universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Real-Time on Demand Crowdsourcing Framework for Multimodal Process"

verfasst von / submitted by

Roman Habitzl, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2018 / Vienna 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 935
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Medieninformatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas

Declaration of Originality

I hereby declare that except where specific reference is made to the work of others, the content of this thesis is original and has not been submitted in whole or in part for any other degree or qualification in this, or any other university. I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others.

Vienna, ...
(Signature)

Abstract (Deutsch) Crowdsourcing nutzt die Intelligenz der Masse von freiwilligen Akteuren, um Aufgaben zu bewältigen, welche zu komplex sind, als das Computer sie alleine lösen könnten. Es existieren zwar bereits einige Crowdsourcing-Plattformen, diese besitzen aber meist bestimmte Einschränkungen, wie beispielsweise die begrenzte Erreichbarkeit und Verfügbarkeit der Nutzer. Diese Arbeit versucht verschiedenste, aktuelle Methoden aus dem Bereich des Crowdsourcing zu verbinden, um eine einzige Plattform zu schaffen, welche akzeptable Ergebnisse in Echtzeit und auf Abruf liefern kann. Die weite Verbreitung von mobilen Endgeräten wie Smartphones wird genutzt, um die Verfügbarkeit der Nutzer zu steigern. Die getroffenen Design-Entscheidungen bezüglich Architektur und Implementierung werden detailliert präsentiert und auf die verwendeten Technologien wird eingegangen. Der entwickelte Prototyp wurde schließlich auf seine qualitativen und quantitativen Eigenschaften getestet, wodurch die Nutzerfreundlichkeit und technische Leistung bestimmt werden konnten. Letztendlich präsentiert die Arbeit zukünftige Verbesserungs- und Erweiterungsmöglichkeiten.

Abstract (English) Crowdsourcing uses the collective intelligence of voluntary crowd workers to handle tasks which are currently too complex to be solved by computers alone. While there already exist quite a few crowdsourcing platforms, most of them come with certain limitations, such as the very restricted availability of the user base. This work attempts to investigate and combine different existing state-of-the-art approaches to build a single platform capable of providing acceptable results in real-time and on demand. By utilizing the wide spread of mobile devices like smartphones, this project aims to increase the crowd workers' availability. The work presents the design decisions in terms of architecture and implementation in detail. Additionally, the used technologies are introduced and discussed. The developed prototype was tested in terms of quality and quantity, determining its capabilities regarding usability and technical performance respectively. Finally, the work presents possible ways of further improving the system and extending its features.

Table of Contents

List of Tables

1 Introduction

1.1 Motivation and Problem

Using "Machine Learning" techniques, algorithms can be specified that give computers the ability to learn without being explicitly programmed. This works by predicting the data based on a preceding learning process. However, the results of such automated systems are often inaccurate due to limited training data. With the use of crowdsourcing and the collective intelligence of voluntary crowd workers, these training sets and therefore the accuracy of the resulting outcome can be improved.

On the other hand, there are tasks that are too complex to be solved without human intelligence [4]. Good examples of such tasks are text simplification measurements [17] and activity recognition like identifying dependency relationships between actions [18][19].

Figure 1 illustrates the concept of crowdsourcing and shows the interaction of both the requester and the crowd workers with the crowdsourcing platform.

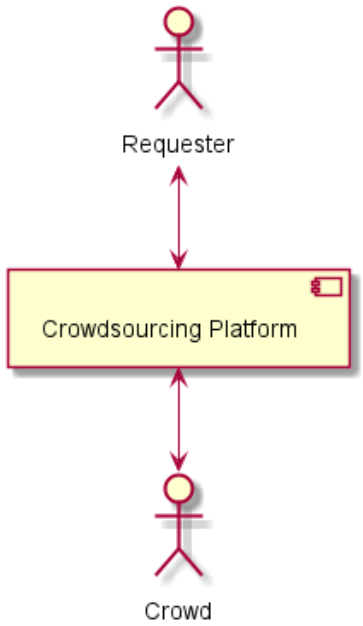

Fig. 1. Concept of crowdsourcing

Most existing crowdsourcing platforms come with certain limitations. Several tasks often require the crowd workers to be available at a specific time or location. When working from a desktop computer it may not be very appropriate to

provide solutions to these problems. Because of the wide spread of mobile devices a mobile application for crowdsourcing could provide results from any place at any time.

The challenge of this thesis is to combine different concepts to build a single platform capable of providing acceptable results in consideration of all the requirements that are presented in section 1.2. The implementation of the mentioned requirements leads to the development of a mobile crowdsourcing framework which enables real-time crowd feedback on demand. In addition to the basic functionality this framework prototype shall address the problems of real-time results and feedback of workers from a specific location. As it is often desired to receive answers for a specific task as soon as possible, interactive systems require real-time results with low crowd latency. Although it is possible to reduce this latency by recruiting a retainer crowd that waits for the task, the use of mobile devices has the potential to further improve solutions concerning this requirement significantly.

Another issue is the warranty of quality. Crowd workers sometimes tend to rush through tasks to earn their fee as fast as possible. While it is desired to gather responses in seconds, the feedback quality should not be affected in a negative way.

As it is not desired to over-complicate this project, a few limitations that should not be part of the work have been defined. Also, a crowdsourcing application is only useful when providing an appropriate community of crowd workers. However, the process of growing such a user base and spreading the application will not be part of this thesis.

1.2 Specification of Functional Requirements

This section covers the specific functional requirements that have to be implemented in order to achieve the objective of the project prototype. A more detailed and in-depth description of the implementation and design can be found in section 4.

1.2.1 Task Creation The basic functionality of this framework consists of the creation of tasks with varying complexity. These tasks can contain different media types like simple texts or more complex media such as video or audio files. Also, there should be different answering options to configure. The possible answering options are plain text, single choice, multiple choice.

1.2.2 Task Distribution Aside from that it shall be possible to distribute the tasks among crowd workers in consideration of location and availability. In order to further improve the task assignment, other attributes such as age and gender can be specified to find appropriate workers.

1.2.3 Real-Time Results The platform should try to guarantee real-time results by utilizing technologies that lower crowd latency. To improve this aspect of the framework, it is required to support the recruitment of retainer crowds.

1.2.4 Score System While it is not part of the project to implement a way of paying crowd workers, there should be a simple score system that keeps track of the requesters' and workers' activities.

1.2.5 On Demand Availability Clients shall be able to use the functions of the platform on demand. This means that system should be implemented according to the "Software as a Service" (SaaS) delivery model. Operators of the platform shall be able to configure the platform to fit their needs. Users of the platform on the other hand, should be able to interact with the system via a thin client application.

1.2.6 Expandability As some features will only be implemented in a very basic way, the system shall be realized as a framework to provide the possibility of extending features easily. With this approach, other developers can build their own features using the developed prototype as a foundation.

1.3 Research Questions

Taking all the requirements presented in section 1.2 in consideration, following main research questions can be identified and will be answered as resulting outcome of this thesis.

1. How to design a mobile crowdsourcing framework capable of providing crowd feedback in real-time on demand?
   This question does not only refer to the used concepts and approaches, but also to the technologies and software design considerations.
2. What problems occur when trying to combine existing approaches to implement such a system?
   The implementation of such a system will most probably come with certain limitations and problems.
3. How to evaluate such a system in terms of quantity and quality?
   While quantity refers to the technical parameters of the system, the evaluation in terms of quality refers to the perceived user experience.

2 State of the Art

In this section existing approaches to solve problems in the context of crowdsourcing applications or to improve their performance are presented. Some of the referred strategies are implemented in the prototype developed for this project.

2.1 Crowd-Powered Interfaces

Typically user interfaces struggle to fulfill all the needs of the high-level tasks that they desire to support. For example, tools for writing a paper might support actions like creating a proper layout and finding spelling errors. However, there are no solutions for more complex tasks, which require an actual understanding of the written content, such as helping with writing decisions [4].

So-called crowd-powered interfaces take advantage of human intelligence to provide support for those complex tasks. A study presented five different systems that have been developed, all of them aiming to provide user interfaces by utilizing human computation and crowdsourcing concepts. The conclusion of their work suggests that those interfaces succeed in being more intuitive and efficient than other typical user interfaces [4].

2.2 Real-Time Crowd-Powered Interfaces

In order to provide an appropriate level of user experience, interactive systems need to respond to user input within seconds. In the case of crowdsourcing applications, the biggest time loss might be the crowd latency itself. Therefore it is required to implement techniques that lower crowd latency. There are two different approaches that aim to solve this problem [5]. A retainer model shall enable payment for crowd workers who simply wait for a task and then respond quickly. On the other hand, a technique called rapid refinement tries to find consistency in received answers to dynamically improve the search space to focus on promising answers.

2.2.1 Retainer Model The proposed retainer model is responsible for producing reliable crowds that deliver results with a significantly reduced response time. It does so by recruiting a certain amount of workers in advance and notifying them when a task is available. In order to justify the waiting time for a new task, workers are getting paid a small reward for participating in a retainer crowd. The study showed that this solution is able to guarantee a fast response time, while also maintaining scalability and still being useful even after a longer wait time [5].

2.2.2 Rapid Refinement This approach is a programming pattern specifically designed for reducing the actual work time. Even tasks of a limited complexity like simply answering questions can take quite a time, especially when expecting results of higher quality. The study suggests that low-latency crowds are synchronous crowds, meaning that the workers handle their tasks simultaneously. This is possible by using the previously mentioned retainer model, which basically ensures that the whole crowd is available shortly after starting the task. While the crowd is still searching for a final answer, the system automatically looks for emerging agreements within the search space and is able to narrow down the possible answer options to a single result. By calculating the likeliness

of the workers to agree, the task can be focused more and more on a smaller search space, guiding the workers to find a final answer [5].

2.3 Task Assignment in Mobile Crowdsourcing Systems

With the increased popularity of smart mobile devices it has become possible to build mobile crowdsourcing systems. Those systems can potentially target a much wider range of workers, as it is not required for a worker to be available and online at home from a desktop computer or notebook. A study on assigning tasks to workers by referring to their daily schedules addresses the dynamic nature of tasks and workers' availability [12]. Another aspect that comes with the advantages of mobile devices is the use of location data. Studies show examples of how to further improve task assignment with this additional improved source of information [22][29].

2.3.1 Worker-Task-Time Triples As workers are most likely to handle tasks in their spare time, many task assignments might be unanswered, because the worker simply was not available at the required time. The proposed solution aims to generate worker-task-time triples in order to increase the coverage of task assignments. The process starts by creating directed graphs containing all combinations of workers and tasks for each time period. Those graphs are then merged into a single "time-extended" graph, allowing the system to find the most optimized worker-task-time triples. The optimization is calculated by minimizing the total cost of the assignment, which is done by considering the time and distance of each combination [12].

2.3.2 Location Based Task Assignment Most mobile devices offer the possibility to track its position by using the Global Positioning System (GPS). The data provided by a device's location service can be used to improve task assignment in crowdsourcing systems.

A study presents the development of a "Location-based Relevant User determination System (LoRUS)", which has the capability of determining the most relevant mobile users in a specified spatial region. The motivation behind the creation of such a system is the fact that users operating on mobile devices do not have a static location. As those users' dynamic location can change quite frequently over time, their relevance for certain tasks can also change. LoRUS tries to determine the most relevant users, which are the ones that will most likely handle tasks related to a certain location [22].

Another study made use of location information by creating a location based forum and implementing an application that provides information about locations based on the data collected with the forum. With the help of Google Maps the platform can map locations to questions and automatically generate answers to questions which have been asked in a similar way before [29].

