



DI Michael Schwarz, BSc

# **Quality Assurance for Human Computation Based Recommendation**

## **MASTER'S THESIS**

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

**Graz University of Technology**

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Alexander Felfernig

Institute for Software Technology

Graz, January 2017

## **AFFIDAVIT**

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

---

Date

---

Signature

# Abstract

Recommender systems are widely used in many application domains. Most of them rely on collaborative filtering or content-based filtering. These methods have the disadvantage that the user either requires a profile or that a good metric to match items is necessary. Using constraint-based recommendations, recommendations are not limited to users having a profile. However, constraint-based recommenders rely on the work of knowledge engineers. This leads to a knowledge acquisition bottleneck for large recommenders. We propose to use Human Computation to enable regular users to supplement the work of the knowledge engineers. One of the biggest problems with this crowdsourcing approach is the quality of the collected data.

In this thesis, we developed several algorithms and techniques to ensure that data collected from users meets a certain level of quality. They cover a broad range of methods to ensure the quality, from unintended inputs to active attacks on the recommender knowledge base. Furthermore, we developed an algorithm to find a minimum set of tasks that users have to solve to improve the quality of the knowledge base.

To test our techniques, we built the first general purpose, constraint-based recommender framework. It is a platform-independent, modular framework that can be used for both research and in productive use. Moreover, we implemented an HTML5 based user interface. Using this system, we conducted a large-scale study to test our algorithms. The results show that the algorithms perform well and that it is feasible to build a high-quality knowledge base on the basis of Human Computation.

# Kurzfassung

Recommender-Systeme sind in vielen Anwendungsbereichen im Einsatz. Die meisten von ihnen setzen auf kollaboratives Filtern oder inhaltsbasierte Filterung. Diese Verfahren haben den Nachteil, dass der Benutzer entweder ein Profil benötigt, oder, dass eine Metrik erforderlich ist, welche die Ähnlichkeit von Entitäten bestimmen kann. Mit constraint-basierten Empfehlungen gibt es diese Einschränkungen nicht. Constraint-basierte Recommender benötigen jedoch die Arbeit von Wissensingenieuren. Dies führt zu einem Wissensengpass für große Recommender, da Wissensingenieure nur begrenzt verfügbar sind. Unser Ansatz ist die Verwendung regulärer Nutzer als Wissenslieferanten. Dies wird auch als Human Computation bezeichnet. Das größte Problem hierbei ist, die Qualität der gesammelten Daten sicherzustellen.

In dieser Masterarbeit entwickelten wir verschiedene Algorithmen und Techniken, um sicherzustellen, dass die von den Benutzern gesammelten Daten ein gewisses Maß an Qualität erfüllen. Die Algorithmen decken ein breites Spektrum an Methoden ab, die sowohl unbeabsichtigte Eingaben als auch aktive Angriffe auf die Wissensbasis erkennen und verhindern können. Darüber hinaus entwarfen wir einen Algorithmus, um die kleinste Menge an Aufgaben für die Anwender zu finden, die bereits zu einer Verbesserung der Wissensbasis beiträgt.

Um unsere Techniken zu testen, konstruierten wir das erste allgemeine, constraint-basierte Recommender-Framework. Es ist ein plattformunabhängiges, modulares Framework, das sowohl für die Forschung als auch im produktiven Einsatz verwendet werden kann. Darüber hinaus entwarfen wir eine HTML5-basierte Benutzeroberfläche. Mit Hilfe dieses Systems führten wir eine großangelegte Studie durch, um die Algorithmen und Methoden zu testen. Die Ergebnisse zeigen, dass unser Ansatz sehr erfolgreich ist, und dass es möglich ist, eine qualitativ hochwertige Wissensbasis mit Hilfe von Human Computation aufzubauen.

# Contents

<b>Abstract</b>	<b>i</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation	1
1.2 Related Work	2
<b>2 The PeopleViews Architecture</b>	<b>4</b>
2.1 Backend	4
2.2 Frontend	5
2.2.1 Communication	6
2.2.2 Mobile Optimization	6
2.3 API	7
2.4 Data Structure	7
2.4.1 Recommender	8
2.4.2 Product Attribute	8
2.4.3 User Attribute	10
2.4.4 Support Values	11
2.4.5 Item	11
2.4.6 Evaluation	13
2.5 Recommendation approach	14
<b>3 The User Interface</b>	<b>16</b>
3.1 Recommender	16
3.2 Items	20
3.3 Recommendations	23
3.4 Search	26
3.5 Knowledge Acquisition	27
3.5.1 Micro Tasks	28
3.5.2 Game	32
3.6 iOS Client	33
<b>4 Quality Assurance</b>	<b>37</b>
4.1 User Management	37
4.1.1 Human Score	38
4.1.2 CAPTCHAs	39
4.1.3 Timing Models	40
4.2 Manipulation Detection	41
4.2.1 Login Behaviour	42
4.2.2 Browsing Behaviour	42
4.3 Data Quality	43
4.3.1 Spam	43
4.3.2 Data Collection	44
<b>5 Evaluation</b>	<b>46</b>
5.1 Study Design	46
5.2 Collected Data	50

5.2.1	Data Quality . . . . .	50
5.2.2	Data Amount . . . . .	51
5.2.3	Worker Limitation . . . . .	51
5.3	Results . . . . .	53
5.3.1	Timing Models . . . . .	53
5.3.2	User Interface . . . . .	53
5.3.3	Prediction Quality Improvement . . . . .	54
5.3.4	Limitations . . . . .	55
<b>6</b>	<b>Conclusion and Future Work</b>	<b>57</b>
	<b>Bibliography</b>	<b>58</b>
<b>A</b>	<b>API</b>	<b>64</b>
A.1	Register / Login . . . . .	64
A.2	Items . . . . .	67
A.3	Recommender . . . . .	69
A.4	User . . . . .	70
A.5	Microtasks . . . . .	71
A.6	Evaluations . . . . .	72
A.7	Data Structure . . . . .	73

# List of Figures

2.1	The communication between the client and the server is accomplished using <i>messages</i> that are handled by a <i>message hub</i> on each side. Every module can register a handler for a specific message type which receives the corresponding messages from the message hub. . . . .	7
2.2	Overview of the data structure . . . . .	8
2.3	Recommender . . . . .	9
2.4	Product attributes . . . . .	9
2.5	User attributes . . . . .	11
2.6	Item . . . . .	13
3.1	The interface to add a new recommender. The user specifies name, image, description and text. Additionally, the user adds item and user attributes. . . . .	17
3.2	Item attributes of a sample recommender . . . . .	18
3.3	User attributes of a sample recommender . . . . .	19
3.4	When adding a new item, the user specifies its name, image, description, tags and link. Moreover, the user specifies all the item attributes that are defined in the corresponding recommender. . . . .	20
3.5	The user can view a list of the items she has created. In this list, it is possible to edit and evaluate the items. . . . .	21
3.6	The details view of an item shows all the information gathered about an item. Additionally, similar items are shown. . . . .	22
3.7	Recommendation screen . . . . .	24
3.8	The explanation of a recommended item shows whether the item is the best (“Expensive”) or the worst (“Hiking”) among all items regarding this requirement. . . . .	24
3.9	Comparison of two items . . . . .	25
3.10	Search results for the term “metropolis” . . . . .	26
3.11	Evaluation of an item . . . . .	28
3.12	Microtask types 1 to 6. . . . .	31
3.13	A question from the game that is shown to the playing user. After selecting the answer, the user can advance to the next question. The displayed question is from the City recommender. . . . .	32
3.14	Summary of a game . . . . .	33
3.15	A recommendation session in the iOS client. . . . .	34
3.16	Comparison and item details in the iOS client . . . . .	35
3.17	Evaluation and a sample micro task in the iOS client . . . . .	35
3.18	A game session in the iOS client . . . . .	36
4.1	An implicit CAPTCHA (microtask type 5). The has to decide which item (“EOS 7D” or “Paris”) belongs to a specific recommender (“City”). . . . .	40
4.2	The time it took the user to answer a microtask, broken down into the single microtask types. . . . .	41
4.3	A user is notified if there was a login from a different city than usual. . . . .	42
4.4	Heat map for the recommendation screen. . . . .	43
4.5	An item was unpublished due to the majority vote, i.e., 75% of the users stated that the item is spam. . . . .	44
5.1	Study welcome screen . . . . .	47
5.3	The list of tasks for the study . . . . .	47
5.2	List of all items to evaluate . . . . .	48
5.4	The regional distribution of the workers. . . . .	50

5.5	The survey for worker selection. Four out of five questions have to be answered correctly to be forwarded to the study. . . . .	52
5.6	The time it took the user to answer a micro task, broken down into the single micro task types.	53
5.7	Used browsers. . . . .	54
5.8	The position of items in recommendation list and the probability that they were chosen. . .	55
5.9	The fewer items were considered by the user (“top n items considered”), the higher the improvement by the quality assurance. In other words, the quality assurance ensured that the best fitting items are ranked higher. If only the first 3 items are considered by the user, the probability increases by more than 20% (blue line) that the best matching item is among those 3 items. Without the ground truth mechanisms, the improvement was only around 15% (black line). . . . .	56



# List of Tables

2.1	Product attributes of a sample city recommender . . . . .	10
2.3	User attributes of the sample city recommender . . . . .	12
2.4	Three sample cities with one item attribute and one user attribute including its support values. . . . .	15
3.1	Micro-task types supported in PEOPLEVIEWS . . . . .	30
4.1	Human Score Example . . . . .	39
4.2	All user interactions that trigger microtask generation with its corresponding microtasks. . . . .	44
4.3	This is an example of four cycles for one type of microtask for a specific recommender. The algorithm always starts with the current number of microtasks that were added to the agenda. If the data is good, i.e., there is no uniform distribution and there are no missing answers, after all microtasks have expired or were answered, this number might have been too high and gets decreased (Cycle 2 and Cycle 3). Otherwise, the number is increased (Cycle 1 and Cycle 4). . . . .	45
5.2	Item attributes of the Canon DSLR recommender . . . . .	49
5.4	User attributes of the Canon DSLR recommender . . . . .	49
5.6	Average timing for micro task completion. The values $\mu$ and $\sigma$ are the parameters for the approximated lognormal distribution. . . . .	54

# Chapter 1

## Introduction

### 1.1 Motivation

Combining human computation [67] with recommenders is a new approach [19, 43, 64]. Letting users collect the knowledge of a recommender has the potential to create enormous knowledge bases. Especially for constraint-based recommenders that rely on human-generated constraint sets [16], this can reduce the workload of knowledge engineers dramatically [18]. We already presented the basic idea for this in RecTurk [18] and refined it in PEOPLEVIEWS [19]. We introduced micro tasks that are solved by the users. The answers from these micro tasks are used to build the knowledge base of the recommender. What we did not take into account was the quality assurance of the data. Is the data quality of random users good enough to build a knowledge base from this? How to handle abuse and manipulation of these ideas? To research those problems, we use a full-fledged recommender system and describe as well as evaluate several techniques for quality assurance.

Although recommender systems are very common, it is hard to find a recommender framework that is open source, extensible and easy to use. We could not use closed-source projects, as the quality assurance has to be applied to several parts of a recommender which requires being able to change the source code. For our project, it was essential to have a platform that allows to collect knowledge from a crowd of users, implement algorithms for quality assurance, and then compare whether the recommendations were improved. Furthermore, we also required real users to complete the study. Therefore, the platform had to be device-independent and user-friendly.

During our research, we looked at many projects. First, we looked at recommendation libraries. HAPIGER<sup>1</sup> provides an easy to use recommendation engine using the *Good Enough Recommendations (GER)*<sup>2</sup> that uses collaborative filtering. It is a very simple recommendation library without a user interface. Another comparable project is PREDITOR<sup>3</sup> which can calculate recommendations based on the Jaccard similarity coefficient, a statistical measurement to measure the similarity between two sets [35]. Both systems were quite widespread but provided only basic functionality. LIBREC<sup>4</sup> is another recommendation library written in Java. LIBREC provides more than twenty recommendation and ranking algorithms that are state-of-the-art although there are no constraint-based recommendation algorithms.

Second, we also investigated full recommendation frameworks that are still under active development. DATO<sup>5</sup> is a company that provides predictive services. As their customers include Pandora<sup>6</sup> and StumbleUpon<sup>7</sup>, the software would certainly be a great framework for implementing our quality assurance approaches,

---

<sup>1</sup><http://www.hapiger.com/>

<sup>2</sup><https://www.npmjs.com/package/ger>

<sup>3</sup><https://github.com/Pathgather/predictor>

<sup>4</sup><http://www.librec.net/>

<sup>5</sup><https://dato.com/products/predictive-services/>

<sup>6</sup><http://www.pandora.com/>

<sup>7</sup><http://www.stumbleupon.com/>

but only a part of it is open source. A research-only framework is provided by RECOMMENDERLAB [28]<sup>8</sup>. However, this framework is specialized on binary data, i.e. ratings can only be *true* or *false*. LENSKIT [14]<sup>9</sup> is a research project by Texas State University and GroupLens Research at the University of Minnesota. It provides a framework for researchers and a toolkit to build own recommenders. This project came as close as possible to our vision. Still, it does not provide any means of quality assurance and it does not support constraint-based recommendations.

As we could not use any of the existing libraries, we developed our own framework, the PEOPLEVIEWS framework. The PEOPLEVIEWS system is a general purpose recommender framework. Its goal is to provide a platform that is suitable for different use cases. We want to provide a system, which companies and individuals can easily integrate into their products or which can be run by a community. Furthermore, with the PEOPLEVIEWS platform, we also provide an environment for research and education purposes. On the one hand, researchers can experiment with novel approaches for recommendations and quality assurance without having to deal with user interfaces and usability. On the other hand, psychologists and user interface designers can use the platform to study user behavior and to design new user interfaces.

## 1.2 Related Work

Traditional recommendation approaches that are wide-spread include content-based filtering [56] and collaborative filtering [40]. In *content-based filtering*, items are recommended based on the user profile. The user profile is built from items which the user liked and viewed previously. Items are described by features that can be compared. Most of the time, these features are keywords extracted from an item's description. A recommendation of a content-based recommender is then based on the similarity of an item's features with the features saved in the user profile [59]. For example, if a user liked several action movies in the past, the system will learn to recommend action movies [48].

*Collaborative filtering* gives recommendations based on the similarity between user profiles. To calculate a recommendation, the recommender considers items that other users with similar preferences liked. The similarity between user-profiles is based on their rating history [59]. As only user-profiles are compared, this recommendation approach is also known as "people-to-people correlation" [61]. For example, user *A* likes mainly *action* movies, but also some movies from the *drama* genre. Although user *B* liked only *action* movies so far, he will also get recommendations for *dramas* once in a while, as his user-profile is similar to the user-profile of user *A*.

These methods are well-suited for certain types of products, where the quality can be evaluated by simple ratings, such as books or movies [59]. The disadvantage of these methods is that users always require a user-profile and are not able to define certain constraints to get useful recommendations. Our aim was to develop a general purpose recommender system. Every user should be able to use a recommender in our system without needing to register or creating a profile first. These were some of the reasons we could not use one of these methods. Moreover, these methods do not support constraints specified by the user.

Instead, we use a constraint-based recommendation approach [16]. Constraint-based recommenders are based on knowledge. Using a constraint-based recommender, we do not depend on user profiles or item similarities. Constraint-based recommendation systems collect requirements from the user and rely on a knowledge base with explicit rules [59]. These rules are used to decide how well items match the user's requirements.

However, constraint-based recommendations suffer from the *knowledge acquisition bottleneck*. For every new product, the knowledge base has to be extended. In a predecessor of the system, we presented the idea of using Human Computation to build the knowledge base [18]. We also demonstrated how the constraint-based recommendation approach works based on Human Computation [19]. The main idea is to ask users to rate how well certain requirements are fulfilled by an item. The user's answers are aggregated and used as rules (constraints) for the knowledge base.

Sinha et al. [62] showed that users prefer to get recommendations from friends rather than from online recommender systems. As humans are still better at completing certain tasks, von Ahn [66] proposed to

---

<sup>8</sup><http://lyle.smu.edu/IDA/recommenderlab/>

<sup>9</sup><http://lenskit.org/>

use those human skills in the form of human computation. Krishnan et al. [43] demonstrated that human computation can help to improve the results of online recommender systems.

Quality assurance for data collected through crowdsourcing has been identified as a serious challenge [78]. Users and their contributions have to be evaluated before the data is integrated into the system. To accomplish this, techniques to block, detect and deter malicious users have to be installed [13]. Even if such techniques are used, it still occurs that malicious data gets into the system. In these cases, the system has to be able to undo these contributions [13].

In this thesis, we research how we can ensure the quality of the collected data to get better recommendation results. We have to prevent manipulation attacks such as shilling attacks and generic manipulations. Shilling attacks are attacks aimed at product recommenders. Users try to influence the recommender to drive their sales. They enter ratings such that the system will recommend their products more often than products of their competitors [45]. Various attack strategies exist to *push* or *nuke* products [52]. The basic idea is always to have the own product recommended more often by either pushing the own product or nuking competing products. These techniques focus mainly on automated attacks. However, there are also attacks carried out by ordinary users in a non-automated fashion. Users try to convince other users to manipulate the recommender in favor of their products [46].

To prevent manipulation, we present a novel kind of CAPTCHAs based on the ideas of Baird et al. [2] and Google [24]. They demonstrate implicit CAPTCHAs that are not easily detectable as such. These CAPTCHAs try not to interfere with the user's workflow. In the best case, a user is not able to recognize that his current action is an automated test to detect whether the request is generated by a bot. Additionally, Google used data about the user's behavior to generate simpler tests while still protecting a site from automated requests.

Various techniques for quality assurance with different costs have been proposed [34]. They can be split into two groups, *run-time quality assurance* and *design-time quality assurance*. Gadiraju et al. [22] analyzed the behavior of users in crowdsourcing domains and reasoned, that it is necessary to understand the user's behavior. They classified malicious workers on crowdsourcing platforms. From the behavioral pattern of crowd workers, they derived a guideline for designing crowdsourced surveys with limited malicious activity. Therefore, we introduce the concept of timing models based on the ideas of Zaidan et al. [75]. With timing models, we can describe task durations using models and use the models to classify the users.

Furthermore, we use well-known algorithms from the field of credit card and mobile phone fraud detection [7] to find anomalies in the user's behavior. We combine them with distributed denial of service (DDoS) and bot detection algorithms [51]. Oikonomou et al. [51] demonstrate how to distinguish bots from human by analyzing the user's behavior. These algorithms are used to detect large-scale automated requests and spam. Chung et al. [8] showed that a Beta-distribution could also improve the detection of malicious profiles. By modeling certain parameters using the Beta-distribution, it is possible to get an additional confidence for the profile classification.

We developed a new algorithm to calculate the required knowledge acquisition tasks based loosely on the working set algorithm [10]. The tasks are distributed to the users to collect required knowledge. Our scheduling algorithm tries to minimize the number of tasks that have to be scheduled by continuously evaluating the resulting quality.

The problem of finding the best matching users for this task is not scope of this master's thesis. However, ideas for solving this problem are given by Khazankin et al. [39] and Aharon et al. [1]. Instead of filtering the data after collection, selecting appropriate workers for the tasks is a different way of quality assurance. The selection can be based on a certain skill-set of the worker [39] or on the user's interaction history [1]. Jung et al. [37] present algorithms for worker selection in crowdsourcing environments. They argue that data quality can benefit greatly from a careful worker selection. To verify the quality of such a worker selection, the collected data is can be verified to a ground truth [57]. The ground truth describes data that is objectively collected by experts.

## Chapter 2

# The PeopleViews Architecture

As there was no suitable base for our development, we designed the PEOPLEVIEWS framework from scratch. We aimed for a scalable, extendable framework that is easy to use but provides a lot of functionality.

The system runs as a standalone web application that can be used on any platform supporting Java. The frontend runs in a web browser. Therefore, there is no platform dependency.

We divided the framework into two main parts, the frontend and the backend. Both parts are organized as modules that have only minimal dependencies to other modules. This organization makes it flexible and easily extendable.

The design goals of the framework were

### **Platform independency**

The frontend for the user has to be fully platform independent. Moreover, the backend should run on at least Microsoft Windows and Linux.

### **Configurability**

It should be possible to configure as many parameters as possible, for example, the used recommendation approach or the way results are shown to the user.

### **Scalability**

The system has to work as single user system for research and testing as well as for a large user base in productive use. There should not be a limit on the number of users or the amount of data the framework can handle.

### **Versatility**

It must be possible to adapt the framework to a wide variety of use cases. Every component has to be either generic, configurable or exchangeable.

To allow different use cases, we separated the backend entirely from the frontend. This separation makes the front-end interchangeable as it only has to use well-defined interfaces of the backend. These interfaces are described in Section 2.3.

In this chapter, we describe the underlying architecture, APIs, and data structures. Section 2.1 provides the backend details of the system. In Section 2.2, we explain the frontend as well as the communication with the backend. The APIs are described in Section 2.3, and the data structure is described in Section 2.4.

## 2.1 Backend

The backend is the main part of the PEOPLEVIEWS framework. It is written in Java 8 using the open-source framework Spring<sup>1</sup>. Spring is a popular framework for web applications that is employed in many projects. It provides various extensions for building web applications, e.g. object-relational mapping tools. We chose Spring as it is well supported and does not depend on any platform.

---

<sup>1</sup><https://spring.io/>

As we did not want to impose any limits, such as a maximum number of users, we had to settle for a design that scales automatically. Still, the framework should also run efficiently on an average PC to be able to develop and experiment with it. Scalability is a big issue for such a complex framework. A system is scalable if adding more hardware leads to a proportional increase in performance capacity [31]. We also have to decide between *horizontal* and *vertical* scalability. When using the vertical scaling approach, we always use one single instance of the software and only upgrade the hardware to get higher performance. The horizontal scaling approach is to run multiple instances of the same application on different machines [27].

To get a scalable system, Hadlow [27] described a few basic principles to follow:

- Stateless** Do not keep a state in the instance. Every state that resides in the instance is hard to scale. A load balancer has to be aware of these states, as the state is bound to the instance and the user cannot be re-scheduled to another instance.
- Enclosed API** API calls have to be designed in a way that every action takes exactly one API call. If actions are divided into multiple API calls, the instance would have to keep track of this state.
- Idempotent** The software has to be tolerant if the same message is delivered not once, but multiple times.
- KISS - Keep It Small and Simple**  
Divide the software into many small and simply modules that have only one responsibility. The modules should ideally be decoupled.

Our framework implements a hybrid approach to scalability. It scales well horizontally, which makes it easy for testing and small scale applications. Moreover, we are also able to scale the system vertically, as the instances itself are stateless, and the state is stored in a scalable way.

In our implementation, the instance itself does not have a state. User sessions are handled externally as described in Section 2.1. All persistent data is stored in a MySQL database. For storing persistent data, we use object-relational mapping (ORM). The Spring JPA<sup>2</sup> extension allows us to define the data access objects (DAO) only and let Spring handle the structure of our database. This creates well-optimized queries and a normalized database. This is not only beneficial in the viewpoint of efficiency but also when considering security. Using these automatically generated queries makes the system safe against SQL injections.

