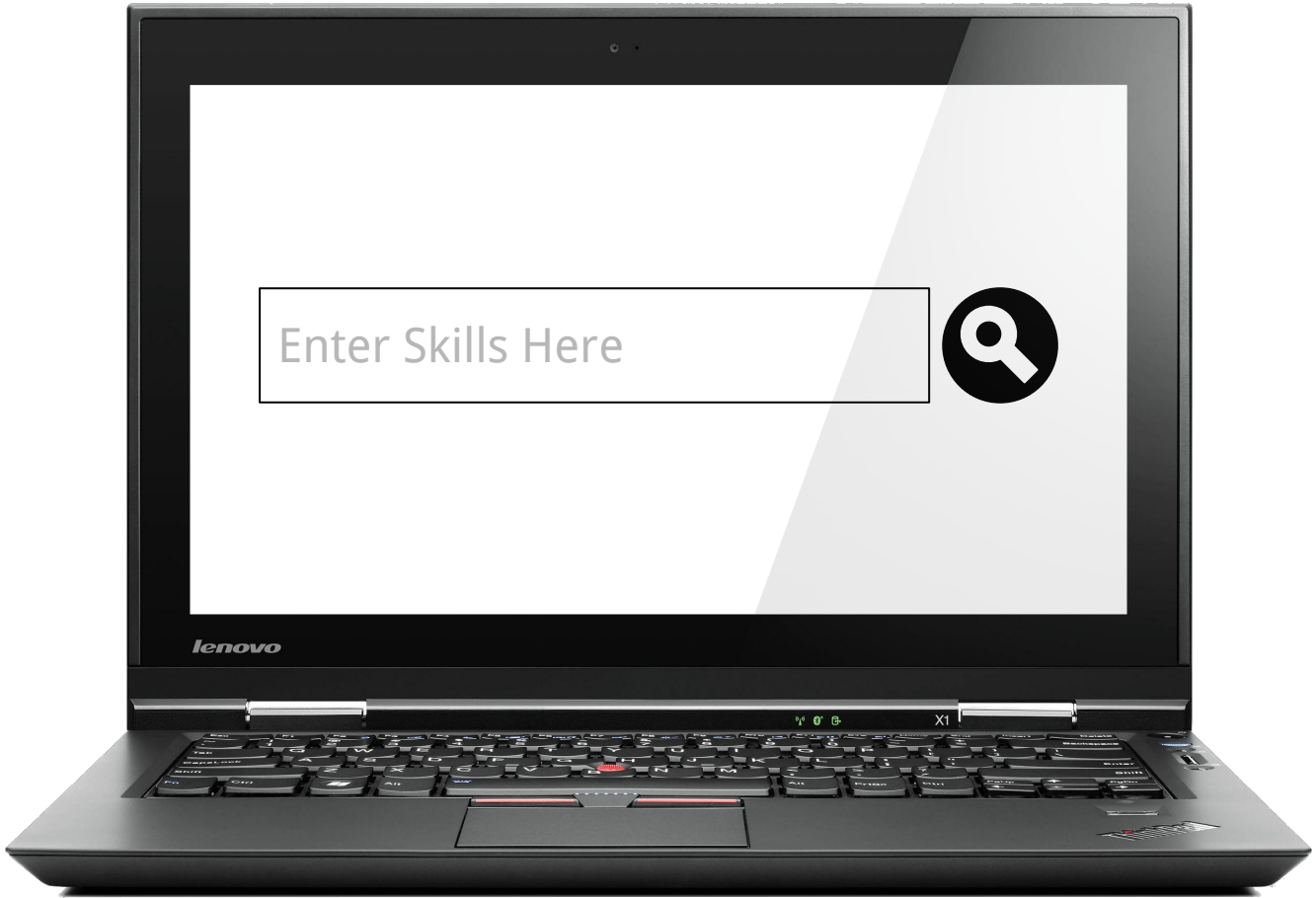


- People search for other people that have specific skills
- Create a central source of information
- Focus: Motivation and Cooperation
- This thesis: Backend only
  - Visual concept / Frontend: Strecker

# Concept

# Walkthrough





# Walkthrough

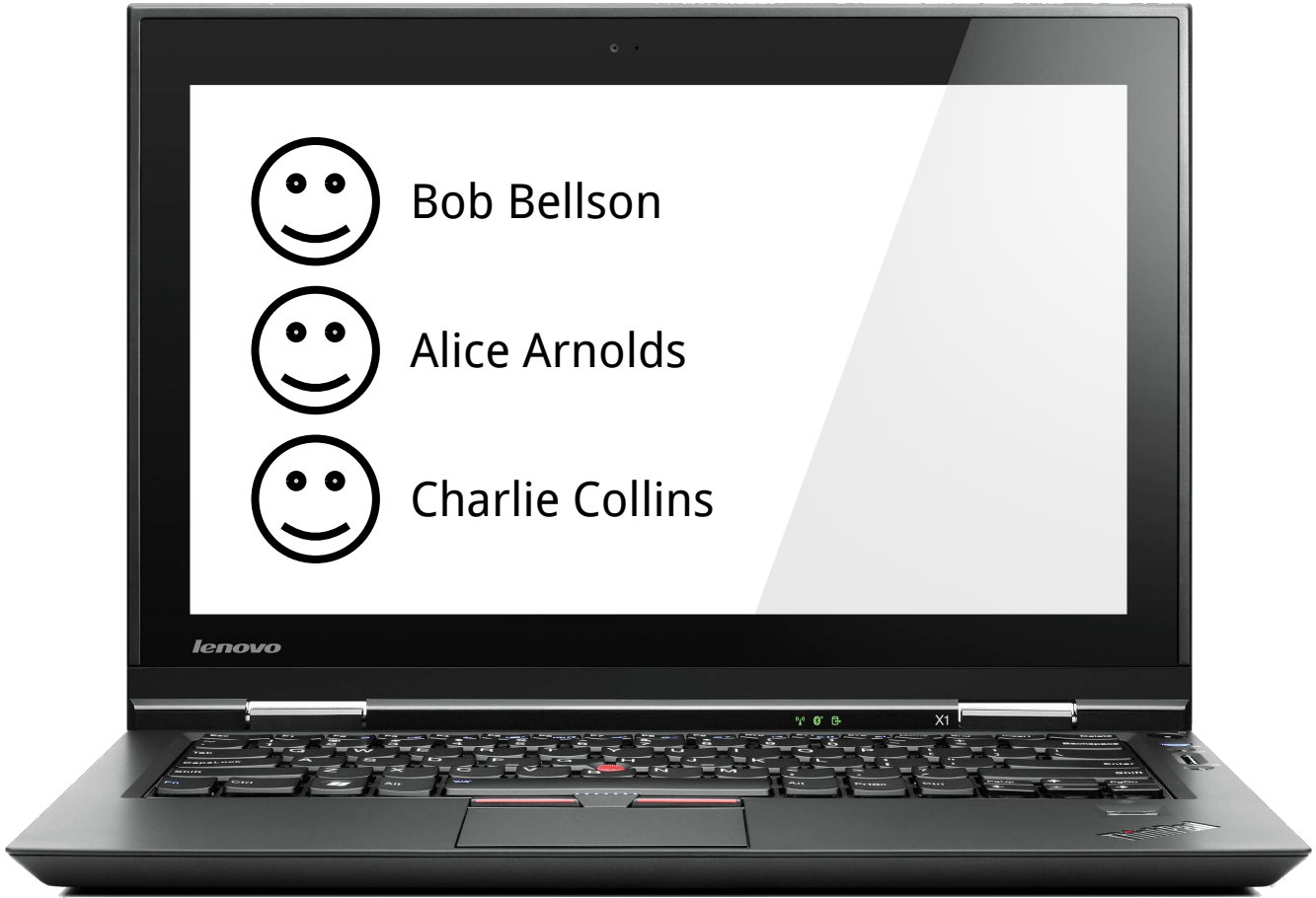


# Walkthrough



# Walkthrough





# Walkthrough



# Main Features



- Person Search
- Recommending skills to search
- Recommending similar profiles

# Person Search

# Definitions



- Skill: ability of a specific person
- Levels
  - Skill level: knowledge
  - Will level: motivation
  - Four Step Scale (0-3)
- Fitness: Measurement of how well a person fits into a searched skill set

# Four Step Scale



| | Skill Level | Will Level |
|---|---|---|
| 0 | Novice | Uninterested |
| 1 | Basic Knowledge | Indifferent |
| 2 | Advanced Knowledge | Somewhat Interested |
| 3 | Expert | Highly Interested |

# Four Step Scale

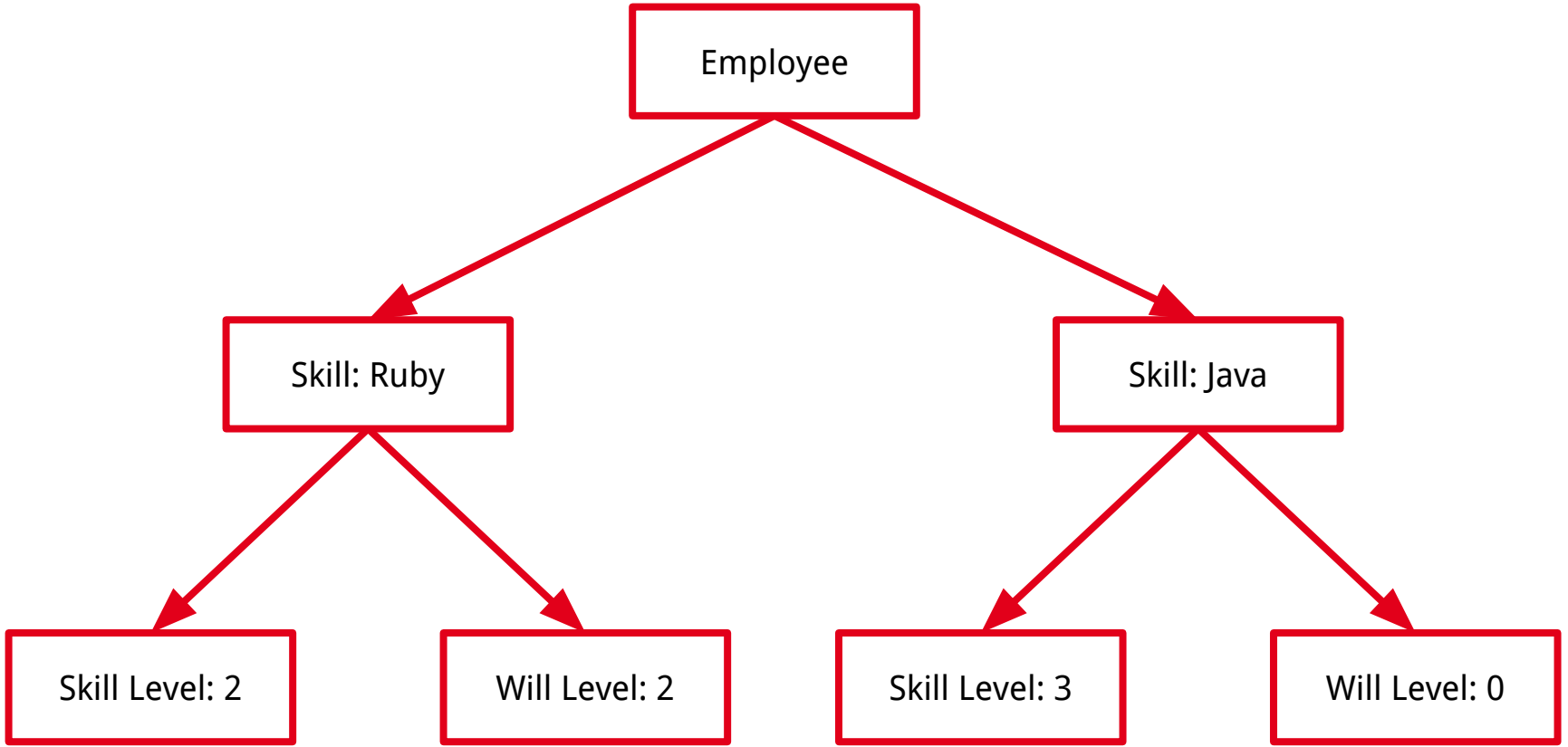


| | Skill Level | Will Level |
|---|---|---|
| 0 | Novice | Uninterested |
| 1 | Basic Knowledge | Indifferent |
| 2 | Advanced Knowledge | Somewhat Interested |
| 3 | Expert | Highly Interested |

- Skill Level = 0 → Person has little knowledge
- No knowledge → Skill not present

# Definitions





# Person Search

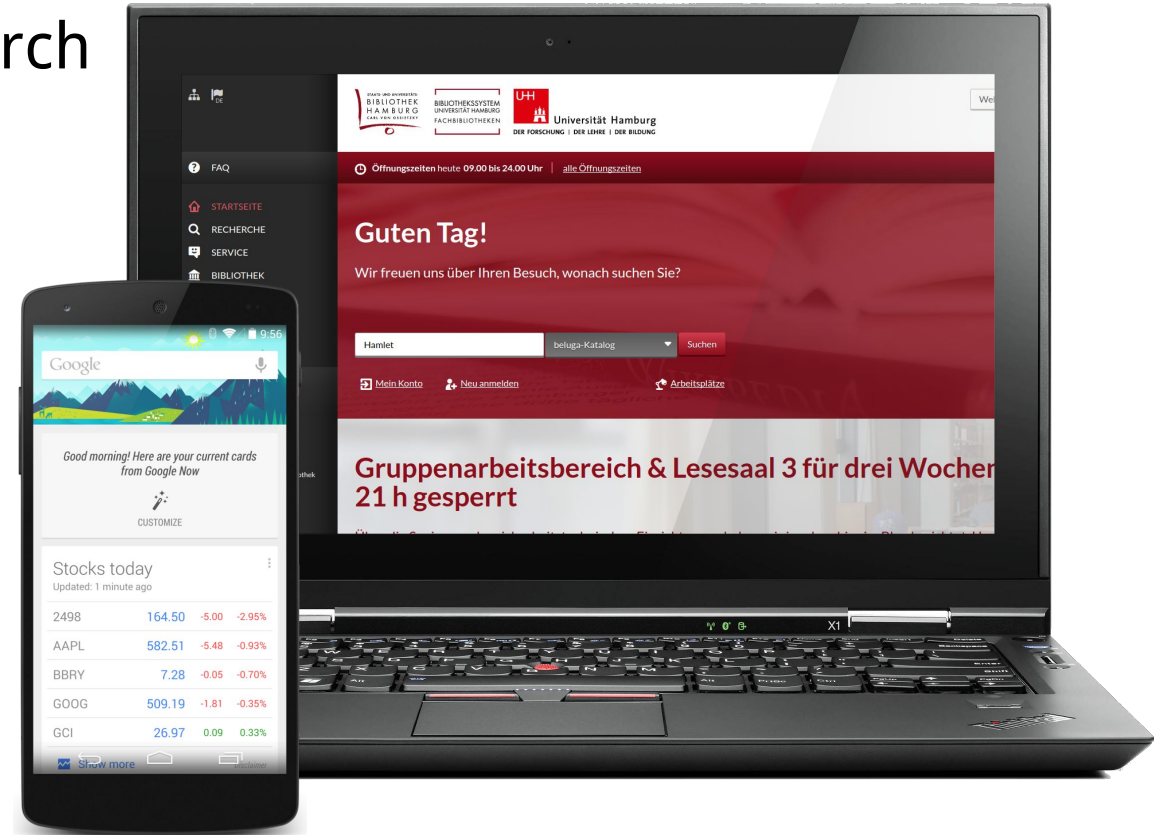


- User enters skills to look for
- Systems presents list of results
  - People that have all skills
  - Best match on first Position
- IR System

# IR Systems (examples)



- Google
- Facebook Graph Search
- Siri/Alexa
- grep
- …



# Information Retrieval (IR) Systems



"Information retrieval (IR) is finding material [...] that satisfies an information need from within large collections."