2.4 Worker Performance

Crowdsourcing systems not only rely on the intelligence, but also on the eagerness and accuracy of the mass in order to deliver high quality results. This is given by the fact that in an unsupervised system, workers could simply provide random results for their tasks, only aiming to finish their work as quickly as possible. The reliability of the results would suffer from the lack of diligence.

Another aspect is the crowd's eagerness to deliver results. If there is no reason for a worker to handle an assigned task, it will be hard to fulfill the application's requirement of delivering results in real-time.

Therefore, one big question is how to improve the performance of workers in the system.

2.4.1 Gamification One way of improving the quality of the results is by applying reward concepts known from gaming to the system. This process is known as gamification and has been proven to have a significant, positive impact on the participation in crowdsourcing applications [23].

A more practical example for applying gamification to a crowdsourcing system is the "SimplyCity" project. This mobile crowdsourcing platform was developed to help collecting local information for cities that intend to become a smart city. Most of the times, the first step of evolving into a smart city is making city related data publicly available. As many emerging cities do not have the capabilities of doing so, the "SimplyCity" prototype encourages citizens to help gathering the required data. On the one hand, the project introduced various types of task, depending on the requirement of the data to collect. These task types range from simple questionnaires to location tagging and taking pictures. The other problem was to encourage people to participate in the data collection process. This was achieved by setting up a competitive leaderboard for individuals as well as for whole cities. Contribution was rewarded by points that improve the ranking in each leaderboard [24].

2.4.2 Moral Reminder Reducing an abusive behavior of crowd workers and therefore improving the quality of the retrieved results can be achieved with quite simple methods. A study showed that a simple moral reminder is enough to significantly improve the performance of workers, especially on short surveys. The moral reminder was the signing of a short statement, in which participants basically committed to honesty before working on the task. The results proved that technique to be a very efficient improvement possibility, given the simplicity of the method [14].

3 Architecture

This section covers the design decisions in terms of a software architectural point of view. Following main topics occurred when thinking about a possible architectural solution:

- Structure for a distributed application:
  As it is necessary to provide the system's information to an arbitrary number of users, it makes sense to go for a distributed application structure. The two most commonly used approaches are the client-server model and peer-to-peer networking.
- Messaging:
  The data which is generated by users and utilized by the crowdsourcing platform needs to be exchanged between every interested participant in the system. Therefore it is needed to choose a reasonable messaging pattern which defines how data is shared between users and the system.
- Architectural pattern:
  There are plenty of architectural software patterns which can be used to structure an application logically. Those patterns do not aim to solve a very specific detailed problem, but address a broader aspect such as the basic organization of an application.
- Persistence:
  Some data needs to be stored persistently in order to be available for the platform at any time, even after restarting or reinstalling the system. Usually this is done via a database, which is collecting and storing the data in its own format. There exists a variety of different database types, with the most commonly used ones being relational and non-relational database systems.

3.1 Client-Server Model

The client-server model is an architectural structure for distributed applications that distinguishes between client and server components. Servers provide various services and functionalities and clients can request the usage of such a service. Clients and servers can communicate over a computer network with each other and do not share any resources. Because there can be an arbitrary number of clients, each one requesting a server's services at any given time, the server listens for incoming requests. Therefore it is the client's role to initiate a communication session. Figure 2 illustrates the concept of the client-server model.

There are multiple different reasons why the client-server model was preferred over a peer-to-peer approach for this project.

Management and maintenance Tasks will be created and managed by a variety of users, the same goes for answering tasks and sending results. All this resulting data, ranging from user information to actual tasks and results, must be stored and persisted. By choosing the client-server approach the whole system becomes much easier to manage and maintain, as there will be one single endpoint which

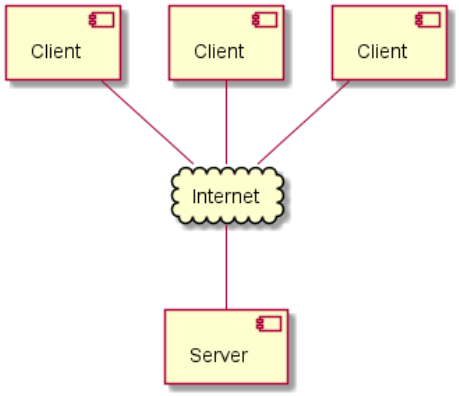

Fig. 2. Client-server architecture

is completely in control of an administrator. The data can be persisted in a database and is quickly obtainable for modifications and calculations. Even if errors occur, the central server can provide a log output to allow a faster problem analysis.

Performance and scalability Furthermore, the calculations done by the system might be more complex and therefore require more computational power. User devices, especially when talking about mobile devices, may suffer from performance problems when those expensive operations are calculated on them. When relocating this concern on a centralized hardware like a server, the user will not experience any performance-disturbing issues at all, which will lead to a much better user experience. The client software itself only has to deal with displaying the present data and triggering the server's services in order to manipulate the data. With a growing community and more data being stored and managed by the platform, the operations will become more costly. In terms of scalability the server's hardware can easily be upgraded, leading to more computational power. Also, the server's performance can be monitored by an administrator, deciding when to perform hardware upgrades or spotting bottlenecks in the software's performance.

Front end interchangeability Another advantage of the client-server model is the possibility of easily developing different client applications for varying devices. The client software only has to deal with displaying and visualizing the data. Depending on the customer's need, different client software solutions can be implemented, each utilizing a different subset of the provided services by the server.

Conclusion Overall the client-server architecture might come with a higher cost in terms of hardware and resources compared to a peer-to-peer solution, especially in the beginning phase of a project. But in the long run it definitely pays

off due to the fact that the whole system becomes way more manageable and maintainable.

3.2 Messaging

Messaging defines the way how two separate parts of the system communicate with each other. This section however does not only cover the choice of an appropriate message pattern, but also the way of how resources are accessed and manipulated.

In terms of message patterns, there exist following commonly used concepts [8]:

- Request-response: A service requestor sends a message to a service provider. The service provider returns a resulting response for the according request.
- Publish-subscribe: With this approach the sending application is not interested in the identity of the receiving applications. Interested parties subscribe themselves for receiving notifications and automatically get notified when the sender publishes a message to the communication infrastructure. Figure 6 illustrates this pattern.
- Fire-and-forget (either one-to-one or one-to-many): This one-way messaging approach is used when the sender should not be affected by the receiver, for example if the receiving party is not available.

3.2.1 Request-Response Message Pattern As described in section 3.1, the system architecture of choice for this project is the client-server model. The typical message exchange between a client and a server is done via the request-response message pattern. That means that the client usually sends a request to the server, which responds with data based on the outcome of the performed operation. This approach enables a two-way conversation between different endpoints using a single communication channel.

3.2.2 Representational State Transfer This architectural style for distributed hypermedia systems defines a number of constraints based on the Hypertext Transfer Protocol (HTTP). Following constraints are defined for the Representational State Transfer (REST) style [9]:

Client-Server The first constraint is the use of a client-server architecture. This brings the advantage of a separation in terms of concerns, splitting the user interface from the business logic and data persistence. As discussed in section 3.1, the improvements of maintenance, scalability and front end portability are important benefits of this constraint.

Statelessness Furthermore, the communication used for the interaction between a client and the server has to be stateless. In detail this means that any request sent by the client to the server needs to contain all information in order to be

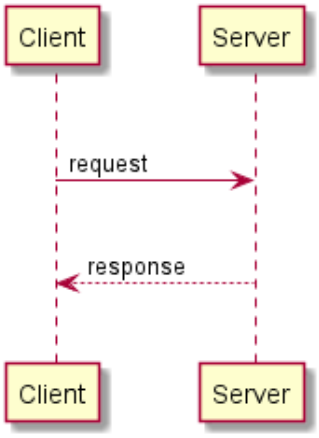

Fig. 3. Request-response pattern

processed correctly. There is no stored session data which gives context to any subsequently sent messages.

By keeping the communication stateless the system's scalability and reliability can be further improved, because there is no room for partial failures between requests and the back end can free resources after each transaction without losing data. Another benefit of the stateless nature is the fact that the workflow becomes more clear. Monitoring systems, for example, can determine the purpose of each message without analyzing further messages.

Cache This constraint aims to reduce the required load sent over a network. By giving response data the possibility of being cached on the client side, this information can be reused for later requests which are equivalent to the initial one.

Uniform Interface This design decision is considered as an essential part of any application implementing the REST architectural style. The uniform interface between components simplifies the whole application and further decouples the architecture. The key restrictions to achieve this feature are the following:

1. Resource identification in requests:
   A resource is uniquely identified by a uniform resource locator (URL) or uniform resource identifier (URI).
2. Manipulation of resources through representations:
   The response represents the resource identified by the URL or URI. The resource can be manipulated directly via the resource identifier.
3. Self-descriptive messages:
   Each message contains enough metadata to fully describe the message itself and how to process it.

4. Hypermedia as the engine of application state:
   Given the initial URL or URI for the REST application, all other states should be accessible and possible to discover. The system should provide links to all other available actions.

Layered System The scalability can be further improved by adding a layer constraint, which enables a hierarchically structured system. Components do not know to which layer they are actually connected and have no awareness of what is behind the layer they are interacting with, allowing intermediary layers to manage certain aspects such as load balancing or security.

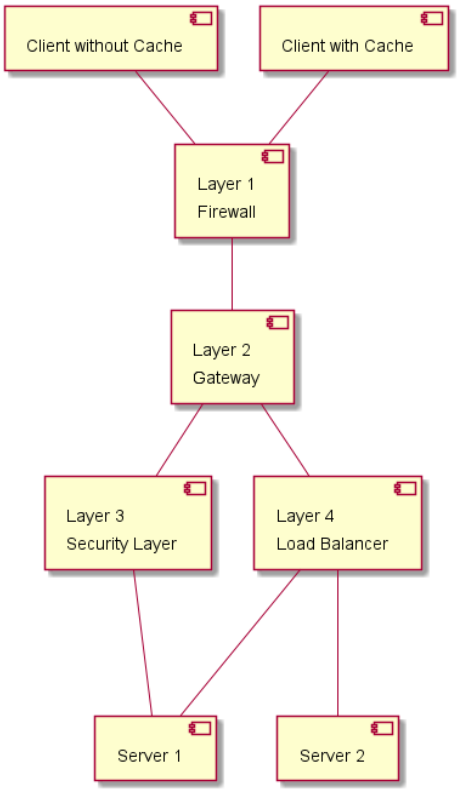

Fig. 4. REST concept example: Uniform-Layered-Client-Cache-Stateless-Server

HTTP method mapping When using HTTP for a RESTful interface, following basic HTTP methods are typically utilized for accessing and manipulating resources:

- GET:
  This method is used to retrieve a specific resource or a collection of resources. It should not change the state of the system and have no side-effects, meaning the GET method is a nullipotent method.
- PUT:
  This method replaces the according resource with the newly passed data. If no resource for the addressed identifier currently exists, a new resource will be added. The PUT method is idempotent, meaning that the state of the system will always be the same, regardless of how many times the same request is processed.
- POST:
  This method updates an existing resource.
- DELETE:
  This method simply deletes the addressed resource. Like the PUT method, the DELETE method is an idempotent operation.

Conclusion Because of all the above mentioned advantages of using the REST architectural style, such as improved maintenance, scalability, simplicity and client portability, the implementation for this project was done in a RESTful approach.

3.2.3 JavaScript Object Notation The widely used JavaScript Object Notation (JSON) is a lightweight data interchange text format, which also is independent of any programming language [16].

JSON uses a text representation which can be read easily by humans. In comparison with other data exchange formats like the Extensible Markup Language (XML), the JSON format offers a shorter and better human-readable representation. Figure 5 illustrates the schema of valid JSON structures.