**Session management** The only data containing a state is the current users' session. As we had the requirement to design the architecture in a scalable manner, we decided to do the session management externally. For this, we relied on Redis, an “open source (BSD licensed), in-memory data structure store”<sup>3</sup>. Redis itself is scalable and can be easily integrated into Spring. It can run on the same server as well as on any other server or as a provider in the cloud.

A user can either authenticate himself using the user account he created on the page, or through single sign-on (SSO) using his Google or Facebook account. As soon as the user is authenticated, the session is created and connected to the user through a session cookie. After 30 minutes of inactivity, the session is destroyed, i.e., the user has to log-in again.

## 2.2 Frontend

To have a cross-platform frontend, we decided to use a HTML5 and CSS3 implementation. This means that the frontend is browser based, and the user only needs a modern web browser to use the PEOPLEVIEWS system. To get a uniform design throughout all components and support a broad range of devices, we settled for the Bootstrap framework<sup>4</sup>. Bootstrap is one of the most popular front-end frameworks for responsive web projects [53].

---

<sup>2</sup><http://projects.spring.io/spring-data-jpa/>

<sup>3</sup><http://redis.io/>

<sup>4</sup><http://getbootstrap.com/>

In addition to this user-interface framework, we also used jQuery<sup>5</sup> as a cross-browser JavaScript library and jQuery UI<sup>6</sup> for cross-browser widgets. On top of those frameworks, we developed an attribute-oriented programming (AOP) framework in JavaScript to hide the client-server communication from the developer. AOP allows the user to use attributes for describing the behavior of the program [55, 70]. The website is completely asynchronous, i.e., the user can interact with it like a normal native application without noticing that it runs in the browser.

To make the system as flexible as possible, the framework also supports grammar-aware localization by implementing the ICU MessageFormat [32]. This message format allows us to support pluralization and gender for all languages. The displayed language is automatically selected based on the users' browser language.

We tested the site extensively with Google Chrome 48, Mozilla Firefox 44, Opera 28, Apple Safari 9.1, Microsoft Edge and Microsoft Internet Explorer 11. Apart from Internet Explorer, there are no known issues with any of the tested browser. To be as backward compatible as possible, we developed numerous workarounds for Internet Explorer such that the page is also usable with this browser.

### 2.2.1 Communication

Both backend and frontend are subdivided into independent modules. These modules need a method to communicate with each other. We want the server and the client to stay decoupled. Therefore, we do not want to communicate directly between the modules. Moreover, it should be irrelevant to which instance we talk if multiple instances are running.

We use a message passing approach between the client and the server. A message consists of a *type* and *content*. The type is a string that uniquely links messages to modules. Every module can announce message types that it wants to handle. The content of the message can be arbitrary data that is encoded in JSON format.

Both server and client have a *message hub* as communication endpoint. The message hub acts as a router between the different modules. It receives all messages and redirects them to a local module or another message hub if the receiver is a remote module. Modules register their message handler at the message hub and send messages to other modules via the message hub.

This design additionally allows us to exchange modules on both the client and the server side without having to change other modules.

The structure of this communication is shown in Figure 2.1.

### 2.2.2 Mobile Optimization

The support for mobile browsers becomes more and more important. Especially now, as they have already overtaken the desktop browsers [5]. Due to their high HTML5 compliance, we can provide the same functionality as with desktop browsers [12, 21].

We decided to use the same code base for both mobile and desktop clients. To take the different screen resolutions and orientations into account, we used the media queries provided by Bootstrap. Media queries allow a responsive web design, by changing the layout and visibility of elements depending on the devices' screen size. This is necessary, as mobile devices are usually in portrait mode, whereas desktop clients use landscape mode. Moreover, we had to change the navigation, as there is not enough space to always display the navigation on small screens.

As we already built the user interface using the Bootstrap components and classes, optimizing it for mobile devices was not as hard as expected. We did not have to create an entirely new user interface, which is convenient as updates can be applied to both mobile and desktop clients simultaneously. We tested the mobile site on various devices running Android and iOS, and we were able to use the full functionality of PEOPLEVIEWS without any limitations.

---

<sup>5</sup><https://jquery.com/>

<sup>6</sup><https://jqueryui.com/>

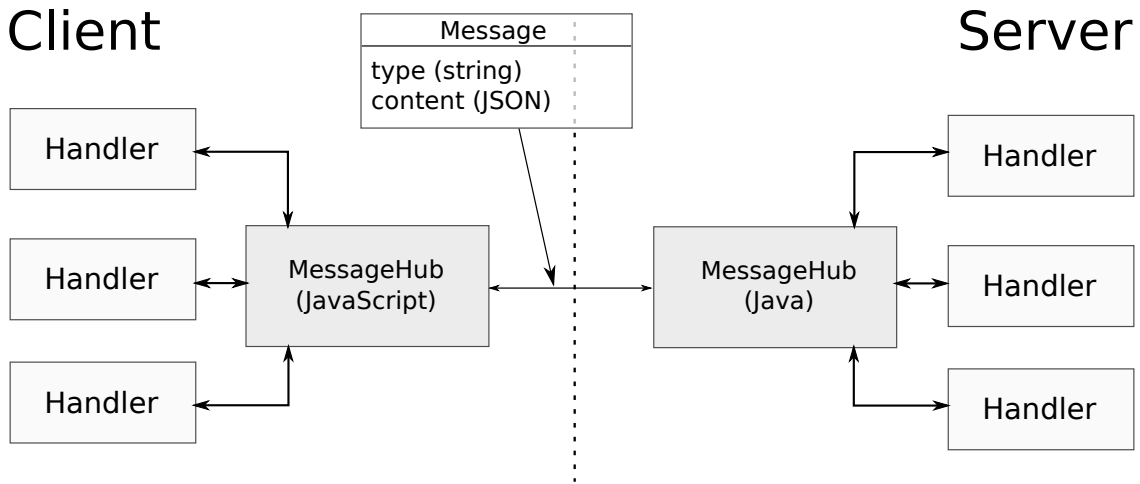


Figure 2.1: The communication between the client and the server is accomplished using *messages* that are handled by a *message hub* on each side. Every module can register a handler for a specific message type which receives the corresponding messages from the message hub.

Another frontend especially designed for mobile iOS devices was developed within a bachelor’s thesis at the Institute. We will describe this interface in Section 3.6.

## 2.3 API

As we designed the backend in a scalable way, we also had to create a stateless API. On the one hand, the API has to provide access to more or less static resources such as images. As these resources do not change often, we would like them to be cached for performance reasons. On the other hand, we also have API calls that execute certain actions such as getting recommendations. The result of such an action is volatile and must not be cached.

Therefore, we decided to split the API into a static and a dynamic part. For the static, cacheable part that is used for resources, we decided to use a RESTful API. REST stands for representational state transfer. According to Fielding et al. [20], it induces certain architectural properties such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. With REST, we can directly address resources such as image via a unique URL.

The main API for all actions is based on JSON messages. We exposed our message hub so that it can not only receive messages from the frontend message hub but also act as an API endpoint. In other words, any application can use our API by sending JSON messages to our API endpoint URL. We describe the supported API calls in Appendix A.

## 2.4 Data Structure

We were looking for a data structure that can represent all possible types of recommenders. The data structure should not be the limiting factor for recommender types, but it still has to be efficient. Multiple independent recommenders must be supported, each of them with different properties.

Figure 2.2 gives a brief overview of the data structure and the individual components. This class diagram covers only the most important parts of a recommender. It does not show the relation between recommenders and users as this is not required for explaining the data structure. In this section, we describe all the displayed components in detail. To make this section less abstract, we will also give an example using a city



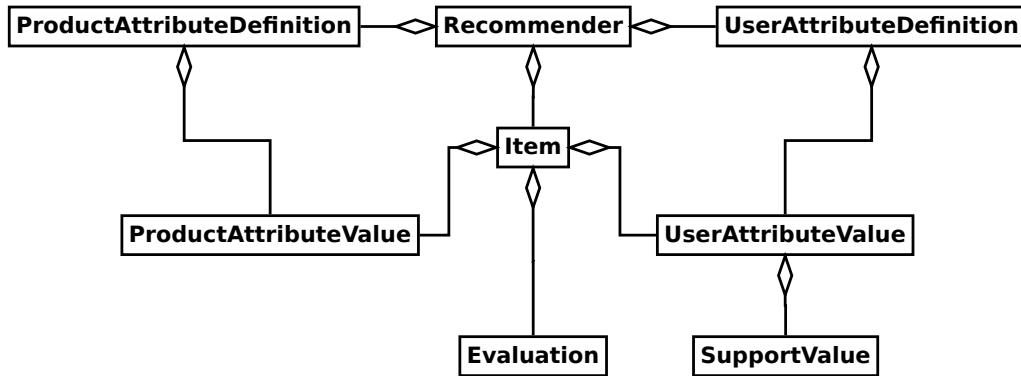


Figure 2.2: Overview of the data structure

recommender. The goal of the recommender is to provide people with a decision guidance when planning their next holiday. Therefore, we use attributes that are relevant for a vacation.

### 2.4.1 Recommender

The root of the hierarchy is always a *recommender*. This class holds the definition of a recommender. Every recommender has the following properties that describe the recommender itself.

<b>Name</b>	The name of the recommender. It should make clear what the recommender is used for, e.g. “City”.
<b>Description</b>	A few sentences that describe the recommender. Those should describe the target users and what type of items it contains. The search uses the description if a user wants to find a recommender. A description can, for example, be “Looking for a City for your next holiday?”
<b>Image</b>	Every recommender has a small thumbnail that makes it easy to distinguish this recommender from other recommenders.
<b>Tags</b>	The creator of a recommender can tag the recommender with several keywords. These keywords are used in the search. Furthermore, the similarity of recommenders is given by the number of matching tags. Tags are, for example, “Holiday”, “Vacation” or “City”.
<b>Published</b>	A recommender can be <i>published</i> or <i>unpublished</i> . Any user can use published recommenders. Owners can also see their unpublished recommenders.

In addition to these properties, the class defines the structure of the recommender. Each recommender is described using *product attributes* (Section 2.4.2) and *user attributes* (Section 2.4.3). It is also the container which holds the *items* (Section 2.4.5). Figure 2.3 shows the UML diagram of a recommender.

### 2.4.2 Product Attribute

Each product (or item) has various properties that we can divide into two categories: facts and opinions. A fact is a verifiable property of a product. For example, this can be the brand or the weight of an item.

Product attributes describe item properties that are considered a fact. Every product attribute describes one of these item’s properties. A product attribute has to identify the described property, i.e. name the property, and provide a value for the property.

We split the product attribute into two models, a *definition*, and *value*. The corresponding classes *ProductAttributeDefinition* and *ProductAttributeValue* are shown in Figure 2.4. A recommender contains only product attribute definitions. A product attribute value contains the instantiation of a product attribute definition of a specific item. Therefore, this class is not part of the recommender but part of the item.

A product attribute definition has the following attributes.

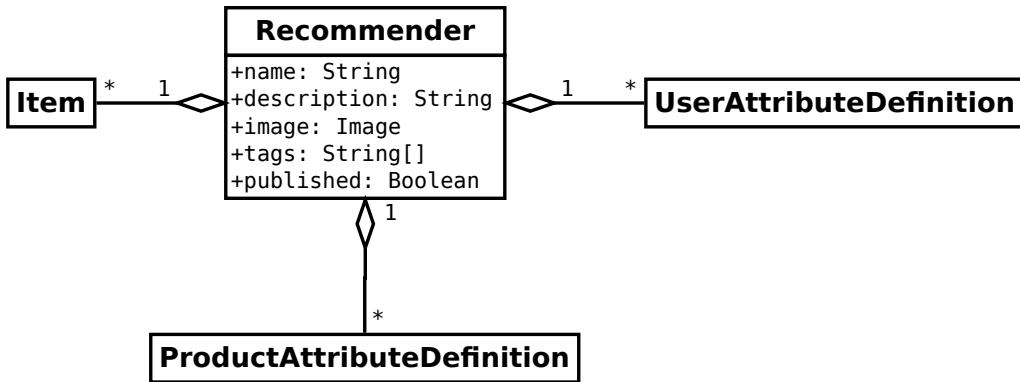


Figure 2.3: Recommender

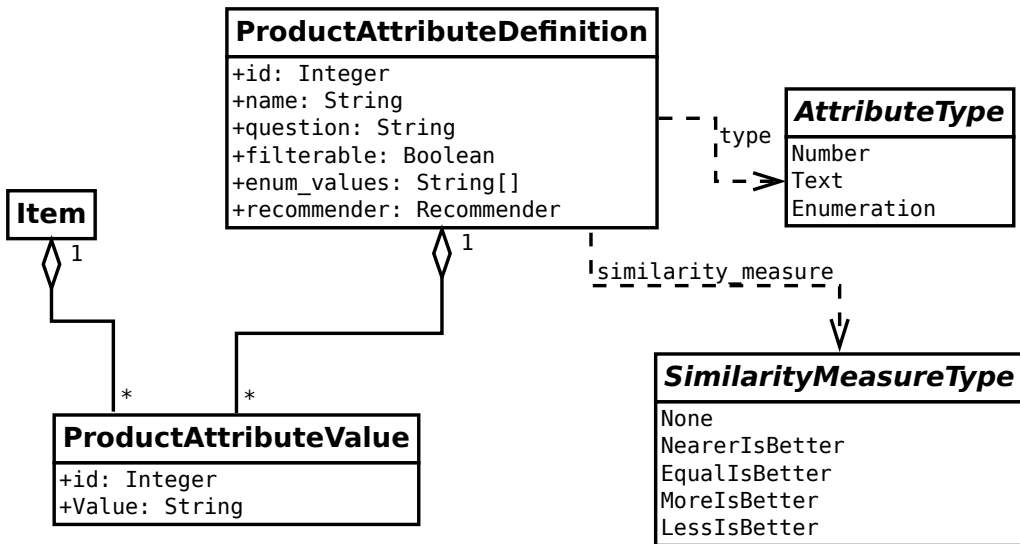


Figure 2.4: Product attributes

Name	Question	Type	Similarity Measure
Population	Population of this city?	Number	nearer is better
Country	In which country is this city located?	Text	exact match
Main Language	What is the main language spoken in this city?	Text	exact match

Table 2.1: Product attributes of a sample city recommender

<b>Name</b>	The name that identifies the property, e.g. “Country”.
<b>Question</b>	The question a user sees when using the recommender. It should be short and concise, e.g. “Country of the city?”.
<b>Filterable</b>	Decides whether the property can be used to get recommendations. If a property is not filterable, it is only shown to the user. A not filterable property would be, for example, “pronunciation”.
<b>Type</b>	Defines the data type of the product attribute value. The type can either be a number, a text or an enumeration of pre-defined values. Example product attributes for these types are “Population” (for number), “Country” (for text) or “Continent” (as enumeration).
<b>Enumeration Values</b>	If the type is an enumeration, this field contains the possible answers to the question. Taking the above example of the attribute “Continent”, possible values could be “Europe”, “America”, “Africa” and “Asia”.
<b>Similarity Measure</b>	Depending on the <i>type</i> of the product attribute, the similarity measure defines how to compare two items regarding this product attribute. We use the similarity metrics defined by McSherry et al [49].

When creating new items in a recommender, each product attribute definition is answered with a product attribute value for this item. The product attribute value is, depending on the type of the definition, either a number, a text or one of the answers defined in the enumeration values.

Table 2.1 shows the filterable product attributes of a sample city recommender.

Note that we will refer to the product attributes as item attributes in the following chapters. This is the term that the user sees in the user interface.

### 2.4.3 User Attribute

Unlike product attributes, user attributes describe opinions on item properties. An opinion is a personal viewpoint on a specific property. In contrast to facts, opinions are subjective and in general not verifiable. Usually, user attributes are the best way to describe most properties of an item.

As with the product attributes, user attributes are also split into a definition and a value model. Figure 2.5 shows the corresponding class diagram. A recommender contains only user attribute definitions. Every user attribute value contains an answer for its user attribute definition. Using *support values*, user express how well they think that the user attribute value fits the item. In fact, a support value is nothing more than a percentage of how well the answer matches the question.

A user attribute definition has the following attributes.

<b>Name</b>	The name that identifies the property, e.g. “Sights”.
-------------	---

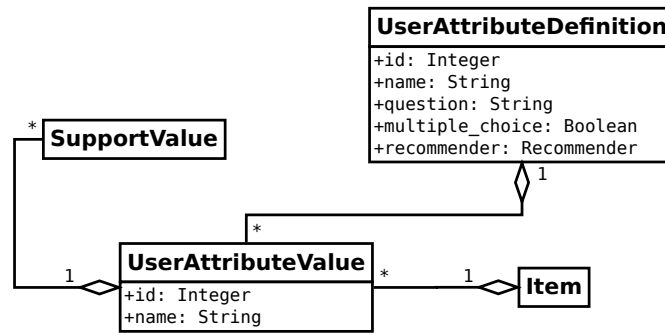


Figure 2.5: User attributes

**Question** The question a user sees when using the recommender. This should be short and concise, e.g. “What are the points of interest worth visiting in this city?”. The possible answers to this question are defined as *user attribute values*.

#### Multiple Choice

If the user attribute is defined as multiple choice, the user is allowed to select multiple answers to get recommendations. Otherwise, the answers are mutually exclusive, and the user is limited in choosing only one answer. An example for mutually exclusive answers is “Yes” and “No”.

#### User Attribute Values

A user attribute might not have an objective answer, but still it has to define possible answers from which a user can choose. User attribute values represent these possible answers (we refer to them as *domain*).

Support values for user attribute values are not assigned when creating a new item. Since their nature is subjective, it would be counter-intuitive to let the creator of the item determine them. Hence, the opinion of the crowd is collected in the form of support values. Each user has the possibility to contribute her opinion in the form of *support values*.

Table 2.3 shows the user attributes of the sample city recommender including their possible answers.

### 2.4.4 Support Values

As already stated, user attributes are subjective and cannot be assigned to a single value. Therefore, we introduce the concept of support values. A user attribute has different possible answers, also called a domain. For each answer, a user can specify how accurate the answer for this item is. This accuracy is expressed as a support value.

A support value is more or less a percentage that the user defines. The user can also change it, but every user can only provide one support value per item and user attribute answer. For a specific answer, the total support value is the aggregated support value of all users. The way of aggregation depends on the recommendation algorithm and is not in the scope of this thesis.

### 2.4.5 Item

The main part of a recommender are the items. An item represents a product, object or entity. The user utilizes a recommender to get items that match their criteria best. Each recommender provides the container for the items and defines their structure. The item structure has to be flexible so that it is possible to describe any entity that can be recommended.

We describe an item using the following general properties that apply to all items.

**Name** The name of the item. It is used everywhere where an item is displayed. An example is “Shanghai”.

Name	Question	Multiple Choice	Values
Price Level	What is the general price level in this city?	No	<ul style="list-style-type: none"> <li>• Inexpensive</li> <li>• Moderate</li> <li>• Expensive</li> <li>• Overpriced</li> </ul>
High Season	When is the best time to visit this city?	Yes	<ul style="list-style-type: none"> <li>• January-March</li> <li>• April-June</li> <li>• July-September</li> <li>• October-December</li> </ul>
Sights	What are the points of interest worth visiting in this city?	Yes	<ul style="list-style-type: none"> <li>• Historic Sites</li> <li>• Museums</li> <li>• Religious Sites</li> <li>• Sport Venues</li> <li>• Scenic Places</li> </ul>
Activities	What are the activities worth doing in this city?	Yes	<ul style="list-style-type: none"> <li>• Nightlife</li> <li>• Shopping</li> <li>• Dining</li> <li>• Hiking</li> <li>• Swimming</li> </ul>
Security	What is the level of security in this city?	No	<ul style="list-style-type: none"> <li>• Dangerous</li> <li>• Acceptable</li> <li>• Safe</li> </ul>
English Speaking Population	Who speaks English in this city?	No	<ul style="list-style-type: none"> <li>• Everyone</li> <li>• Most People</li> <li>• Only Young People</li> <li>• Nearly No One</li> </ul>
Public Transport	Which places are reachable using public transport?	No	<ul style="list-style-type: none"> <li>• None</li> <li>• Important Places/Sites</li> <li>• Whole Inner City</li> <li>• Everything</li> </ul>

Table 2.3: User attributes of the sample city recommender

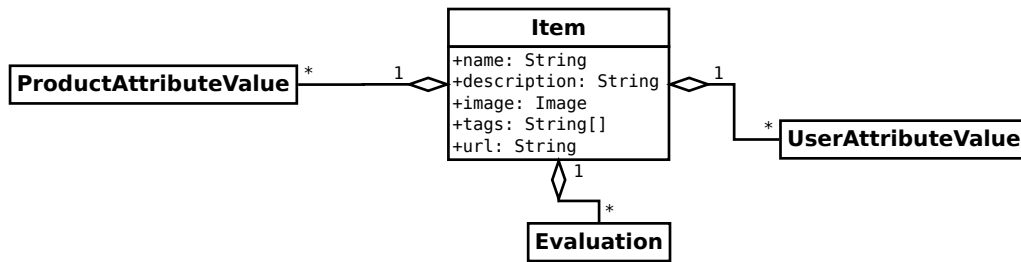


Figure 2.6: Item

<b>Description</b>	The description gives more details about the item. It should be short and concise but provide the user with additional information, e.g. “Shanghai is the largest Chinese city by population and the largest city by population in the world.”
<b>Tags</b>	An item can be tagged with multiple terms. These terms are used for the search, to show the user similar products and also to collect information about the interests of a user. Example tags are “Metropolis”, “Traffic” or “Modern”.
<b>Image</b>	The image allows users to distinguish items quickly from each other. Depending on the type of item, this can be a photograph or a symbol.
<b>URL</b>	The URL links to a page that can provide further information to an item. The URL allows extending the limitations given by the tags and description.

Moreover, each item contains the product attribute values and support values of the user attribute values unique to its recommender. Just as the general properties, the product attributes are assigned when creating the item. Only the creator of the item can change those properties. The user attributes, in contrast, are collected from the user base. Section 2.4.6 describes how users can influence the user attributes. Figure 2.6 shows the UML diagram of the item.

Not only the owner (or creator) of a recommender can create items. Items can be added by any user of the system as soon as the recommender is published.

### 2.4.6 Evaluation

Users have to provide support values for items somehow. They provide these using evaluations. An evaluation contains the support values of one user for a specific item. There is a distinction between a *full* evaluation and *partial* evaluation.

In a full evaluation, every user attribute value has a support value. It is the case if a user evaluates an item and assigns a support value to every answer. There can only be one full evaluation as every answer of an item can only be rated once by a user. A partial evaluation contains only support values for some of the user attributes. There can be multiple partial evaluations, as long as there is no full evaluation and the support values are for distinct attributes. Otherwise, evaluations get merged or updated.

Every evaluation for an item consists of one or more user attribute values, and the corresponding support values as a percentage.

## 2.5 Recommendation approach

The recommendation approach consists of two phases [19]. First, we find all *candidate items*. These are the items which match the criteria specified by the user. Second, we rank the candidate items.