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.

# Information Retrieval (IR) Systems



**Employees**

"Information retrieval (IR) is finding material [...] that satisfies an information need from within large collections."

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.

# Information Retrieval (IR) Systems



**Employees**

"Information retrieval (IR) is finding material [...] that satisfies an information need from within large collections."

**Search Query**

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.

# Information Retrieval (IR) Systems



**Employees**

"Information retrieval (IR) is finding material [...] that satisfies an information need from within large collections."

**Search Query** **Staff**

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.

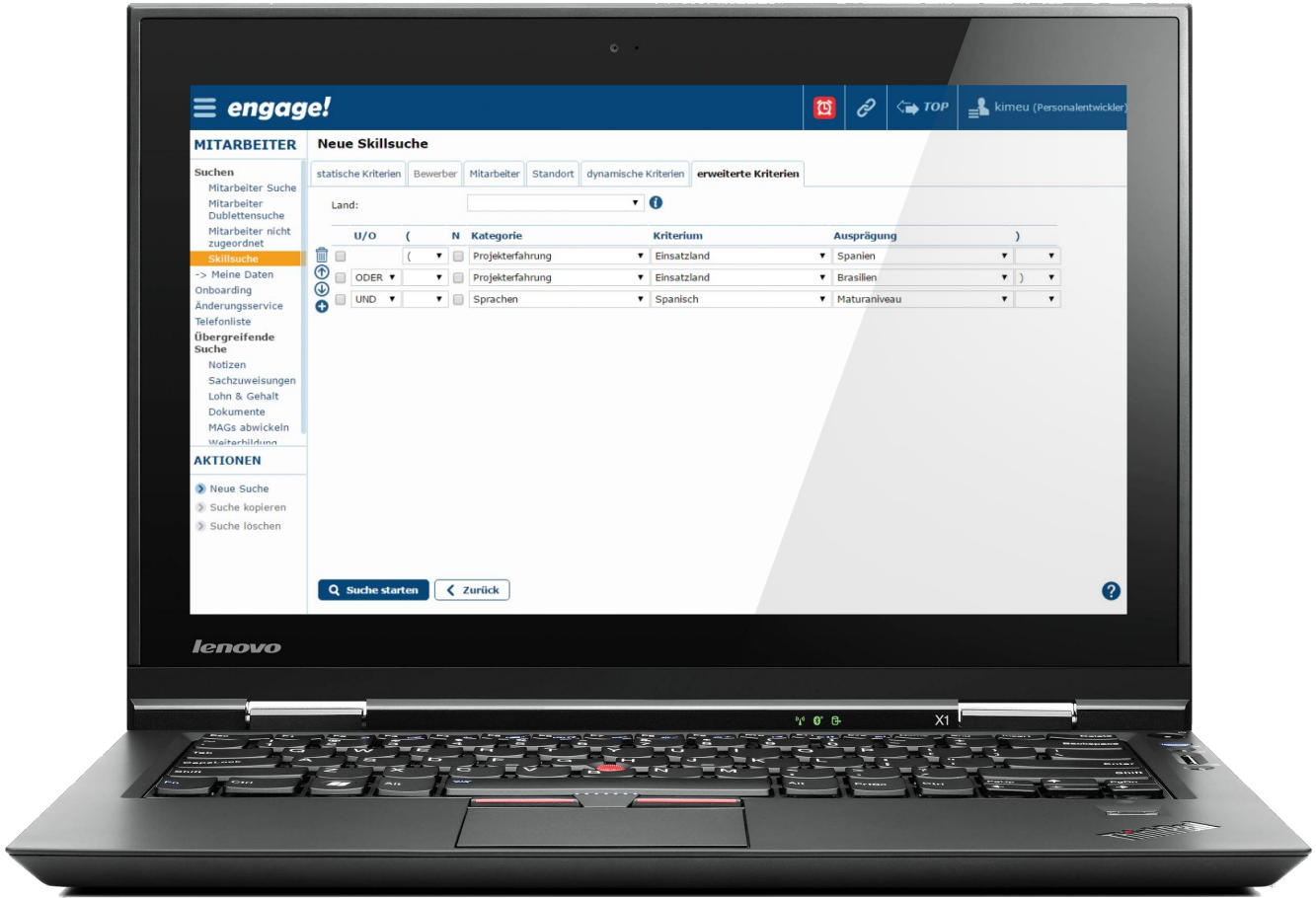
# Boolean and Ranked IR



- Boolean IR systems
  - Boolean operators
  - Example: Jira Query Language
    - “priority in (Blocker, Critical) AND project in (ProjA, ProjB, ProjC)”
- Ranked IR systems
  - Items ranked→ Best match first
  - Example: Google

# Person Search in Talent Management (engage!)





# Person Search in Talent Management (engage!)



# Person Search (Commercial Solution)



- Boolean Systems
- Complex queries
- Bloated interfaces
- No ranking

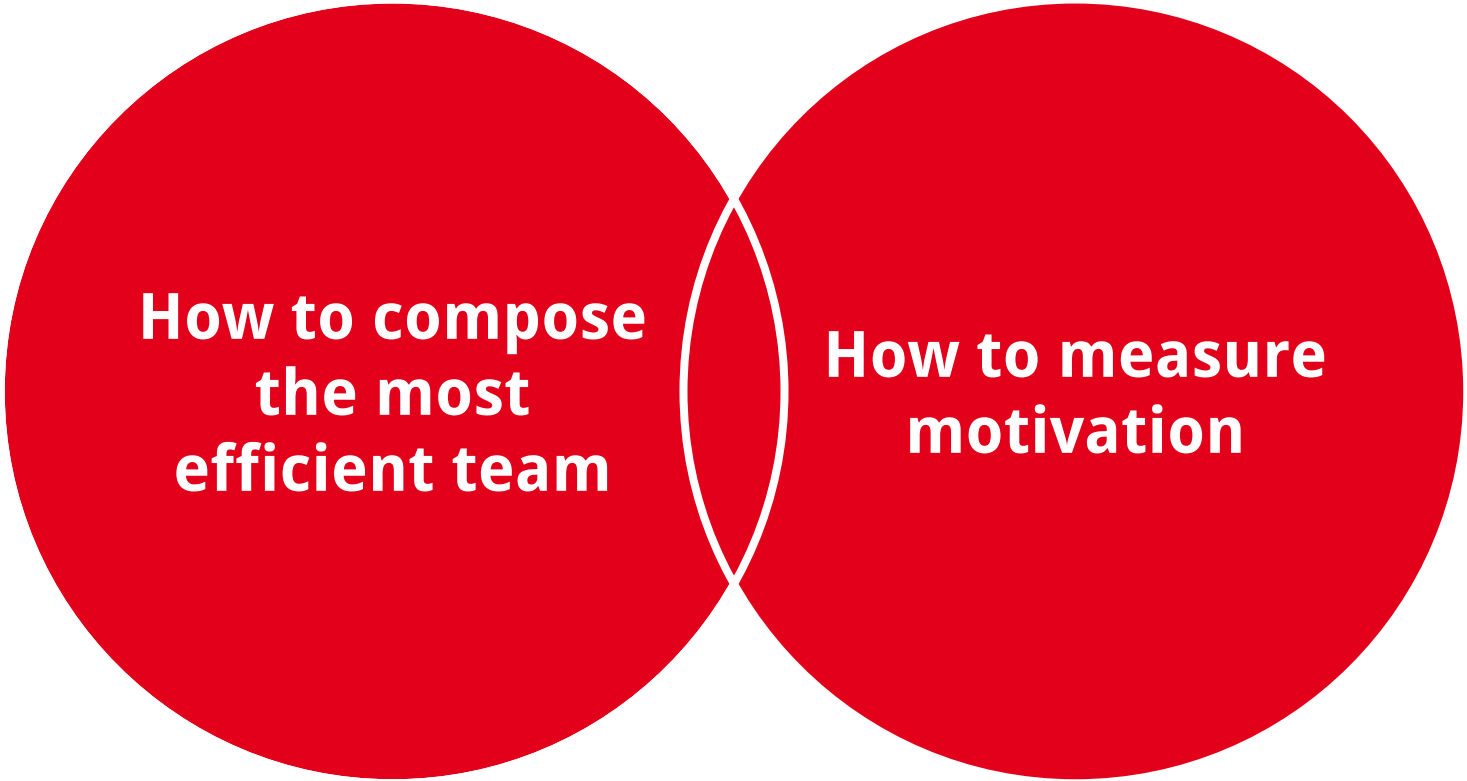
# Person Search (Chosen Approach)



- Ranked IR system
- No complex queries
- Best match first
  - Fitness Score

# Fitness Score (Research)





# Fitness Score (Basic Principle)



- Adaption of Spoonamore et al.
- Weighted mean of factors
  - Average skill level in searched skills
  - Average will level in searched skills
  - Specialization (skill levels)
  - Specialization (will levels)
- Weighting parameters configurable

# Example (Java and Ruby in HH)




- $w_{as} = w_{aw} = w_{ss} = w_{sw} = 0.25$
- Notation: skill level|will level
- Java and Ruby
- In Hamburg

# Example (Java and Ruby in HH)




| 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 |

# Example (Java and Ruby in HH)




| 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 |

# Example (Java and Ruby in HH)




| 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 |

# Example (Java and Ruby in HH)




| 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 |

# Example (Java and Ruby in HH)




| 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 |

# Example (Java and Ruby in HH)




| Person | Location | Java | Ruby | C++ | f |
|---|---|---|---|---|---|
| Alice | Hamburg | 2\|1 | 2\|2 | 3\|3 | 0.44 |
| Bob | Hamburg | 2\|3 | 0\|3 | 0\|1 | 0.71 |
| Charlie | Hamburg | 3\|3 | 2\|1 | 1\|2 | 0.69 |
| Donald | Hamburg | 3\|3 | - | 2\|2 | |
| Erika | Frankfurt | 1\|1 | 2\|3 | 3\|1 | |

# Example (Java and Ruby in HH)




| # | Person | Location | Java | Ruby | C++ | f |
|---|---|---|---|---|---|---|
| 3 | Alice | Hamburg | 2\|1 | 2\|2 | 3\|3 | 0.44 |
| 1 | Bob | Hamburg | 2\|3 | 0\|3 | 0\|1 | 0.71 |
| 2 | Charlie | Hamburg | 3\|3 | 2\|1 | 1\|2 | 0.69 |
| | Donald | Hamburg | 3\|3 | - | 2\|2 | |
| | Erika | Frankfurt | 1\|1 | 2\|3 | 3\|1 | |

# Example (Java and Ruby in HH)

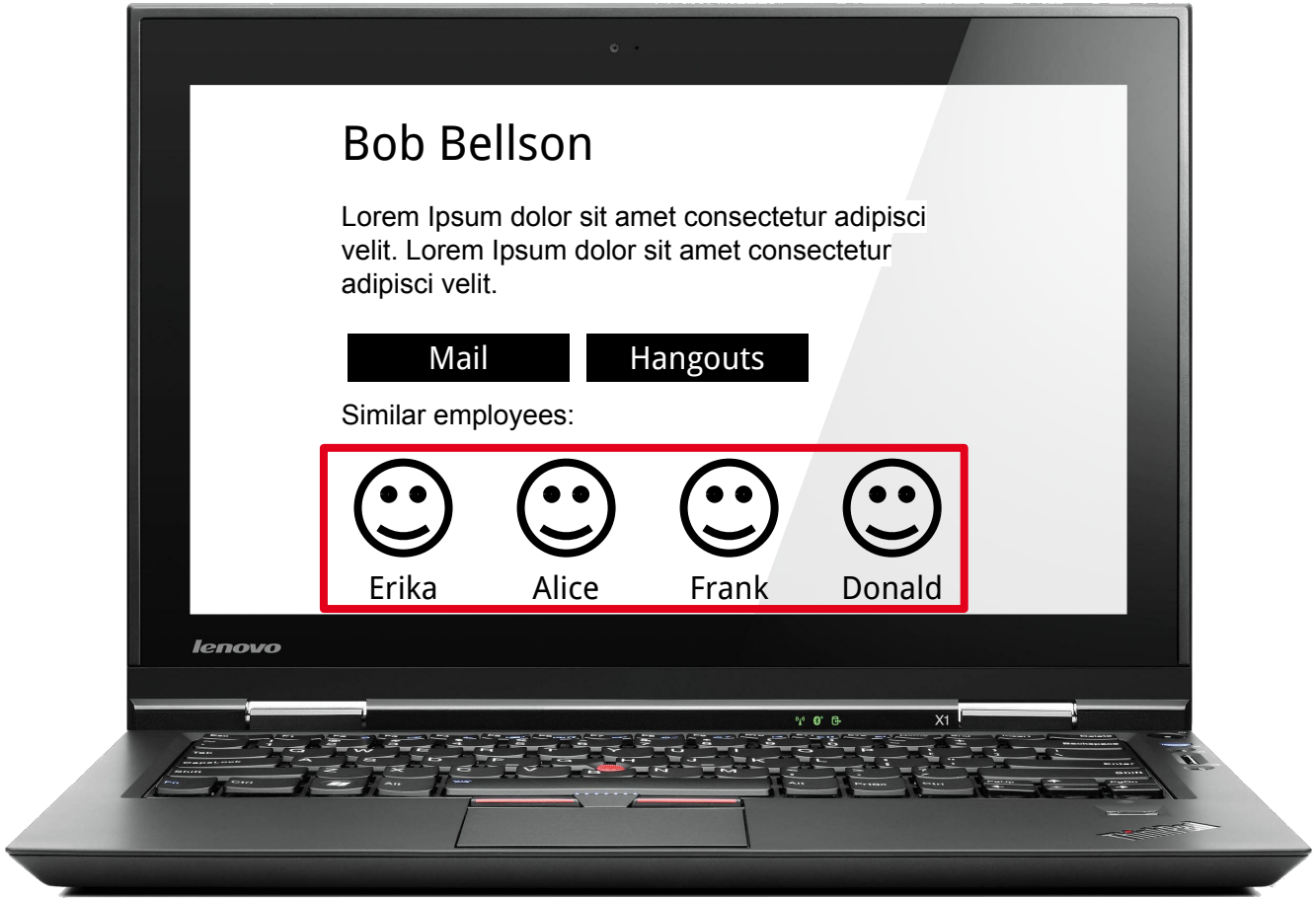



| # | Person | Location | Java | Ruby | C++ | f |
|---|---|---|---|---|---|---|
| 1 | Bob | Hamburg | 2\|3 | 0\|3 | 0\|1 | 0.71 |
| 2 | Charlie | Hamburg | 3\|3 | 2\|1 | 1\|2 | 0.69 |
| 3 | Alice | Hamburg | 2\|1 | 2\|2 | 3\|3 | 0.44 |