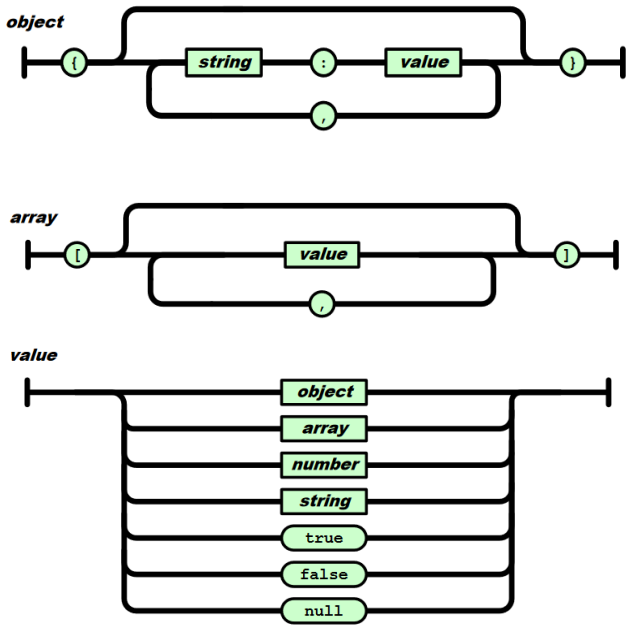

Fig. 5. Graphical representation of valid JSON structures [16]

Due to the discussed advantages, its great popularity and the resulting variety of supporting technologies, the JSON data format was used as serialization technology for the implementation of this project's prototype. The platform itself offers the possibility to be extended easily with other formats, but for the reference implementation only this approach was used.

3.2.4 Push Notifications Although the whole system's main messaging style is based on the request-response pattern, the crowd latency can be further reduced by sending notifications to the client devices. With this approach, the front end application does not have to consistently query the back end for changes or updates, but can simply wait to be notified about a new event. This is especially useful for mobile devices, which are the key to the low response time of the crowd.

The appropriate technology for this scenario is called push notifications, where a remote server publishes a notification to the client application. In order to make this happen, the client application usually has to subscribe itself for receiving notifications first. After registering at the back end with a unique identifier, the server can notify the client about a certain event by sending a notification via a previously negotiated protocol, such as HTTP.

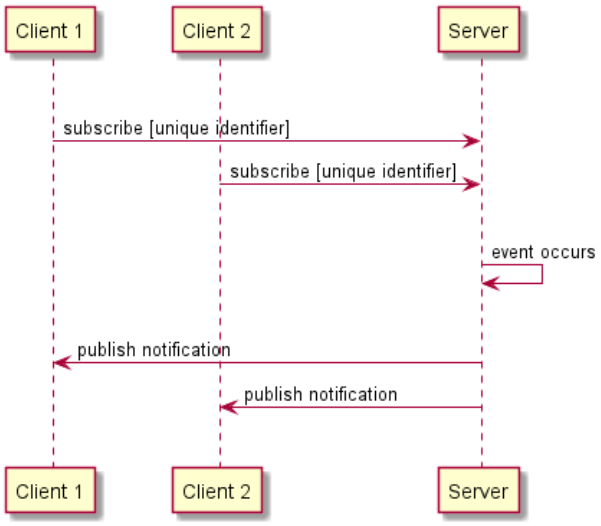

Fig. 6. Publish-subscribe pattern

3.3 Model-View-Controller Design Pattern

The idea of the Model-View-Controller (MVC) architectural software pattern is to split the user interface from the underlying data model. As the name suggests, it consists of three different aspects:

- Model
  The model aspect of this pattern represents the available data and information the system is based on. It also includes the logic for manipulating this data.
- View
  This aspect describes the way of how information is shown to the user. It is possible for data to have more than one view representing its information.
- Controller
  The controller aspect is used for processing user input and triggering specific logic for the model and view.

Figure 7 shows the intended interactions between the components of the MVC architectural pattern.

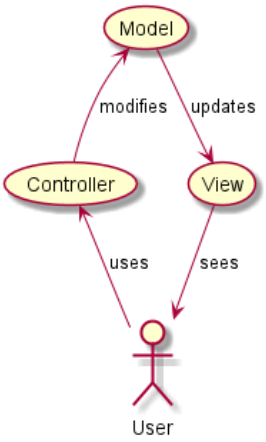

Fig. 7. Model-View-Controller pattern interactions

The description and advantages of using the Model-View-Controller pattern are described as follows:

> "Use of the MVC and PAC design pattern makes it easier to develop and maintain an application since:
> – the application's 'look' can be drastically changed without changing data structures and business logic.
> – the application can easily maintain different interfaces, such as multiple languages, or different sets of user permissions.
> Colloquially, the term 'MVC' has been extended to describe the way that large-scale changes to an application's Model are driven by a Controller that is responsible, but not for logic that changes an application's overall state in response to the event created by the user's interaction. In response to changes in the Model, the Controller initiates creation of the application's new View." [20]

So although the MVC pattern initially was developed for low-level user action and desktop applications, it still can be applied to web applications as well. In doing so, the three different aspects of the pattern are divided between the server and the client part of the system [20].

3.4 Persistence

Basically there are two different database approaches applicable to this project. The possible options are called relational and non-relational databases, with the well-known and widely used solutions MySQL and MongoDB respectively. MySQL is a typical relational database management system, storing data in tables and using the Structured Query Language (SQL) for accessing and managing the data. MongoDB on the other hand is a document-oriented database,

storing data as documents in the Binary JSON (BSON) format. Figure 8 displays the different data structure used for each database.

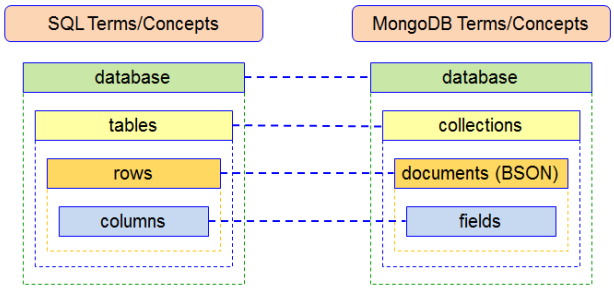

Fig. 8. Comparision of SQL and MongoDB data structures [21]

3.4.1 Comparison of MySQL and MongoDB Both MySQL and MongoDB have proven themselves to be useful and powerful database management systems, although the core differences between them are in the very basic approach. Each one of them is by now famous enough to offer a relatively easy setup and a wide range of support by different technologies.

When deciding which of the two approaches to use, it is necessary to take a look at their respective advantages and features.

MySQL As the data is stored in tables and their predefined rows, the SQL solution is a perfect fit for data with a known schema that fits in those tables and rows. Furthermore the relational approach supports atomic transactions, meaning that database operation are irreducible. Either all of of them are applied or no changes occur at all, preventing wrong partially database updates. The downside of supporting features like atomicity is that a lot of work has to be done behind the scenes, which reduces the scalability in terms of write operations, as many complex operations cannot be distributed to multiple machines.

MongoDB Here data is stored in documents, being an optimal solution for data sets with an unstable schema. By dropping the support for atomicity, MongoDB manages to reduce the time of tables being locked. Also, by using a technique called "sharding", MongoDB enables horizontal scaling for its database. "Sharding" refers to an architecture that partitions database entries by key ranges, allowing to distribute the data between multiple databases.

> "MongoDB provided lower execution times than MySQL in all four basic operations, which is essential when an application should provide support to thousands of users simultaneously.
> We can choose MongoDB instead of MySQL if the application is data intensive and stores many data and queries lots of data." [11]

Conclusion Although the non-relational approach seems to be a better fit for growing and big data sets, the traditional relational database solution with the MySQL implementation is used for this project. This decision was made because in the system the data schema is well-known beforehand and there will not be a huge amount of data in the prototype implementation. Also, the persistence layer is implemented in an easily exchangeable way, allowing the database to be switched at any time without too much effort.

3.4.2 Persistence Layer The persistence layer of the application is implemented in a way that offers the possibility of replacing the logic easily. With this approach, the interface connecting to an underlying database can be switched on the developer's demand without much trouble.

The reference implementation of the prototype uses a relational MySQL database. As the data model is written in an object-oriented programming language, it is incompatible to the data structure used by the database. To overcome this issue, a programming technique called Object-Relational Mapping (ORM) is used. Such tools can convert the complex objects used in the application's data model to the scalar values which are used in the database structure.

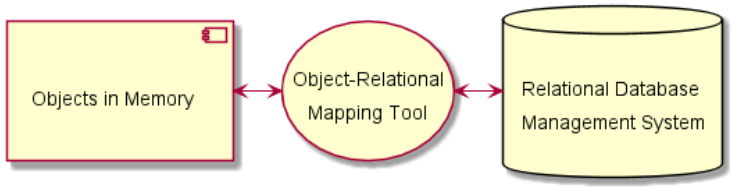

Fig. 9. Object-Relational Mapping as mapping tool between objects in memory and a relational database management system

3.5 Summary of Architectural Decisions

As a result of the previously described architectural considerations and decisions, the whole system was implemented as a distributed application, splitting the responsibilities between a server and clients. The clients connect to the back end over the internet and use request-response messaging to access and manipulate resources. The server is providing an appropriate interface in a RESTful manner. Especially useful for mobile devices, the client devices can register themselves on the server to subscribe for push notifications, allowing the platform to inform users about notable events.

The logical structure of the software follows a Model-View-Controller style, relieving client devices from performance-intense calculations and leaving only the representation of the view in their responsibility. Controller components process requests sent by clients and trigger actions on the model. The persistence is realized with a relational database management system — more precisely a

MySQL solution. For converting the objects in the application's memory to the relational data structure, an Object-Relational Mapping tool is used on the boundary between business logic and database.

A complete overview of the resulting system architecture is shown in figure 10.

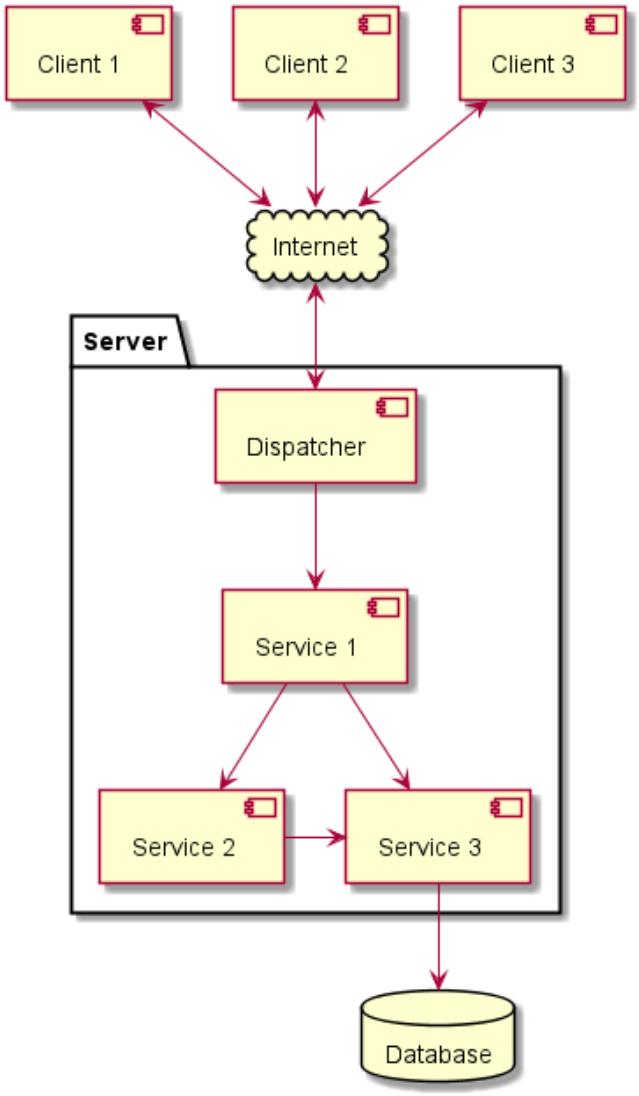

Fig. 10. System architecture overview

4 Design and Implementation

This section intends to cover the process of identifying the requirements of the crowdsourcing framework in terms of use cases and features and the resulting design decisions. The functional requirements mentioned in section 1.2 give a rough overview on the initial specifications and are refined and elaborated. Furthermore, the used technologies in the meaning of third party code components will be introduced and explained. Finally, a list of all implemented features is documenting the capabilities of the finished prototype software.