We denote the requirements given the user as  $REQ$ . They are divided into requirements for user attributes,  $REQ_{ua}$ , and item attributes,  $REQ_{ia}$ . It holds that  $REQ = REQ_{ua} \cup REQ_{ia}$ . The set of all candidate items is  $RS$ . It contains all items which match the user's requirements. An item matches the requirements, if the support values for each of the user attribute and item attribute requirements are larger than zero. In other words, every item that is recommended has to fulfill all user requirements. A more formal description is shown in Equation 2.1.

$$RS = \{item \mid \forall uav \in REQ_{uav} : s(item, uav) > 0\} \quad (2.1)$$

$$\cap \{item \mid \forall ia \in REQ_{ia} : s(item, ia) > 0\}$$

**Equation 2.1:** The candidate item set  $RS$  includes all items  $item$  for which the following conditions hold: for all user attribute  $uav$  and item attributes  $ia$  specified by the user in  $REQ$ , the item must have a support  $s$  greater than 0.

The support for user attributes was already explained in Section 2.4.4. For item attributes, the support depends on the similarity metric defined by McSherry [49]. Equation 2.2 shows how the recommender algorithm uses the similarity metric.

$$\text{support}(item, ia, v) = \begin{cases} 1 & \text{if } v = \text{val}(item, ia) & \text{else } 0 & \text{EIB} \\ 1 - \frac{|v - \text{val}(item, ia)|}{\max(I, ia) - \min(I, ia)} & & & \text{NIB} \\ \frac{\text{val}(item, ia) - \min(I, ia)}{\max(I, ia) - \min(I, ia)} & & & \text{MIB} \\ \frac{\max(I, ia) - \text{val}(item, ia)}{\max(I, ia) - \min(I, ia)} & & & \text{LIB} \end{cases} \quad (2.2)$$

**Equation 2.2:** Depending on the order relation, the support of the item attribute  $ia$  for an item  $item$  can be determined using this formula. The item attribute value of the item  $item$  is called  $\text{val}$ , whereas  $v$  is the value given by the user as a constraint.  $\min(I, ia)$  and  $\max(I, ia)$  are the smallest and largest support values respectively for the item attribute  $ia$  over all items  $I$ .

For example, given the three cities in Table 2.4. The user wants to have a safe holiday in an average-sized city. As requirements, he selects the user-specific filter constraints  $REQ_{ua} = \{Safe\}$  and  $REQ_{ia} = \{2,000,000\}$ . The item attribute has *nearer is better* (NIB) as similarity measure. The support is therefore calculated using  $1 - \frac{|v - \text{val}(item, ia)|}{\max(I, ia) - \min(I, ia)}$ . Tokyo, for example, has a support value of

$$1 - \frac{|2,000,000 - 13,510,000|}{13,510,000 - 987,000} \approx 0.08$$

The support values for Mumbai and Capetown are 0.20 and 0.92 respectively. Tokyo and Mumbai are included into the candidate item set, as their support value for “Safe” and “Population” is not zero. Capetown is not included, as the support value for “Safe” is zero.

In the second phase, the items have to be ranked. The ranking is based on the *utility* function. The utility function is the sum of all support values which the user specified in her requirements. Equation 2.3 gives the formal definition.

To rank the items in the candidate item set  $RS$ , i.e. the cities Tokyo and Mumbai, we calculate their utility. We sum their support values for “Population” and “Safe”. This results in a utility of  $0.08 + 0.9 = 0.98$

City	User attribute “Security”			Item Attribute
	Dangerous	Acceptable	Safe	Population (NIB)
Tokyo	0.0	0.2	0.9	13,510,000
Mumbai	0.4	0.8	0.3	11,980,000
Capetown	0.9	0.7	0.0	987,000

Table 2.4: Three sample cities with one item attribute and one user attribute including its support values.

$$\text{utility}(item, REQ) = \sum_{uav \in REQ_{ua}} s(item, uav) + \sum_{ia \in REQ_{ia}} s(item, ia) \quad (2.3)$$

**Equation 2.3:** Calculating the utility of an item  $item$  for the given constraints  $REQ$ .  $s$  gives the support for an item-attribute pair.

for Tokyo and  $0.20 + 0.3 = 0.5$  for Mumbai. The top recommended item is therefore Tokyo, followed by Mumbai.



## Chapter 3

# The User Interface

The PEOPLEVIEWS system can be used in two different modes, the *modeling* and the *recommendation* mode. In the modeling mode, recommenders and items can be created and modified. Furthermore, knowledge is also acquired exclusively in the modeling mode. The recommendation mode, in contrast, is only there to use the recommender, i.e., let the system find the best-matching items for given constraints.

In this chapter, we will show how the user interface is designed. As an illustration, we will show all actions on the same recommender. Continuing from the last section, we keep the city recommender as our sample recommender. We will first create and then use this recommender to show all aspects of the user interface.

The UI is generic and allows all types of recommenders and items to be created. An item can also be a non-real entity, such as an algorithm or a management strategy. We chose the city recommender, as we used it for the evaluation as well. There, we required the users to have a certain knowledge of the items.

### 3.1 Recommender

Every user can create a new recommender that can be used by other users. Figure 3.1 shows the screen for building a new recommender with the values of our sample recommender. When creating a recommender, the creator has to model the recommender using the basic properties described in Section 2.4.1. Furthermore, the user can add item attributes and user attributes. Per default, the recommender stays unpublished. Only the creator can see it and add items to the recommender. This allows the creator to build a good recommender which she can also test before everyone uses it.

## Add Recommender

Recommender name

Image



Select image

Description

Tags (comma separated)

Item attributes ?

User attributes... ?

Add one or more user attributes.

**Published** (This recommender should be visible for all other users)

Figure 3.1: The interface to add a new recommender. The user specifies name, image, description and text. Additionally, the user adds item and user attributes.

Figure 3.3 shows the user interface for the item attributes. We defined three item attributes as described in Table 2.1. A user can add as many item attributes as he wants. Item attributes are the *hard facts* that are unambiguously determinable. As outlined in Section 2.4.3, each attribute has a name, type, question and similarity measure. Moreover, it can be defined whether an item attribute is usable for recommendations or if it is only displayed in the item view.

In addition to the item attributes, a user must also define user attributes. User attributes are *soft facts* that are not objectively determinable. Again, there is no limitation of how many user attributes a recommender can have. Each user attribute is described by a name, a question for the user when getting recommendations and the possible answers. The details of these properties can be found in Section 2.4.3. Figure 3.3 shows the user interface for the user attributes of our sample recommender.

Item attributes ?

Population	Number	Nearer is better	Remove
Population of this city?		<input type="checkbox"/> Usable as recommendation filter	<span>?</span>

---

Country	Text	Exact match	Remove
In which country is this city located?			

---

Main Language	Text	Exact match	Remove
What is the main language spoken in this city?			

---

+ Add attribute

Figure 3.2: The item attributes of the city sample recommender. Each attribute has a name, type, question, and similarity measure. Existing item attributes can be removed and new item attributes can be added.

User attributes... ?

Price Level

**Attribute name**

Price Level Remove

**Question to user when evaluating item**

What is the general price level in this city?

**Possible answers**

inexpensive Remove

moderate Remove

expensive Remove

overpriced Remove

+ Add answer  Allow multiple answers

- High Season
- Sights
- Activities
- Security
- English Speaking Population
- Public Transport

+ Add attribute

Figure 3.3: The user attributes of the city sample recommender. Each user attribute has a name, a question for the user when getting recommendations and the possible answers.

## 3.2 Items

Items are the essential part of the PEOPLEVIEWS system. An item represents the entity with which the recommender works. If a user uses the recommender, the items are displayed to him. An item belongs to exactly one recommender. Its structure is described in Section 2.4.5. Figure 3.4 illustrates the corresponding user interface. As it can be seen, the item attribute values have to be specified when creating the item whereas there is no possibility to enter the support values of the user attributes. The creator of an item can provide the support values through an evaluation as described in Section 3.5.


Every user can add an item to an existing published recommender. The creator of a recommender can already add items to her recommender while it is not published yet. Items of unpublished recommenders are invisible to other users until the creator of the recommender decides to publish the recommender. Items are even invisible if the item itself is published but the recommender is unpublished.

### Add item

Item name

---

Image



Select image

---

Description

Tags (comma separated)

metropolis x traffic x many people x

Link

<https://en.wikipedia.org/wiki/Shanghai>

Item attributes

Population of this city?

24000000

In which country is this city located?

People's republic of China

What is the main language spoken in this city?

Chinese

**Published** (This item should be visible for all other users. Only published items are shown in recommendations.)

Cancel

Save









Figure 3.4: When adding a new item, the user specifies its name, image, description, tags and link. Moreover, the user specifies all the item attributes that are defined in the corresponding recommender.

A user can always see the list of his created items. To make it easier for the user to find an item, the items are categorized by the recommender. Figure 3.5 shows the items we created for the sample city recommender. Here, the user can edit individual items and also evaluate them to provide initial support

values for the user attributes. The evaluation is described in detail in Section 3.5.

Figure 3.6 demonstrates the item detail view. This screen shows how an item is presented to the user. The upper part of the screen consists of the general basic item properties. These are the name, image, tags, description, and link. Afterward, the item attributes (also called *hard facts*) are displayed. On the bottom, the user gets a list of similar items. The similarity measure between those items is based on their user attributes, i.e., an item is similar to another item if its support values resemble the support values of the other item.

## My Items

Item	Action
 <p><b>Berlin</b> Berlin is the capital of Germany. With a population of 3.5 million people it is the second most populous city and the se...</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>
 <p><b>Beijing</b> Beijing is the capital of the People's Republic of China and one of the most populous cities in the world.</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>
 <p><b>Shanghai</b> Shanghai is the largest Chinese city by population and the largest city by population in the world.</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>
 <p><b>Tokyo</b> Tokyo, officially Tokyo Metropolis, is one of the 47 prefectures of Japan, and is both the capital and largest city of J...</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>
 <p><b>Sydney</b> Sydney is the state capital of New South Wales and the most populous city in Australia and Oceania. Located on Australia...</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>
 <p><b>New York City</b> The City of New York, often called New York City or simply New York, is the most populous city in the United States.[1] ...</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>
 <p><b>Mumbai</b> Mumbai (also known as Bombay, the official name until 1995) is the capital city of the Indian state of Maharashtra. Mumb...</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>
 <p><b>Moscow</b> Moscow is the capital and the largest city of Russia. It is a major political, economic, cultural, and scientific center...</p>	<input type="button" value="Edit"/> <input type="button" value="Evaluate"/>

« 1 - 8 9 - 10 »

Figure 3.5: The user can view a list of the items she has created. In this list, it is possible to edit and evaluate the items.

# Shanghai

← Back



## Tags

metropolis

traffic

many people

## Description

Shanghai is the largest Chinese city by population and the largest city by population in the world.

## Link

<https://en.wikipedia.org/wiki/Shanghai>

## Item attributes

Attribute	Value
Population	24000000
Country	People's republic of China
Main Language	Chinese

Evaluate

Edit

## Similar items

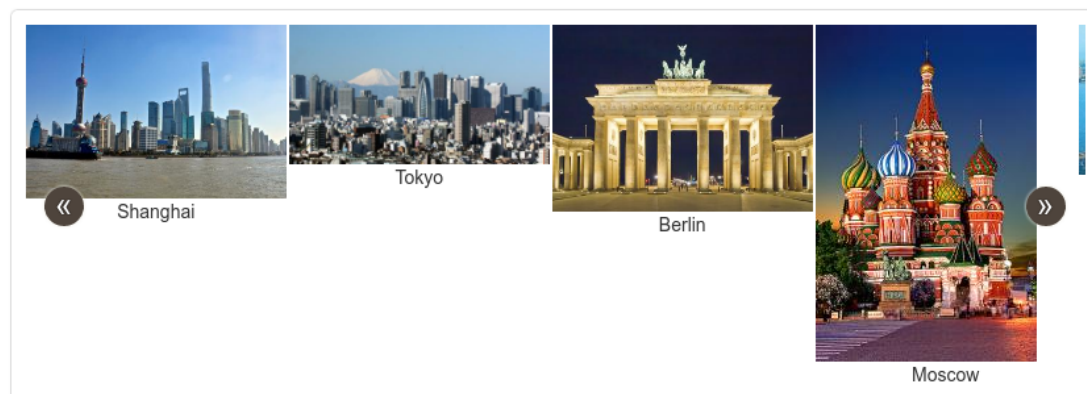


Figure 3.6: The details view of an item shows all the information gathered about an item. Additionally, similar items are shown.

### 3.3 Recommendations

The main purpose of the PEOPLEVIEWS system is to determine recommendations for users. Getting a recommendation is the only functionality in the recommendation mode. The recommendation algorithm calculates the best matching items which fulfill the user-given constraints as explained in Section 2.5. The resulting items are shown in order of relevance to the user. Moreover, the user gets a short explanation why the items are ranked in this way.

Figure 3.7 shows the user interface to get recommendations. On the left side, the user can specify her constraints or filter criteria. The constraints consist of the user attributes and the item attributes which are filterable. While the user selects as many requirements as he wants, the recommendations are updated in real time. The user can also save the selected requirements as a *filter*. Filters are displayed on the start page and allow to repeat a recommendation query. They are particularly useful if the user is interested in a particular type of items that she wants to query frequently, e.g., inexpensive, safe cities for her holiday from June to September.

To make the recommendation process more transparent to the user, we give an explanation as to why the top items are the most relevant ones, and the last items do not fit the requirements well. The idea behind this is quite simple. For each requirement, we tag the best and the worst item. This gives the user additional information that he can usually not infer from the ranking. Figure 3.8 shows an example of this explanation. In this case, the user has specified multiple requirements. Overall, the top recommended item matches the requirements best, but the item is the worst item for the requirement “Hiking”. The explanation might be a valuable help for the user to decide which item she selects.

For more details, the user can also compare two items by selecting them and pressing the “Compare Items” button. Figure 3.9 shows this screen. There, the items are shown side-by-side. For a quick overview on which item performs better, an overall score is calculated for both items. This score is a percentage. It indicates in how many attributes the item is better than the item it is compared to.

Then, a comparison of the user attributes is shown. To make the comparison quickly comprehensible, the individual user attributes are shown as a radar chart. According to Chambers et al. [6], a radar chart is a useful way to display multivariate observations with an arbitrary number of variables. Radar charts are a typical way to display performance metrics, similarities, and outliers [3, 50]. For each user attribute, we display all aggregated support values as a percentage. The percentage makes it easy to see similarities and differences in the compared items.

Finally, the item attributes are displayed. As they are hard facts, we can just display their value. Additionally, if they have a similarity metric defined, we highlight the better item according to this metric.



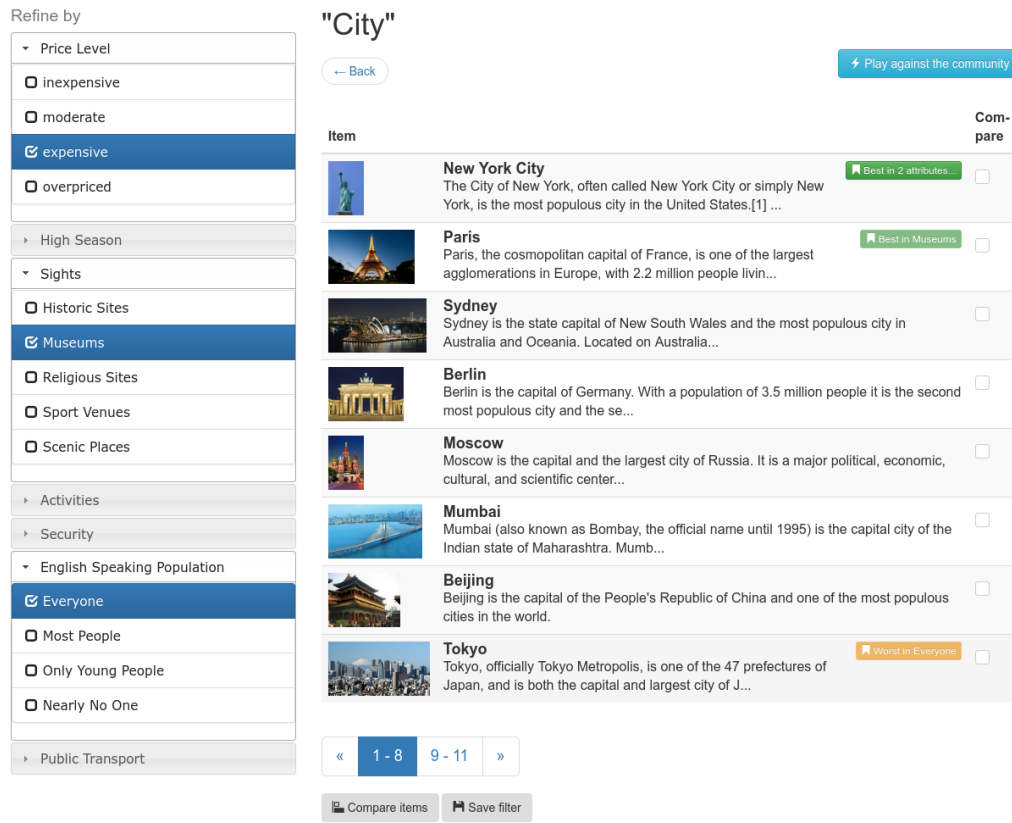


Figure 3.7: The user chooses his filter criteria on the left. On the right, the recommended items are displayed, sorted by how well they fit the user’s criteria.



Figure 3.8: The explanation of a recommended item shows whether the item is the best (“Expensive”) or the worst (“Hiking”) among all items regarding this requirement.

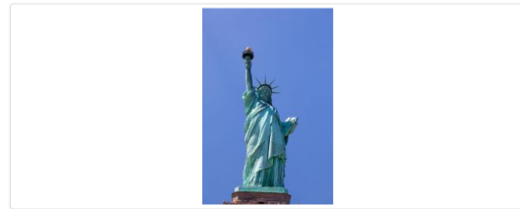
## Comparison

[← Back](#)



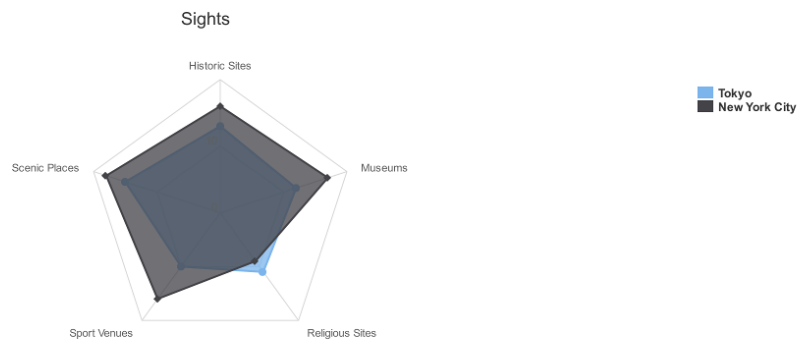
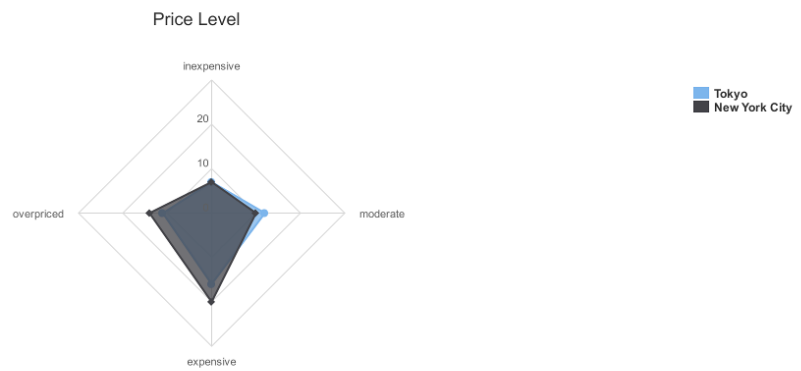
Tokyo

23



New York City

68



Population	13500000
Country	Japan
Main Language	Japanese

Population	8500000
Country	United States of America
Main Language	English

Figure 3.9: The user attributes of two items are compared using radar charts. This gives a first impression of the individual user attribute values. Item attributes are listed side-by-side on the bottom of the screen.

### 3.4 Search

Recommendations are not the only possibility to view items. The user can also query the system using a targeted search. This search functionality is available everywhere in the system. Per default, the search looks for items and recommenders but the user can change this behavior.

The search backend can find recommenders and items by doing a full-text search. It analyzes the name, description, and tags of items and recommenders. Furthermore, for items, also item attributes are indexed. The results are sorted by the number of occurrences of the search term.

The system also provides an implicit search. This search is triggered when clicking on an item's tag. This feature gives the user the possibility to find similar items quickly in different recommenders or recommenders with similar purposes.

## Search result for "metropolis"



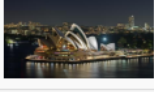


Recommenders

Included

Items

Included

---

Result	Type
 <p><b>Beijing</b> Beijing is the capital of the People's Republic of China and one of the most populous cities in the world.</p>	Item
 <p><b>Tokyo</b> Tokyo, officially Tokyo Metropolis, is one of the 47 prefectures of Japan, and is both the capital and largest city of J...</p>	Item
 <p><b>Sydney</b> Sydney is the state capital of New South Wales and the most populous city in Australia and Oceania. Located on Australia...</p>	Item
 <p><b>Shanghai</b> Shanghai is the largest Chinese city by population and the largest city by population in the world.</p>	Item
 <p><b>Mumbai</b> Mumbai (also known as Bombay, the official name until 1995) is the capital city of the Indian state of Maharashtra. Mumb...</p>	Item

« 1 - 5 »

Figure 3.10: The PEOPLEVIEWS system provides a always accessible search bar at the top of the page. Entering a search term such as “metropolis” leads to a result page as in this figure. On the search page, the user can decide whether to include only recommenders, only items or both in the search.

## 3.5 Knowledge Acquisition

Up to this point, we have seen various elements of PEOPLEVIEWS that allow managing items and recommenders as well as using the information provided by the system. In this section, we will show the user interface for knowledge acquisition. The knowledge acquisition is the most important aspect of a recommender system as otherwise the system is nothing more than a collection of data. Constraint-based recommenders require attributes which describe how well the item supports the requirements specified by the user.

A considerable problem for knowledge acquisition is the users in-existent readiness to complete time-consuming tasks [44, 54]. Consequently, we designed micro tasks which can be solved within seconds. Users are asked short questions that they can answer in a matter of seconds. Section 3.5.1 shows the different types of micro tasks we designed and their use. In addition to the micro tasks, we also provide two separate interfaces to collect knowledge from the user. One interface that is very similar to micro tasks is the game that is described in Section 3.5.2. Users have a need to get feedback (*i.e.* points) about the performance regarding their work [38]. Therefore, we introduce the use of game design elements known as gamification [11]. Games have a dual benefit. First, they keep the users on the platform for a longer period. Second, information gathered from games has on average a higher data quality than data entered by the user [41].