# Recommending Similar Users

# Recap: Walkthrough





# Recommending Similar Users

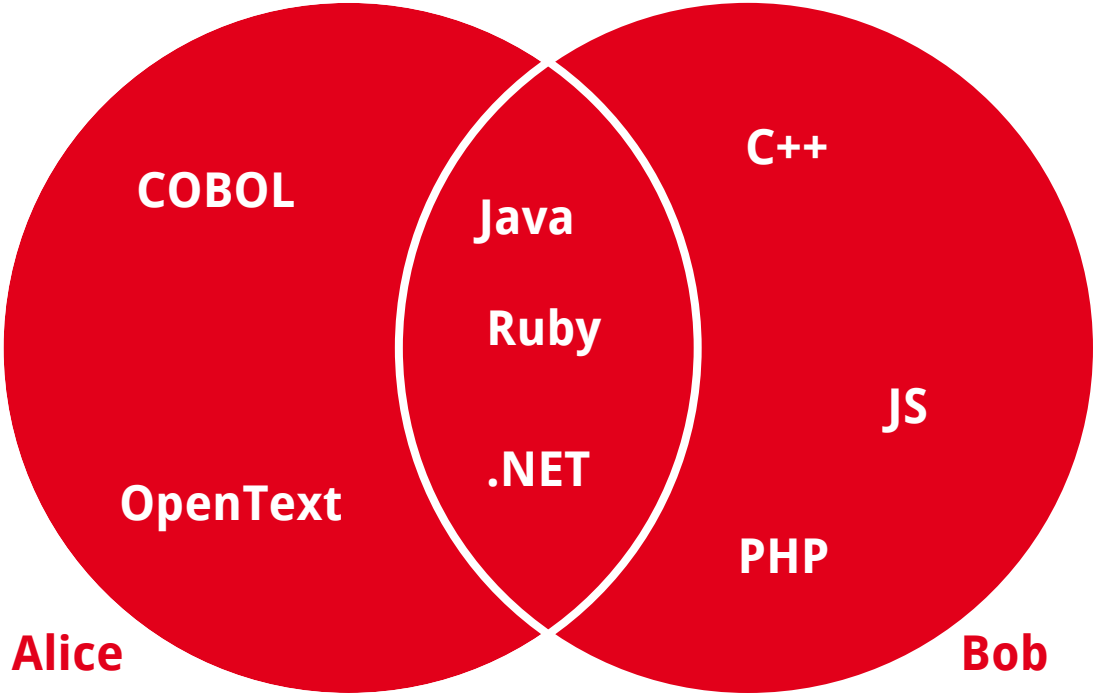


- Recommender System
  - Content-based filtering
- User = Set of Skills
- Jaccard Similarity Coefficient (JSC)

# Recommending Similar Users (JSC)



- Size of intersection: 3
- Size of union: 8
- JSE: 3 / 8  (37.5%)



# Recommending Skills to Search

# Recap: Walkthrough



# Recommending Skills to Search



- Skills: name and icon
- No user context
- Item-based (collaborative) filtering
  - Examine what skills other users' searched for
  - Recommend those skills

# Recommending Skills to Search (Markov Chain)



- Markov Chain
- Model to predict transitions between states
- Only the current state determines the next state

# Recommending Skills to Search (Outline)

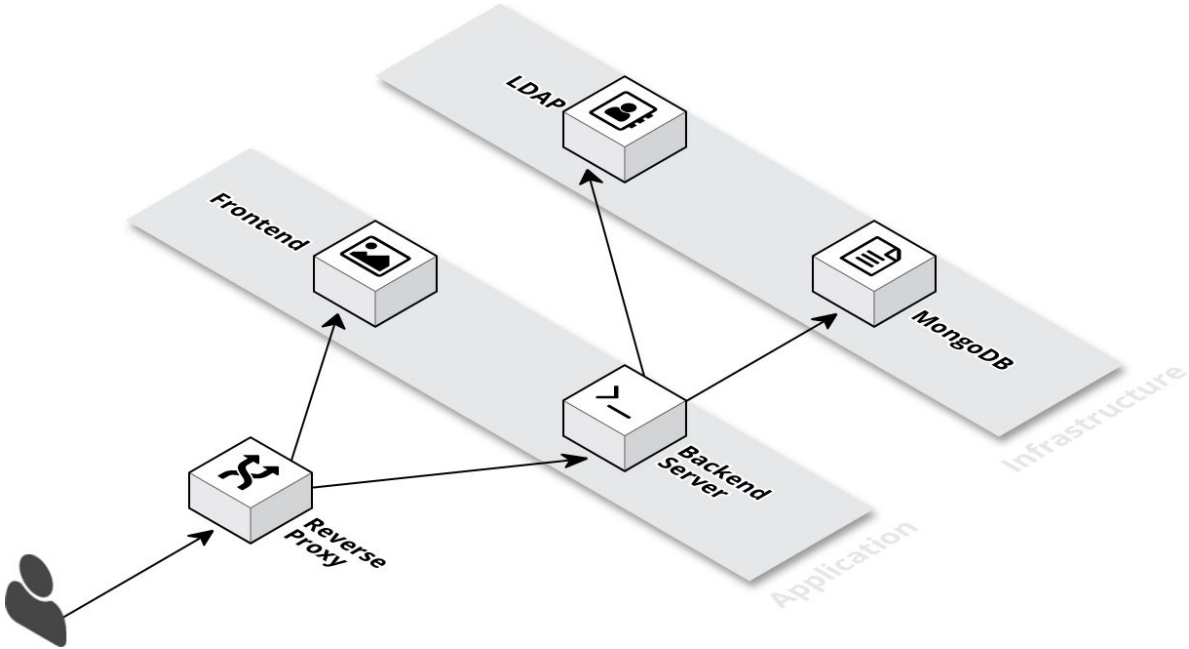


- States: Single Skills
- Transitions: Count of joint searches
- Generate Recommendations:
  - Get all suggestions from entered items
  - Aggregate suggestions (add counts)
  - Remove entered items
  - Sort by search count
  - Return first n items

# Implementation

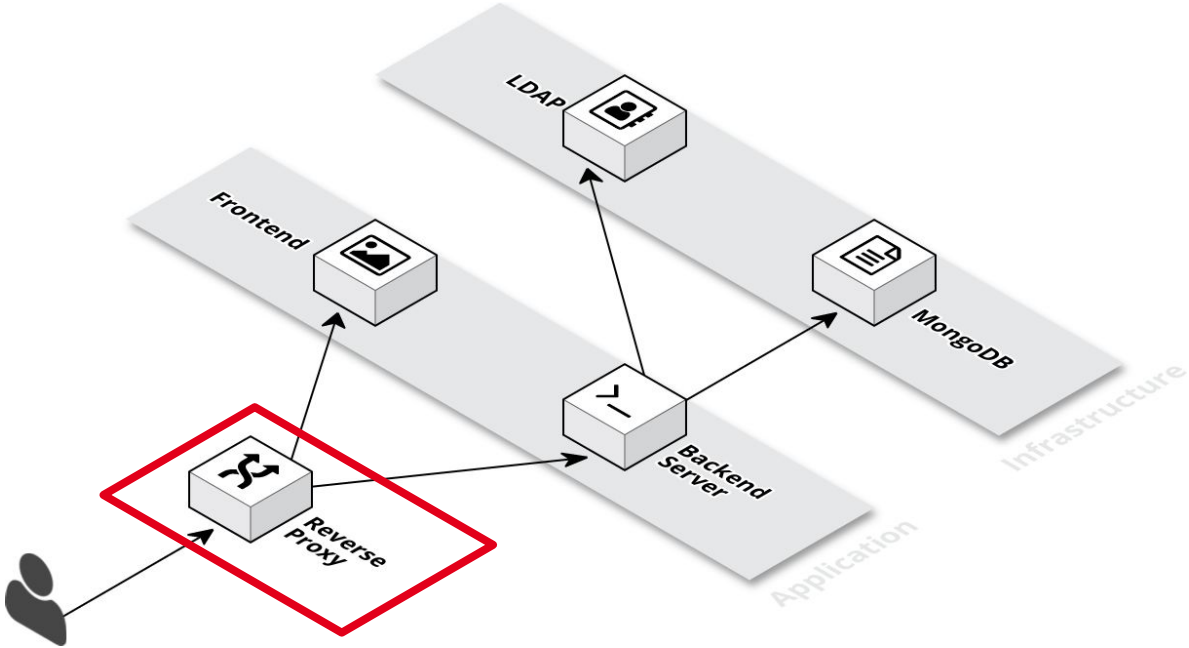
# Application Structure





# Application Structure

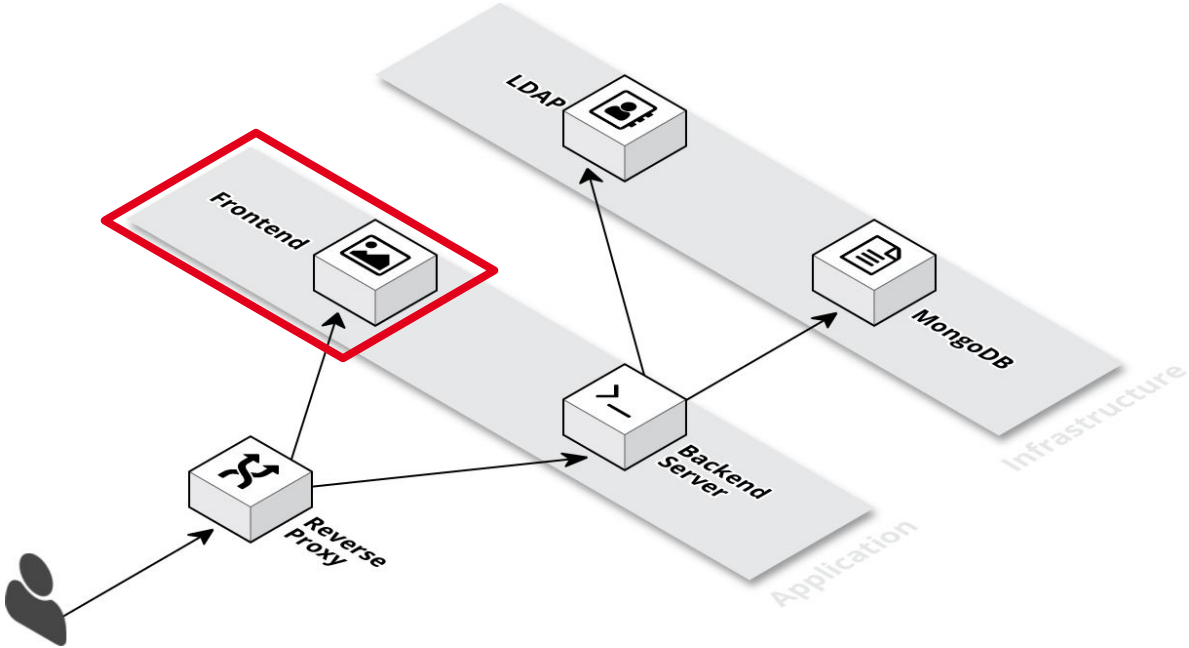





- Serve static files
- Forward API calls
- SSL endpoint

# Application Structure

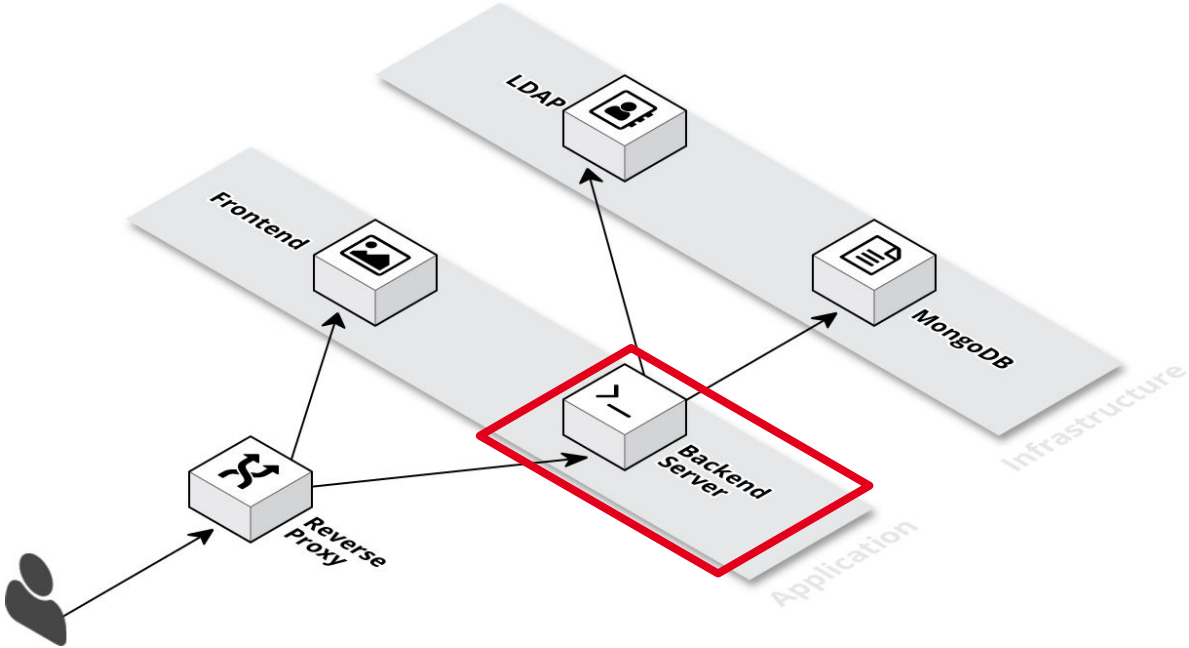





- Graphical Interface
- HTML, CSS, JS, Assets
- Executes API calls (AJAX)