4.1 Use Cases

The activities of a user can be categorized in two different roles, being called "Requester" and "Worker". As a requester, the user can create tasks and use the platform to get results for those tasks. The worker role lets the user act as a crowd worker and as a part of the collective intelligence of the crowd. Workers get tasks automatically assigned by the system and can send results for their assigned tasks. The use cases can therefore be described for three different actors:

- User
- Requester
- Worker

4.1.1 User Role The "User" role applies for anyone using the application. Use cases which can be performed by all users at any time without any context are applicable to this role.

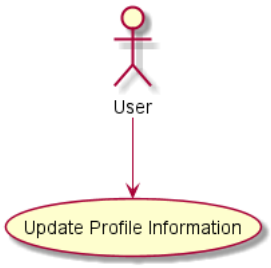

Fig. 11. Use cases for the actor "User"

There was only one use case identified:

- Update Profile Information:
  Users should be able to update their profile information at any point. This implies personal information such as their name, email address, gender or

birthday, as well as their availability. Furthermore, the information about participation in certain system functionalities should be changeable. In particular this refers to the option to participate as an active crowd worker or to be available for retainer crowd assignments.

4.1.2 Requester Role As a "Requester" users act to create and manage their tasks, which will eventually be worked on by the crowd. Figure 12 illustrates the identified use cases in two different diagrams, one showing general use cases and the other one focusing on the use cases related to task creation and management.

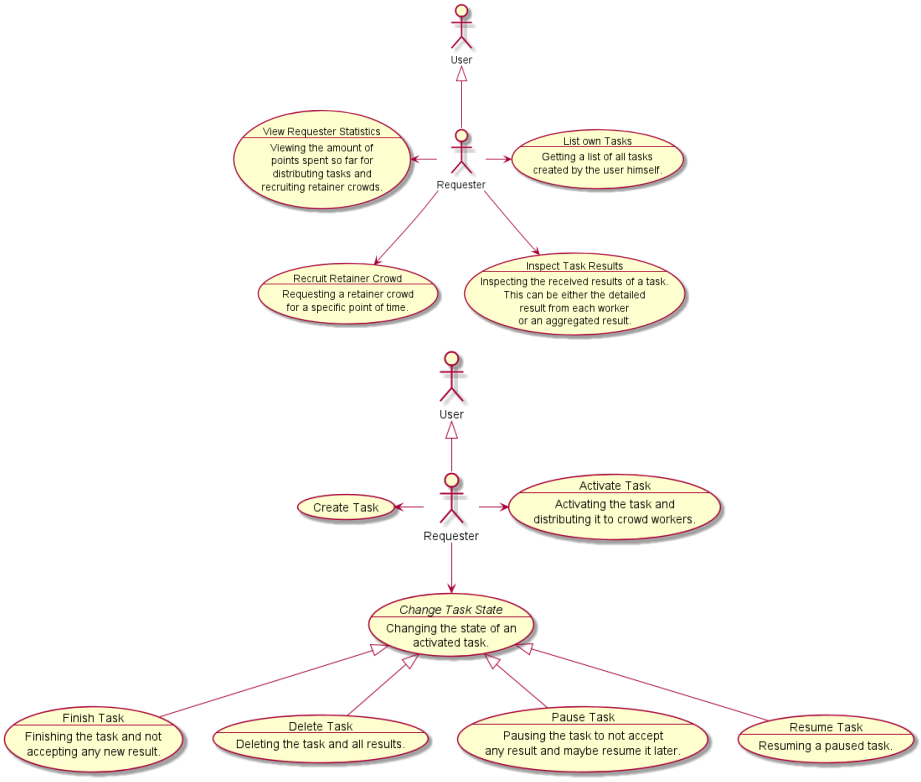

Fig. 12. Use cases for the actor "Requester"

Following use cases have been implemented in the platform:

– Create Task:
Creating tasks is considered one of the basic requirements for this project. Requesters are allowed to create tasks at will. Each task may consist of varying complexity, containing different media types such as a simple text,

video or audio. The number of questions for a task is also free to choose, with each question having different answering options to configure. Possible answering options can be an open text box with free text, single choice options or multiple choice options. Besides from the questions and answer options, there should be task settings to configure that specify the task assignment conditions. A requester might want to filter the crowd workers for a task according to special criteria like nationality or age.

- Activate Task:
  Once a task has been created, its state shall be changeable by the initial task author. Activating the task should trigger an automatic distribution to crowd workers. Once a task is active, results can be received for it.
- Finish Task:
  Finishing the task should irreversibly end the task, allowing no more results to be submitted. An ended task cannot be activated again.
- Delete Task:
  Deleting a task will completely remove the task and all received results from the system with no option to restore the information.
- Pause Task:
  Pausing a task should put the task in a state where no more results can be submitted. A paused task however can still be resumed later.
- Resume Task:
  Resuming a paused task will make it active again, allowing results to be submitted again.
- View Requester Statistics:
  The requester should have an overview on the points he has spent on getting task results from the crowd. Also, some more sophisticated features such as recruiting a retainer crowd might cost points. The statistics should at least provide information about the spent points and the amount of created tasks, with a possible extension to more data.
- List own Tasks:
  Any created task has to be stored and available to be listed for the initial creator. The task list should at least show the title of each task and the creation date, with user interaction options to access or manipulate each task.
- Inspect Task Result:
  Naturally the requester wants to see the received result for each of his tasks. The results should be presented in two different ways, an aggregated result to clearly display the general opinion of all workers and a detailed view which shows each single result on its own.
- Recruit Retainer Crowd:
  The possibility to recruit a retainer crowd is useful for reserving crowd workers for a specific point of time. Using this technique, the crowd latency can potentially be lowered. Recruiting a retainer crowd however should require the requester to spend score points. Otherwise, a requester could always reserve an arbitrary amount of workers without any cost.

4.1.3 Worker Role In the role of a "Worker" users participate as crowd workers and act as part of the intelligence of the crowd. Workers get tasks automatically assigned by the platform and can work on them.

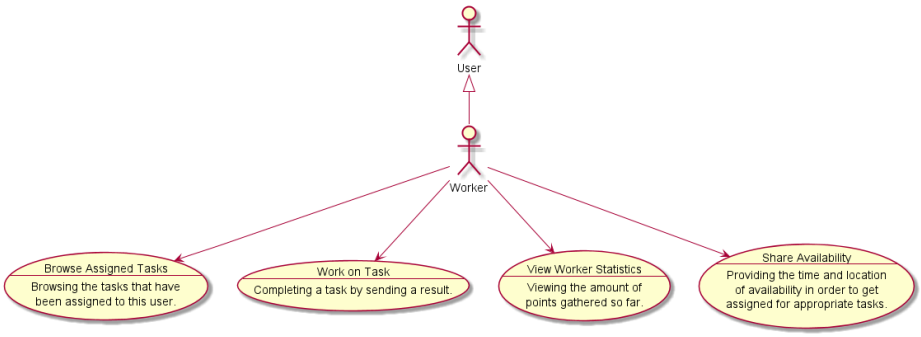

Fig. 13. Use cases for the actor "Worker"

The identified use cases for the worker role include following:

- Browse Assigned Tasks:
  A worker should be able to see the tasks which have been assigned to him. The overview of assigned tasks should at least provide information about the task title and the reward for completing this task. Additionally, the user should have the options to either accept or decline the assigned task.
- Work on Task:
  Submitting results for created tasks is another very basic requirement for the crowdsourcing framework. In order to submit results, workers have to be able to work on a task, meaning that they can access assigned tasks, answer the questions as defined by the task author and finally send the result.
- View Worker Statistics:
  The worker should have the possibility to monitor their earned points for completing tasks. The statistics view should at least provide an overview about the score, the completed tasks and the declined tasks.
- Share Availability:
  In order to enable a better task assignment for the system and to increase a worker's chance of getting tasks assigned, a worker should be able to share his availability with the platform. By providing the time and location of availability the system is able to identify appropriate workers for task assignments and retainer crowds more easily.

4.2 Technologies

There are many third party frameworks, libraries and tools available that help building the foundation of this project's software or support the development of

a certain feature. In this section the used technologies are introduced, including the motivation of using them.

4.2.1 Spring Framework The Spring framework is an open-source Java framework developed by Pivotal Software. It provides infrastructure support for Java applications and handles infrastructure related concerns, so that the developer only has to deal with his application. Furthermore, Spring enables the development of software even of enterprise application size using "plain old Java objects" (POJOs) [15]. By using only POJOs it is not needed to run the application on an application server that manages Enterprise JavaBeans (EJB) — running a simple robust servlet container is sufficient.

Another central feature of the framework is its inversion of control (IoC) container. Inversion of control, also called "Hollywood's Law" [28], is a design principle that inverts the control flow of a program:

> "The framework dictates the architecture of your application. It will define the overall structure, its partitioning into classes and objects, the key responsibilities thereof, how the classes and objects collaborate, and the thread of control. A framework predefines these design parameters so that you, the application designer/implementer, can concentrate on the specifics of your application. The framework captures the design decisions that are common to its application domain. Frameworks thus emphasize design reuse over code reuse, though a framework will usually include concrete subclasses you can put to work immediately.
> Reuse on this level leads to an inversion of control between the application and the software on which it's based." [10]

The Spring framework implements the inversion of control principle as dependency injection (DI). In doing so, objects define their dependencies themselves, for example by using constructor parameters. The container then automatically injects the required dependencies when creating the object. Spring calls such objects that are managed by the inversion of control container "beans". Beans build the foundation of the application.

The modular organization of the Spring framework allows to only use the needed parts, making it a lightweight framework with a broad range of services. Following modules have been utilized developing the crowdsourcing framework:

- Web Model-View-Controller framework
- Transaction management
- Object Relational Mapping Data Access
- Security

Web Model-View-Controller framework This module offers an MVC architecture and various components to develop a flexible web application while also supporting RESTful web services:

"The Spring Web model-view-controller (MVC) framework is designed around a `DispatcherServlet` that dispatches requests to handlers, with configurable handler mappings, view resolution, locale, time zone and theme resolution as well as support for uploading files. The default handler is based on the `@Controller` and `@RequestMapping` annotations, offering a wide range of flexible handling methods.
With the introduction of Spring 3.0, the `@Controller` mechanism also allows you to create RESTful Web sites and applications, through the `@PathVariable` annotation and other features." [15]

Those features make it a perfect fit for the needs of the crowdsourcing platform and the desired architecture described in section 3.

Figure 14 graphically shows the life cycle of a request being handled in the Spring Web MVC framework.