For both game and micro task, the user can not actively select which kind of knowledge she wants to provide. The questions there are semi-random and influenced by the user's profile. The system collects every action of a user and distributes the micro tasks according to this information. The distribution approach is loosely based on the *ExcUseMe* algorithm by Aharon et al. [1]. The exact mode of operation is not part of this master's thesis and will not be explained here.

The second interface that the system provides is the *evaluation*. Figure 3.11 shows the screen to evaluate an item.

For the evaluation, the user can actively select a specific item from any recommender. In the evaluation, all user attributes are listed with their corresponding user attribute values. The user can give a support value to each user attribute value or select “?” if she does not want to evaluate an answer. A user can evaluate every item only once. Nonetheless, it is possible to view and change the evaluation at any time.

## Evaluate »Tokyo«

### Recommender "City"



#### Description

Tokyo, officially Tokyo Metropolis, is one of the 47 prefectures of Japan, and is both the capital and largest city of Japan.

▾ What is the general price level in this city?

inexpensive	<input type="range"/>	? 0%	100%
moderate	<input type="range"/>	? 0%	100%
expensive	<input type="range"/>	? 0%	100%
overpriced	<input type="range"/>	? 0%	100%

▸ When is the best time to visit this city?

▸ What are the points of interest worth visiting in this city?

▸ What are the activities worth doing in this city?

▸ What is the level of security in this city?

▸ Who speaks English in this city?

▸ Which places are reachable using public transport?

Figure 3.11: The user can evaluate each user attribute value of a specific item using a support value. The sliders are used to select the support value between 0% and 100%. If the user does not want to give a support value for a user attribute value, she can select “?”.

### 3.5.1 Micro Tasks

As the recommender’s knowledge depends on the wisdom of the crowd, we require a way to get this data into our system. Therefore, we defined micro tasks as short tasks to be done by users. We designed these tasks such that a user can answer them within seconds. As human answers to questions are subjective and imprecise, we implemented a fuzzy rating scale [9]. It means that a user cannot only select an answer but also indicate how well the answer fits. Using fuzzy rating scales for user interactions was introduced by Hesketh et al. [30]. This method gives us two advantages, compared to letting a user only select the best matching answer. First, we get richer data as we get not only the answer but also a percentage that determines how confident the user is, that the given answer is the best one. Second, we disburden the user in deciding what the best answer is. A user can select an answer that matches not perfectly in his opinion and point out that fact.

The PEOPLEVIEWS system supports six different types of micro tasks. Table 3.1 gives an overview of the

different types including an example of each type.

**Support (Type 1)**

This micro task asks the user to specify how well an item supports a specific user attribute value. There are two use cases for this micro task. First, when a new item is created, we collect initial information about all user attribute values for the item using this micro task. Second, PEOPLEVIEWS games use only this micro task. It allows the game to build upon the existing micro task technology.

**Comparison (Type 2)**

One user attribute and two items are chosen. The user has to decide which item matches the user attribute best. Furthermore, the user can indicate how well the selected item matches the user attribute. It gives us the direct information which item is better suited for a particular user attribute choice. It creates the ranking information on an item level as discussed in Section 2.5.

**Single choice (Type 3)**

With this type, the user has to select which user attribute value is most likely the best choice for the given user attribute. Furthermore, the user is able to provide a percentage for her support, indicating how well she thinks that the given answer fits. It gives us a ranking on user attribute level, i.e., it determines which user attribute value is seen as the correct answer by the majority.

**Multiple choice (Type 4)**

This micro task is essentially a combination of Type 1 and Type 3. Instead of showing one user attribute value as in Type 3, we show all user attribute values and let the user choose the support for each user attribute value. It gives us more information than Type 1 which makes it also effective for initial data collection for new items.

**Captcha (Type 5)**

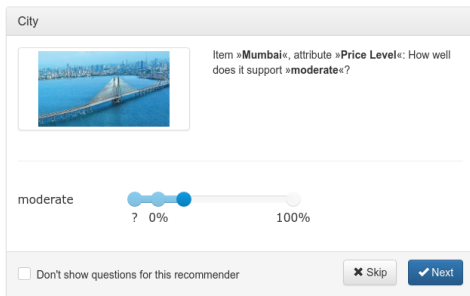
This type is used as CAPTCHA. It selects two items from different recommenders and displays their image. The user has to answer which item belongs to the first recommender. It is quite easy for users to recognize the images but hard for computers. Its purpose is not to add knowledge to the recommendation knowledge base. We use this type only for quality assurance as described in Chapter 4.

**Binary decision (Type 6)**

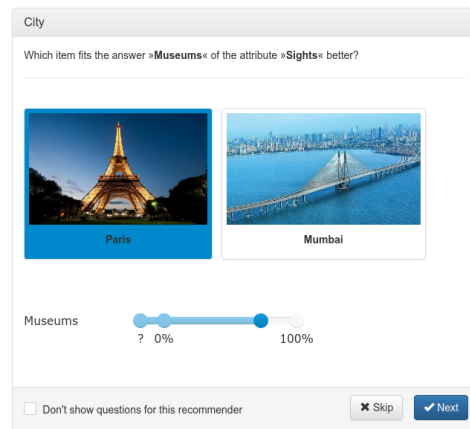
As with the Type 5, this type of micro task is used for the quality assurance as well. It is used to verify that new items are neither spam nor in the wrong recommender.

type	description	example	Figure
1	Evaluation of one user attribute with regard to one specific value	Item <i>Mumbai</i> , attribute <i>Price Level</i> : How well does it support <i>moderate</i> ?	3.12a
2	Selection and evaluation of best-performing item with regard to a specific attribute value	Which item fits the answer <i>Museums</i> of the attribute <i>Sights</i> better ( <i>Paris</i> or <i>Mumbai</i> )?	3.12b
3	Evaluation of one single-choice user attribute with regard to all possible values	Item <i>Berlin</i> : Which answer fits the attribute <i>Activities</i> best? ( <i>Nightlife</i> , <i>Shopping</i> , <i>Dining</i> , <i>Hiking</i> , <i>Swimming</i> )?	3.12c
4	Evaluation of one multiple-choice user attribute with regard to all possible values	How would you evaluate the attribute <i>high season</i> for the item <i>Sydney</i> ( <i>January-March</i> , <i>April-June</i> , <i>July-September</i> , <i>October-December</i> )?	3.12d
5	Captcha-style check	Which of the following two items belongs to the recommender “City”?	3.12e
6	Yes-No question	Does the item “Beijing” belong to the recommender “City”?	3.12f

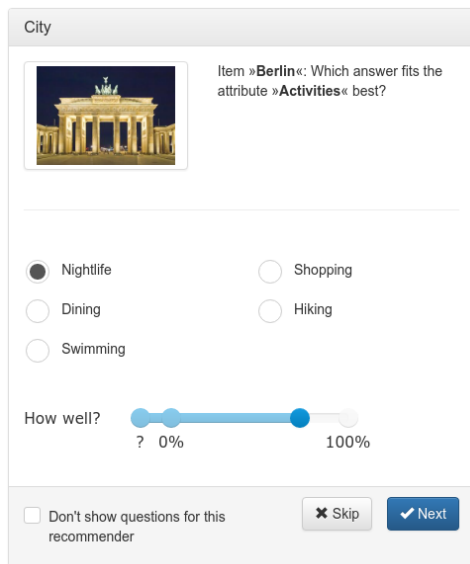
Table 3.1: Micro-task types supported in PEOPLEVIEWS .



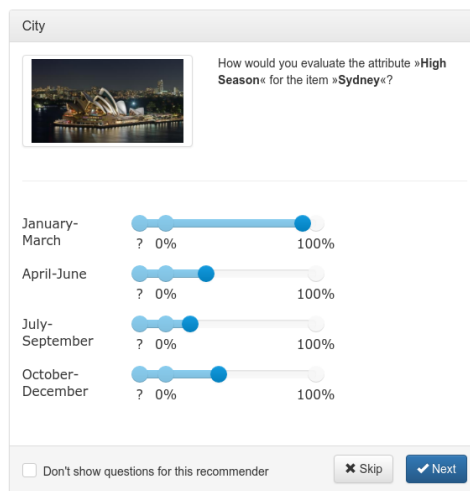
(a) MicroTask 1



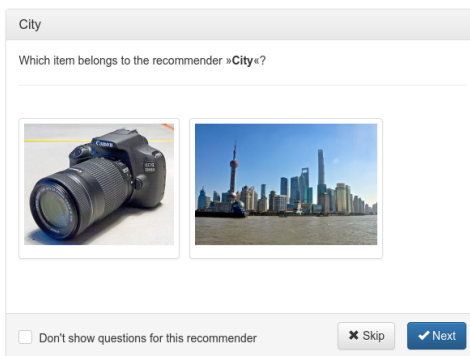
(b) MicroTask 2



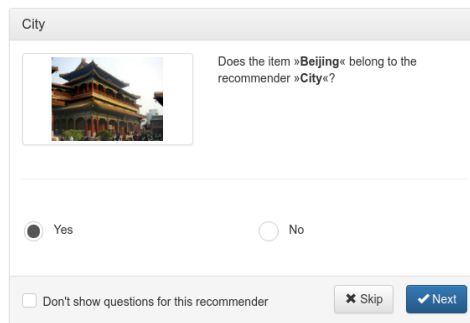
(c) MicroTask 3



(d) MicroTask 4



(e) MicroTask 5



(f) MicroTask 6

Figure 3.12: Microtask types 1 to 6.



### 3.5.2 Game

Users can also play a simple game on the platform. The game has a dual benefit. First, it keeps the users on the platform for a longer period of time. Second, information gathered from games has on average a higher data quality than data entered by the user [41]. At the recommendation screen, the user can decide to check his knowledge about the topic by playing the game.

If the user decides to play a game, ten question-answer pairs are generated for random items in the current recommender. The actual game is, in fact, trivial. The user sees the pairs one after another and decides how well the answer fits the question for the particular item. The knowledge base determines the correct answer, i.e., the truth is what the majority of the community thinks.

The game is closely related to the micro tasks. The user is asked how well the item supports one specific value of a user attribute. Figure 3.13 illustrates a sample question for a game in the “City” recommender. This question is, in fact, the same as the micro task shown in Figure 3.12a.

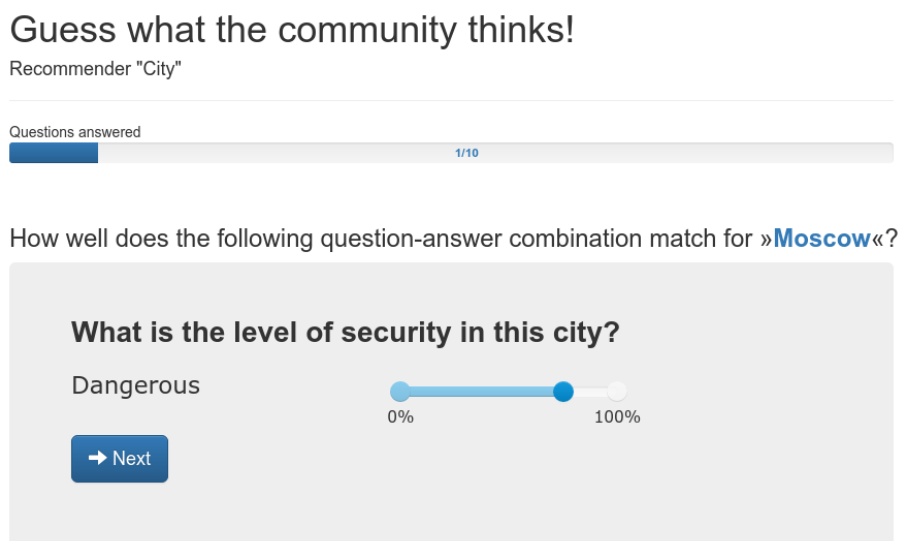


Figure 3.13: A question from the game that is shown to the playing user. After selecting the answer, the user can advance to the next question. The displayed question is from the City recommender.

After the player has answered all questions, the game displays a summary of the user’s answers and the corresponding correct answers. Figure 3.14 shows this screen. The less the user’s answer deviates from the actual answer, the more points are awarded to the user.

The game also features a high score list. The list shows the Top 10 players as well as the own rank. We designed the game in such a way that it can be finished within one minute. Players should feel tempted to play multiple times to improve their score without requiring much time.

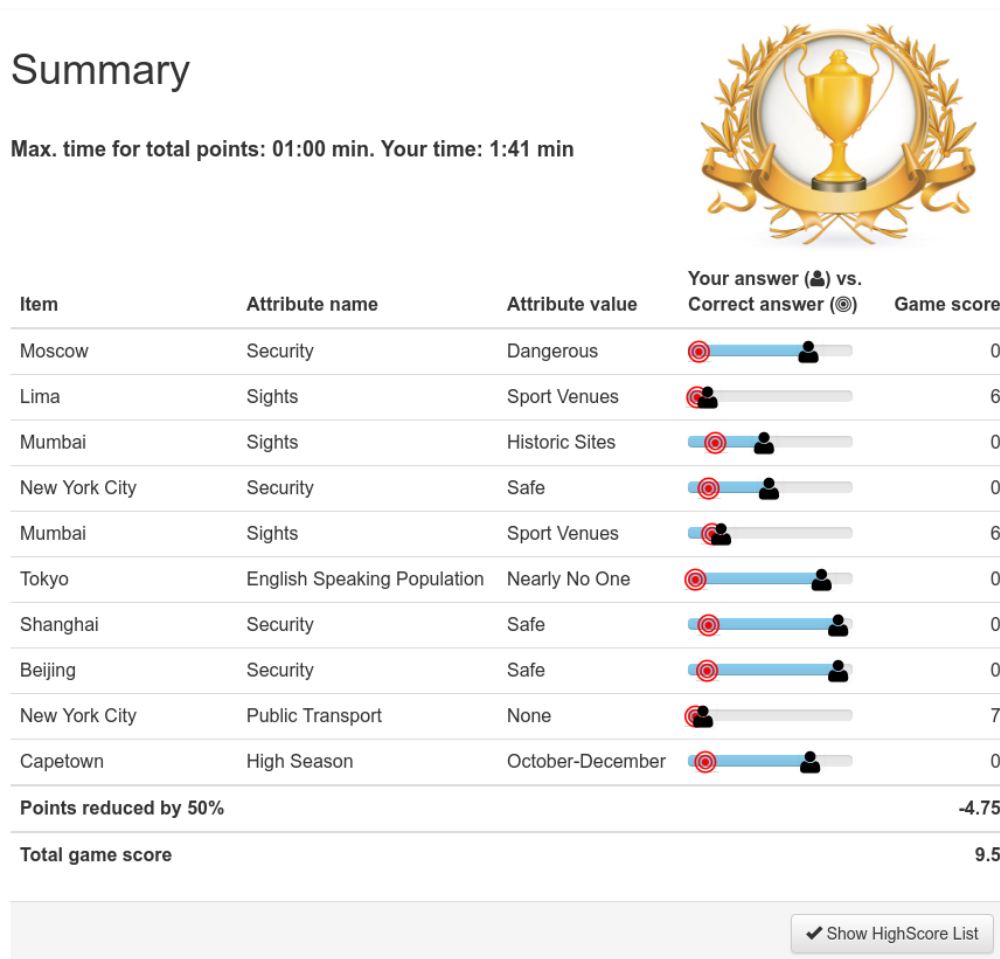


Figure 3.14: Summary of a game

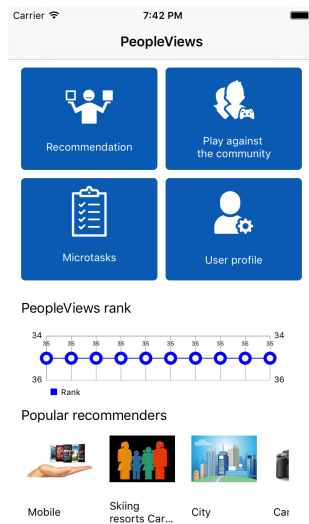
## 3.6 iOS Client

In addition to our web-based frontend, a native iOS client was developed by Angela Promitzer within a bachelor's thesis. We only provided the API documentation and had no further influence on the project. It shows that the API is in no way specific to any technology and can be used with any language.

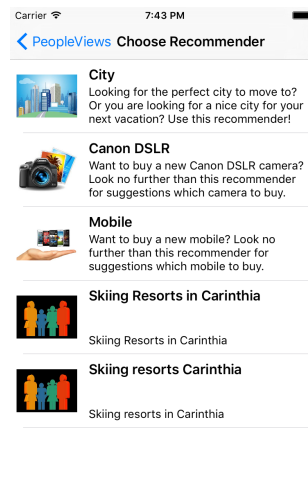
The developed iOS client is written in Swift and runs on all devices running iOS 8 or higher. It implements the recommendation mode and additionally micro tasks, evaluations, and the game.

Figure 3.15 shows the recommendation session on the iOS client. The menu is displayed in the form of a desktop when opening the app. Getting a recommendation requires only two steps: selecting the recommender and choosing the filter criteria.

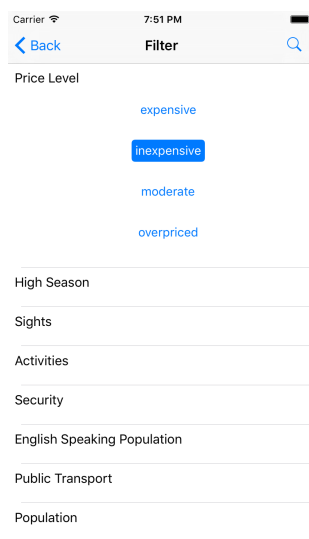
As with the web client, there is the possibility to compare and view items. Figure 3.16 shows the corresponding user interface. Figure 3.17 illustrates the application's possibility to collect knowledge. It supports all types of micro tasks as well as the evaluation of items. Furthermore, Figure 3.18 shows how the iOS client implements the game.



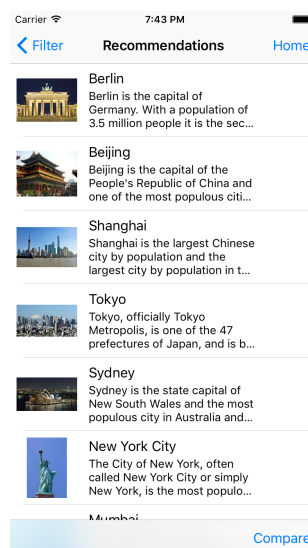
(a) Home screen



(b) Choose recommender



(c) Choose filter criteria



(d) Get recommendations

Figure 3.15: A recommendation session in the iOS client. Figure 3.15a shows the home screen when opening the application. Figure 3.15b displays a list of all available recommenders from which the user can choose. In Figure 3.15c, the user has to select his filter criteria for the recommendation. Finally, Figure 3.15d displays the items that match the filter criteria.

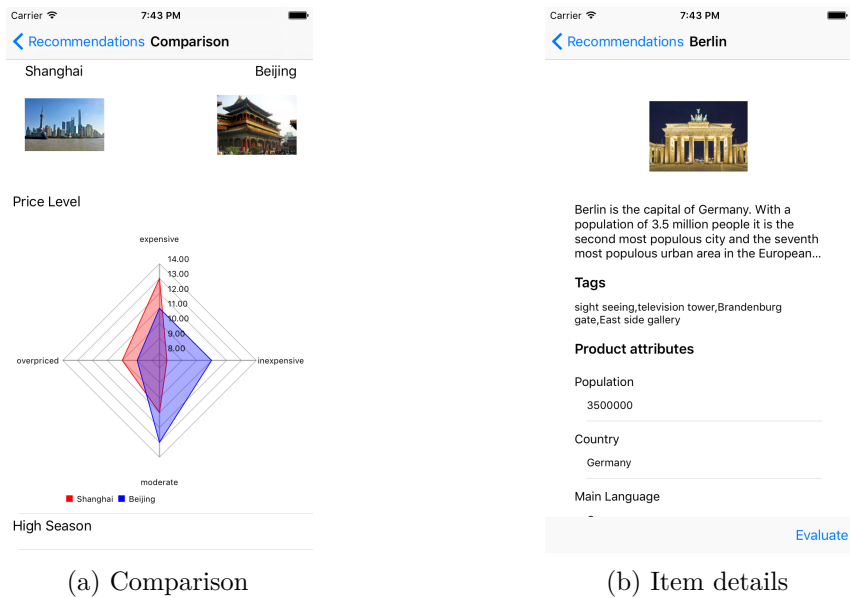


Figure 3.16: The left screen shows the comparison of two items. The advantages and disadvantages of the items are combined into a radar chart. The right screen shows the detailed item view for a specific item.

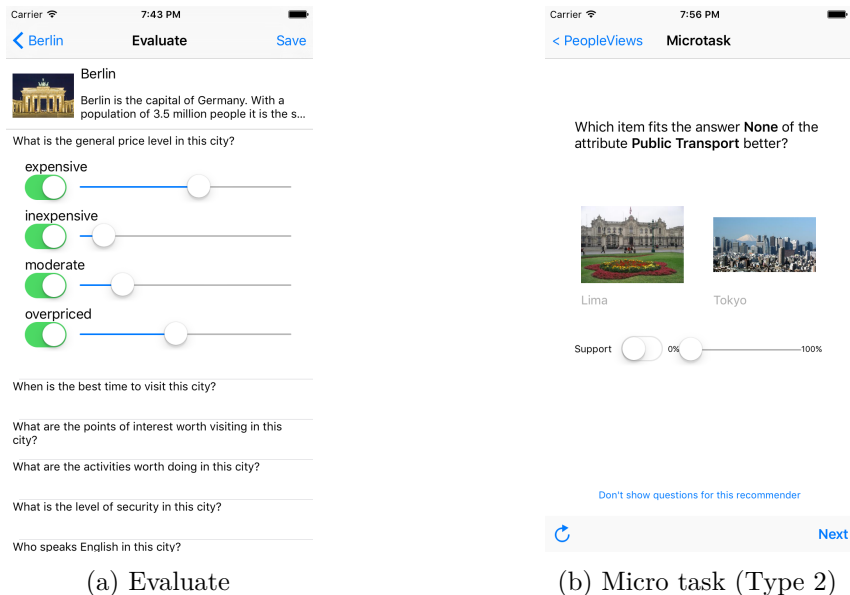
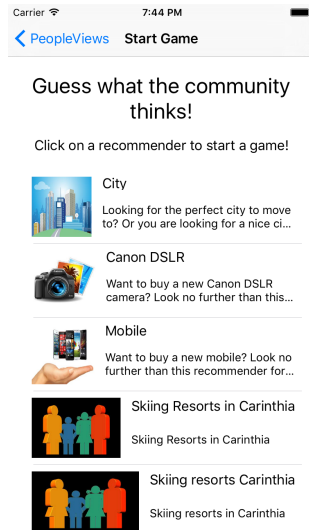
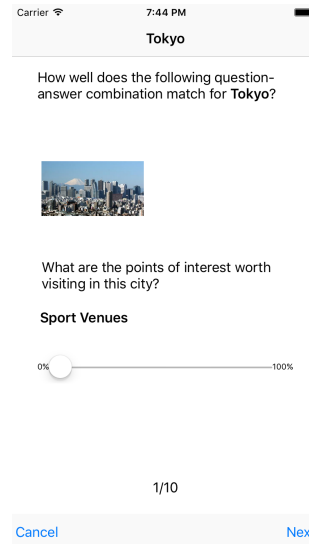


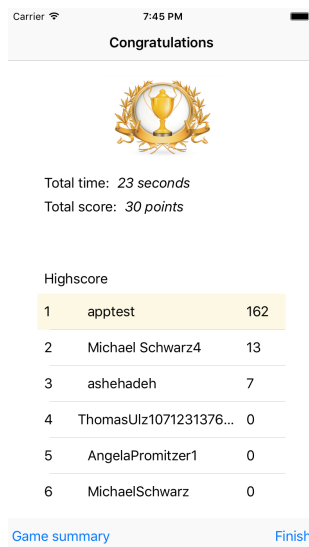
Figure 3.17: On the left, the evaluation screen is shown. Similar to the web frontend, the user has a slider to select the support value and a button to choose whether she wants to rate the corresponding user attribute value or not. The right screen shows a sample micro task.