# Application Structure

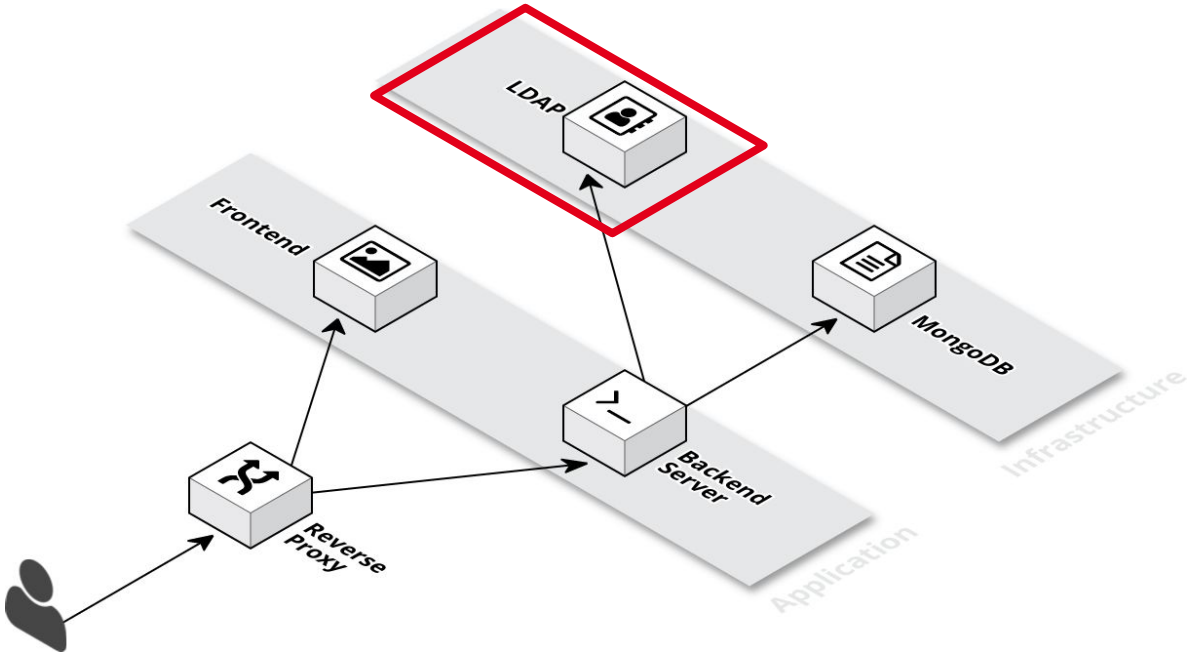





- Main Application
- REST API

# Application Structure

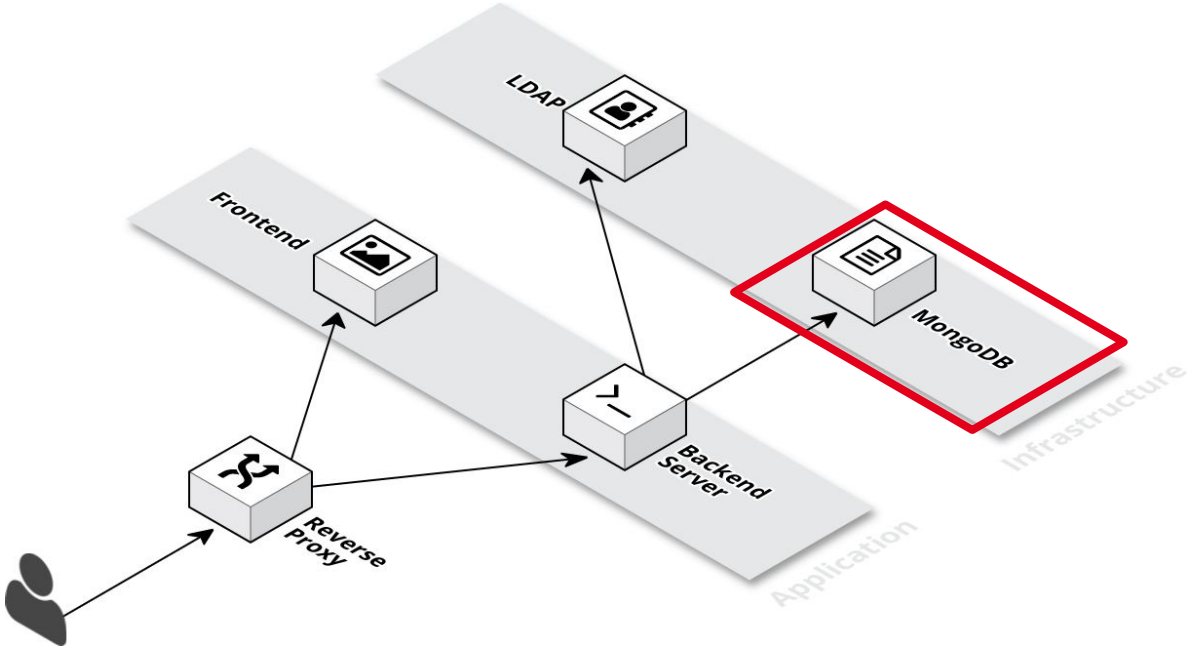





- Personal Data
- Authentication
- Used for most internal services

# Application Structure






- Application data
- NoSQL

# Backend Technologies



# Backend Technologies



# Backend Technologies



# Backend Technologies



# Backend Technologies



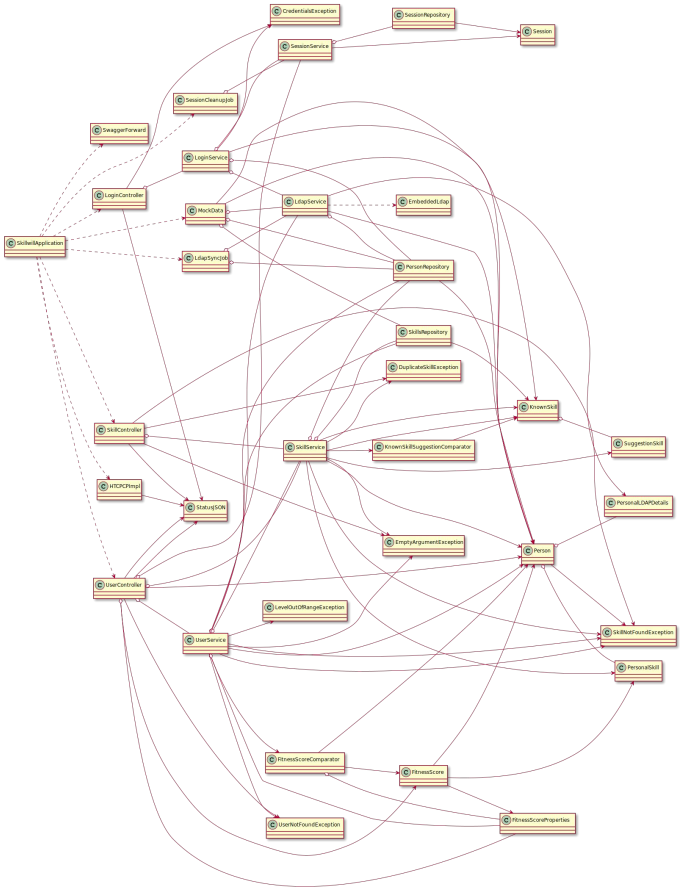
# Backend Technologies



# Backend Structure

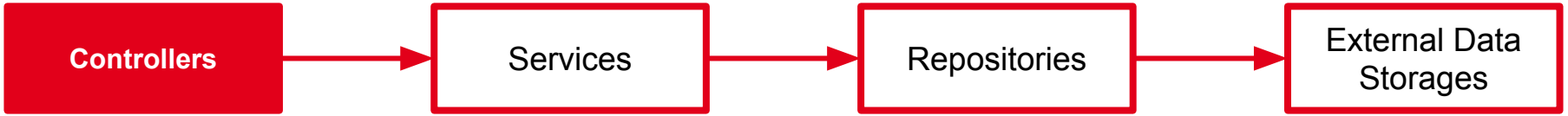


- Controllers
- Services
- Repositories
- Jobs
- Helpers
- Data Types



# Backend Structure

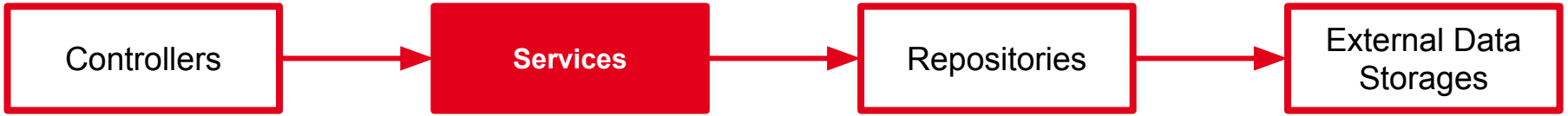





- Listen to API endpoints
- Use services to
  - Get data
  - Send data
  - Send commands
- Convert Objects to JSON

# Backend Structure

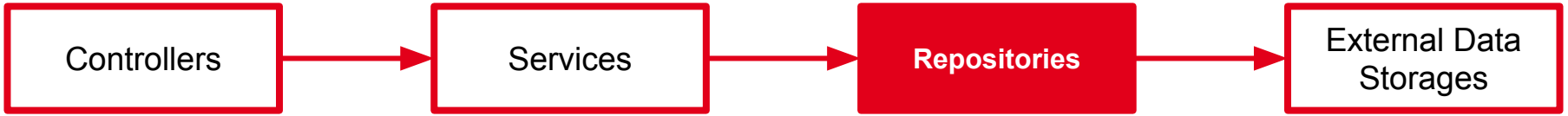





- Get, filter, transform, edit data
- Use repository objects to
  - Retrieve data from external sources
  - Write data to external sources
- Contain business logic

# Backend Structure

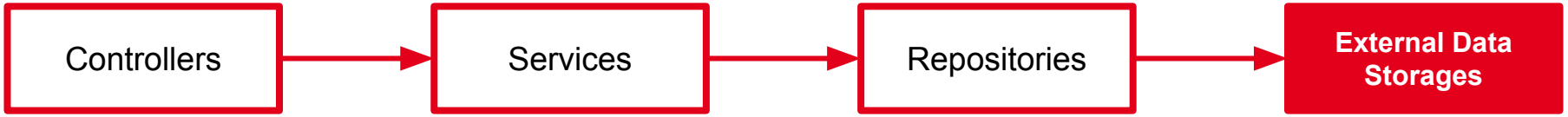





- Spring Data repository objects
- Provide Methods for CRUD operations
- Wrappers to simplify storage access

# Backend Structure






- MongoDB
- Skills
- Persons
- Sessions

# Evaluation

# Evaluation



- Scalability
- Response times (1s)
- Search algorithm
  - Are results distributed uniformly?
  - Do algorithm’s and managers’ ratings correlate?

# Evaluation



- Scalability ✔
- Response times (1s) (✔)
- Search algorithm
  - Are results distributed uniformly?
  - Do algorithm’s and managers’ ratings correlate?

# Evaluation



- Scalability ✔
- Response times (1s) (✔)
- Search algorithm
  - Are results distributed uniformly? ✔
  - Do algorithm’s and managers’ ratings correlate?

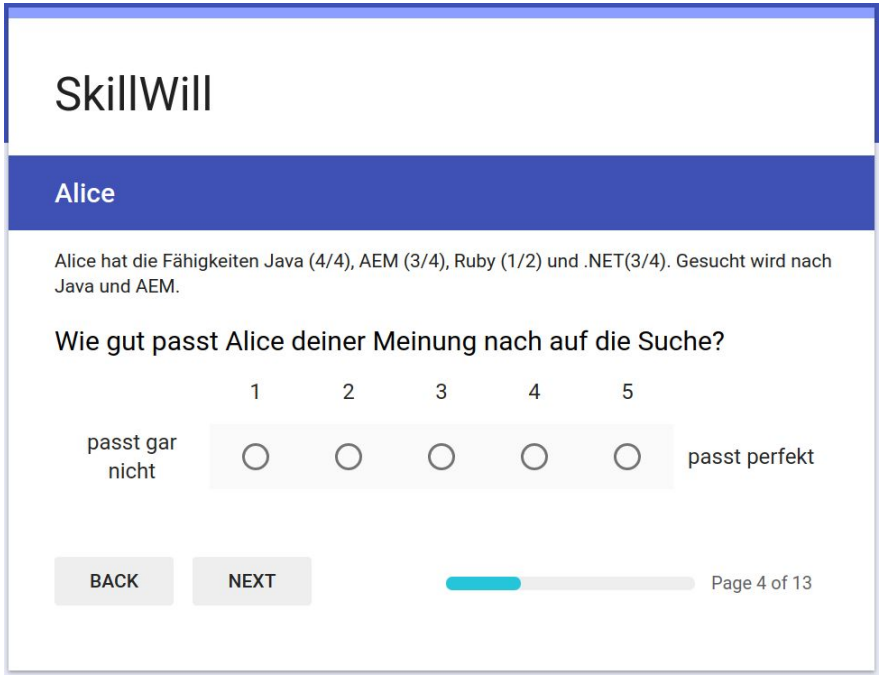
# Evaluation (Survey)