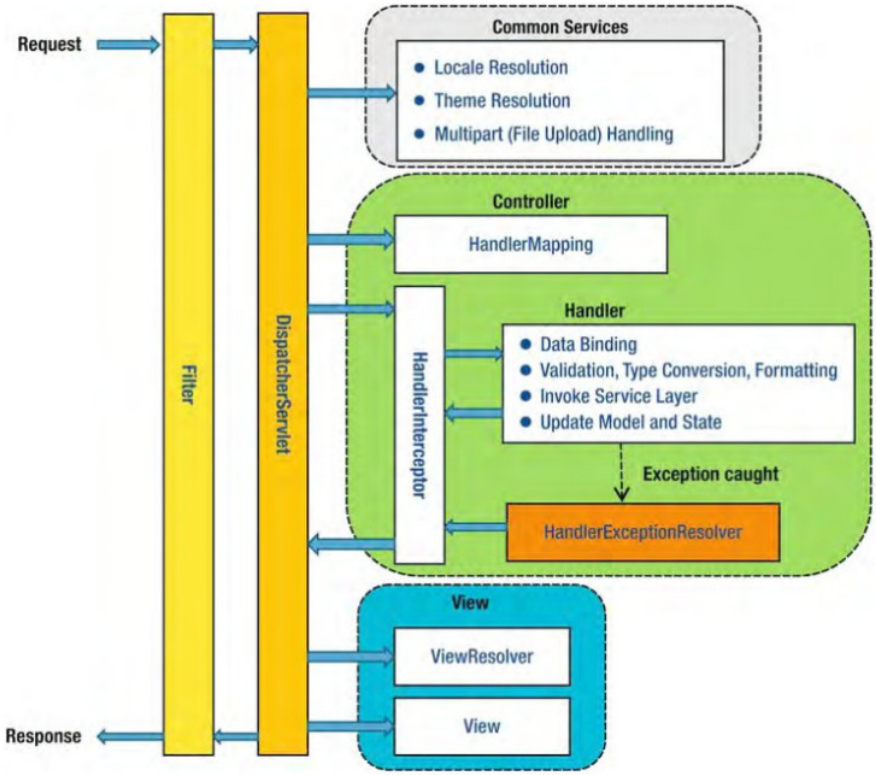

Fig. 14. Request life cycle in the Spring Web MVC framework [25]

The Spring framework's view resolution is designed to be really flexible, as it allows view configuration through content negotiation via the `Accept` header of

HTTP requests, file extensions and many more options. The model, implemented as a `Map` interface, will be transformed into the required format, which is JSON in the case of this project's prototype.

The crowdsourcing framework project uses the Spring Web MVC framework of version `4.3.1.RELEASE` of the Spring framework.

Transaction management A database transaction is defined as a sequence of operations that affect the database. Each of those operations is considered as a single unit of work and should either be applied to the database altogether or not at all. This is necessary to assure data integrity and a consistent data set.

Spring's transaction management module supports a generalized and simple interface for a variety of different transaction management application programming interfaces (APIs), for example Java Transaction API (JTA), Java Database Connectivity (JDBC), Java Persistence API (JPA), Java Data Objects (JDO) or Hibernate. Furthermore the module provides great integration with Spring's very own data access abstractions.

The prototype implementation uses Spring's transaction management module of version `4.3.1.RELEASE` of the Spring framework.

Object Relational Mapping Data Access The Spring framework comes with support for various data access technologies, including first-class support for Hibernate integration. Via dependency injection all supported features for ORM tools can be configured quite easily. There are plenty of good reasons to use the ORM data access module [15]:

- General resource management:
  Hibernate's `SessionFactory` instances are configured and managed by the Spring application context, providing an easy, efficient and safe resource handling.
- Integrated transaction management:
  By using the `@Transactional` annotation the semantics of a transaction can be declared, enabling an automatic management of exception handling for rollbacks and other transaction specific concerns.
- Common data access exceptions:
  With different transaction management APIs come a variety of custom exceptions. The Spring framework converts them to a common exception hierarchy, allowing the developer to handle exceptions in a much easier and generalized way.
- Easier testing:
  With Spring's inversion of control container and dependency injection the different implementations of persistence related classes can be changed quite easily, simplifying unit testing of the persistence code.

As Hibernate was chosen as ORM tool for this project, the built-in support from the Spring framework is a convenient feature.

In particular, Spring's `HibernateTransactionManager` class was used for integrating Spring's data access abstraction with Hibernate.

Spring's ORM data access module of version `4.3.1.RELEASE` of the Spring framework was used for the development.

Security Spring Security is a Spring sub-project providing security features like authentication and authorization for applications. The framework is considered the recommended standard for integrating security into Spring-based applications. It supports solutions for the two major security aspects "authentication" and "authorization" [2]:

- Authentication:
  The sub-project provides strategies for different authentication models and integration with a wide set of technologies, including form-based authentication, the Lightweight Directory Access Protocol (LDAP) and OpenID.
- Authorization: Spring Security supports three main authorization concerns, which are authorizing web requests, method invocation and access to individual domain object instances.

Version `4.1.3.RELEASE` of the Spring Security project was used for developing this prototype.

4.2.2 Hibernate The Hibernate framework developed by Red Hat provides a set of data related solutions for Java applications. The particular modules used for developing the crowdsourcing back end are Hibernate ORM, an object-relational mapping framework and Hibernate Validator, a bean validation API.

Hibernate ORM The object-relational mapping tool provides data persistence in relational databases while supporting object-oriented concepts like inheritance, polymorphism, association and composition. Furthermore, it has built-in support for the Java collections framework. The strengths of the framework include high performance, scalability, reliability and extensibility.

Hibernate Validator The bean validation API offers generalized, annotation-based constraints to validate domain model objects. Metadata annotations such as `@NotNull`, `@NotEmpty` or `@Size` can be used to automatically validate objects received via client requests on the back end side. Besides that the built-in constraints can easily be extended by writing custom constraints.

The implementation uses version `5.2.2.Final` of the Hibernate ORM framework and version `5.3.0.Final` of the Hibernate Validator.

4.2.3 Jackson The Jackson project is a collection of Java data-processing tools, primarily known for its high-performance JSON parsing processor. It also supports a variety of other data formats such as CSV, Protobuf, XML, YAML and more. It is developed by FasterXML and aims to be a fast and lightweight library. [26]

For this project, version `2.8.3` of the Jackson databind core module (`https://github.com/FasterXML/jackson-databind`) is used for serializing and deserializing between the JSON data format and Java objects.

The support for the Java Specification Request 310 (JSR 310), which essentially is the Java 8 date and time API, is enabled by importing version `2.8.3` of the module `https://github.com/FasterXML/jackson-datatype-jsr310`.

4.2.4 opencsv The opencsv library is an open-source CSV parser developed by Glen Smith. It provides a simple interface and supports useful configuration options such as custom quotation and separation characters. Furthermore, there is also the possibility to bind data values directly to bean fields via annotations [27].

The prototype implementation uses the opencsv library to parse and validate the uploaded CSV files, which can contain reusable data values and predefined question definitions.

A comparison of CSV parsers for Java shows that the opencsv library is considerably slower than other solutions for a large amount of records [7]. As the use cases of the crowdsourcing framework only cover a relatively rare usage of CSV files with a low number of lines, the performance impact is not that important. In favor of API simplicity and easy configuration the implementation uses version `3.9` of the opencsv parser.

4.2.5 Firebase The Firebase platform is a mobile and web application development platform by Firebase Inc., a subsidiary company of Google. It provides a wide range of different services, covering analytics, stability and performance monitoring, mobile advertising, user base growth and development features like authentication, data storage, hosting and cloud messaging [1].

In order to broadcast real-time notifications to mobile clients, the crowdsourcing framework makes use of the Firebase Cloud Messaging (FCM) service. The service is free to use and aims to reliably deliver messages to cross-platform clients. It is also capable of delivering messages in three different ways:

- Target single devices
- Target groups of devices
- Target devices subscribed to a specific topic

FCM differentiates between two types of messages, which are called notification and data messages. Notification messages are only handled by the client application if the app is currently active. If the app is in the background, an automatically generated notification is displayed. However, FCM traditionally is used with data messages, which are handled directly by the client app, regardless whether it currently is in the foreground or background.

The prototype uses version `5.3.0` of the Google Firebase software development kit (SDK) for delivering push notifications to mobile clients.

4.3 System Overview

The system consists of multiple features that work on domain objects. A domain object is a logical entity that describes an aspect of the domain and reflects a real-world concept.

Each feature has its own set of operations that can be triggered by requests from external clients or other internal services. The handlers for those requests are so-called controllers and map an HTTP request targeting a certain URL to a service operation. Domain objects may also be persisted in the database using repositories, which provide an interface to store the data model in a database specific way.

Figure 15 shows the concept of controllers and services. Each controller can have access to other services, but not to other controllers. Services can access each other, but also do not have access to controllers. Furthermore, services can have access to repositories to load and save domain objects.

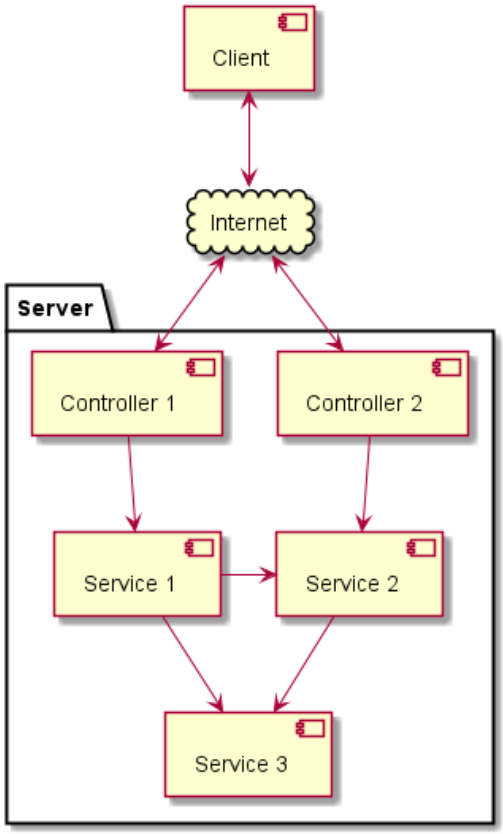

Fig. 15. Concept of controllers and services

4.3.1 Messaging Contract As described in detail in section 3.2.3, JSON is the preferred data format for exchanging information between server and clients.

All the available properties for each request are described in section 4.4 as well as in the documentation pages of the crowdsourcing framework, which can be .

For example, following request data might be sent to create a user in the system:

```
{
    "username": "john123",
    "password": "password",
    "firstName": "John",
    "lastName": "Doe",
    "email": "john.doe@email.com",
    "workerSettings": {
        "activeWorker": true,
        "retainerCrowdParticipant": true,
        "gender": "MALE",
        "birthday": "1992-07-31",
        "languages": [
            "en", "de"
        ],
        "location": {
            "country": "Austria",
            "city": "Vienna"
        }
    }
}
```

For each request, the system will answer with a response containing at least a `success` flag indicating the successful handling of the request. Furthermore, there might be additional information after processing the request successfully. In an error case, there will be an `error` field with a short textual description of the error.

The following data might be sent as a response to the previous request for creating a user:

```
{
    "success": false,
    "error": "Username already exists"
}
```

4.3.2 Configuration The framework allows the configuration of certain parameters at system startup. Those configuration options are called deployment properties, because they can only be defined at the initial deployment of the application. Changing their values during run-time is neither supported nor advised, as most of them have a high technical or semantic impact on the platform.

There are two different files that specify the available deployment properties, both located in the `WEB-INF\classes` directory of the deployed web application:

– `application.properties`
– `database.properties`

The `application.properties` file is used to define application specific settings, while the `database.properties` file allows the specification of technical settings regarding the connection to the database.