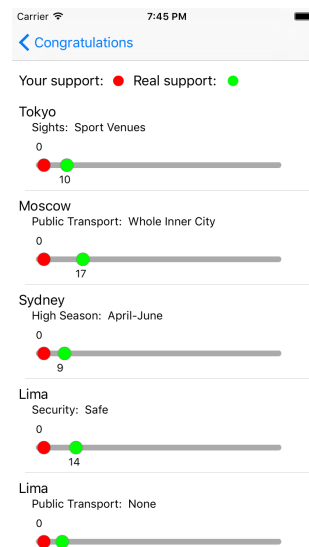
(a) Start the game



(b) A sample question of the game



(c) The high score



(d) The detailed view that shows the given (your support) as well as the correct answer (real support).

Figure 3.18: A game session in the iOS client. When starting the game, the user has to choose a recommender in which to play. Then, ten questions are asked that are similar to micro tasks. At the end, the high score is displayed and the user is able to view the detailed summary.

## Chapter 4

# Quality Assurance

One problem that occurs when collecting knowledge from the user is that we do not know anything about the quality of the data. Quality assurance for data collected through crowdsourcing has been identified as a serious challenge [13, 78]. As our platform relies solely on crowdsourcing to acquire knowledge, we have to implement ways to assess the quality of the completed tasks. As data is entered by different persons, we require methods to ensure that we use only correct data to build the knowledge base. As already described, there are many ways for users to submit data. This also means, that there are many ways for users to enter wrong data. Whether this is intended by the user or not, we have to cope with these erroneous inputs.

Not only do we have to consider wrong data that was entered by accident, through the carelessness of the user, or by a misunderstanding of the tasks. We also have to take active manipulation of malicious users into account. Here, we have to distinguish between different types of attacks, namely target and non-target attacks. A non-targeted attack will just try to be destructive, for example, spam bots that insert spam products. Targeted attacks seek to manipulate the ranking of a recommended item, either to be at a better or a worse location [45, 46].

Furthermore, another important part of quality assurance is to establish a valuable recommendation knowledge base. Therefore, the quality assurance has to decide whether new data has to be generated with the help of microtasks or existing data has to be removed. We have to detect and remove outliers permanently to increase the usefulness of the recommender. Manipulation detection is a part of the quality assurance as we will see in this chapter.

### 4.1 User Management

In PEOPLEVIEWS , there are *anonymous users* and *registered users*. Anonymous users are users who are not logged in. They are primarily using the recommender without contributing to the knowledge base. Those users do not have a profile and do not get any microtasks. They can only use the system in recommendation mode.

To use PEOPLEVIEWS in modeling mode, a user has to create an account. The system provides two possibilities for this. An anonymous user can create a dedicated PEOPLEVIEWS account in the system. If the user decides to do so, she has to enter her email address. She then receives an activation link to activate and use the account. The activation ensures that a user cannot create unlimited accounts. Furthermore, the user can reset his password using this email address. The other possibility to use the system in modeling mode is to login in via an identity provider. We support the Google and Facebook single sign-on system. It allows a user to use his existing Google or Facebook account to log into PEOPLEVIEWS .

Every registered user has a user profile in the PEOPLEVIEWS system. This profile contains the basic information about the user, such as, for example, her name, email address, and last login. Furthermore, the profile keeps track of every activity inside the system. Collecting this information is beneficial both for the user and the system. Knowing a user's interests allows us to select microtasks where the likelihood is

greater that the user will answer them. Having users answer questions where they know the answer is of course also favorable for the knowledge base.

### 4.1.1 Human Score

The human score is a measurement of trust. The idea is similar to the *reputation* used by Stack Overflow [15]. A new user is not trustworthy and should not be able to influence the knowledge base. Whereas we fully trust a long term user who already made many valuable contributions. We express this trust as a percentage that we call the human score.

Gadiraju [22] claimed that it was necessary to understand the user's behavior when crowdsourcing tasks. Therefore, the human score is updated continuously to reflect the user's behavior. Every action that has an influence on the knowledge base is automatically evaluated and integrated into the human score. In the current version, the evaluation is based on CAPTCHAs (Section 4.1.2), timing models (Section 4.1.3), login behavior (Section 4.2.1) and browsing behavior (Section 4.2.2). All these actions increase or decrease the *human score points* for the user. The exact amount is configurable for all actions. Suitable values have to be determined empirically. The actual human score of user  $u$  is calculated by summing over all human score points and normalizing the sum with the average human score points of the top users, i.e., users with the most human score points. This results in a human score between 0 and 1. Equation 4.1 shows this more formally.

$$HS(u) = \min \left( 1, \left( \sum points(u) \right) \cdot \frac{|topuser|}{\sum_{u \in topuser} points(u)} \right) \quad (4.1)$$

We use the human score to weight all data from the user. For example, the answer of a user with a human score of 1 is as valuable as two answers from users with human score of 0.5. Consequently, the answer from a user with a human score of 0 is not taken into account at all.

The weighted support value  $s_w$  of one answer  $uav$  is calculated by

$$s_w(item, uav) = \frac{\sum_{support \in uav} support(item, uav, user) \cdot HS(user)}{\sum HS(user)}. \quad (4.2)$$

Every support value is multiplied by the human score of the user that provided the support value. The sum of these weighted support values is divided by the sum of the human scores of all these users to get the weighted average.

Table 4.1 shows an example for weighting the support values of a user attribute with two user attribute values.

User	Human Score	Support			
		Answer 1	Answer 2	Answer 1 (weighted)	Answer 2 (weighted)
User 1	1	0.8	0.3	0.8	0.3
User 2	0.5	0.9	0.4	0.45	0.2
User 3	0.5	0.6	0.5	0.3	0.25
User 4	0	0.2	0.7	0	0
<i>Sum</i>	<b>2</b>	2.5	1.9	1.55	0.75
<i>Average</i>	-	$\frac{2.5}{4} = 0.625$	$\frac{1.9}{4} = 0.475$	$\frac{1.55}{2} = 0.775$	$\frac{0.75}{2} = 0.375$

Table 4.1: Four different users and their support value for two user attribute values (Answer 1 and Answer 2). Their human score is used to weight their support value. The aggregated or average support is the sum of the support values divided by the sum of the human scores. We can see that a user with human score 0 does not influence the result at all.

### 4.1.2 CAPTCHAs

To maintain a high-quality dataset, it is necessary to detect bots. A well known and broadly used concept are CAPTCHAs. Those are an automated way to tell computers and humans apart by generating and grading tests that most humans can pass, but current computer programs cannot pass [68]. The drawback of CAPTCHAs is that they require a significant effort by the person answering them [2]. This disrupts the user’s workflow and is considered as an annoyance. Google showed in a study that Artificial Intelligence technologies can solve most text-based CAPTCHAs with a success rate of 99.8% [23]. Therefore, Google introduced a new type of CAPTCHA in 2014 that analyzes the user’s behavior and uses the profile information that Google has already collected [24, 25].

Building on the ideas of Baird et al. [2] and Google [24], we implemented implicit CAPTCHAs. Implicit CAPTCHAs should not be recognizable. Therefore, they do not disrupt the user’s workflow. We designed them to look like a normal microtask. Figure 4.1 shows a sample CAPTCHA. It is composed of the same elements as a normal microtask illustrated in Figure 3.12. The hardness of this CAPTCHA comes from the Computer Vision problem of labeling images [24, 77]. For a human, it is easy to distinguish the shown images whereas a computer has to recognize the images and decide to which recommender they belong. This is considered a hard problem that cannot be solved currently.

As the user does not get feedback whether she submitted a correct or a wrong answer, she will not know that the microtask was a CAPTCHA. The result is only recorded in her profile and influences the human score.



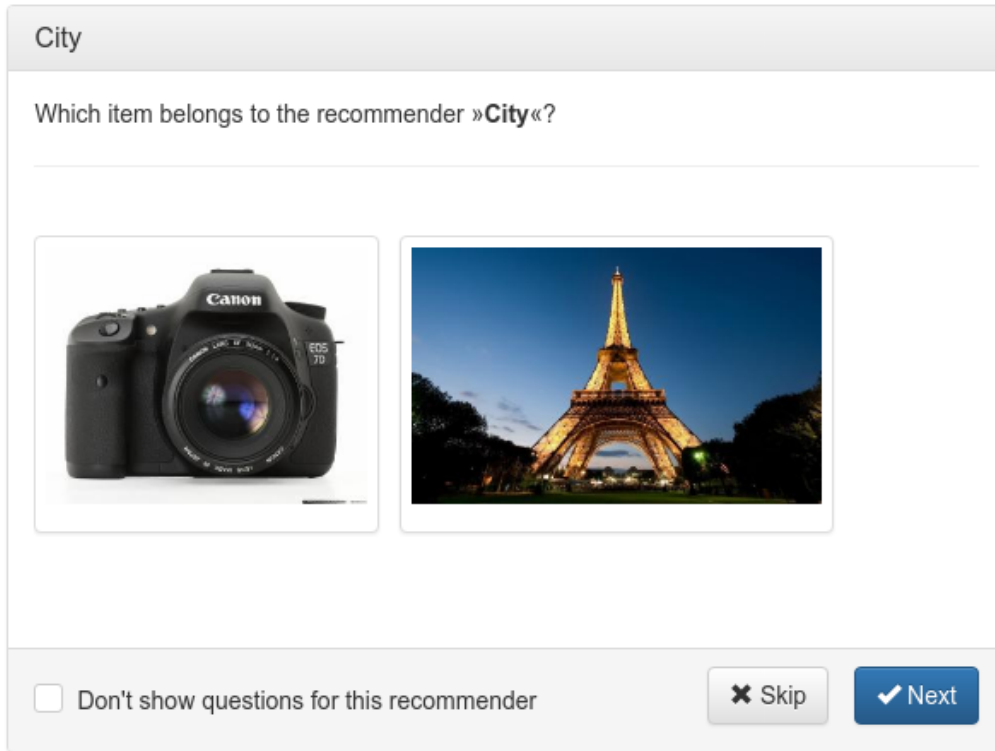


Figure 4.1: An implicit CAPTCHA (microtask type 5). The has to decide which item (“EOS 7D” or “Paris”) belongs to a specific recommender (“City”).

### 4.1.3 Timing Models

Microtasks are the main source of data that is integrated into the knowledge base. Therefore, we have to ensure that the entered information is as useful as possible. Our main approach on rating data is based on models. Building upon the ideas of Zaidan et al. [75], we provide rough models for certain parameters of a task, e.g. the time it takes to answer a microtask. Having these models, we can match collected data to the applicable models using the Kullback-Leibler distance ???. Based on the results we incorporate the data into the knowledge base. If the information does not match the model at all it is discarded (*i.e.*, malicious input or spam). Otherwise, the information is weighted depending on the conformance with the model. The models are updated regularly based on the average parameters of all contributing users.

We conducted a small study with 40 participants to analyze how long the average user keeps himself busy with one microtask. These results are used to determine the maximum workload of a microtask. The timings fit a log-normal distribution model which we verified using a Pearson’s chi-squared test. The log-normal distribution can be used to describe a variety of human behavior, e.g., the item response time [65] or dwell time on online resources [74]. Our reasoning is based on the ideas of Gros et al. [26] and Sobkowicz et al. [63]. They show that if there are two factors of freedom related to a particular information, the lognormal distribution is present [63]. In the case of the microtask, one factor is the time the user has to think about the item itself. The second factor is the time it takes to find the best answer from the users point of view. Figure 4.2 shows that our assumption matches the data collected from a pre-study.

The probability density function for the lognormal distribution with a random variable  $x$  is given as

$$\ln \mathcal{N}(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp \left[ -\frac{(\ln x - \mu)^2}{2\sigma^2} \right], \quad x > 0 \quad (4.3)$$

In our case, the random variable  $x$  is the time it takes to answer the microtask. To approximate the parameters  $\mu$  and  $\sigma$ , we have to determine maximum likelihood estimators. Given the probability density

function in Equation 4.3, the maximum likelihood estimators are

$$\hat{\mu} = \frac{\sum_k \ln x_k}{n} \quad (4.4)$$

for the mean as well as

$$\hat{\sigma}^2 = \frac{\sum_k (\ln x_k - \hat{\mu})^2}{n}. \quad (4.5)$$

for the variance. Additionally, we intend to calculate the global maximum of the probability density function. This maximum is called the mode. It solves the equation  $(\ln f)' = 0$  and is therefore given as

$$\text{Mode}[X] = e^{\mu - \sigma^2} \quad (4.6)$$

Finally, we also calculate the median. It is the point  $x$ , where  $\int_0^x f dx = 0.5$ :

$$\text{Median}[X] = e^{\mu} \quad (4.7)$$

We can update these values regularly with the collected timing information to improve the models. A microtask timing that does not fit the corresponding model is considered an outlier and the user's human score points are decreased. Otherwise, the human score points are increased. This has the effect that many useless answers are penalized with reduced trust in the user. The positive effect on the knowledge base is that data which comes from users with a low human score has negligible influence to the knowledge base. This results in an automatic outlier removal.

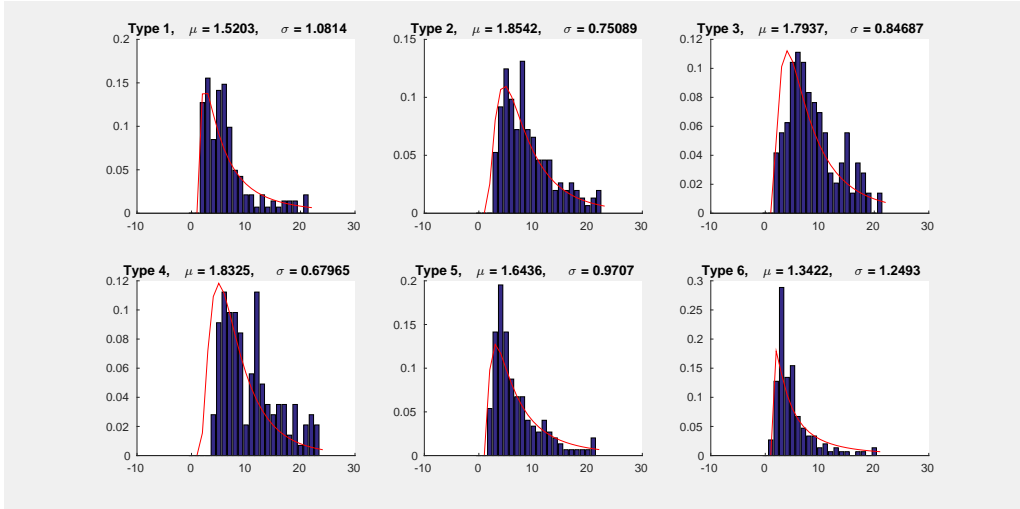


Figure 4.2: The time it took the user to answer a microtask, broken down into the single microtask types.

## 4.2 Manipulation Detection

When collecting data, we have to anticipate manipulation attempts. Lam et al. [45] introduced shilling attacks against recommenders based on collaborative filtering. In those attacks, an attacker creates *attack profiles* that are similar to other users and rates a particular item to either increase or decrease the rating of this item. O'Mahony et al. [52] classify those attack into *push attacks* (position an item at the top of the recommendations) and *nuke attacks* (prevent an item from being recommended). Both of the described attacks rely on the fact that recommendations are calculated based on user profiles. As constraint-based recommendations do not rely on the user profile, the discussed attacks do not apply to our system. This is a countermeasure that is inherent to constraint-based recommenders.

Nonetheless, we can mount a similar attack by creating attack profiles that give good or bad ratings to one particular item on the basis of support values. This is, however, limited by the fact that every item can

only be rated once by a user. Moreover, the user cannot choose items in micro tasks at all. This attack is only a minor threat and can only be mounted manually due to our behavioral analysis of users.

As described in Section 4.1.1, each profile is rated by the human score. This score, which consists of multiple behavioral aspects of a user allows us to prevent manipulations quite effectively. As this score decreases with every suspicious or abnormal activity, the influence of the user's interactions is negligible to non-existent for malicious users. This section focuses on two analysis methods we implemented specifically to detect and block non-human users.

### 4.2.1 Login Behaviour

We expect a user to usually log in from the same country. Logins from varying countries are a suspicious activity. We will apply contextual anomaly detection based on history as it is used for mobile phone fraud detection [7]. A user's history of login locations serves as two-dimensional dataset creating a multivariate normal distribution. The expected clusters are determined using the expectation-maximization algorithm. A high number of clusters indicates profile misuse or a bot and leads to a decrease in the human score points. Currently, the limit is set to two new locations within 24 hours. Figure 4.3 shows the notification a user gets about a potential abuse of his account. Clicking on the notifications reveals further information about the detected anomaly, such as login time, city, browser and operating system.

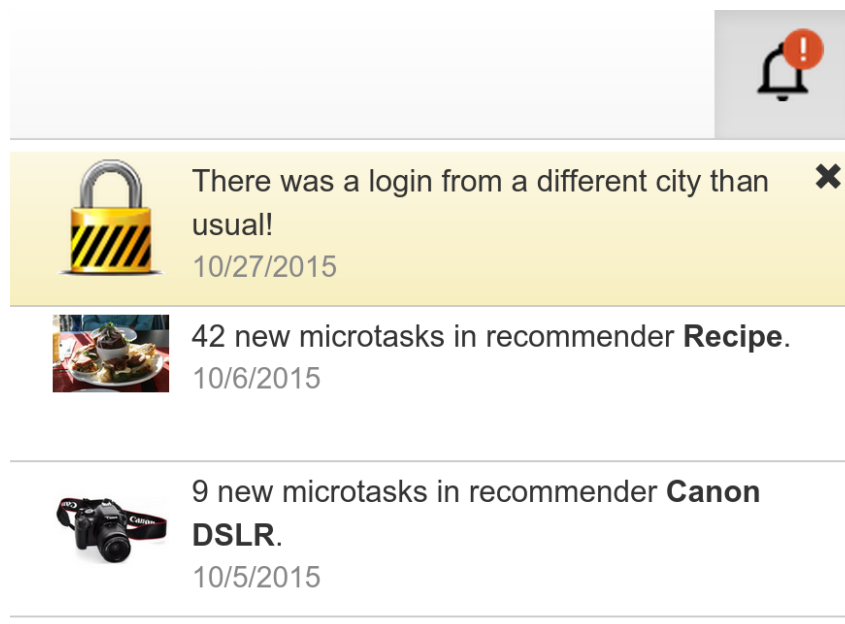


Figure 4.3: A user is notified if there was a login from a different city than usual.

### 4.2.2 Browsing Behaviour

Non-human users, i.e., bots, and human users behave differently when surfing the web [73]. This fact is also widely utilized for detecting bots and distributed denial of service attacks (DDoS) [51].

Normal users use their mouse and/or keyboard to navigate through the website. This generates pseudo-random data which is usually not faked by bots. Moreover, their browsers are individual regarding installed fonts or plugins. We generate and save heat maps of every page to determine if the usage is plausible. Heat maps without traces, such as mouse movement or scrolling are suspicious and therefore decrease the human score points. Furthermore, we can use these heat maps in future studies to analyze the user behavior and improve the system. Figure 4.4 shows two heat maps of the recommendation screen, one for normal behavior and one for abnormal behavior.

If a user has less interactions, i.e., mouse clicks or touches, as the average user, we consider this as suspicious and decrease the human score points. Moreover, we cluster the heat map into *hot areas* as indicated by the red areas in Figure 4.4. If a user is active in none of the hot areas, we also decrease the human score points as this is an abnormal behavior.

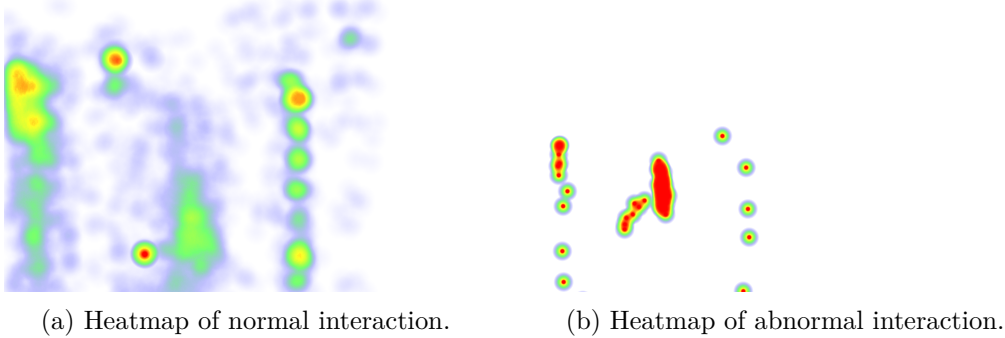


Figure 4.4: The heat map for the recommendation screen. The red areas are parts of the page where the user interaction is high. As we can see, the average user uses mostly the first few filter criteria. Moreover, most of the time the first item is selected.

## 4.3 Data Quality

To get good recommendations, it is essential that the underlying data has a high quality. For this, we have to collect data from the users continuously to update our knowledge base. While collecting complete information for all items, we have to prevent spam.

### 4.3.1 Spam

As users are allowed to create new items, we also face the problem of spam. Spam items are items that are either in the wrong recommender or unrelated items such as advertisement. Especially products are very interesting for spammers, as many users see them [36]. There are already many spam detection methods based on different techniques, for example using neural networks [60], rankings [42], graph theory [4] or machine learning [71]. However, according to Harris et al. [29], the best spam detection mechanism is still a human expert.

Our system is already heavily based on human computation. This allows us to use the users for spam detection. We use microtasks for this task, specifically the Type 6 (cf. Section 3.5.1, Table 3.1). With this microtask, we ask users whether a given item belongs to a specific recommender or not. As soon as we have more than ten answers to such a microtask, we calculate

$$spam = \frac{\text{does not belong to recommender}}{\text{does not belong to recommender} + \text{belongs to recommender}}$$

If this *spam* value is above 0.75, i.e., 75% of the users say that the item is spam, we unpublish the item automatically and notify the owner as it can be seen in Figure 4.5. This threshold was determined empirically and performed well in our tests. Additionally, we also decrease the user's human score points who created this item.

	Microtask Type					
	1	2	3	4	5	6
Create User Account					✓	
Create Item	✓	✓	✓	✓	✓	✓
Change Item		✓				✓
Add User Attribute				✓		
Change User Attribute Domain	✓		✓			

Table 4.2: All user interactions that trigger microtask generation with its corresponding microtasks.