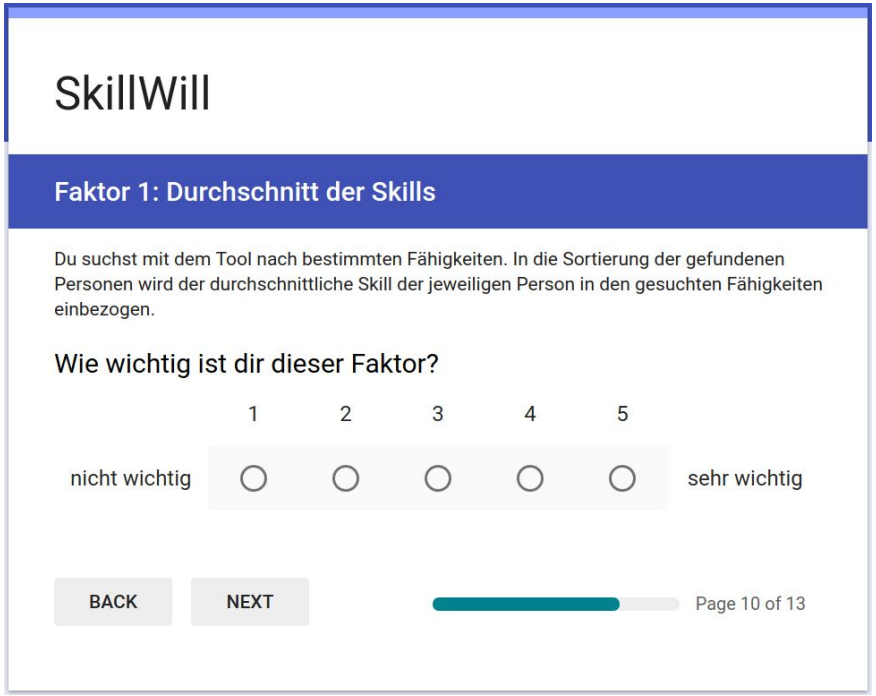


- Online Survey (Google Forms)
- 41 participants (8% of the staff)
- 9 Questions

# Evaluation (Survey)

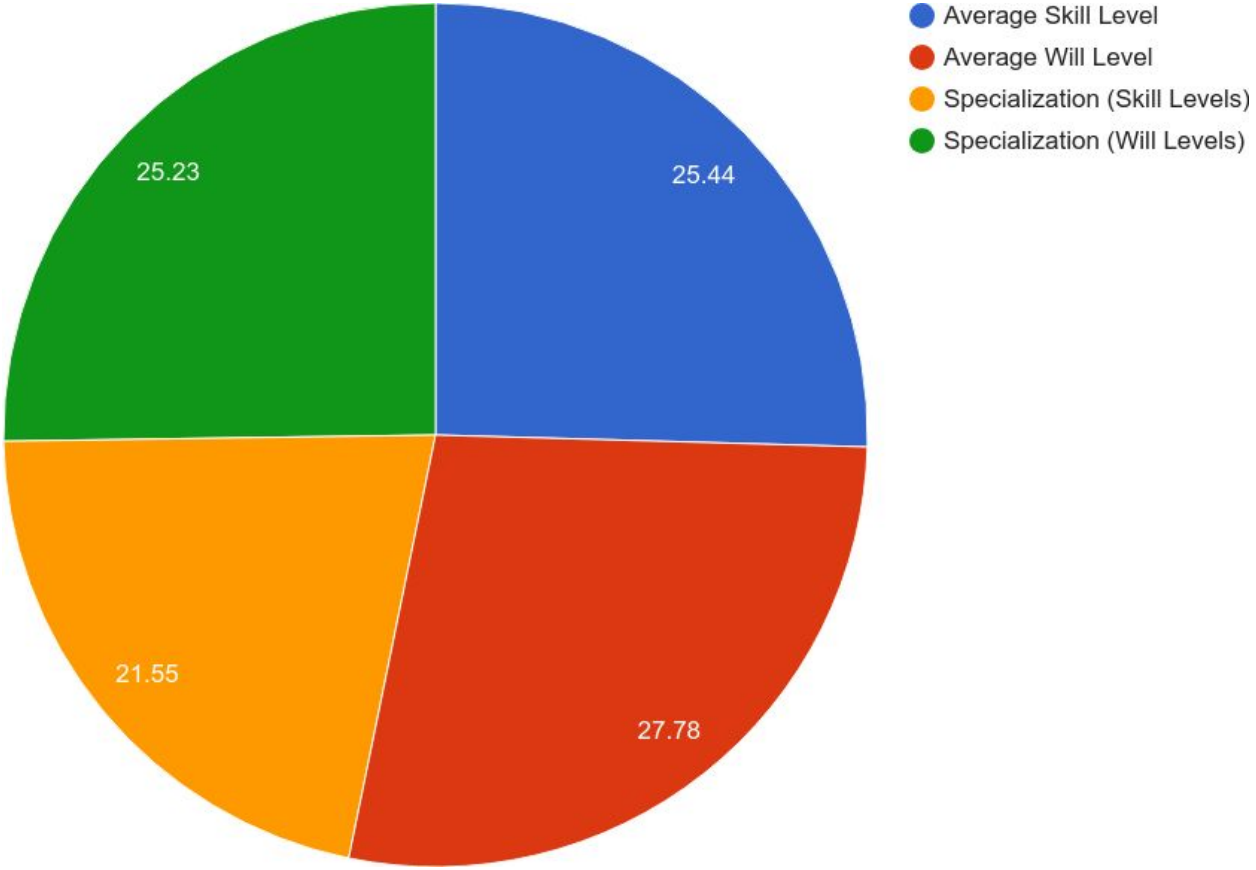




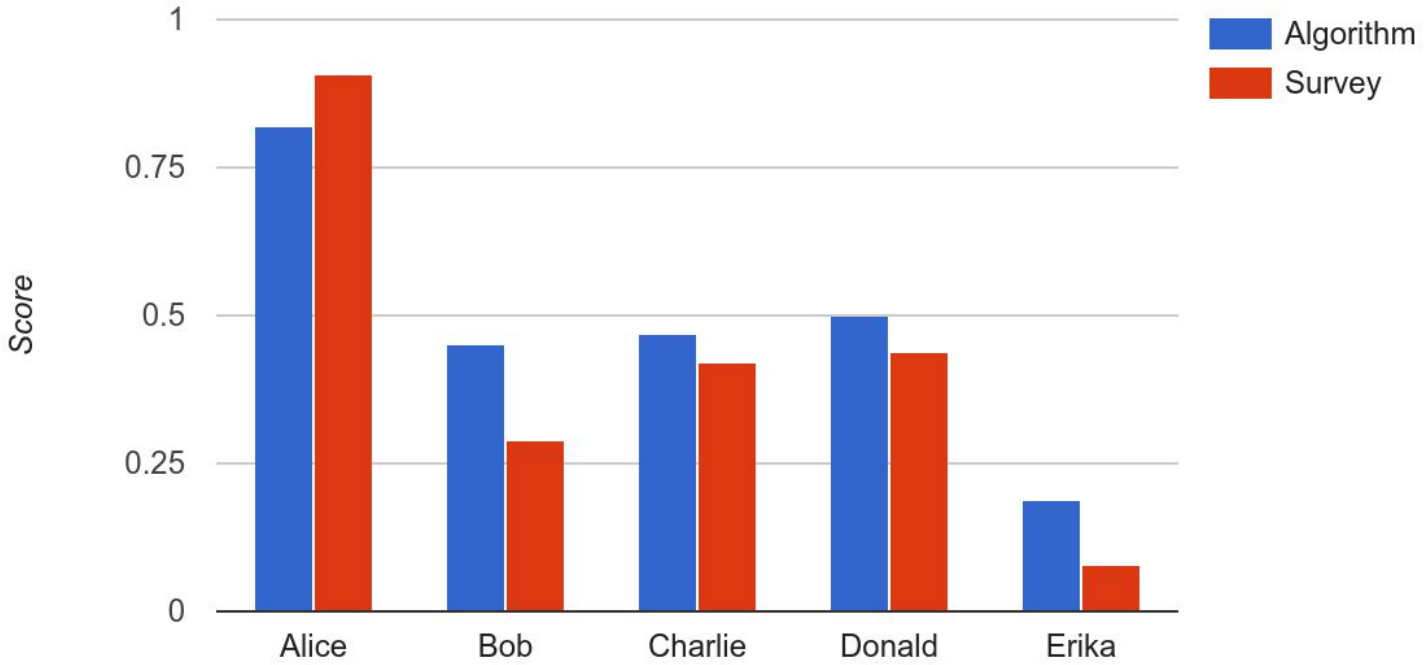
# Evaluation (Survey)





# Evaluation (Survey)





# Evaluation (Survey)



- Two-tailed heteroscedastic T-Test
- $p \geq 0.1 \rightarrow$ No significant deviation
- $p \geq 0.05 \rightarrow$Two data rows deviate significantly

# Evaluation



- Scalability ✔
- Response times (1s) (✔)
- Search algorithm
  - Are results distributed uniformly? ✔
  - Do algorithm’s and managers’ ratings correlate?

# Evaluation



- Scalability ✔
- Response times (1s) (✔)
- Search algorithm
  - Are results distributed uniformly? ✔
  - Do algorithm’s and managers’ ratings correlate? ✔

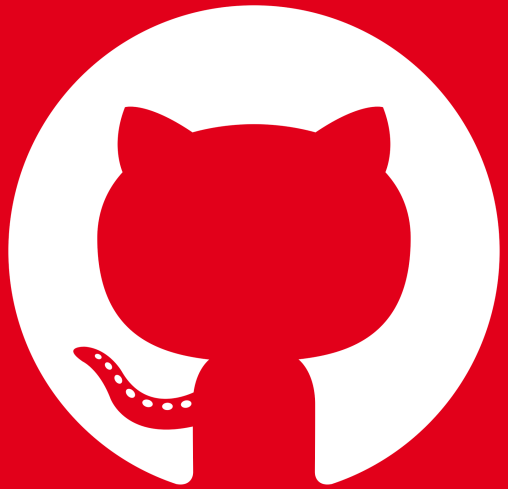
# That’s it

# Demo

# Demo

# https://demo.torben.xyz

# Thanks



github.com/t0rbn/BSc
github.com/sinnerschrader*

*soon

# Contact

Github: @t0rbn
Mail: 3reetz@informatik.uni-hamburg.de
More: http://torben.xyz

# Image Sources

Thinkpad X1: https://www.bhphotovideo.com
Google Now on Nexus 5: ttps://www.androidcentral.com/google-now
Screenshot engage!: https://www.infoniqa.com/hr-software/talent-management
Application Architectures created with https://cloudcraft.co/
Octocat: https://github.com/logo
Java Logo: https://upload.wikimedia.org/wikipedia/en/thumb/3/30/Java_programming_language_logo.svg/412px-Java_programming_language_logo.svg.png
OpenJDK Logo: https://upload.wikimedia.org/wikipedia/commons/thumb/1/18/OpenJDK_logo.svg/2000px-OpenJDK_logo.svg.png
Swagger Logo: https://avatars2.githubusercontent.com/u/7658037?v=3&s=400
Junit Logo: http://junit.org/junit4/images/junit5-banner.png
Maven Logo: https://maven.apache.org/images/maven-logo-black-on-white.png
Git Logo: https://git-scm.com/images/logos/logomark-orange@2x.png
Gitlab Logo: https://upload.wikimedia.org/wikipedia/commons/thumb/1/18/GitLab_Logo.svg/2000px-GitLab_Logo.svg.png
Flapdoodle Logo: https://avatars0.githubusercontent.com/u/1661811?v=3&s=280
Unboundid  Logo: https://media.licdn.com/mpr/mpr/shrink_200_200/AAEAAQAAAAAAAAUtAAAAJGEyYmI1MDlmLWFkMjktNGU3ZS1hOTk2LWM0MDJjZTQ5MzA2Mw.png
Jacoco Logo: https://cdn.liviutudor.com/wp-content/uploads/2016/02/Jacoco.png
Spring Logo: https://upload.wikimedia.org/wikipedia/en/2/20/Pivotal_Java_Spring_Logo.png

# Q&A

# Bonus Slides

# Requirements

# Functional Requirements



- Person search
  - Enter skills → find best matching person
- User profiles
  - Skills, personal data, direct contact
  - Enter own skills (knowledge, motivation)
  - Login
- Management of registered Skills
  - Pool of predefined skills
  - Add new skills
  - Rename skills
  - Delete skills

# Non Functional Requirements



- Different devices
  - Primary: Desktops
  - Mobile devices optional
- Browser support
  - Chrome, Firefox, Safari
  - No support for IE/Edge
- Response Times (RAIL)
  - 100ms to acknowledge input
  - 1s to finish rendering results
- Scalability
  - Increased number of users should not be a problem
  - Enlarge storage and computing resources

# Realated Work

# Fitness Score (Research)



- Ivanovska et. al: Algorithms for Effective Team Building
- Canós-Darós: An algorithm to identify the most motivated employees
  - General Motivation ↔ Task specific
  - Asking employees to rate their motivation generates suitable data
- Spoonamore et. al: Matching Sailors to Positions Based on Skill

# Matching Sailors to Positions Based on Skill



- Multiple Factors
  - Rating
  - Pay grade
  - NECs
- Basic priciple: weighted mean of factors
  - $S = \alpha$ ratingscore + $\beta$ paygradescore + $\gamma$ NECscore

# Fitness Score Algorithm

# Fitness Score (Definitions and Helpers)



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

$$S = \{Java, Ruby, C++, ...\}$$

$$E = \{x \in S \mid \text{employee has skill x}\}$$

$$Q = \{x \in S \mid \text{user searches for skill x}\}$$

$$v_s : E \mapsto V$$

$$v_w : E \mapsto V$$

# Fitness Score (Factors)



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

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

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

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

# Fitness Score (Basic Algorithm)



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

# Refining the Fitness Score Algorithm



- Estimated Factors ≅ 0.25
- Setting all to 0.25
  - Simplifies algorithm
  - No drastic effect on accuracy

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

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

# Recommender Systems

# Recommender Systems (Definition)



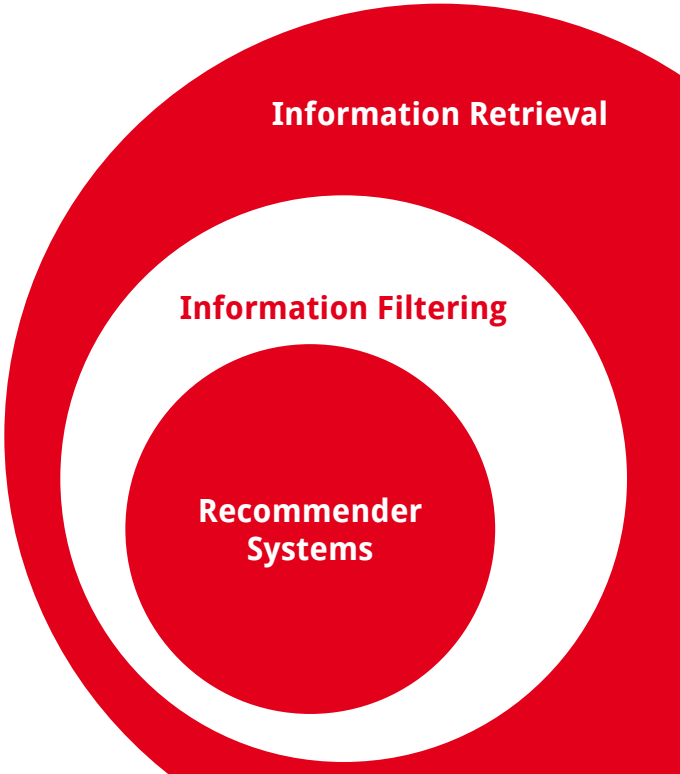
“information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of [...] information  [sic]”

I SINKAYE , F.O. ; F OLAJIMI , Y.O. ; O JOKOH , B.A.: Recommendation systems: Principles, methods and evaluation.

# Recommender Systems and IR



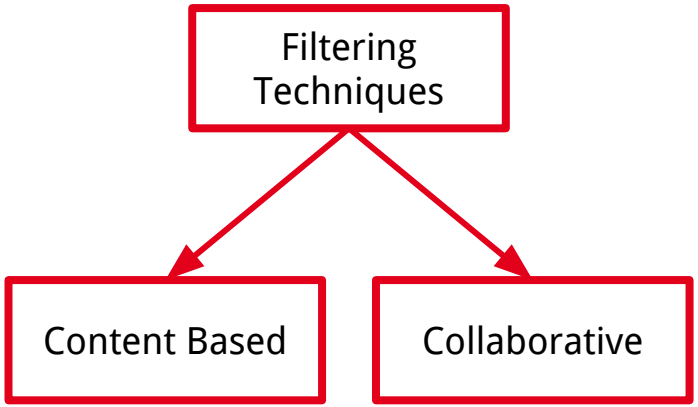
- Subset of IR systems
- Both find relevant information
- Recommender Systems → “zero query” IR
  - IR: user actively searches
  - Recommender: system proactively recommends
  - Person search is not a recommender system



# Filtering Techniques



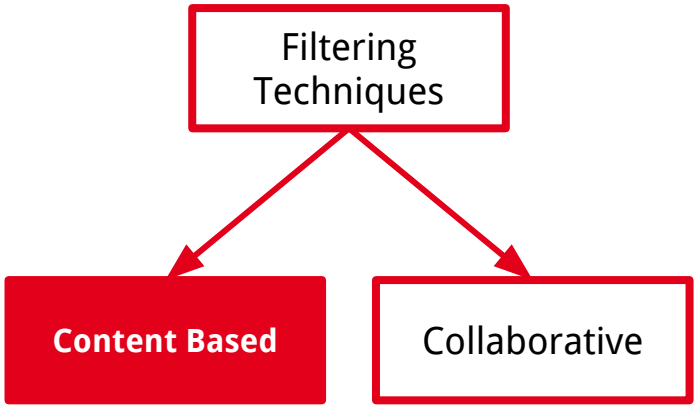
- Isinkaye et al.
- Techniques to find the items to suggest
- Hybrid filtering: combine multiple techniques



# Filtering Techniques (Content Based)



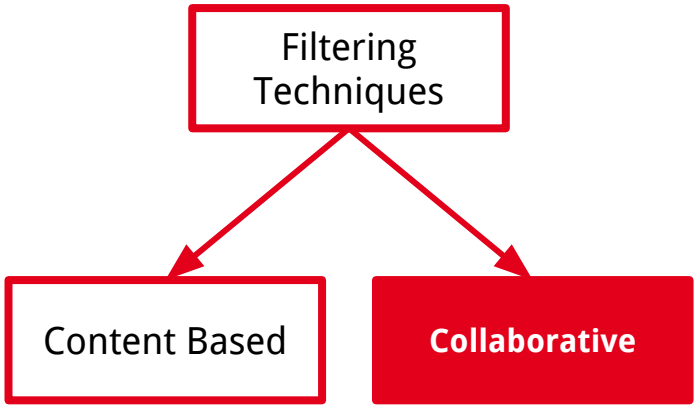
- Examine content
- Find items that are similar to the ones the user interacted with