The database deployment properties consist of following settings:

– `jdbc.driverClassName`
  The Java Database Connectivity (JDBC) interface needs the fully qualified class name of the driver that will be used. For a typical MySQL database this usually is set to `com.mysql.jdbc.Driver`.
– `jdbc.url`
  The URL that is used to connect to the database.
– `jdbc.username`
  The username used to authenticate to the database.
– `jdbc.password`
  The password used to authenticate to the database.
– `hibernate.dialect`
  This property defines the fully qualified class name of the dialect that should be used. Hibernate needs this information to generate the correct database queries.
– `hibernate.show_sql`
  This flag defines whether SQL queries should be printed to the console output.
– `hibernate.format_sql`
  This flag defines whether the printed SQL queries should be formatted automatically for better readability.
– `hibernate.hbm2ddl.auto`
  This property sets the mode that is used for automatically validating or exporting schema DDL to the database when the `SessionFactory` is created. Following modes are supported:
  – `validate`: Validates the schema without changing the database.
  – `update`: Updates the schema.
  – `create`: Creates the schema, removing previously stored data.
  – `create-drop`: Drops the schema when the `SessionFactory` is closed explicitly, for example when the application is shut down.

The following lists contain descriptions of all available application deployment properties. Most properties have a close relation to certain features, which are explained in depth in the feature description of section 4.4.

General configuration properties:

– `retainer.crowd.cost.per.minute.and.user`
  This integer property defines the amount of points that need to be paid to each user per minute for participating in a retainer crowd. The default is 10, which sums up to a cost of 600 points for an one hour retainer period per user.

– `cloud.messaging.token`
  The authorization token used by Google's FCM service for sending push notifications.

Logging properties:

– `logging.enabled`
  This flag can be used to configure whether logging should be enabled. Logging messages are print to the standard output stream and additionally written to a specified log file.
– `logging.level`
  This property can be used to set the granularity of logging messages. The default value is `DEBUG`, which enables all logging information. Other possible values to set are `INFO`, `WARN` and `ERROR`.
– `logging.file.folder`
  Setting this value defines the directory to which the log file should be written. The server running this application must have writing permissions to this directory.
– `logging.file.name`
  This property defines the name of the logging file. If both logging folder and logging file name are not configured, no logging information will be written to any file.

File upload properties:

– `file.upload.folder.data`
  The upload folder for regular data files, which are restricted to CSV files in the prototype implementation.
– `csv.separator.character`
  This property allows the definition of a custom CSV separator character. The default character is `,`.
– `csv.quote.character`
  This property allows the definition of a custom CSV quote character. The default character is `"`.

Scheduled tasks properties:

– `task.assignment.check.interval.millis`
  The interval to perform a validation of all assignments in milliseconds. If open assignments, which are in the state `ASSIGNED` or `ACCEPTED`, belong to a finished task, the assignments will change to `EXPIRED` or `MISSED`.
– `task.time.check.interval.millis`
  The interval to perform a validation of start and end time of tasks in milliseconds. If a task's start or end time is reached, the task will automatically be started or finished.
– `result.check.interval.millis`
  The interval to perform a validation of all results in milliseconds. If results belong to a task that no longer exists, the results will be deleted.

- `distribution.check.interval.millis`
  The interval to perform a validation of all task distributions in milliseconds. If task distributions belong to a task that no longer exists, the distribution information will be deleted.
- `availabilities.check.interval.millis`
  The interval to perform a validation of every user's availability information in milliseconds. If an availability has an end time that is already in the past, the availability will be deleted.
- `retainer.crowds.check.interval.millis`
  The interval to perform a validation of all retainer crowds in milliseconds. If a retainer crowd has an end time that is already in the past, the crowd will be deleted and all according points will be assigned.

4.4 Framework Features

This section dives a little bit deeper into the technical implementation details of the prototype. The implemented features and realized concepts are described and documented thoroughly, providing a complete overview on the whole software. Some of the used concepts are inspired by state-of-the-art approaches mentioned in section 2.

The set of features implemented in this prototype can be differentiated by considering the domain object that will be targeted primarily by the client action. For the following list of features, each implemented domain object is described separately with all the available operations on it.

4.4.1 Users Any client interacting with the platform in an authenticated way is required to register itself before. With a registration a user is created, holding the login data and some personal information for an optimized task assignment. A user represents both aspects of interacting with the crowdsourcing platform, the requester part as well as the worker part. While participating as a requester requires someone to actively create a task, the worker role is partially passive, because the system decides which worker will get a task assigned. Therefore each user also has settings that inform the system whether they want to participate as a worker.

Creating a user, which basically is a registration in the system, can be done by sending a POST request to `/register`. The JSON content of the request consists of properties defined in table 1. If the user creation succeeds, a successful response with the generated user ID is returned.

```
{
    "success": true,
    "userId": "320600c4-dada-445f-bf26-f556aba76b70"
}
```

Table 1. Properties of a request for creating a user

Property name	Required	Description
username	yes	The username which will be used for login. Must be unique in the system and therefore not already in use.
password	yes	The password for this user.
firstName	yes	The first name of the user.
lastName	yes	The last name of the user.
email	no	The email address of the user.
userRights	no	The user rights. Must be USER or ADMIN. This property can only be set if the sender of this request is an authenticated admin user.
workerSettings	no	The settings for the worker role as defined in table 2.

Table 2. Properties of worker settings

Property name	Required	Description
activeWorker	yes	A flag that indicates whether this user wants to get tasks assigned and act as a worker.
retainerCrowdParticipant	yes	A flag indicating whether this user wants to get assigned to retainer crowds, if his availability information is fitting.
languages	yes	The list of spoken languages of the user. Must be in form of IETF BCP 47 language tags.
gender	no	The gender of the user, must be MALE or FEMALE.
birthday	no	The birthday in the format yyyy-MM-dd.
location	no	The usual location of the user as defined in table 3.
availabilities	no	The list of availability information for the user as defined in table 4.

Table 3. Properties of a location

Property name	Required	Description
country	yes	The country of this location.
city	no	The city of this location.

Table 4. Properties of an availability

Property name	Required	Description
startTime	yes	The start time of this availability in the format yyyy-MM-dd HH:mm.
endTime	yes	The end time of this availability in the format yyyy-MM-dd HH:mm.
location	no	The location during this time period as defined in table 3.

In order to receive a complete list of all users, clients can send a GET request to `/user`. It is possible to retrieve information about a specific user by sending a GET request to `/user/{userId}` or to `/user/username/{username}`. If no information can be found for the given user ID or username, `null` is returned.

A client can update specific user information by sending requests to the corresponding URL. There are two different update options, one for changing user information and one for changing the user password. In order to change a user, the requesting client must either be authenticated as the user to change or possess admin privileges. If the user updated was performed successfully, a success response containing the user ID is returned. Updating the password of a user is supported by sending a PUT request to `/user/{userId}/password`. The JSON content of the request simply has to be the new password as a value. The other user information can be updated by sending a PUT request to `/user/{userId}`. The JSON data of the request supports the properties listed in table 5. Properties that are not defined or set to `null` are not affected by the update operation.

Table 5. Properties of a request for updating a user

Property name	Required	Description
firstName	no	The first name of the user.
lastName	no	The last name of the user.
email	no	The email address of the user.
gender	no	The gender of the user, must be MALE or FEMALE.
birthday	no	The birthday in the format yyyy-MM-dd.
location	no	The usual location of the user as defined in table 3.
languages	no	The list of spoken languages of the user. Must be in form of IETF BCP 47 language tags.
activeWorker	no	A flag that indicates whether this user wants to get tasks assigned and act as a worker.
retainerCrowdParticipant	no	A flag indicating whether this user wants to get assigned to retainer crowds, if his availability information is fitting.

Users can update their availability information by sending a POST request to `/user/{userId}/availability`. The JSON content needs to be an availability object as defined in table 4. By sending a DELETE request to the very same URL, all availability information for this user will be deleted.

If a client desires to delete a user, a DELETE request to `/user/{userId}` has to be sent. The requesting client must be either authenticated as the user to delete or possess admin privileges.

4.4.2 Tasks The primary feature of the crowdsourcing platform is the possibility to create tasks and work on them. A task represents a set of questions with different types of answers and answering options. Furthermore, tasks and questions can also have a variety of media attached, such as images, audio and video files. Each task also contains configurable assignment conditions, which can be used to specify conditions that must be fulfilled by workers. The system will automatically assign tasks only to workers that match the specified requirements.

Any user can create a new task by sending a POST request to `/task`. The JSON content of the request must represent a valid task, consisting of the properties defined in table 6. If the task creation succeeds, a success response with the automatically generated task ID is returned.

```
{
    "success": true,
    "taskId": "320600c4-dada-445f-bf26-f556aba76b70"
}
```

Another way of creating new tasks is by cloning an existing one by sending a POST request to `/task/{taskId}/clone`. The JSON data only consists of one optional property, which is called `dataId` and defines the ID of a dynamic task data. The newly created task has exactly the same settings as the original task with following exceptions:

- The new task will have a new randomly generated task ID.
- The task state is set to the initial CREATED state of new tasks.
- The ID of the associated dynamic task data might be overwritten, if a new one is sent with the request.

A task's lifetime is represented by four different states. Figure 16 shows the task state machine and the capabilities of each state. A newly created task always starts in the CREATED state. This state represents the preparation of a task, which is not yet distributed to workers. The task creator can activate a task in this state and trigger the transition to the ACTIVE state. With this transition, the system automatically determines optimal task assignments for workers and distributes the task to them. As soon as a task is in this state, assigned workers can submit their results. The task creator can always pause an active task and set it to the PAUSED state. A task in this state cannot accept new results until it is resumed again. At any point, the creator of the task can end the task and transition it to the FINISHED state. This transition is irreversible, meaning that a finished task will always stay finished, incapable of receiving new results.

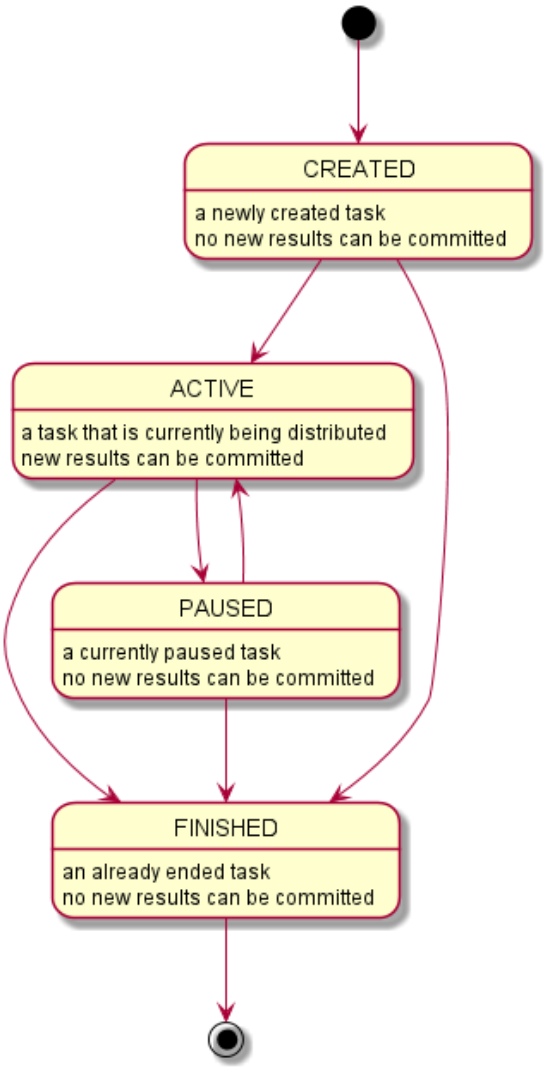

Fig. 16. State machine of tasks

Table 6. Properties of a request for creating a task

Property name	Required	Description
taskSettings	yes	The task settings represent a general configuration of the task. They consist of properties as defined in table 8.
questions	yes	A list of defined questions for this task. Each question consists of properties as defined in table 7.
dataId	no	The ID of the dynamic task data. This feature is explained in section 4.4.3.