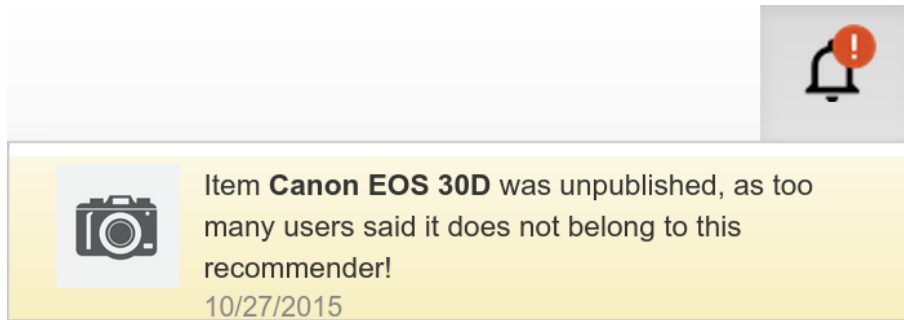


Figure 4.5: An item was unpublished due to the majority vote, i.e., 75% of the users stated that the item is spam.

### 4.3.2 Data Collection

The collection of new knowledge is a process that has to be triggered. There are various types of actions that trigger a collection. Table 4.2 displays all actions and the generated microtasks for these actions. There are two types of actions, changes in the recommender and changes in an item.

Changes in a recommender itself require new knowledge. The possible actions that lead to a generation of microtasks are the following.

- Adding a new item attribute
- Adding a new user attribute
- Changing the domain of a user attribute

Moreover, if a new item is added to a recommender, no knowledge for this item is available. Therefore, new microtasks are generated to collect knowledge for this item. The generated microtasks have the following aim.

- Verify if the item is valid and in the correct recommender (Type 5 and Type 6)
- Verify the entered user attributes (Type 1 and Type 3)
- Collect support values for user attribute values (Type 2 and Type 4)

The number of generated microtasks for each type is determined dynamically. We call this set of microtasks an *agenda*. The initial number of microtasks will vary depending on the recommender. To get an optimal number of microtasks for the agenda, we developed a new algorithm. It is loosely based on the local working set algorithm for task scheduling [10]. At first, the number of tasks will be fixed at a small value, as there are not many users using a new recommender. If a task on the agenda was completed, the result influences the number of tasks for the future. There are two scenarios where the number of micro tasks was too low and the number must be increased for the future. First, if the collected data is uniformly distributed, i.e., every answer was chosen roughly the same number of times. Second, when there are attributes without an answer. Otherwise, the number can be decreased to settle down at a minimum number of needed tasks.

	# of microtasks	Answered	Data is good	New # of microtasks
Cycle 1	10	4	no	$10 \times 1.5 = 15$
Cycle 2	15	11	yes	$15 \times 0.75 = 11$
Cycle 3	11	6	yes	$11 \times 0.75 = 8$
Cycle 4	8	4	no	$8 \times 1.5 = 12$

Table 4.3: This is an example of four cycles for one type of microtask for a specific recommender. The algorithm always starts with the current number of microtasks that were added to the agenda. If the data is good, i.e., there is no uniform distribution and there are no missing answers, after all microtasks have expired or were answered, this number might have been too high and gets decreased (Cycle 2 and Cycle 3). Otherwise, the number is increased (Cycle 1 and Cycle 4).

This mechanism not only ensures a dynamic adaptation to the user base of a recommender, but also ensures that the number of tasks is at a minimum to not bother users more than necessary. Table 4.3 shows an example over four cycles for one type of microtask. A cycle starts if no task of this type is on the agenda anymore. The factors for increase and decrease can be adapted to let the system react faster or slower. The chosen values delivered good results in our tests.

## Chapter 5

# Evaluation

To test multiple aspects of our framework, we designed an extensive study. To get many people as test persons, we chose the MICROWORKERS platform<sup>1</sup>. This platform allows creating micro jobs that are distributed among a huge user basis. Every completed task can be checked either manually or automatically, and the worker is then paid a predefined amount of money.

We selected an above-average payment of 20 cents per successfully completed task. Moreover, we only allowed workers with good ratings to take part in the study to get a realistic dataset. For the whole study, we invested 200\$ and got 1307 users who completed the tasks successfully. As the MICROWORKERS platform allows all of the workers to work in parallel on a task, we got more than the 1000 users we paid for.

To evaluate the quality of the workers, we did a second study that limited the workers to people with a knowledge of the recommendation domain used in the field study. We enforced that by adding a quiz-like survey before accepting workers for the study.

### 5.1 Study Design

For the study, we created a new *study* mode in PeopleViews. It is an extension to the recommendation mode, where the following actions are allowed

- Solving micro tasks
- Evaluating items
- Getting recommendations

To each worker, we assigned seven small tasks that we described on the start page. Figure 5.1 shows this welcome screen.

---

<sup>1</sup><https://microworkers.com/>

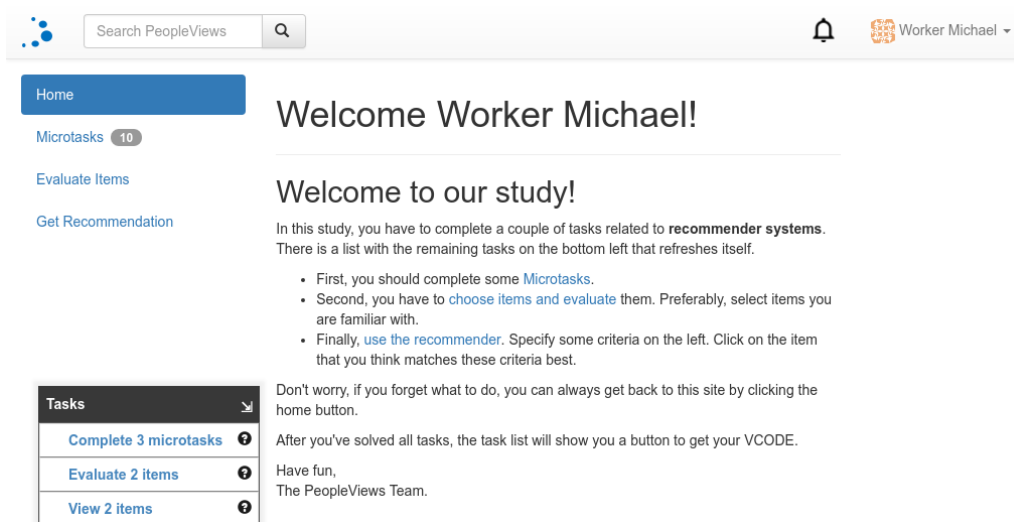


Figure 5.1: Welcome screen of the study including a short introduction and the reduced menu.

We designed the tasks to be straightforward and fast so that a worker has to spend less than 15 minutes on the page. The tasks every worker had to do where:

#### Complete 3 microtasks

We generated ten random micro tasks and displayed them to the worker. The worker was able to skip or answer a micro task. After successfully completing three micro tasks, she was notified that this part of the task was completed.

#### Evaluate 2 items

For this task, we created a new user interface, where we show all items to the user (Figure 5.2). There, the worker had to select the items he wanted to evaluate. We suggested that she chose the items in such a way that she had at least some knowledge about the chosen items.

#### View 2 items

The worker had to use the recommendation functionality to complete this task. We changed the recommender to not show any recommendations when no filter criteria are specified. This forced the worker to specify some criteria to see recommendations. From this recommendations, she was supposed to pick the item that matches her filter criteria best.

We always displayed a task list with the remaining tasks that the worker had to complete. Figure 5.3 shows this task list, where one sub-task is already completed. Each entry has an additional help button to explain the task in more detail.

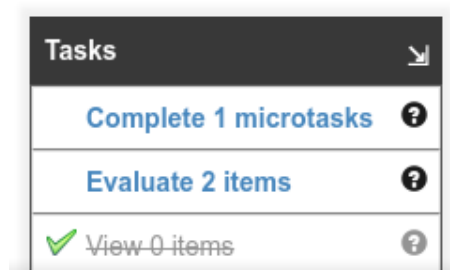










Figure 5.3: The list of tasks for the study

We did not use the recommendation algorithm to calculate the results but displayed all items of the recommender in a random order. On the one hand, this shows us which item the user expected to get for



## All Items

Item		Action
	<b>Berlin</b> Berlin is the capital of Germany. With a population of 3.5 million people it is the second most populous city and the se...	<a href="#">Evaluate</a>
	<b>Beijing</b> Beijing is the capital of the People's Republic of China and one of the most populous cities in the world.	<a href="#">Evaluate</a>
	<b>Shanghai</b> Shanghai is the largest Chinese city by population and the largest city by population in the world.	<a href="#">Evaluate</a>
	<b>Tokyo</b> Tokyo, officially Tokyo Metropolis, is one of the 47 prefectures of Japan, and is both the capital and largest city of J...	<a href="#">Evaluate</a>
	<b>Sydney</b> Sydney is the state capital of New South Wales and the most populous city in Australia and Oceania. Located on Australia...	<a href="#">Evaluate</a>
	<b>New York City</b> The City of New York, often called New York City or simply New York, is the most populous city in the United States.[1] ...	<a href="#">Evaluate</a>
	<b>Mumbai</b> Mumbai (also known as Bombay, the official name until 1995) is the capital city of the Indian state of Maharashtra. Mumb...	<a href="#">Evaluate</a>
	<b>Moscow</b> Moscow is the capital and the largest city of Russia. It is a major political, economic, cultural, and scientific center...	<a href="#">Evaluate</a>

« 1 2 »

Figure 5.2: List of all items to evaluate

his filter criteria so that we can evaluate our recommendation approach. On the other hand, we can analyze if there is a bias towards a specific item position when selecting an item, i.e. if users tend to select only items that are at a certain position of the recommended items list.

For the study, we provided two recommenders with predefined items. The first recommender is the city recommender that was already used in the last chapters. The item attributes and user attributes are listed in Section 2.4.3 and Section 2.4.2 respectively. As a second recommender, we created a Canon DSLR recommender. Table 5.2 and Table 5.4 list the item attributes and user attributes respectively. Workers were randomly assigned to one recommender for evaluating items and the other recommender for getting recommendations. That ensures that a user does not choose the recommended item that she evaluated before.

The city recommender and Canon DSLR recommender contained eleven, respectively, ten items.

Name	Question	Type	Similarity Measure
Megapixel	What is the maximum resolution of this camera in megapixels?	Number	more is better
Maximum ISO	What is the maximum ISO value this camera can support?	Number	more is better
Price	What is the suggested retail price of this camera?	Number	less is better

Table 5.2: Item attributes of the Canon DSLR recommender

Name	Question	Multiple Choice	Values
Field of Application	For which field of application is this camera suited for?	Yes	{Macro, Sport, Portrait, Tele, Landscape}
Experience Level	What is the suggested level of experience a user should have using this camera?	No	{Beginner, Amateur, Expert}
Durability	What level of durability would you expect from this camera?	No	{Bad, Moderate, Good}
Value for Money	What do you think is the value you get for your money when buying this camera?	No	{Good Deal, Price is OK, Too Expensive}
Loss of Value	How fast do you think will this camera lose its value if bought new?	No	{Stable Value, Loses in Value Slowly, Loses in Value Fast}

Table 5.4: User attributes of the Canon DSLR recommender

## 5.2 Collected Data

The study was open for four days. In this time, 1307 tasks were completed successfully. We limited the study to workers with a good rating to get useful data. There are various strategies for worker selection which are not part of this thesis [47, 58, 76]. For the study, we used the automatic worker selection of the microWorkers platform. The majority of the workers are from Eastern Europe, especially from Serbia, Romania, Bosnia and Herzegovina, and Croatia. Many workers are also from the United States, South-Eastern Asia, and Northern Africa. Figure 5.4 shows the regional distribution of the workers that were part of our study. As we can see, we had workers from various countries all over the world, eliminating a possible cultural bias.

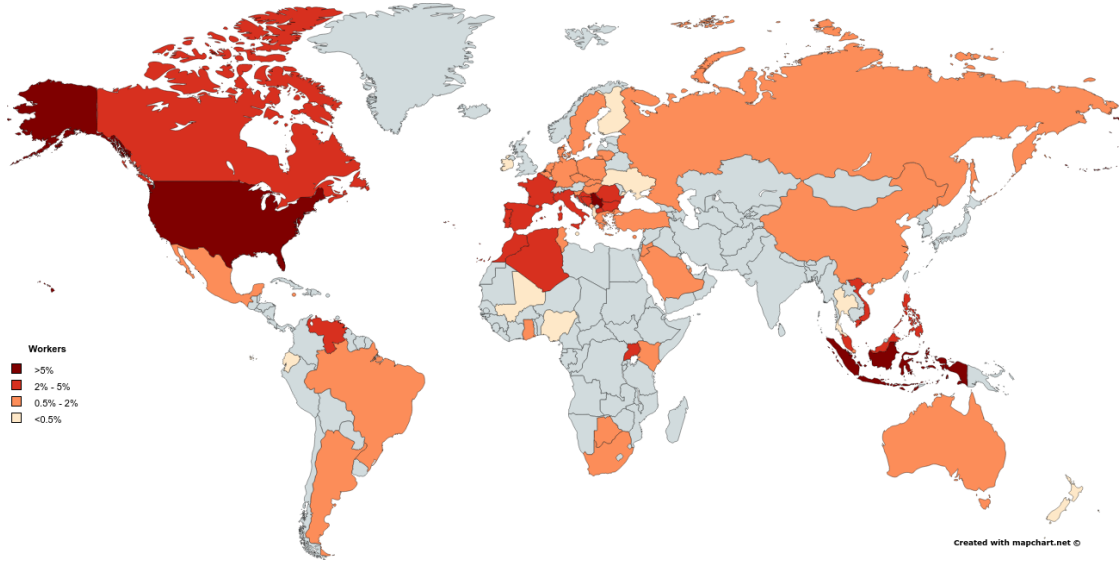


Figure 5.4: The regional distribution of the workers.

### 5.2.1 Data Quality

To assess the overall quality of the collected data, we analyzed the entered CAPTCHAs. The CAPTCHAs are the hidden CAPTCHAs that were explained in Section 4.1.2. When using hidden CAPTCHAs, a user does not realize that she is answering a CAPTCHA. Therefore, we can use the number of wrong CAPTCHAs as a useful metric for the overall quality of the data.

Each user got at least one hidden CAPTCHA as it is also the case in a real-world application of the system. The users answered 1671 CAPTCHAs. From these, only 5% were answered incorrectly. From the wrong CAPTCHAs, 34% were submitted by only 26 distinct users who answered all their CAPTCHAs wrong. Most likely, these users' contribution is only spam as they just clicked through the tasks. This resulted in a human score of zero for these workers and consequently in filtering out their data.

We manually analyzed the incorrect CAPTCHAs further to see whether there is an explanation for them. The wrong CAPTCHAs did not correlate with the country. All wrong answers are uniformly distributed over all countries. This distribution suggests that the CAPTCHAs are understandable and do not depend on any cultural background. We suggest that the spam is caused by the relatively high payment and the automated verification. This combination might entrap users to complete this study by just clicking through all tasks to get the money.

### 5.2.2 Data Amount

From the 1307 completed tasks, we collected more data than we expected. We predicted that the average user would only do the tasks that are required to get the money. Surprisingly, that was not the case and most users completed more tasks than necessary.

For the micro tasks, we got 6264 answers although we only expected 3921. It means that the average user answered 4.5 instead of 3 micro tasks. That is remarkable, as even though the user is notified of the task completion, they continued answering the micro tasks. As the micro tasks were chosen randomly, each item was evaluated by at least 179 micro tasks.

We got 2917 unique evaluations for the 22 items. That are 2.2 evaluations per user, which is still 10% higher than expected. The evaluations were distributed uniformly between the city and the Canon DSLR recommender. Each item was evaluated at least 37 times using an evaluation.

### 5.2.3 Worker Limitation

We conducted another study to see how the worker quality changes when limiting the study to experts. The study design stayed the same. We only performed a selection of the workers beforehand.

To be eligible for the study, each worker had to do a short survey before being approved for the study. The survey can be seen as *ground truth*. Figure 5.5 shows the survey. It consists of five multiple choice questions from which at least four have to be answered correctly. To reduce the number of users who try all answers until they have four correct answers, the page is reloaded if less than four answers are correct. This resets all the answers, and the user has to start from the beginning.

We designed the questions with two criteria in mind. First, the questions should be easily answerable for users who have expertise in the field. These users should be able to complete the survey without having to look something up. Second, the questions should be hard to answer for users without knowledge in the field. Even with the help of search engines, it has to be tedious to find the answer. To fulfill both criteria, we formulated the questions similar to a quiz. The posed questions are nothing someone would usually ask on the internet and therefore hard to find using a search engine. However, with basic knowledge of the field and simple logic, the question can be answered easily.

The survey at the beginning of the study leads to a drastic reduction of workers. To be able to compare the results, we let the study run for four days as well. Due to the survey, we had a reduction of 73% from 900 to 245 workers. This suggests that workers do not necessarily select tasks for which they are eligible. If certain knowledge is a requirement, this has to be enforced by the system.

We compared the quality of the worker based on their human score. For the first study without any limitation on workers, we got an average human score of  $0.265 \pm 0.022$  where the maximum human score is 1. Note that the low value is due to the limited amount of data we are able to collect from one user. When enforcing the limitations through the survey in the beginning, the human score for the same target group increases to  $0.595 \pm 0.035$ . This is an increase of approximately 124%.

As a conclusion, we can reason that a pre-selection of workers indeed improves the quality of the data. However, the improved quality is at the expense of data quantity. In this study, we improved the quality of the average worker by 124% while reducing the amount of data by 73%.

**If you take a photo of a moving car on a race track which Mode Setting is the best to use?**

Aperture Priority (AV or A)

Shutter Priority (TV or S)

Program (P)

**What is the effect if you increase ISO?**

You can produce a crisper photo

You can take photographs in lower light

**Which of the following ones is the largest Aperture?**

f 8

f 16

f 2.8

**Which shutter speed would not produce good results if using a handheld camera with a lens at a focal length of 100mm (and no image stabilizer)?**

1/8 of a second

1/125 of a second

1/250 of a second

**Which of the following is especially suitable for use in low-light photography?**

Low ISO setting (e.g. 100 ISO)

Fast Shutter Speed

Wide Aperture

Figure 5.5: The survey for worker selection. Four out of five questions have to be answered correctly to be forwarded to the study.

## 5.3 Results

The collected data allows us to strengthen our hypothesis about timing models and the user interface. Moreover, we show that the quality assurance indeed increases the recommendation quality.

### 5.3.1 Timing Models

As described in Section 4.1.3, we expected the micro task timings to follow a log-normal distribution. Using the data from the study, we were able to confirm our assumption. Furthermore, we calculated the model parameters and use them as initial values for the models.

Figure 5.6 shows the assumed log-normal distribution applied to the data collected in the study. The exact parameters are displayed in Table 5.6. As planned, the time it takes to answer a micro task is well below one minute. The average time a user spends on a micro task is 16 seconds.

The average time it takes a user from opening the evaluation screen until completely filling it out is 61.5 seconds. This is about three times longer than the longest micro task takes. This time difference and the fact that users are more willing to complete multiple micro tasks as stated in Section 5.2.2 implies that the micro task is a good way to collect knowledge from the user.

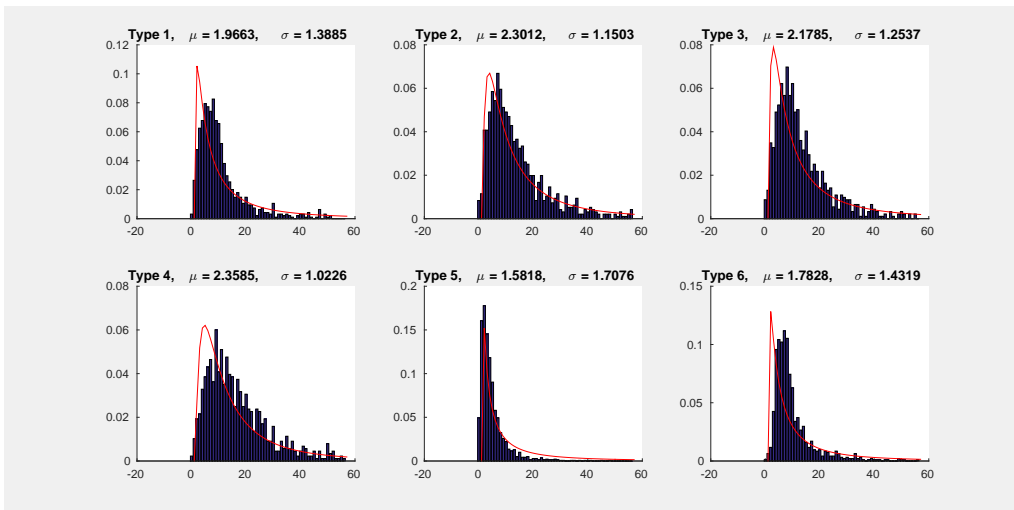


Figure 5.6: The time it took the user to answer a micro task, broken down into the single micro task types.

### 5.3.2 User Interface

We analyzed the information that we collected about the users on our system. It helped us to improve the user interface and decide whether we have to put more focus on any browsers or operating systems. We were especially interested in the used browser and whether the user visited the system using a desktop or a mobile client.

Out of the 1502 workers who visited the page, only 8.3% used a mobile client. The distribution of the browsers is illustrated in Figure 5.7. It follows a long-term trend that is continuously analyzed by w3schools. Most workers used either Google Chrome or Mozilla Firefox, both of them supporting HTML5 and CSS3. As we optimized and tested our system mainly with those two browsers, at least 96% of the users experienced the page as intended.

Type	Amount	$\mu$	$\sigma$	Mode	Median	Avg. time	Description
1	965	1.966	1.389	1.039	7.145	14.8s	Match question-answer with percentage
2	1007	2.301	1.150	2.659	9.986	19.8s	Choose one item out of two regarding one attribute
3	966	2.179	1.254	1.834	8.833	19.3s	Single choice answer
4	926	2.359	1.023	3.716	10.575	22.7s	Multiple choice answer
5	1436	1.582	1.708	0.263	4.864	7.6s	Captcha
6	964	1.783	1.432	0.765	5.947	11.5s	Yes/no question

Table 5.6: Average timing for micro task completion. The values  $\mu$  and  $\sigma$  are the parameters for the approximated lognormal distribution.

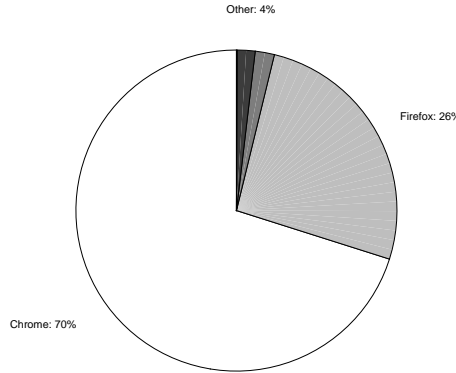


Figure 5.7: Used browsers.