# Filtering Techniques (Collaborative)



- Users behave similarly
- Find “neighbours”
- Recommend items neighbours interacted with
- Subclasses:
  - Model based
  - Memory based
    - Item based
    - User based



# Example: Amazon (Collaborative)



# Example: Heise (Content Based)



# Recommending Skills to Search (Example)

# Recommending Skills to Search (Example)




- Entered: Java, PHP
- Number to recommend (n): 1

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

# Recommending Skills to Search (Example)




- PHP (7)
- CSS (3)
- COBOL (1)
- Java (7)
- CSS (9)
- COBOL (5)

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

# Recommending Skills to Search (Example)




- PHP (7)
- Java (7)
- CSS (12)
- COBOL (6)

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

# Recommending Skills to Search (Example)




- CSS (12)
- COBOL (6)

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

# Recommending Skills to Search (Example)




- CSS (12)
- COBOL (6)

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

# Recommending Skills to Search (Example)




- CSS (12)

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

# API Endpoints

# REST API Endpoints (Login)



| URL | Method | Feature |
|---|---|---|
| /login | POST | User Login |
| /logout | POST | User Logout |

- Login to edit user’s skills
- No login needed to search

# REST API Endpoints (Users)



| URL | Method | Feature |
|---|---|---|
| /users | GET | Main person search function |
| /users/{user} | GET | Get specific user's details |
| /users/{user}/skills | POST | Add/Update user's skills |
| /users/{user}/skills | DELETE | Remove skill from user's profile |
| /users/{user}/similar | GET | Recommend similar users |

- New users added on their first login
- No removing of users

# REST API Endpoints (Skills)



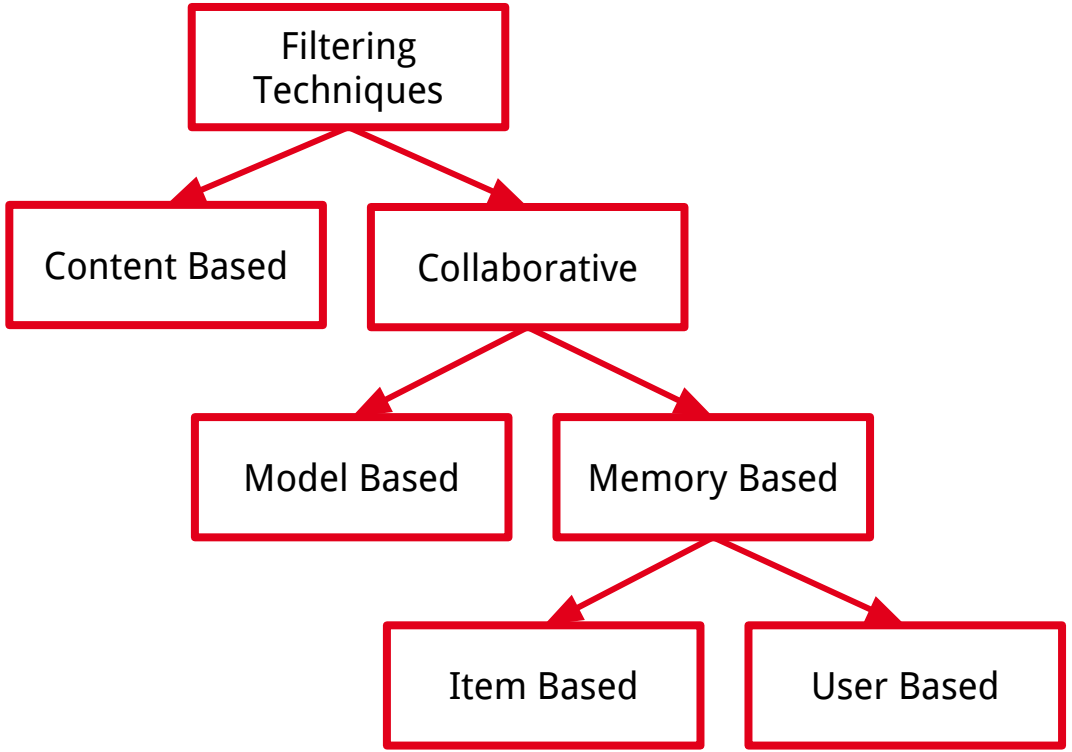
| URL | Method | Feature |
|---|---|---|
| /skills | GET | Get all skills/text autocomplete |
| /skills | POST | Create new Skill |
| /skills/next | GET | Recommend next skill to enter |
| skills/{skill} | PUT | Edit skill (rename) |
| skills/{skill} | DELETE | Delete skill |

# Concept: Collaborative Filtering Techniques

# Filtering Techniques



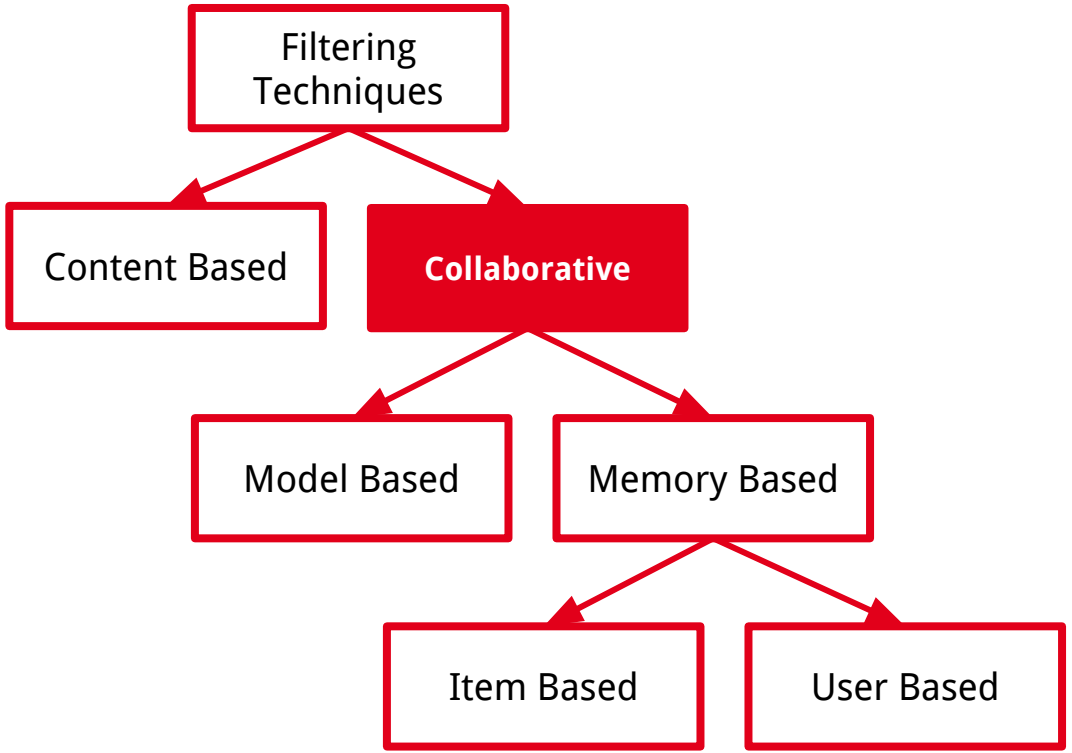
- Isinkaye et al.
- Techniques to find the items to suggest
- Hybrid filtering: combine multiple techniques



# Filtering Techniques (Collaborative)



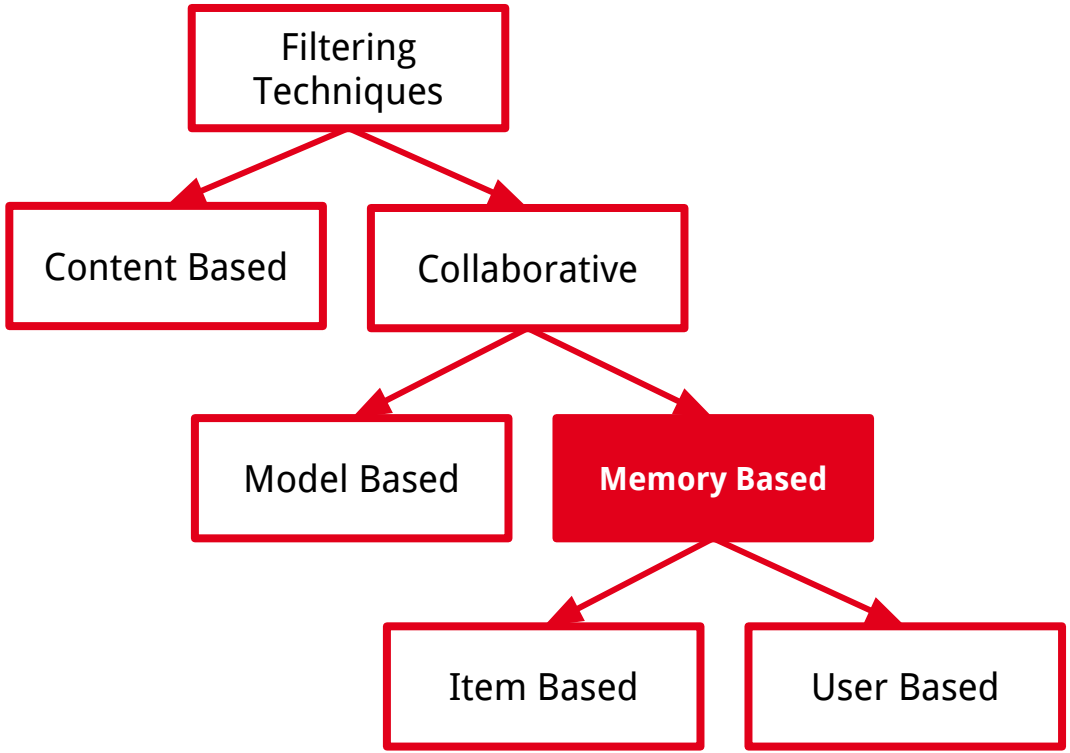
- People behave similarly
- Find neighbours



# Filtering Techniques (Memory Based)



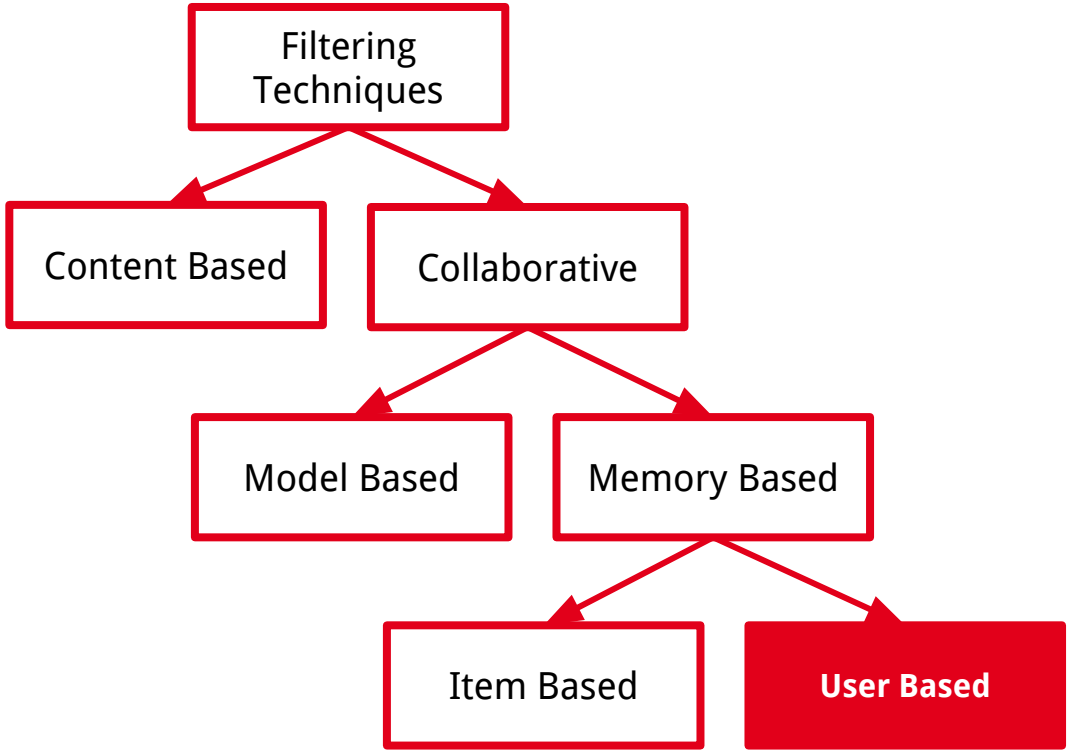
- Subset of collaborative filtering
- Operates directly on saved interaction history



# Filtering Techniques (User Based)



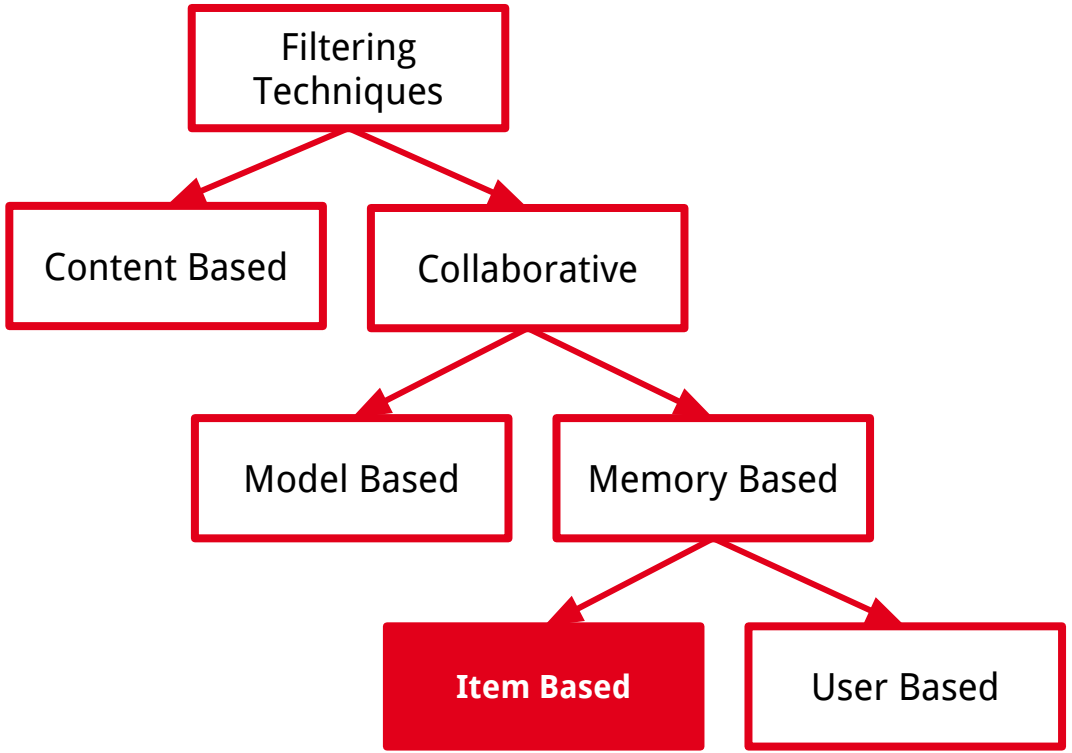
- Subset of memory based filtering
- Save interaction history per user
- Find groups of users that have similar interaction histories