Table 7. Properties of a question

Property name	Required	Description
questionType	yes	The type of this question. Supported values are: – SINGLE_CHOICE – MULTIPLE_CHOICE – DROPDOWN – OPEN
questionText	yes	The text that contains the actual question.
answerRequired	yes	A flag that specifies whether this question must be answered by the worker.
answerOptions	yes	A list of possible answer options. This property is not taken into consideration if the question is of type OPEN.
media	no	An optional list of media resources attached to the task description. The structure of a media resource is defined in table 10.

Table 8. Properties of task settings

Property name	Required	Description
title	yes	The main title of this task.
description	yes	A more detailed description of this task.
media	no	An optional list of media resources attached to the task description. The structure of a media resource is defined in table 10.
reward	no	The reward points that should be paid to a worker for completing this task. The range is from 0 to 1000 with a default value of 0.
maximumWorkers	no	The maximum amount of workers that this task will be assigned to. The default value is 3, which is also the minimum of this property.
startTime	no	The planned start time of this task in the format yyyy-MM-dd HH:mm:ss. If this property is set, the system will automatically activate the task at this point of time and trigger the task assignment.
endTime	no	The planned end time of this task in the format yyyy-MM-dd HH:mm:ss. If this property is set, the system will automatically finish and end the task.
assignmentConditions	no	The assignment conditions used for defining conditions for a more sophisticated task assignment. The structure of this property is documented in table 9.

Table 9. Properties of assignment conditions

Property name	Required	Description
requiredLanguage	no	The spoken languages of the worker. Must be in form of IETF BCP 47 language tags.
requiredMinimumAge	no	The minimum age of the worker.
requiredMaximumAge	no	The maximum age of the worker.
requiredGender	no	The gender of the worker. Supported values are: – MALE – FEMALE
requiredLocation	no	The location of the worker as described in table 3. The location specified in an availability of a worker is always prioritized over his general location.

Table 10. Properties of a media resource

Property name	Required	Description
url	yes	The URL of the media file.
description	no	An additional description of this media resource.
mediaType	no	A property used to help clients handling the media by providing its type. Supported values are: – IMAGE – VIDEO – AUDIO – UNKNOWN The default value is UNKNOWN.

In order to receive information about a certain task, clients can send a GET request to `/task{taskId}`. The platform responds with the very same information as defined in table 6 for creating tasks. Additionally, the properties `requesterId` with the identifier of the task creator and `taskState` with the current state of the task are provided.

Clients can also send a GET request to `/user/{userId}/assignments`, which is responded with a list of all tasks assigned to this user. To receive all tasks that have ever been distributed to a worker, even the already finished ones, clients can send a GET request to `/user/{userId}/assignments/all`. The requesting client must either be the worker to whom the assignments belong of possess admin privileges.

Changing the state of a task can be done by sending a PUT request to `/task/{taskId}/state`. The JSON content has to be a simple value representing the desired new state. In order to successfully change the state of a task, the action must be either requested by the initial creator of the task or an user with admin privileges. The state change from CREATED to ACTIVE triggers the automatic task assignment by the system and distributes tasks to a number of suitable workers, which is actually higher than the specified maximum amount of workers in the task settings. This is done by the system to increase the chance of receiving results as quickly as possible. For this reason, the platform monitors the percentage of quickly accepted task assignments and uses the information to calculate the amount of workers that will get a chance to handle the task. For example, if 80% of all tasks are accepted within 10 minutes and a user requested 100 results for a task, the system will distribute the task to `100 / 80% = 125` workers. To prevent flooding the system with task distributions, there is a defined limit of 60%. By assigning more workers to a task, this mechanism compensates the lack of motivated and reliable workers in an environment. If the performance of workers increases, it will automatically perform fewer task distributions in the future. Lastly, changing the state of a task to FINISHED ends the task irreversibly. All supported task states and state transitions are shown in figure 16.

Clients can delete tasks by sending a DELETE request to `/task/{taskId}`. The requesting client must be the creator of the task or possess admin privileges.

4.4.3 Dynamic Task Data Tasks can be customized by adding dynamic task data via the `dataId` property during task creation. The property must be set to a valid identifier of a previously created dataset. Such a dataset consists of a name for identification purposes and a list of values which can act as exchangeable parameters for the task description, question texts and answer options. Table 11 lists the properties of such a dataset. In combination with the task cloning feature described in the previous paragraph, the dynamic task data can be used to easily recreate similar tasks with different parameters.

In order to reference a data value in a text, it is required to insert a certain token, which also holds the index of the referenced value. Such tokens are surrounded by square brackets and start with the keyword data, followed by a colon

and the index. For example, referencing the first value can be done by adding `[data:0]` to the text.

Table 11. Properties of a dataset for dynamic task data

Property name	Required	Description
dataSetName	no	A custom name to identify this dataset. The default value is "Data Set".
dataValues	yes	A list of values that can be referenced in tasks and questions. The values can be referenced with a specific token including the index of the value. For example, referencing the first value can be done via `[data:0]`.

Clients can receive dynamic task data by sending a GET request to either `/task/data/{dataId}` or `/user/{userId}/data`. The system responds with the specific dataset or a list of all datasets created by the user respectively.

In order to create a new dataset, a user has to send a POST request to `/task/data` with JSON content as defined in table 11. Additionally, task data can also be uploaded in form of a CSV file by sending a POST request to `/upload/data`. The uploaded CSV file must be in a certain format, separated by the character of the configured CSV separator deployment property described in section 4.3.2. All values have to be in the first line of the file. Given the default value separator, the CSV file would have following structure:

```
[value 1]|[value 2]|[value 3]|...
```

The dataset name will be the original file name of the uploaded file.

Additionally to providing task data it is possible to define questions within the CSV file. A task referencing a dataset via its `dataId` property will automatically include the predefined questions. The questions from the CSV file are simply added after the questions which have been defined in the normal task definition. While the first line of the CSV file is reserved for data values, every line afterwards can be used to define a new question. The structure of a question definition also depends on the configured CSV separator character:

```
[type]|[required]|[text]|[answer 1]|[answer 2]|...
```

The values correspond to the properties of a question listed in table 7. It is also possible to directly reference the defined values of the first line of the CSV file in the answer options afterwards. The following content shows a full example of a CSV file defining both data values and multiple questions:

```
food|drinks
DROPDOWN|true|What is your gender?|male|female
```

```
SINGLE_CHOICE|true|Pizza belongs to...|[data:0]|[data:1]
SINGLE_CHOICE|true|Cola belongs to...|[data:0]|[data:1]
MULTIPLE_CHOICE|false|I like...|none|[data:0]|[data:1]
OPEN|false|Do you have anything to say about this survey?
```

In this example the data values are used as topics that need to be allocated to different terms. By cloning a task and simply changing the data values of the first line it is possible to easily create a whole new task with different topics to assign.

Users can delete their created datasets by sending a DELETE request to `/task/data/{dataId}`. This action requires the user to be the creator of this dataset or to possess admin privileges.

4.4.4 Task Assignments The crowdsourcing platform automatically chooses suitable workers for each task. Such a task assignment contains the information which task is assigned to which worker. Task assignments also have different states, representing the actions a worker has taken and which operations can be performed. Figure 17 illustrates the possible state transitions for task assignments.

A new assignment initially starts in the ASSIGNED state, which means that a worker has just been assigned to this task an can either accept or decline the assignment. Depending on the taken action, the state changes respectively to ACCEPTED or DECLINED. When a worker finishes his work on a task, the assignment state will change to FINISHED. If a worker misses to accept, decline or finish the assigned task and the task is ended, the assignment state transitions to MISSED. On the other hand, if a worker fails to finish an already accepted assignment before the task is ended, the assignment changes to the EXPIRED state.

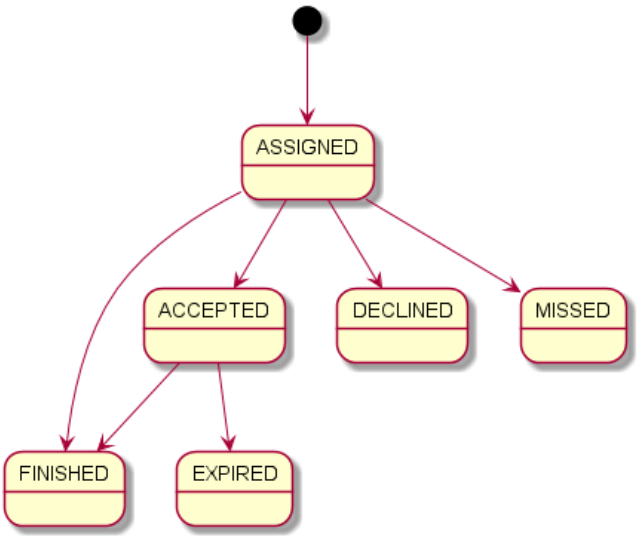

Fig. 17. State machine of task assignments

The differentiation between finished, missed and expired tasks can be used to calculate a reputation for workers. Users who have missed a lot of tasks show a lack of diligence and might be lower prioritized when being considered for a task assignment.

Task assignments cannot be created directly by users, they are automatically determined by the system. The platform iterates throw a series of steps in order to optimize the way of finding suitable workers. In a first step workers who got recruited for a retainer crowd are taken into consideration. The concept of retainer crowds is explained as a separate feature in this section. The second step tries to find workers that have declared their availability at the current point of time. This means that workers who provide their availability information are always favored for being assigned to tasks. Lastly all other workers are considered.

The amount of workers that are assigned to a task depends on the `maxWorkers` setting defined during the task creation, which is explained in table 6. In order to minimize the time needed for receiving a sufficient amount of results, the system calculates the overall percentage of assignments that are accepted or finished by a worker within ten minutes. The task will get distributed to a number of workers based on this factor and the defined `maxWorkers` setting of the task.

Clients can receive task assignments by sending a GET request to the URL `/assignment/user/{userId}/new`. The system responds with a list of all new assignments for this worker. This only includes assignments that have not yet been accepted, declined or finished by the worker. Table 12 lists the properties that describe a task assignment.

Table 12. Properties of a task assignment

Property name	Required	Description
id	yes	The identifier of this task assignment.
workerId	yes	The identifier of the worker to whom the task is assigned.
taskId	yes	The identifier of the task that is assigned to the worker.
taskAssignmentState	yes	The current state of this task assignment.
taskState	yes	The current state of the task corresponding to this assignment. This property is provided to increase the usability for clients, so that they can hint users whether the task is ready to receive results.

New task assignments can be accepted or declined by workers by sending a PUT request to `/assignment/{assignmentId}`. The JSON content of the request must simply hold a value representing the desired new state, which is either ACCEPTED or DECLINED. If the state change was successful, the system answers with a success response containing the identifier of the assignment and the new state.

```
{
    "success": true,
    "assignmentId": "320600c4-dada-445f-bf26-f556aba76b70",
    "taskAssignmentState": "ACCEPTED"
}
```

4.4.5 Results Workers have the responsibility to finish the tasks that have been assigned to them. Therefore, when receiving a task assignment, a user can submit a new result for the task. This can be done by sending a POST request to `/result`. The JSON content must represent a task result as listed in table 13. Users can only submit results if they have been assigned to the task and have no other result submitted for this task so far.

Figure 18 illustrates the workflow of creating tasks, activating and distributing them to workers. Finally a worker submits a result, which can then be retrieved by the creator of the task.

Table 13. Properties of a task result

Property name	Required	Description
taskId	yes	The identifier of the task that corresponds to this result.
questionResults	yes	A list of results for questions contained in the task. The structure of a question result is defined in table 14.

Table 14. Properties of a question result

Property name	Required	Description
questionId	yes	The identifier of the question.
chosenAnswerOptions	yes	A list of answer options that have been chosen for this question. Depending on the question type, this list can be empty or contain a single or multiple values. The values must accurately represent the provided answer options of the question.