As 90.9% of the workers were able to complete the given tasks, we are confident that the user interface is straightforward and comprehensible. A classic user interface study is subject to future work.

### 5.3.3 Prediction Quality Improvement

Willemsen et al. [72] conducted a study which showed that users do not only consider the top items in a list of recommendations. Contrary to this study, the users considered mostly the first item in the list of recommended items. In fact, 71% chose one out of the first four items only. Figure 5.8 illustrates the probabilities that the user selects the recommended item at a given position. This is the motivation to show the best matching items at the top positions by improving the recommendation quality. Felfernig et al. [17] showed, that showing useful recommendations improves the user satisfaction compared to showing an unsorted product list.

To test the impact on the recommendation quality, we again used the data from the study. We fed the worker's filter criteria into the recommender and checked whether the item that the worker selected was among the top  $N$  recommended items, where  $N$  was between 1 and 10. Then, we applied the quality assurance mechanisms. We weighted the data using the worker's human score and removed outliers using the timing models. By using this filtered data for the recommendation, we expected the results to be better. The improvement was better than anticipated as it is shown in Figure 5.9.

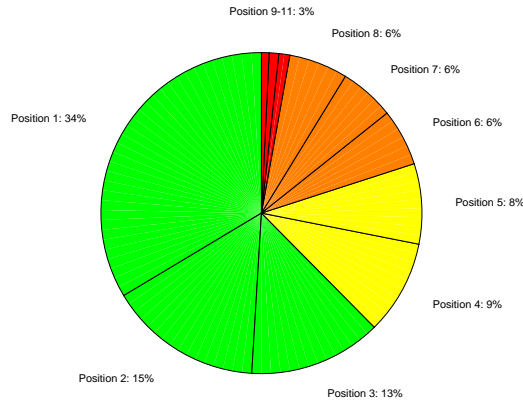


Figure 5.8: The position of items in recommendation list and the probability that they were chosen.

To evaluate the improvement, we let the recommendation algorithm calculate a recommendation for known constraints and evaluated whether the correct item was among the top recommended items. We did this with applied quality assurance and with the original data without any quality assurance.

When taking only the top 3 items of the recommendation into account, the quality assurance mechanisms lead to an improvement of more than 20% if we take the ground truth mechanisms into account. Without the ground truth mechanisms, we get an improvement of around 15%. Although this result is still better than the recommendation without quality assurance, it is considerably worse than quality with enabled ground truth mechanisms. Thus, we use quality assurance with the ground truth mechanisms. Then, the probability that the best matching item is among the top 3 recommended items increases by more than 20%. From Figure 5.8, we know that most of the users selected an item that was among the first 3 items. Therefore, it is plausible to say that the quality assurance has a significant impact on the recommendation quality, independent from the used recommendation approach.

### 5.3.4 Limitations

As all the collected data comes from a study with paid users, the data might not exactly reflect a real-world use case. We see this on the basis of the data quality, where we identified a small number of spamming users. In a real-world application of the system, we would also have to deal with spam, but with a different motivation. In the study, the workers were paid by just completing tasks and therefore created useless inputs to finish the tasks as fast as possible, e.g., they just clicked somewhere to finish a task. Whereas in reality, users would try to use spam to promote their own items.

Moreover, we only tested a subset of the PEOPLEVIEWS system. The workers used the recommender in the same way we expected an average user to use it. However, there was no possibility for the workers to try any other functionality of the system. It was intended that workers did not have the possibility to create new items. In a previous study with an early predecessor of the system, we experienced a very sparse data set when letting the users create own items [18]. Not only is the number of evaluations per item much lower, as there are nearly as many items as evaluations, but the users are also biased. They tend to evaluate their created item or select it in the recommendation list. Therefore, we decided to predefine the items to get a much denser data set without any bias.

We tested the remaining system as well, although with a smaller user group that did the tests voluntarily. Over the period of creating and improving the system, we constantly had users creating recommenders and items to test the functionality and usability of the system. A thorough user interface analysis is subject to future work.

Furthermore, we were not able to test the full extent of the human score, as the workers had only a short interaction period with the system. Nevertheless, even for this brief interaction, the quality assurance was



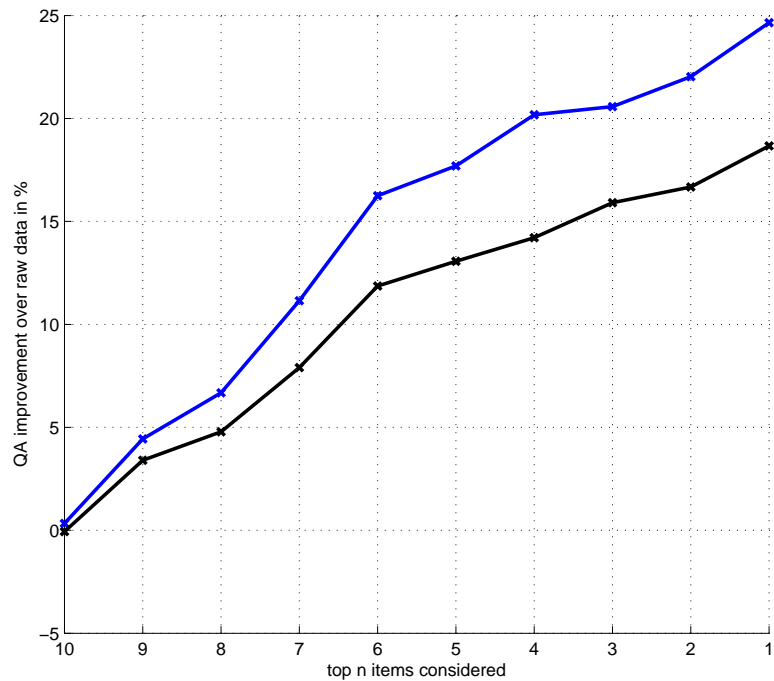


Figure 5.9: The fewer items were considered by the user (“top n items considered”), the higher the improvement by the quality assurance. In other words, the quality assurance ensured that the best fitting items are ranked higher. If only the first 3 items are considered by the user, the probability increases by more than 20% (blue line) that the best matching item is among those 3 items. Without the ground truth mechanisms, the improvement was only around 15% (black line).

able to improve the data and recommendation quality significantly. Also, the distribution of the micro tasks was not tested in the study for the same reason. This was, however, tested in a small scale within our research group and the algorithm yielded good results.

## Chapter 6

# Conclusion and Future Work

Our work showed that it is possible to create a versatile recommender system that collects the data using only human computation. We developed a platform-independent, modular system that can be used both for research and as a ready-to-use recommender. The recommendation approach is constraint-based. This makes it ideal for every use case where there is not much knowledge of the user, i.e., a user does not need a profile to get good recommendations. The constraint set that is used by the recommender is derived solely from user input. The recommender can be utilized as a standalone application using the provided web interface, or it can be embedded in any other project by using the full-fledged API.

Furthermore, we developed quality assurance mechanisms for human computation. We can successfully detect spam and manipulation attempts and take automatic countermeasures to protect the knowledge base. We adapted a well-known scheduling algorithm to adjust the amount of collected data dynamically. It ensures that the data collection is limited to a minimum to not bother the users more than necessary. Moreover, we presented a method to detect and remove outliers automatically using timing models.

We evaluated all our methods using a large-scale study. This study showed that the system is a simple-to-use and robust framework that generates recommendations from a high-quality knowledge base. We proposed that it is not necessary to build recommenders using knowledge experts anymore and that human computation is a viable way to create great recommenders.

The limitations that arise with our system are only marginal. As the quality assurance is a continuous process that has to work on an extensive amount of data, it needs a significant amount of processing power. Although we have a universal framework that is split into modules, we only support constraint-based recommendations. Apart from these two limitations, there are no further known limitations, and we advise the PEOPLEVIEWS platform as a robust basis for further work in this area.

**Future Work** The user interface was developed to be as straightforward and comprehensible as possible. Still, user interface design or user experience design was not a focus of this master's thesis. Therefore, it is certainly a potential for future improvement. From the study, we know, that the users succeeded in using the interface. However, a thorough usability study is missing. Moreover, a further study should be conducted to see whether there are misunderstandings in the questions uses for the microtasks.

Furthermore, a native client for Android would complement the iOS and web client. Android is with a market share of over 80% the biggest player on the mobile operating system market ([33]). This would give the system an even larger user base.

Chung et al. [8] suggested using a Beta distribution to detect malicious rating profiles. This method could further improve the recommendation quality when applying it to the human score. The Beta distribution gives a confidence value for the calculated value. It means that in addition to the weighting we get the confidence in the calculated score as well. This additional metric might lead to even better results.

# Bibliography

- [1] M. Aharon, O. Anava, N. Avigdor-Elgrabli, D. Drachsler-Cohen, S. Golan, and O. Somekh. ExcUseMe. In *Proceedings of the 9th ACM Conference on Recommender Systems - RecSys '15*. Association for Computing Machinery (ACM), 2015. DOI 10.1145/2792838.2800183. URL <http://dx.doi.org/10.1145/2792838.2800183>. (cited on p. 3, 27)
- [2] H. S. Baird and J. L. Bentley. Implicit CAPTCHAs. In E. H. B. Smith and K. Taghva, editors, *Document Recognition and Retrieval XII*. SPIE-Intl Soc Optical Eng, jan 2005. DOI 10.1117/12.590944. URL <http://dx.doi.org/10.1117/12.590944>. (cited on p. 3, 39)
- [3] R. Basu. *Implementing Quality: A Practical Guide to Tools and Techniques : Enabling the Power of Operational Excellence*. Thomson Learning, 2004. ISBN 9781844800575. URL <https://books.google.at/books?id=JHdT8rF4GCwC>. (cited on p. 23)
- [4] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. Know your neighbors: Web spam detection using the web topology. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '07*, pages 423–430, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7. DOI 10.1145/1277741.1277814. URL <http://doi.acm.org/10.1145/1277741.1277814>. (cited on p. 43)
- [5] D. Chaffey. Mobile marketing statistics compilation, 4 2016. URL <http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/>. (cited on p. 6)
- [6] J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. Graphical Methods for Data Analysis. *The Wadsworth Statistics/Probability Series*. Boston, MA: Duxury, 1983. (cited on p. 23)
- [7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 7 2009. ISSN 0360-0300. DOI 10.1145/1541880.1541882. URL <http://doi.acm.org/10.1145/1541880.1541882>. (cited on p. 3, 42)
- [8] C.-Y. Chung, P.-Y. Hsu, and S.-H. Huang.  $\beta$ p: A novel approach to filter out malicious rating profiles from recommender systems. *Decision Support Systems*, 55(1):314–325, apr 2013. DOI 10.1016/j.dss.2013.01.020. URL <http://dx.doi.org/10.1016/j.dss.2013.01.020>. (cited on p. 3, 57)
- [9] S. de la Rosa de Saa, M. A. Gil, G. Gonzalez-Rodriguez, M. T. Lopez, and M. A. Lubiano. Fuzzy rating scale-based questionnaires and their statistical analysis. *IEEE Trans. Fuzzy Syst.*, 23(1):111–126, feb 2015. DOI 10.1109/tfuzz.2014.2307895. URL <http://dx.doi.org/10.1109/TFUZZ.2014.2307895>. (cited on p. 28)
- [10] P. J. Denning. The working set model for program behavior. *Commun. ACM*, 11(5):323–333, 5 1968. ISSN 0001-0782. DOI 10.1145/363095.363141. URL <http://doi.acm.org/10.1145/363095.363141>. (cited on p. 3, 44)
- [11] S. Deterding, D. Dixon, R. Khaled, and L. Nacke. From game design elements to gamefulness. In *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '11*. Association for Computing Machinery (ACM), 2011. DOI 10.1145/2181037.2181040. URL <http://dx.doi.org/10.1145/2181037.2181040>. (cited on p. 27)
- [12] A. Deveria. Can i use... support tables for html5, css3, etc., 2016. URL <http://caniuse.com/>. (cited on p. 6)

- [13] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, Apr. 2011. ISSN 0001-0782. DOI 10.1145/1924421.1924442. URL <http://doi.acm.org/10.1145/1924421.1924442>. (cited on p. 3, 37)
- [14] M. D. Ekstrand, M. Ludwig, J. Kolb, and J. T. Riedl. LensKit. In *Proceedings of the fifth ACM conference on Recommender systems - RecSys '11*. Association for Computing Machinery (ACM), 2011. DOI 10.1145/2043932.2044001. URL <http://dx.doi.org/10.1145/2043932.2044001>. (cited on p. 2)
- [15] S. Exchange. What is reputation? how do i earn (and lose) it?, 2016. URL <http://stackoverflow.com/help/whats-reputation>. (cited on p. 38)
- [16] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce, ICEC '08*, pages 3:1–3:10, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-075-3. DOI 10.1145/1409540.1409544. URL <http://doi.acm.org/10.1145/1409540.1409544>. (cited on p. 1, 2)
- [17] A. Felfernig and B. Gula. An empirical study on consumer behavior in the interaction with knowledge-based recommender applications. In *The 8th IEEE International Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (CEC/EEE'06)*, pages 37–37. IEEE, 2006. (cited on p. 54)
- [18] A. Felfernig, S. Haas, G. Ninaus, M. Schwarz, T. Ulz, M. Stettinger, K. Isak, M. Jeran, and S. Reiterer. Recturk: Constraint-based recommendation based on human computation. In *RecSys 2014 CrowdRec Workshop*, pages 1–6, 2014. (cited on p. 1, 2, 55)
- [19] A. Felfernig, T. Ulz, S. Haas, M. Schwarz, S. Reiterer, and M. Stettinger. Peopleviews: Human computation for constraint-based recommendation. In *ACM RecSys 2015 CrowdRec Workshop*, 2015. (cited on p. 1, 2, 14)
- [20] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. In *Proceedings of the 22nd international conference on Software engineering - ICSE '00*. Association for Computing Machinery (ACM), 2000. DOI 10.1145/337180.337228. URL <http://dx.doi.org/10.1145/337180.337228>. (cited on p. 7)
- [21] M. Firtman. Mobile html5 compatibility, 2016. URL <http://mobilehtml5.org/>. (cited on p. 6)
- [22] U. Gadiraju, R. Kawase, S. Dietze, and G. Demartini. Understanding malicious behavior in crowdsourcing platforms: The case of online surveys. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15*, pages 1631–1640, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3145-6. DOI 10.1145/2702123.2702443. URL <http://doi.acm.org/10.1145/2702123.2702443>. (cited on p. 3, 38)
- [23] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013. (cited on p. 39)
- [24] Google. Are you a robot? introducing "no captcha recaptcha", 12 2014. URL <https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html>. (cited on p. 3, 39)
- [25] Google. Street view and recaptcha technology just got smarter, 4 2014. URL <https://security.googleblog.com/2014/04/street-view-and-recaptcha-technology.html>. (cited on p. 39)
- [26] C. Gros, G. Kaczor, and D. Markovic. Neuropsychological constraints to human data production on a global scale. *CoRR*, abs/1111.6849, 2011. URL <http://dblp.uni-trier.de/db/journals/corr/corr1111.html#abs-1111-6849>. (cited on p. 40)
- [27] M. Hadlow. How to write scalable services, 02 2013. URL <http://mikehadlow.blogspot.co.at/2013/02/how-to-write-scalable-services.html>. (cited on p. 5)
- [28] M. Hahsler. recommenderlab: A framework for developing and testing recommendation algorithms, 2011. (cited on p. 2)
- [29] C. G. Harris. Detecting deceptive opinion spam using human computation. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012. (cited on p. 43)

- [30] T. Hesketh, R. Pryor, and B. Hesketh. An application of a computerized fuzzy graphic rating scale to the psychological measurement of individual differences. *International Journal of Man-Machine Studies*, 29(1):21–35, jan 1988. DOI 10.1016/s0020-7373(88)80029-4. URL [http://dx.doi.org/10.1016/S0020-7373\(88\)80029-4](http://dx.doi.org/10.1016/S0020-7373(88)80029-4). (cited on p. 28)
- [31] M. D. Hill. What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, dec 1990. DOI 10.1145/121973.121975. URL <http://dx.doi.org/10.1145/121973.121975>. (cited on p. 5)
- [32] IBM. Icu - international components for unicode, 2016. URL <https://www.icu-project.org/>. (cited on p. 6)
- [33] IDC. Smartphone os market share, 2015 q2, 2016. URL <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. (cited on p. 57)
- [34] D. İren and S. Bilgen. Cost of quality in crowdsourcing. 2014. (cited on p. 3)
- [35] P. Jaccard. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytologist*, 11(2):37–50, feb 1912. DOI 10.1111/j.1469-8137.1912.tb05611.x. URL <http://dx.doi.org/10.1111/j.1469-8137.1912.tb05611.x>. (cited on p. 1)
- [36] N. Jindal and B. Liu. Review spam detection. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1189–1190, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7. DOI 10.1145/1242572.1242759. URL <http://doi.acm.org/10.1145/1242572.1242759>. (cited on p. 43)
- [37] H. J. Jung. Quality assurance in crowdsourcing via matrix factorization based task routing. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 3–8. ACM, 2014. (cited on p. 3)
- [38] J. H. Jung, C. Schneider, and J. Valacich. Enhancing the motivational affordance of information systems: The effects of real-time performance feedback and goal setting in group collaboration environments. *Management Science*, 56(4):724–742, apr 2010. DOI 10.1287/mnsc.1090.1129. URL <http://dx.doi.org/10.1287/mnsc.1090.1129>. (cited on p. 27)
- [39] R. Khazankin, H. Psaiar, D. Schall, and S. Dustdar. *Service-Oriented Computing: 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings*, chapter QoS-Based Task Scheduling in Crowdsourcing Environments, pages 297–311. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-25535-9. DOI 10.1007/978-3-642-25535-9\_20. URL [http://dx.doi.org/10.1007/978-3-642-25535-9\\_20](http://dx.doi.org/10.1007/978-3-642-25535-9_20). (cited on p. 3)
- [40] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, Mar. 1997. ISSN 0001-0782. DOI 10.1145/245108.245126. URL <http://doi.acm.org/10.1145/245108.245126>. (cited on p. 2)
- [41] M. Krause and R. F. Kizilcec. To play or not to play: Interactions between response quality and task complexity in games and paid crowdsourcing. In *Proceedings of the Third AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2015, November 8-11, 2015, San Diego, California.*, pages 102–109, 2015. URL <http://www.aaai.org/ocs/index.php/HCOMP/HCOMP15/paper/view/11575>. (cited on p. 27, 32)
- [42] V. Krishnan and R. Raj. Web spam detection with anti-trust rank. In *AIRWeb*, volume 6, pages 37–40, 2006. (cited on p. 43)
- [43] V. Krishnan, P. K. Narayanashetty, M. Nathan, R. T. Davies, and J. A. Konstan. Who predicts better?: Results from an online study comparing humans and an online recommender system. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 211–218. ACM, 2008. (cited on p. 1, 3)
- [44] P. Kucherbaev, F. Daniel, S. Tranquillini, and M. Marchese. Modeling and exploration of crowdsourcing micro-tasks execution. In *Proceedings of the Third AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2015, November 8-11, 2015, San Diego, California.*, pages 16–17, 2015. URL <http://www.aaai.org/ocs/index.php/HCOMP/HCOMP15/paper/view/11637>. (cited on p. 27)
- [45] S. K. Lam and J. Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 393–402, New York, NY, USA, 2004. ACM. ISBN 1-58113-844-X. DOI 10.1145/988672.988726. URL <http://doi.acm.org/10.1145/988672.988726>. (cited on p. 3, 37, 41)

- [46] J. Lang, M. Spear, and S. F. Wu. Social manipulation of online recommender systems. In *Proceedings of the Second International Conference on Social Informatics, SocInfo'10*, pages 125–139, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16566-4, 978-3-642-16566-5. URL <http://dl.acm.org/citation.cfm?id=1929326.1929336>. (cited on p. 3, 37)
- [47] H. Li, B. Zhao, and A. Fuxman. The wisdom of minority. In *Proceedings of the 23rd international conference on World wide web - WWW '14*. Association for Computing Machinery (ACM), 2014. DOI 10.1145/2566486.2568033. URL <http://dx.doi.org/10.1145/2566486.2568033>. (cited on p. 50)
- [48] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011. (cited on p. 2)
- [49] D. McSherry. Similarity and compromise. In *Proceedings of the 5th International Conference on Case-based Reasoning: Research and Development, ICCBR'03*, pages 291–305, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40433-3. URL <http://dl.acm.org/citation.cfm?id=1760422.1760448>. (cited on p. 10, 14)
- [50] NIST/SEMATECH. e-handbook of statistical methods, 06 2003. URL <http://www.itl.nist.gov/div898/handbook/eda/section3/starplot.htm>. (cited on p. 23)
- [51] G. Oikonomou and J. Mirkovic. Modeling human behavior for defense against flash-crowd attacks. In *2009 IEEE International Conference on Communications*, pages 1–6, June 2009. DOI 10.1109/ICC.2009.5199191. (cited on p. 3, 42)
- [52] M. P. O'Mahony, N. J. Hurley, and G. C. Silvestre. Recommender systems: Attack types and strategies. In *AAAI*, pages 334–339, 2005. (cited on p. 3, 41)
- [53] M. Otto. About bootstrap, 2016. URL <http://getbootstrap.com/about/>. (cited on p. 5)
- [54] A. Papoutsaki, H. Guo, D. Metaxa-Kakavouli, C. C. Gramazio, J. Rasley, W. Xie, G. Wang, and J. Huang. Crowdsourcing from Scratch: A Pragmatic Experiment in Data Collection by Novice Requesters. In *Proceedings of the Third AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2015, November 8-11, 2015, San Diego, California*. Association for the Advancement of Artificial Intelligence (AAAI) Press, 2015. (cited on p. 27)
- [55] R. Pawlak. Spoon: Annotation-driven program transformation — the aop case. In *Proceedings of the 1st Workshop on Aspect Oriented Middleware Development, AOMD '05*, New York, NY, USA, 2005. ACM. ISBN 1-59593-265-8. DOI 10.1145/1101560.1101566. URL <http://doi.acm.org/10.1145/1101560.1101566>. (cited on p. 6)
- [56] M. J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13(5-6):393–408, Dec. 1999. ISSN 0269-2821. DOI 10.1023/A:1006544522159. URL <http://dx.doi.org/10.1023/A:1006544522159>. (cited on p. 2)
- [57] J. Pickles. *Ground truth: The social implications of geographic information systems*. Guilford Press, 1995. (cited on p. 3)
- [58] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *Journal of Machine Learning Research*, 11(Apr):1297–1322, 2010. (cited on p. 50)
- [59] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011. ISBN 978-0-387-85819-7. URL <http://www.springerlink.com/content/978-0-387-85819-7>. (cited on p. 2)
- [60] A. Rothwell, L. Jagger, W. Dennis, and D. Clarke. Intelligent spam detection system using an updateable neural analysis engine, July 2004. URL <https://www.google.com/patents/US6769016>. US Patent 6,769,016. (cited on p. 43)
- [61] J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. In *Applications of Data Mining to Electronic Commerce*, pages 115–153. Springer, 2001. (cited on p. 2)
- [62] R. R. Sinha and K. Swearingen. Comparing recommendations made by online systems and friends. In *DELOS workshop: personalisation and recommender systems in digital libraries*, volume 106, 2001. (cited on p. 2)