# Filtering Techniques (Item Based)



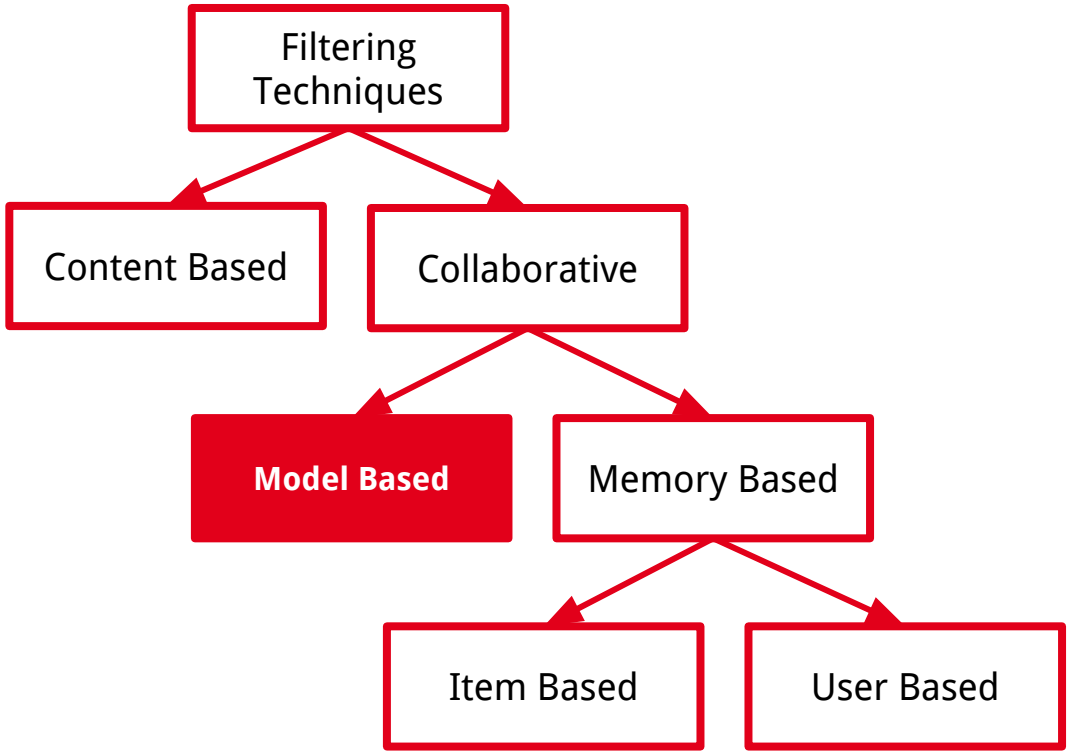
- Subset of memory based filtering
- Save interaction history per item
- Find items that are similar to each other



# Filtering Techniques (Model Based)



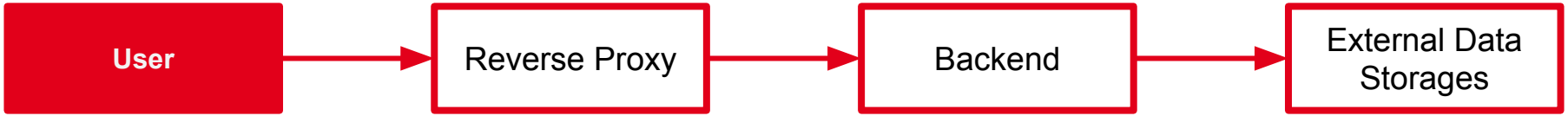
- Subset of collaborative filtering
- Model used to create suggestions
- Interactions to learn the model



# Example: API Call

# API Call (Search for Java and Ruby in HH)

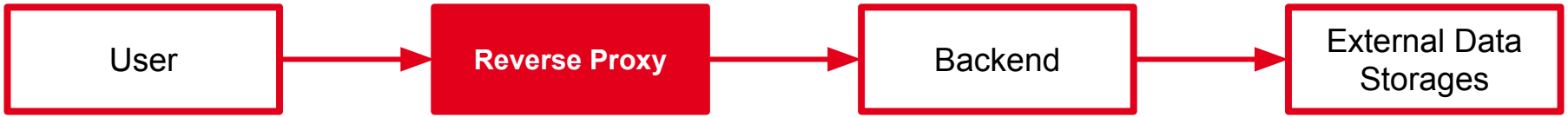





- Search for Java and Ruby in HH
- Browser Calls API
  - *api.some.tld/users?skills=Java,Ruby&location=Hamburg*

# API Call (Search for Java and Ruby in HH)

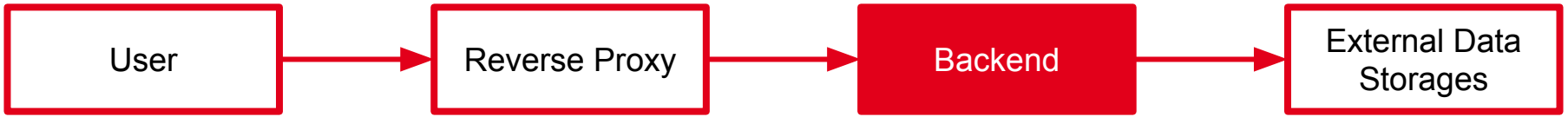





- Recognize API Call (Domain)
- Forward to Backend Server

# API Call (Search for Java and Ruby in HH)

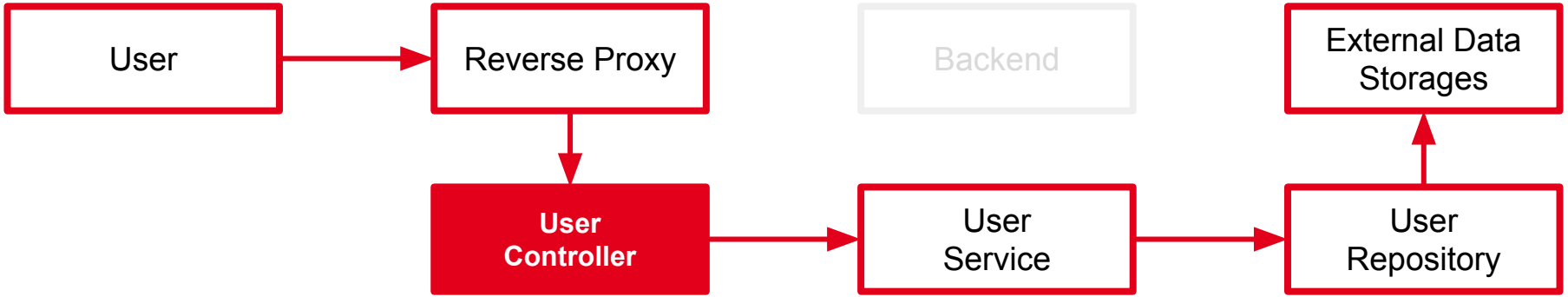





- Waits for HTTP Requests
- Dispatching to Controller

# API Call (Search for Java and Ruby in HH)

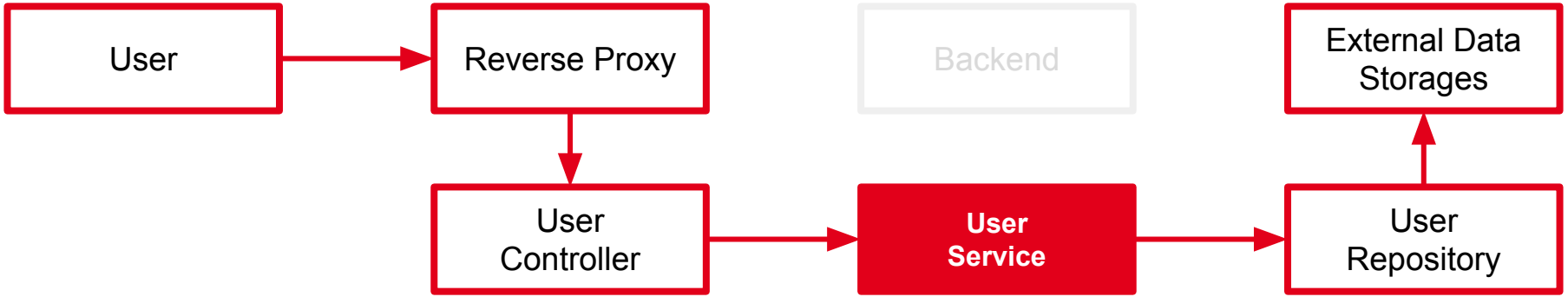





- Call method based on request URL and Parameters
  - /users/
  - ?skills=Java,Ruby&location=Hamburg
- Request matching persons from UserService

# API Call (Search for Java and Ruby in HH)

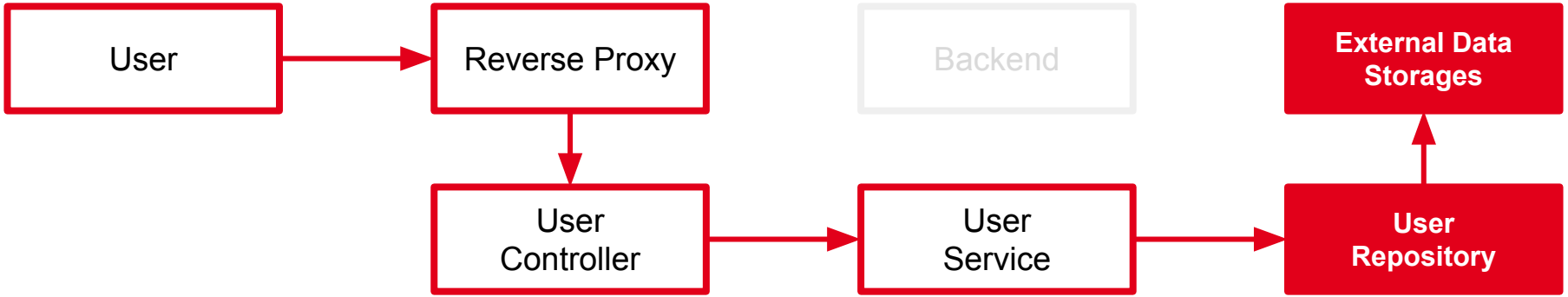





- Request needed users from Repository

# API Call (Search for Java and Ruby in HH)

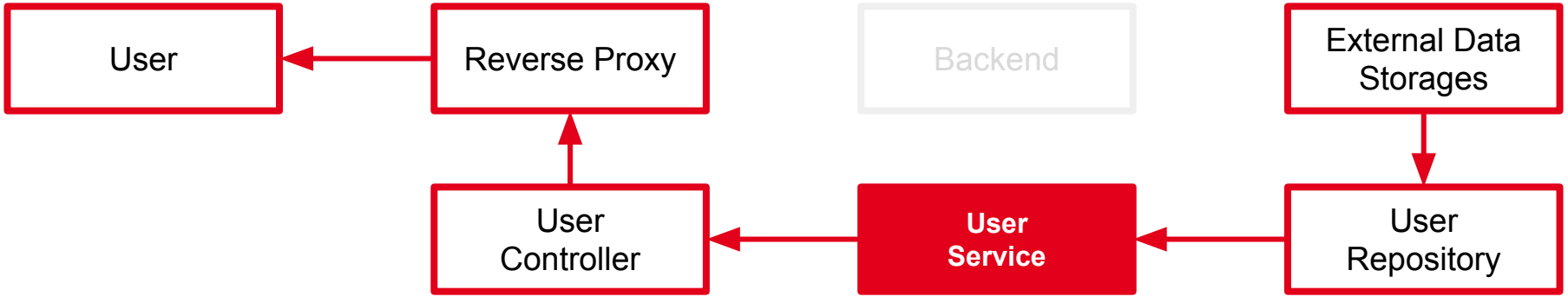





- Get data from MongoDB
- Return user objects to UserService

# API Call (Search for Java and Ruby in HH)

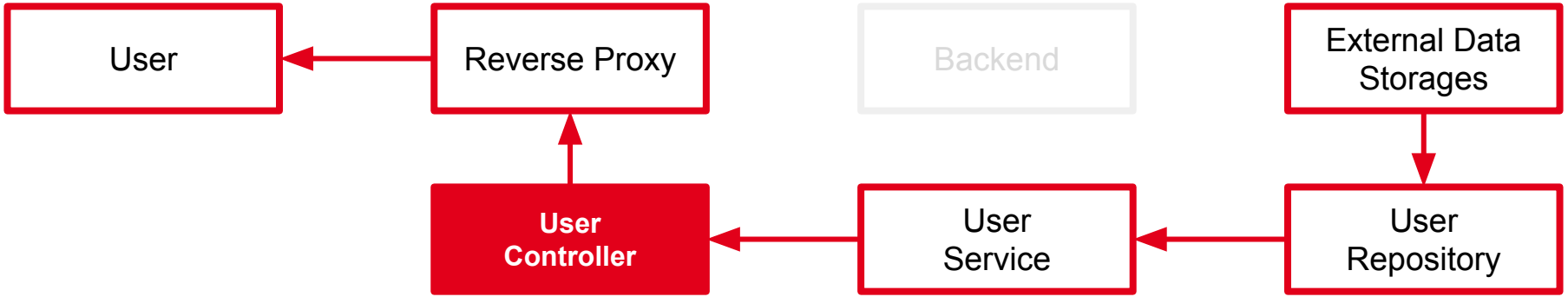





- Process retrieved user objects
- Apply search and fitness score algorithms

# API Call Cycle (Search for Java and Ruby in HH)

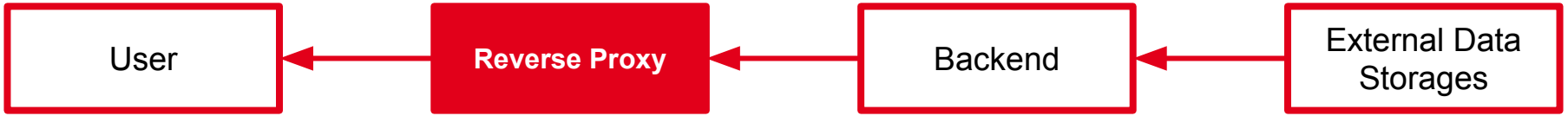





- Convert found user objects to JSON
- Return HTTP Response
  - In case of error, return corresponding HTTP Code

# API Call Cycle (Search for Java and Ruby in HH)

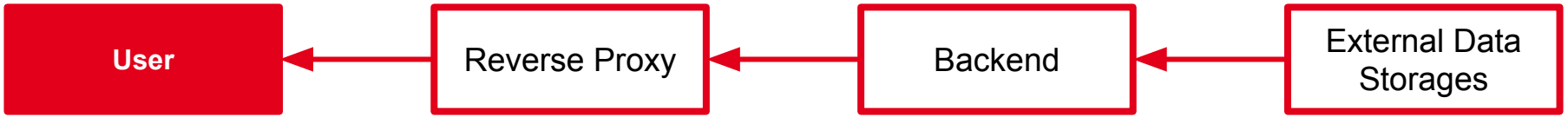





- Forward JSON response to client

# API Call Cycle (Search for Java and Ruby in HH)





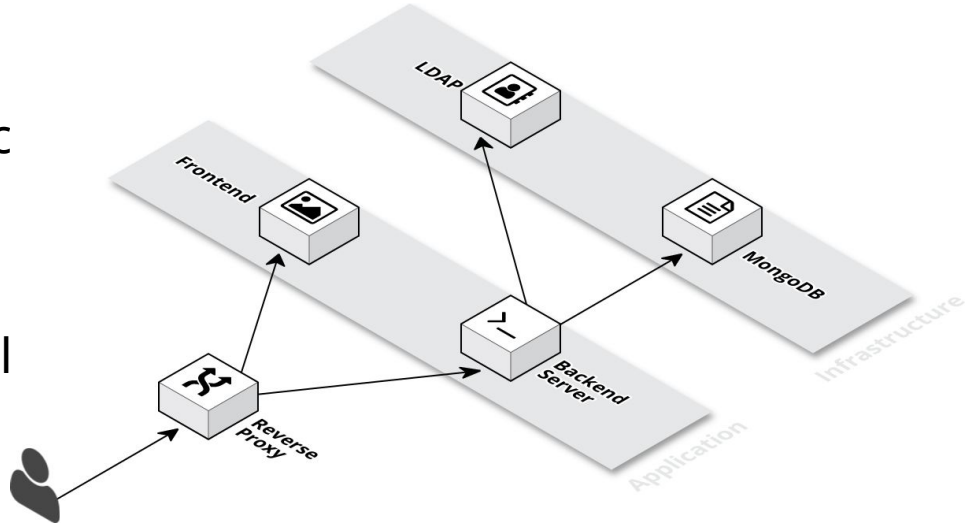

- Parse JSON response
- Render result list

# Evaluation: Scalability

# Evaluation (Scalability)



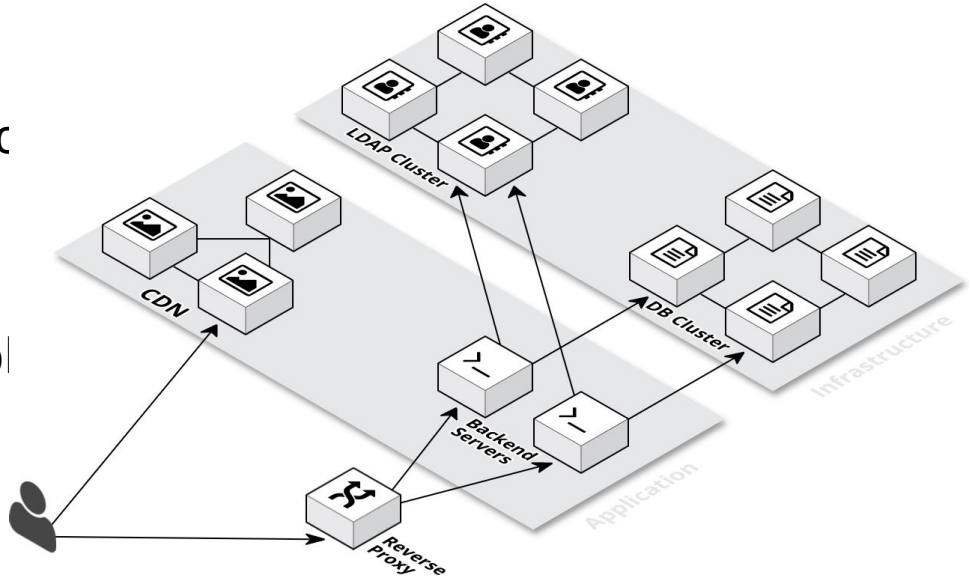
- MongoDB
  - Designed and shown to be sc
- LDAP
  - Six servers
  - Cluster is transparent to appl
- Frontend
  - CDN
- Backend
  - Stateless Application
  - Reverse proxy as load balancer
  - Tested



# Evaluation (Scalability)



- MongoDB
  - Designed and shown to be sc
- LDAP
  - Six servers
  - Cluster is transparent to appl
- Frontend
  - CDN
- Backend
  - Stateless Application
  - Reverse proxy as load balancer
  - Tested



# Evaluation (Scalability)




**SINNERSCHRADER AG UND ACCENTURE VEREINBAREN ZUSAMMENSCHLUSS; ACCENTURE KÜNDIGT FREIWILLIGES ÖFFENTLICHES ÜBERNAHMEANGEBOT FÜR SÄMTLICHE AKTIEN DER SINNERSCHRADER AG AN**

AD-HOC NEWS

20. Februar 2017, 7:21

Die SinnerSchrader Aktiengesellschaft („SinnerSchrader") hat heute nach entsprechenden Beschlüssen von Aufsichtsrat und Vorstand mit der Accenture Digital Holdings GmbH, einer 100-prozentigen... **weiter lesen**

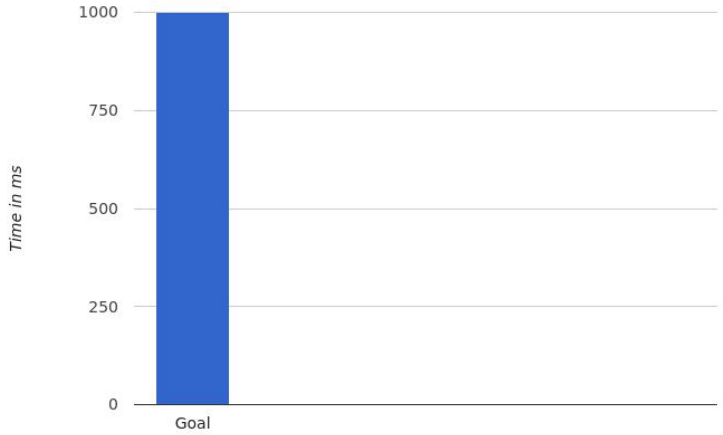
**Alle Finanznachrichten** ›

# Evaluation: Response Times

# Evaluation (Response Times)



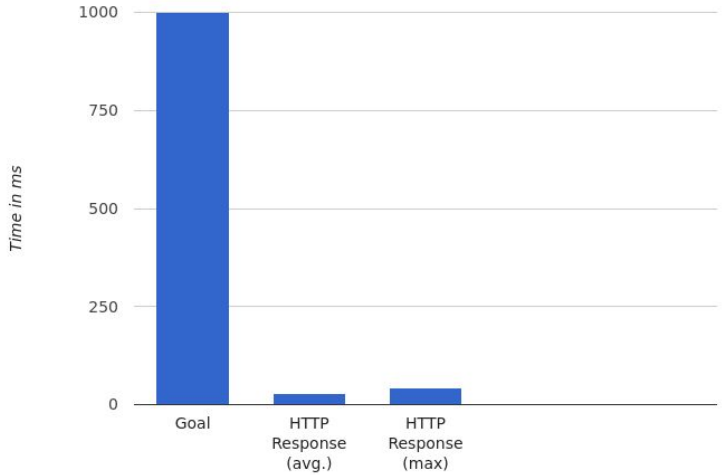
- Goal: 1s max to finish rendering



# Evaluation (Response Times)



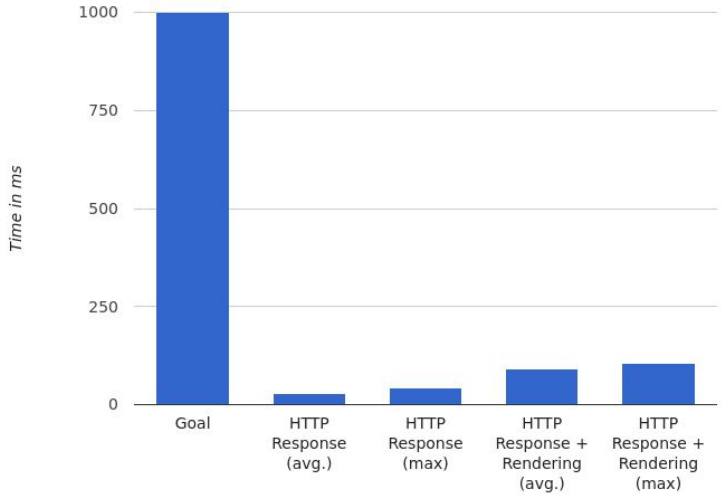
- Goal: 1s max to finish rendering
- HTTP Response: 33ms/44ms (40 samples)



# Evaluation (Response Times)



- Goal: 1s max to finish rendering
- HTTP Response: 33ms/44ms (40 samples)
- + Rendering: 90ms/106ms (16 samples)



# Evaluation: Uniform Distribution of Fitness Scores

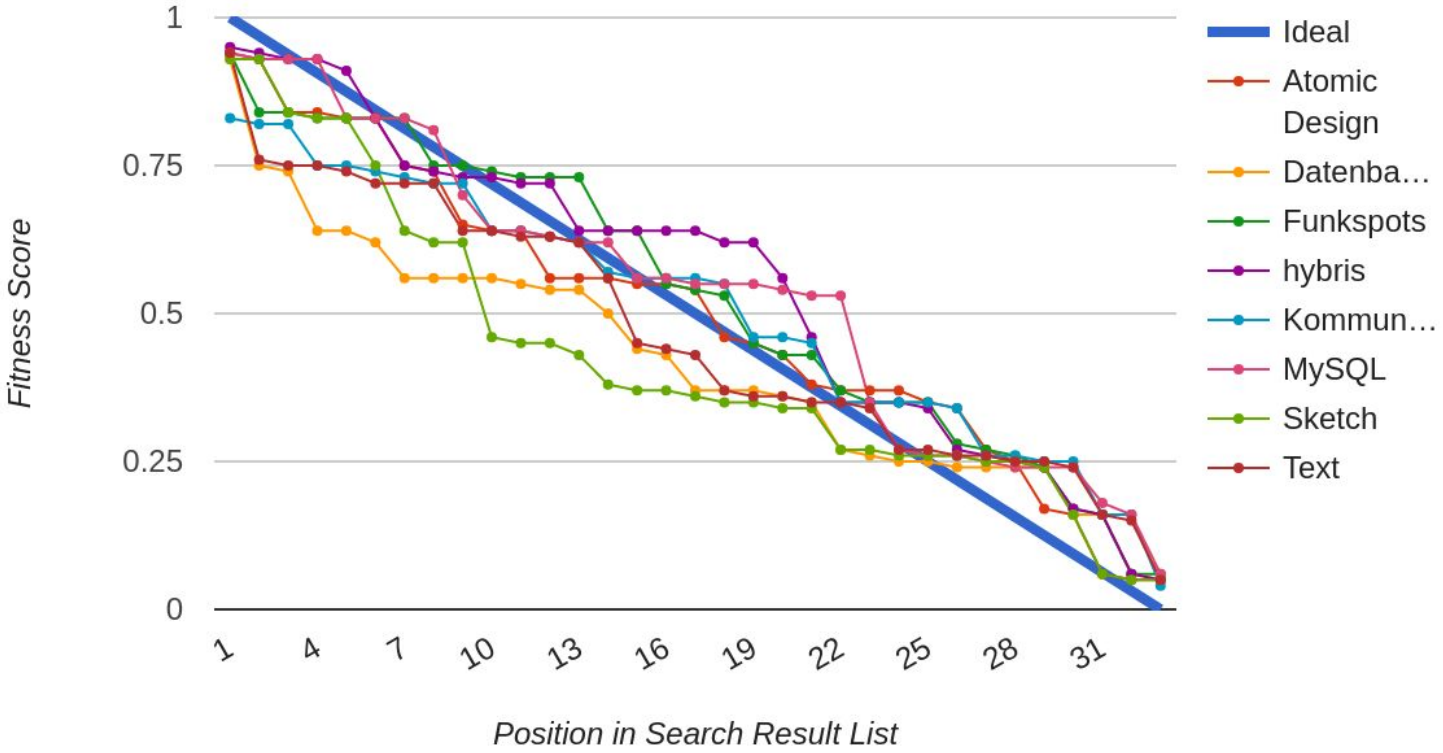
# Evaluation (Distribution of Fitness Scores)



- 100 users
- ≤ 17 skills per users
- Random levels
- Fitness of employees distributed uniformly
- Are fitness scores distributed uniformly?

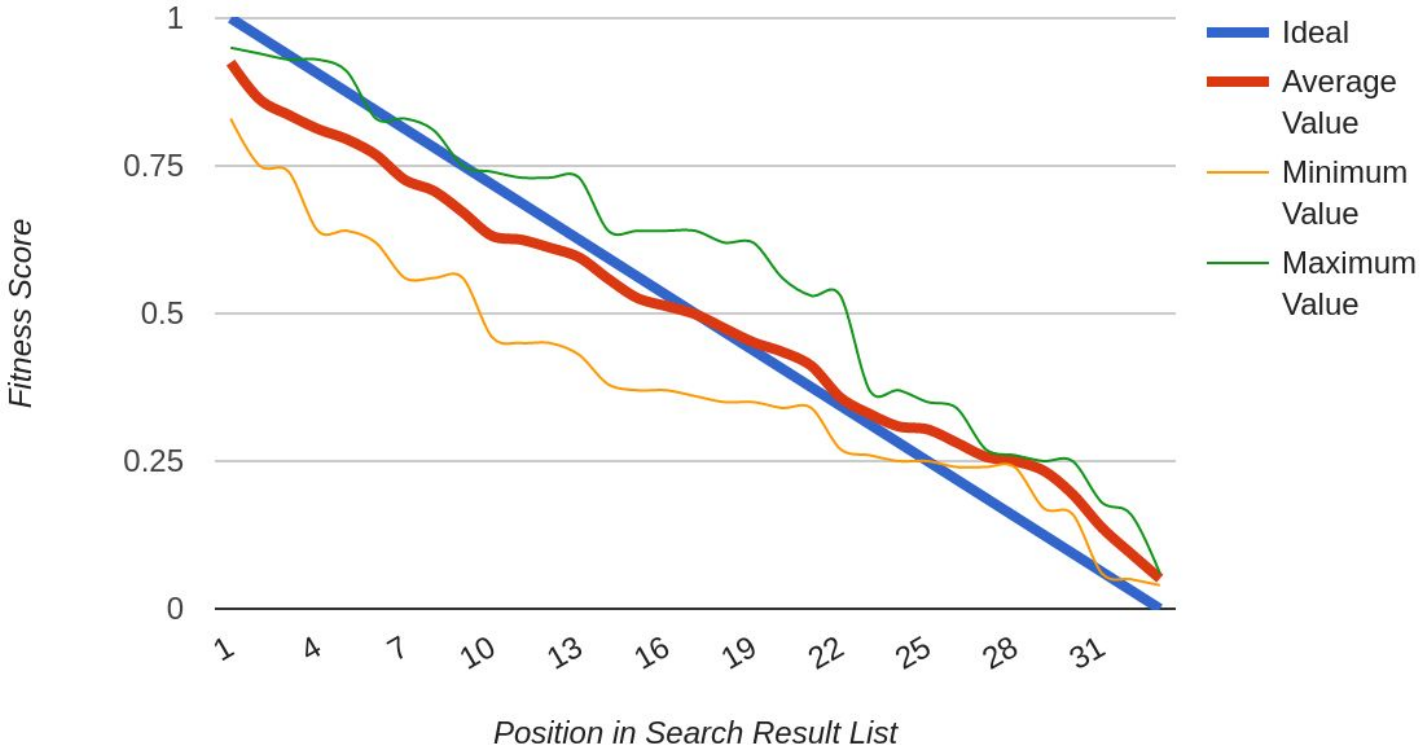
# Evaluation (Distribution of Fitness Scores)





# Evaluation (Distribution of Fitness Scores)





# Evaluation (Distribution of Fitness Scores)



- Average Deviation: 6%
- Maximum Deviation: 27%