- [63] P. Sobkowicz, M. Thelwall, K. Buckley, G. Paltoglou, and A. Sobkowicz. Lognormal distributions of user post lengths in internet discussions - a consequence of the weber-fechner law? *EPJ Data Sci.*, 2(1), feb 2013. DOI 10.1140/epjds14. URL <http://dx.doi.org/10.1140/epjds14>. (cited on p. 40)
- [64] L. Terveen and W. Hill. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*, 1:487–509, 2001. (cited on p. 1)
- [65] W. J. van der Linden. A lognormal model for response times on test items. *Journal of Educational and Behavioral Statistics*, 31(2):181–204, jan 2006. DOI 10.3102/10769986031002181. URL <http://dx.doi.org/10.3102/10769986031002181>. (cited on p. 40)
- [66] L. von Ahn. Human computation. In *Proceedings of the 4th International Conference on Knowledge Capture, K-CAP '07*, pages 5–6, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-643-1. DOI 10.1145/1298406.1298408. URL <http://doi.acm.org/10.1145/1298406.1298408>. (cited on p. 2)
- [67] L. Von Ahn. Human computation. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 418–419. IEEE, 2009. (cited on p. 1)
- [68] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, Feb. 2004. ISSN 0001-0782. DOI 10.1145/966389.966390. URL <http://doi.acm.org/10.1145/966389.966390>. (cited on p. 39)
- [69] w3schools. Browser statistics, 2016. URL [http://www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp). (cited on p. 53)
- [70] H. Wada and J. Suzuki. *Model Driven Engineering Languages and Systems: 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005. Proceedings*, chapter Modeling Turnpike Frontend System: A Model-Driven Development Framework Leveraging UML Metamodeling and Attribute-Oriented Programming, pages 584–600. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32057-9. DOI 10.1007/11557432744. URL <http://dx.doi.org/10.1007/11557432744>. (cited on p. 6)
- [71] A. H. Wang. Don't follow me: Spam detection in twitter. In *Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on*, pages 1–10, July 2010. (cited on p. 43)
- [72] M. C. Willemsen, B. P. Knijnenburg, M. P. Graus, L. C. Velter-Bremmers, and K. Fu. Using latent features diversification to reduce choice difficulty in recommendation lists. *RecSys*, 11:14–20, 2011. (cited on p. 54)
- [73] T. Yatagai, T. Isohara, and I. Sasase. Detection of http-get flood attack based on analysis of page access behavior. In *2007 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 232–235, Aug 2007. DOI 10.1109/PACRIM.2007.4313218. (cited on p. 42)
- [74] P. Yin, P. Luo, W.-C. Lee, and M. Wang. Silence is also evidence: interpreting dwell time for recommendation from psychological perspective. In I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, J. He, R. L. Grossman, and R. Uthurusamy, editors, *KDD*, pages 989–997. ACM, 2013. ISBN 978-1-4503-2174-7. URL <http://dblp.uni-trier.de/db/conf/kdd/kdd2013.html#YinLLW13>. (cited on p. 40)
- [75] O. F. Zaidan and C. Callison-Burch. Crowdsourcing translation: Professional quality from non-professionals. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1220–1229. Association for Computational Linguistics, 2011. (cited on p. 3, 40)
- [76] D. Zhou, S. Basu, Y. Mao, and J. C. Platt. Learning from the wisdom of crowds by minimax entropy. In *Advances in Neural Information Processing Systems*, pages 2195–2203, 2012. (cited on p. 50)
- [77] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. Attacks and design of image recognition captchas. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS '10*, pages 187–200, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0245-6. DOI 10.1145/1866307.1866329. URL <http://doi.acm.org/10.1145/1866307.1866329>. (cited on p. 39)
- [78] S. Zhu, S. Kane, J. Feng, and A. Sears. A crowdsourcing quality control model for tasks distributed in parallel. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, pages

2501–2506, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1016-1. DOI 10.1145/2212776.2223826.  
URL <http://doi.acm.org/10.1145/2212776.2223826>. (cited on p. 3, 37)



# Appendix A

## API

In this chapter we list all the API calls with a brief description and all parameters. The API calls are all JSON messages as described in Section 2.2.1. The all fields of the content are listed including their data type. The `type` is always given next to the API call name.

### A.1 Register / Login

#### Login user

`login`

*Logs in a user that is registered as PeopleViews user. Not used for SSO (OAuth), see auth for this.*

#### Parameters

Field	Description
<code>user</code> <i>String</i>	<b>required</b> <i>The user's username or email address</i>
<code>password</code> <i>String</i>	<b>required</b> <i>The user's password in plaintext</i>
<code>os</code> <i>String</i>	<b>optional</b> <i>The users' operating system</i>
<code>browser</code> <i>String</i>	<b>optional</b> <i>The users' browser name</i>
<code>browserVersion</code> <i>String</i>	<b>optional</b> <i>The version of the users' browser</i>

**Response** `true` if the login was successful, `false` otherwise

#### Response Model

Field	Description
<code>id</code> <i>Long</i>	<b>required</b> <i>The user's internal (unique) ID</i>
<code>username</code> <i>String</i>	<b>required</b> <i>The user's username</i>
<code>email</code> <i>String</i>	<b>required</b> <i>The user's email address</i>

<b>userType</b> <i>enum</i>	<b>required</b> <i>The user type, normal users have null, administrators ADMIN</i>
<b>imageId</b> <i>Long</i>	<b>required</b> <i>The image ID of the user's avatar</i>
<b>overallPvPoints</b> <i>Double</i>	<b>required</b> <i>The user's total sum of PeopleViews Points</i>
<b>pvRanking</b> <i>Integer[10]</i>	<b>required</b> <i>The user's rank in the system of the last 10 days</i>

---

## Logout user

logout

*Logs the current user out.*

**Parameters** none

**Response** true always, false never

**Response Model**  
none

---

## Login State

loginState

*Gets the login state of the current user. Returns the user info if the user is logged in.*

**Parameters** none

**Response** true if the user is logged in, false otherwise

**Response Model**  
same as in login

---

## Single Sign-On (OAuth)

auth

*Logs in a user using Facebook or Google SSO (OAuth).*

**Parameters**

Field	Description
<b>provider</b> <i>String</i>	<b>required</b> <i>The OAuth provider, either facebook or google</i>
<b>accessToken</b> <i>String</i>	<b>required</b> <i>The access token returned by Google/Facebook OAuth login</i>
<b>os</b> <i>String</i>	<b>optional</b> <i>The users' operating system</i>
<b>browser</b> <i>String</i>	<b>optional</b> <i>The users' browser name</i>
<b>browserVersion</b> <i>String</i>	<b>optional</b> <i>The version of the users' browser</i>

**Response** true if the login was successful, **false** otherwise

**Response Model**  
same as in login

---

## Register new user

**register**

*Registers a new user. Not used for SSO (OAuth), see auth for this.*

### Parameters

Field	Description
username <i>String</i>	<b>required</b> <i>The new user's username.</i>
password <i>String</i>	<b>required</b> <i>The new user's password in plaintext</i>
email <i>String</i>	<b>required</b> <i>The new user's email address. Will be used to send the activation code.</i>

**Response** true if the registration was successful, **false** otherwise

**Response Model**  
same as in login

---

## Request password reset

**requestPasswordReset**

*Request a link to reset the user's password*

### Parameters

Field	Description
email <i>String</i>	<b>required</b> <i>The user's email address. Reset link is sent to this address.</i>

**Response** true if the reset was successful, **false** otherwise

**Response Model**  
none

---

## Reset password

**resetPassword**

*Resets the users password using the received reset code.*

### Parameters

Field	Description
password <i>String</i>	<b>required</b> <i>The user's new password in plain.</i>
resetCode <i>String</i>	<b>required</b> <i>The received reset-code.</i>

**Response** true if the reset was successful, **false** otherwise  
**Response Model** none

---

## A.2 Items

### Add/Edit Item

**addAndEditItem**

*Creates a new or edits an existing item*

#### Parameters

Field	Description
id <i>Long</i>	<b>required</b> <i>The ID of the item to edit, or <b>null</b> if it is a new item</i>
recommender <i>Long</i>	<b>required</b> <i>ID of the recommender to which the items belongs to</i>
link <i>String</i>	<b>required</b> <i>Item's URL</i>
name <i>String</i>	<b>required</b> <i>Item's name</i>
tags <i>String</i>	<b>required</b> <i>Tags describing the item, comma-separated</i>
description <i>String</i>	<b>required</b> <i>Item's description</i>
published <i>Boolean</i>	<b>required</b> <i>Whether this item is published or not</i>
imageId <i>Long</i>	<b>required</b> <i>Image ID (or -1 for no image)</i>
productAttributeValues <i>ProductAttributeValue[]</i>	<b>required</b> <i>List of product (=item) attributes and their value</i>

**Response** true if the creation/modification was successful, **false** if there was an error (e.g. item with this name exists already)  
**Response Model** Item

---

### My Items

**myItems**

*Returns a list of items that the user created*

#### Parameters

Field	Description
recommenderId <i>Long</i>	<b>required</b> <i>The ID of the recommender for which the items are retrieved</i>
firstItem <i>Long</i>	<b>optional</b> <i>Index of first item (for pagination)</i>

---

<b>numItems</b> <i>Long</i>	<b>optional</b> <i>Number of items to retrieve (for pagination)</i>
--------------------------------	---

---

**Response** true if the parameters are valid, **false** otherwise

**Response Model**

Field	Description
<b>objectCount</b> <i>Long</i>	<b>required</b> <i>Number of items found in total</i>
<b>array</b> <i>Item[]</i>	<b>required</b> <i>User's items</i>

---

## Get Item

**getItem**

*Retrieves all the information for the given item. The item is specified by its ID.*

**Parameters**

Field	Description
<b>itemId</b> <i>Long</i>	<b>required</b> <i>The ID of the item to retrieve</i>

---

**Response** true if the item exists, **false** otherwise

**Response Model**

*Item*

---

## Get Support Values

**getSupportValues**

*Retrieves the spuort values for the given item.*

**Parameters**

Field	Description
<b>itemId</b> <i>Long</i>	<b>required</b> <i>The ID of the item for which the support values should be retrieved</i>

---

**Response** true if the item exists, **false** otherwise

**Response Model**

*SupportValue[]*

---

## Get similar items

**itemGetSimilar**

*Retrieves a list of items that are similar to the given item.*

**Parameters**

Field	Description
itemId <i>Long</i>	<b>required</b> <i>The base item. Items similar to this item are retrieved.</i>

**Response** true if the item exists, false otherwise

**Response Model**

Item[]

## A.3 Recommender

### Add/Edit Recommender

**addRecommender**

*Creates a new or edits an existing recommender*

**Parameters**

Field	Description
id <i>Long</i>	<b>required</b> <i>The ID of the recommender to edit, or null if it is a new recommender</i>
name <i>String</i>	<b>required</b> <i>Name of the recommender</i>
published <i>Boolean</i>	<b>required</b> <i>Whether this recommender is published or not</i>
tags <i>String</i>	<b>required</b> <i>Comma-separated list of tags</i>
productAttributeDefinitions <i>ProductAttributeDefinition[]</i>	<b>required</b> <i>Array of product (=item) attribute definitions</i>
userAttributeDefinitions <i>UserAttributeDefinition[]</i>	<b>required</b> <i>Array of user attributes</i>

**Response** true if the creation/modification was successful, false if there was an error (e.g. recommender with this name exists already)

**Response Model**

Recommender

### Get Recommender

**getRecommender**

*Returns the recommender specified by its ID*

**Parameters**

Field	Description
recommenderId <i>Long</i>	<b>required</b> <i>The ID of the recommender</i>

**Response** true if the recommender exists, false otherwise  
**Response Model**  
 Recommender

---

## My Recommender

myRecommender

*Returns the recommenders of the current user*

**Parameters** none

**Response** true if the user is logged in, false otherwise  
**Response Model**  
 Recommender[]

---

## Get All Recommenders

getAllRecommenders

*Returns the list of all recommenders*

**Parameters** none

**Response** true always, false never  
**Response Model**  
 Recommender[]

---

## Recommender Editable

isRecommenderEditable

*Returns whether the current user is allowed to edit the recommender*

**Parameters**

Field	Description
id	<b>required</b> The ID of the recommender
Long	

---

**Response** true if the recommender is editable, false otherwise  
**Response Model**  
 null

---

## A.4 User

### User Profile

userprofile

*Retrieves the current users' profile.*

**Parameters** none

**Response** true always, false never

**Response Model**  
UserProfile

---

## Save User Profile

saveProfile

*Saves the current users' profile.*

### Parameters

Field	Description
username <i>String</i>	<b>required</b> <i>The users' username</i>
imageId <i>Long</i>	<b>required</b> <i>The image ID for the user profile</i>
ignoredRecommenders <i>Long[]</i>	<b>required</b> <i>Array of recommender IDs for which the user does not want to get microtasks</i>

**Response** true always, false never

**Response Model**  
none

---

## A.5 Microtasks

### Get Microtask

getMicrotask

*Retrieves a microtask for the current user.*

**Parameters** none

**Response** true if there is an available microtask, false otherwise

**Response Model**  
MicroTask

---

### Save Microtask

saveMicroTask

*Save the result of a microtask.*

### Parameters

Field	Description
id <i>Long</i>	<b>required</b> <i>The microtask's ID</i>



<code>noquestions</code> <i>Boolean</i>	<b>required</b> <i>true if the user wants to ignore this recommender for future microtasks</i>
<code>recommender</code> <i>Long</i>	<b>required</b> <i>Recommender ID to which this microtask belongs</i>
<code>belongstorec</code> <i>Boolean</i>	<b>required</b> <i>(only for Type 6) true for "Yes" as answer, false for "No"</i>
<code>selectedImg</code> <i>Long</i>	<b>required</b> <i>(only for Type 2 and Type 5) Image ID of the selected image</i>
<code>...</code> <i>MicroTask</i>	<b>required</b> <i>All fields of the MicroTask structure, as retrieved by getMicrotask</i>

**Response** true always, false never

**Response Model**  
none

## Skip Microtask

`setSkipped`

*Skips the retrieved microtask.*

### Parameters

Field	Description
<code>id</code> <i>Long</i>	<b>required</b> <i>The microtask's ID</i>
<code>noquestions</code> <i>Boolean</i>	<b>required</b> <i>true if the user wants to ignore this recommender for future microtasks</i>

**Response** true if there is an available microtask, false otherwise

**Response Model**  
MicroTask

## A.6 Evaluations

### Save Evaluation

`saveEvaluation`

*Saves the evaluation for an item.*

### Parameters

Field	Description
<code>itemId</code> <i>Long</i>	<b>required</b> <i>The ID of the item that is evaluated</i>
<code>completionTime</code> <i>Long</i>	<b>required</b> <i>The time it took the user to evaluate the item (in ms)</i>
<code>model</code> <i>UserAttributeDefinition[]</i>	<b>required</b> <i>All evaluated user attributes with their support</i>

**Response** true if the evaluation was saved, false otherwise

**Response Model**  
userAttributeDefinition[]

## Get Evaluations

**getExistingEvaluations**

*Retrieves the evaluations for an item.*

### Parameters

Field	Description
itemId <i>Long</i>	<b>required</b> The ID of the item for which to get the evaluations

**Response** true always, false never

**Response Model**  
userAttributeDefinition[]

## A.7 Data Structure

### Item

*The data structure for an item.*

**Data Type** Item

#### Fields

Field	Description
id <i>Long</i>	<b>required</b> The ID of the item, or <i>null</i> if it is a new item
recommender <i>Long</i>	<b>required</b> ID of the recommender to which the items belongs to
link <i>String</i>	<b>required</b> Item's URL
name <i>String</i>	<b>required</b> Item's name
tags <i>String</i>	<b>required</b> Tags describing the item, comma-separated
description <i>String</i>	<b>required</b> Item's description
published <i>Boolean</i>	<b>required</b> Whether this item is published or not
imageId <i>Long</i>	<b>required</b> Image ID (or -1 for no image)
productAttributeValues <i>ProductAttributeValue[]</i>	<b>required</b> List of product (=item) attributes and their value

---

<code>editable</code> <i>Boolean</i>	<b>optional</b> <i>Whether the item is editable by the current user</i>
---	---

---

## Product/Item Attribute Definition

Structure to define a product (=item) attribute

**Data Type**      `ProductAttributeDefinition`  
**Fields**

<b>Field</b>	<b>Description</b>
<code>id</code> <i>Long</i>	<b>required</b> <i>The ID of the product attribute definition</i>
<code>name</code> <i>String</i>	<b>required</b> <i>Product attribute's name, displayed e.g. in the item details</i>
<code>question</code> <i>String</i>	<b>required</b> <i>Product attribute's question that is asked when creating a new item</i>
<code>attributeType</code> <i>String</i>	<b>required</b> <i>ENUM, NUMBER or TEXT depending on the product attribute's type</i>
<code>similarityMeasure</code> <i>String</i>	<b>required</b> <i>Defines the similarity metric for this attribute. Possible values are EIB, NIB, MIB, LIB</i>
<code>enumValues</code> <i>String</i>	<b>required</b> <i>Comma-separated list of possible answers if the type is ENUM</i>
<code>filterRelevant</code> <i>Boolean</i>	<b>required</b> <i>Whether this product attribute can be used as constraint in a recommendation</i>
<code>minNumericValue</code> <i>Double</i>	<b>optional</b> <i>If the type is NUMBER, this contains the minimum value over all items</i>
<code>maxNumericValue</code> <i>Double</i>	<b>optional</b> <i>If the type is NUMBER, this contains the maximum value over all items</i>

---

## Product/Item Attribute Value

Structure that defines the answer of a product (=item) attribute

**Data Type**      `ProductAttributeValue`  
**Fields**

<b>Field</b>	<b>Description</b>
<code>id</code> <i>Long</i>	<b>required</b> <i>The ID of the product attribute value</i>

---

---

value <i>String</i>	<b>required</b> <i>The actual value of the product attribute</i>
productAttributeDefinition <i>ProductAttributeDefinition</i>	<b>required</b> <i>The corresponding product attribute definition</i>

---

## Support Value

*Structure that describes the support value of an user attribute value*

**Data Type**      ProductAttributeValue  
**Fields**

Field	Description
id <i>Long</i>	<b>required</b> <i>The ID of the support value</i>
value <i>Double</i>	<b>required</b> <i>The actual value of the support value</i>
userAttributeValue <i>UserAttributeValue</i>	<b>required</b> <i>The corresponding user attribute value</i>

---

## User Attribute Definition

*Structure that describes user attribute definition*

**Data Type**      UserAttributeDefinition  
**Fields**

Field	Description
id <i>Long</i>	<b>required</b> <i>The ID of the user attribute definition</i>
name <i>String</i>	<b>required</b> <i>The name of the user attribute definition</i>
question <i>String</i>	<b>required</b> <i>The question for the user attribute definition</i>
multipleAnswers <i>Boolean</i>	<b>required</b> <i>true if this is a multiple choice attribute</i>
userAttributeValues <i>UserAttributeValues[]</i>	<b>optional</b> <i>The corresponding user attribute values for the user attribute definition</i>

---

## User Attribute Value

*Structure that describes user attribute value*

**Data Type**      `UserAttributeValue`  
**Fields**

Field	Description
<code>id</code> <i>Long</i>	<b>required</b> <i>The ID of the user attribute value</i>
<code>name</code> <i>String</i>	<b>required</b> <i>The value of the user attribute value</i>
<code>evaluations</code> <i>SupportValue[]</i>	<b>optional</b> <i>The corresponding support values for the user attribute value</i>

---

## Recommender

*Structure that describes a recommender*

**Data Type**      `Recommender`  
**Fields**

Field	Description
<code>id</code> <i>Long</i>	<b>required</b> <i>The ID of the recommender</i>
<code>name</code> <i>String</i>	<b>required</b> <i>Name of the recommender</i>
<code>published</code> <i>Boolean</i>	<b>required</b> <i>Whether this recommender is published or not</i>
<code>tags</code> <i>String</i>	<b>required</b> <i>Comma-separated list of tags</i>
<code>productAttributeDefinitions</code> <i>ProductAttributeDefinition[]</i>	<b>required</b> <i>Array of product (=item) attribute definitions</i>
<code>userAttributeDefinitions</code> <i>UserAttributeDefinition[]</i>	<b>required</b> <i>Array of user attributes</i>

---

## User Profile

*Structure that describes a user's public profile*

**Data Type**      `UserProfile`  
**Fields**

Field	Description
-------	-------------

---

<b>username</b> <i>String</i>	<b>required</b> <i>Username of the user</i>
<b>email</b> <i>String</i>	<b>required</b> <i>User's email address or SSO ID</i>
<b>imageId</b> <i>Long</i>	<b>required</b> <i>The image ID for the user profile</i>
<b>ignoredRecommenders</b> <i>Long[]</i>	<b>required</b> <i>Array of recommender IDs for which the user does not want to get microtasks</i>

---

## Micro Task

Structure that describes a micro task

**Data Type**      **MicroTask**  
**Fields**

<b>Field</b>	<b>Description</b>
<b>id</b> <i>Boolean</i>	<b>required</b> <i>The micro task's ID</i>
<b>question</b> <i>String</i>	<b>required</b> <i>Question for the microtask that is displayed to the user</i>
<b>type</b> <i>String</i>	<b>required</b> <i>The type of the microtask, one of T1, T2, T3, T4, T5 or T6</i>
<b>userAttributeValues</b> <i>UserAttributeValue[]</i>	<b>required</b> <i>User attribute values for the question, if required</i>
<b>userAttributeDefinition</b> <i>UserAttributeDefinition</i>	<b>required</b> <i>User attribute definition for the question, if required</i>
<b>items</b> <i>Item[]</i>	<b>required</b> <i>Items for the question, if required</i>
<b>recommender</b> <i>Recommender</i>	<b>required</b> <i>The recommender to which this micro task belongs to</i>

---

## User Attribute Filter

Structure that describes a filter criterium using a user attribute

**Data Type**      **UserAttributeFilter**  
**Fields**

<b>Field</b>	<b>Description</b>
<b>userAttributeAnswerId</b> <i>Long</i>	<b>required</b> <i>The ID of the user attribute value</i>

---

---

<b>weight</b> <i>Float</i>	<b>required</b> <i>The weight for this user attribute value in the range [0;1]</i>
-------------------------------	--

---

## Product Attribute Filter

*Structure that describes a filter criterium using a product attribute*

**Data Type**      `ProductAttributeFilter`  
**Fields**

<b>Field</b>	<b>Description</b>
<code>productAttributeId</code> <i>Long</i>	<b>required</b> <i>The ID of the product (=item) attribute</i>
<b>weight</b> <i>Float</i>	<b>required</b> <i>The weight for this product attribute in the range [0;1]</i>
<code>values</code> <i>Float[]</i>	<b>required</b> <i>The bounds for the filter criterium</i>

---