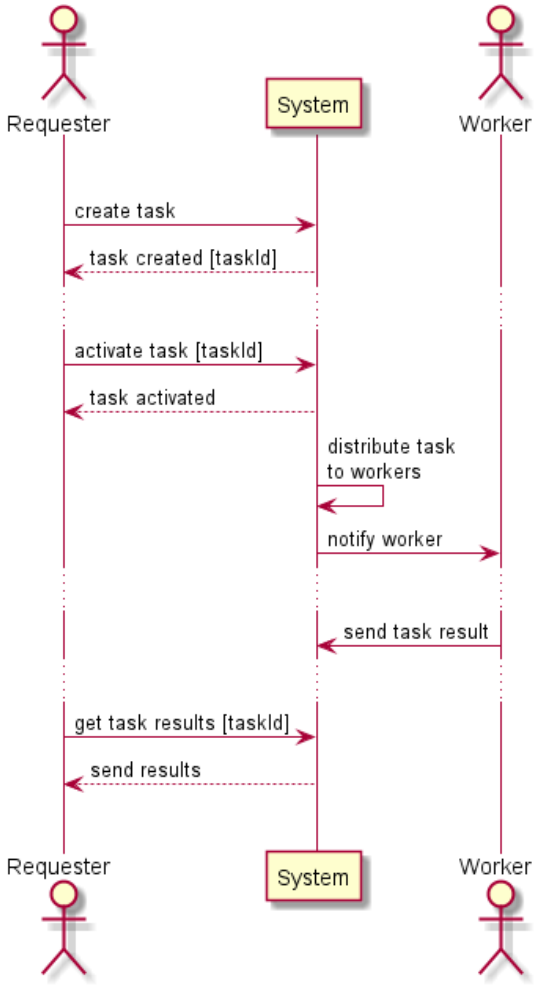

Fig. 18. Workflow of creating and working on tasks

By sending a GET request to `/result/{resultId}` the system responds with the found result. In order to receive all results for a specific task, a client has to send a GET request to `/task/{taskId}/results/detail`. If the requesting client is the initial creator of the task or possesses admin privileges, the platform will respond with a list of all results.

Another feature is the receiving of a statistical overview of all results for a specific task. Clients can send a GET request to `/task/{taskId}/results` to receive statistical information of the received results and some data of the task itself for convenience reasons. The properties of such a statistical task result are listed in table 15.

Table 15. Properties of a statistical task result

Property name	Required	Description
taskId	yes	The identifier of the task.
title	yes	The title of the task. This property is included for convenience reasons, as the overview of such a statistical representation might want to show some context regarding the according task.
receivedResults	yes	The total number of task results received so far for this task.
statisticalQuestionResults	yes	A list of statistical question results. The structure of such a result is described in table 16.

Table 16. Properties of a statistical question result

Property name	Required	Description
questionId	yes	The identifier of the task.
questionText	yes	The actual question text. This property is included for convenience reasons, as the overview of such a statistical representation might want to show some context regarding the according question.
questionType	yes	The type of the question. This property is needed to help the client figure out how to present the received results.
submittedAnswers	yes	The total number of answers received so far for this question.
answerOptions	yes	The list of possible answer options for this task. This property is provided for conveniently displaying the available answers.
chosenAnswers	yes	A list of numbers that indicate the total amount of how often each answer option has been chosen by workers. The number's index in this list corresponds with the answer option's index.

4.4.6 Retainer Crowds A requester has the possibility to recruit a so-called retainer crowd to reserve users for a specific range of time. During this time, workers of the retainer crowd have the topmost priority at getting the task assigned when the requester activates it. An in-depth explanation of the concept of retainer crowds is provided in section 2.2.1.

Users can create a new retainer crowd by sending a POST request to the URL `/retainercrowd`. The request's JSON content must specify the properties as defined in table 17. Users that have requested a retainer crowd have to reward the recruited workers with a configured amount of points per minute of the retainer crowd duration. The configuration is mentioned in section 4.3.2. Upon receiving a request for creating a new retainer crowd, the system automatically searches for workers that have announced their availability within the requested time duration.

Table 17. Properties of a request for creating a retainer crowd

Property name	Required	Description
startTime	yes	The start time of the retainer crowd in the format yyyy-MM-dd HH:mm. Must be a future point of time before the end time.
endTime	yes	The end time of the retainer crowd in the format yyyy-MM-dd HH:mm. Must be a future point of time.
maxNumberOfWorkers	yes	The maximum amount of workers that shall get recruited for the retainer crowd. The value must be at least 3.

It is possible to receive information about a specific retainer crowd by sending a GET request to `/retainercrowd/{retainerCrowdId}`. The information provided for the retainer crowd contains the properties described in table 18. Additionally, the system provides a list of all retainer crowds a user has currently requested. This information can be received by sending a GET request to `/user/{userId}/retainercrowds/requested`. Furthermore, by sending a GET request to `/user/{userId}/retainercrowds/assigned`, a list of all retainer crowds this user has been assigned to is returned.

Table 18. Properties of a retainer crowd

Property name	Required	Description
id	yes	The identifier of the retainer crowd.
requesterId	yes	The identifier of the user who created this retainer crowd.
startTime	yes	The start time of the retainer crowd in the format yyyy-MM-dd HH:mm.
endTime	yes	The end time of the retainer crowd in the format yyyy-MM-dd HH:mm.
workerIds	yes	A list of identifiers of users that have been recruited for participating in this retainer crowd.

4.4.7 Scores The prototype of the crowdsourcing platform uses a simple score system that keeps track of requesters' spent points and workers' earned points. This information is automatically updated by the application upon task related actions. Additionally, some statistical information about tasks is stored within requester scores and worker scores.

A user can receive his requester score by sending a GET request to the URL `/score/requester/{userId}`. The returned response provides information about the requester's spent points and the amount of created tasks. The structure of a requester score is described in table 19.

Table 19. Properties of a requester score

Property name	Required	Description
requesterId	yes	The identifier of the user to whom the requester score belongs.
spentPoints	yes	The total amount of points spent so far for creating tasks and requesting retainer crowds.
createdTasks	yes	The total amount of created tasks.

Similar to the requester score a user can retrieve his worker score by sending a GET request to the URL `/score/worker/{userId}`. Additionally to the earned points, the worker score provides information about the tasks that have been assigned to the user and the amount of times this user has participated in a retainer crowd.

Table 20. Properties of a worker score

Property name	Required	Description
workerId	yes	The identifier of the user to whom the worker score belongs.
workerScore	yes	The total amount of points earned so far by working on tasks and participating in retainer crowds.
completedTasks	yes	The total amount of completed tasks.
completedQuestions	yes	The total amount of completed questions.
expiredTasks	yes	The total amount of expired tasks. A task is expired if the worker fails to finish an already accepted task before the task is ended.
missedTasks	yes	The total amount of missed tasks. A task is missed if the worker does not accept, decline or finish a task before the task is ended.
retainerCrowdParticipations	yes	The total amount of times this user has participated in a retainer crowd.

4.4.8 Push Notifications One goal of the crowdsourcing platform is the ability to provide results as quickly as possible, at best in real-time. This goal is obviously tightly bound to the number of active users, as the pool of suitable and hard-working workers increases. However, even with a high amount of users, the availability of the users is crucible for short response times. By providing an interface that is also accessible on mobile devices, the range of available workers can be vastly extended. People do not need to be at home on their desktop computer or notebook in order to work on tasks. Instead, they can quickly react on a new task assignment wherever they are.

In order to further improve the reaction time of crowd workers, the framework allows clients to register for push notifications. The concept of push notifications is explained in section 3.2.4. For the development of this prototype, the Firebase platform has been used as described in section 4.2.5. Clients can register their device for receiving push notifications by sending a POST request to `/user/device`. The JSON content of the request needs to hold the Firebase device ID token of the client's device. For example, Java clients can query their current token by calling following method provided by the Firebase Java API:

```
FirebaseInstanceId.getInstance().getToken()
```

As Firebase device identifiers can get refreshed, clients have to communicate their updated device ID to the crowdsourcing platform. As the system maps each received device identifier to the logged in user, a client registering for push notifications needs to be logged in when subscribing itself. Therefore it is also possible that one device is used with multiple accounts or one account is used on multiple devices. According to the publish-subscribe messaging pattern, the crowdsourcing platform will publish push notifications to all devices that have ever been registered in association with the affected user identifier. Additionally, the system will automatically detect invalid or outdated device identifiers and remove them from the mapping.

Clients can also explicitly unsubscribe from push notifications by sending a DELETE request to `/user/device`. Just like the subscription request, the JSON content for unsubscribing needs to hold the device identifier as value.

4.4.9 Monitoring The crowdsourcing platform provides a monitoring service which keeps track about certain actions that are performed within the system. In order to retrieve the current state of monitoring information, clients can send a GET request to `/monitoring`. Table 21 describes the available properties of events that are monitored.

Most of the properties are only used for informational purposes. The number of quickly accepted assignments however, which represents the total amount of task assignments that have been accepted by users within ten minutes, is also used for calculating the number of workers that will get assigned to a task. The idea behind this feature is that depending on the average reaction time of the workers, the system can distribute a task to an optimal number of workers in hindsight to minimize the time a requester has to wait for results.

Table 21. Properties of monitoring information

Property name	Required	Description
numberOfUsers	yes	The total number of users that have registered themselves in the system so far.
numberOfTasks	yes	The total amount of tasks that have been created by all users.
numberOfQuestions	yes	The total amount of questions that have been created by all users.
numberOfAssignments	yes	The total amount of task assignments that have been created by the system.
numberOfAcceptedAssignments	yes	The total amount of task assignments that have been accepted by users.
numberOfQuicklyAcceptedAssignments	yes	The total amount of task assignments that have been accepted by users within ten minutes.

Users with admin privileges have the possibility to reset the monitoring information by sending a DELETE request to `/monitoring`. If the reset succeeds, the system responds with the newly initiated monitoring information.

4.5 Client Applications

The prototype implementation not only includes the back end functionality, but also two simple test applications for clients, which provide graphical user interfaces for most of the presented functionality.

4.5.1 Web Client The web client is a really basic homepage that only supports a very limited set of features. Users can login, logout and register via this client application. Additionally, the main purpose of this client is to allow the upload of task data sets as described in section 4.4.3. The monitoring information, which is mentioned in section 4.4.9, can also be queried and reset using the web client's interface. Additionally, the web client can retrieve the results of a logged in requester's tasks in JSON format.

Figure 19 and 20 show screenshots of the client's user interface with all possible actions.

The web client comes with the server and can be installed by deploying the attached `crowdsourcing.war` archive on a Java application server.

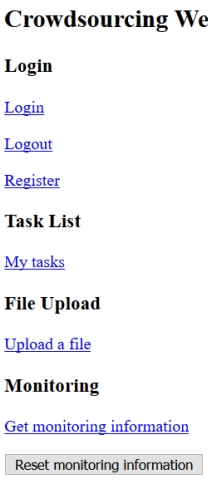

Fig. 19. User interface of the web client

Fig. 20. Receiving all created tasks and a specific result via the web client

4.5.2 Android Application In comparison to the small web client, the vast majority of the prototype's features are integrated in a native Android application. This application also mainly serves as a test client that only provides a simple user interface to work with the implemented back end functionality.

The entry menu of the application is a login screen, which also gives the option to register in the system. After logging in and authenticating against the server, the main menu presents buttons for major features, split by the requester and worker roles. Furthermore, a profile menu allows the user to update his or her information as described in section 4.4.1. There is also the option to to add availability information. Figure 21 shows screenshots of those login and main menu pages.

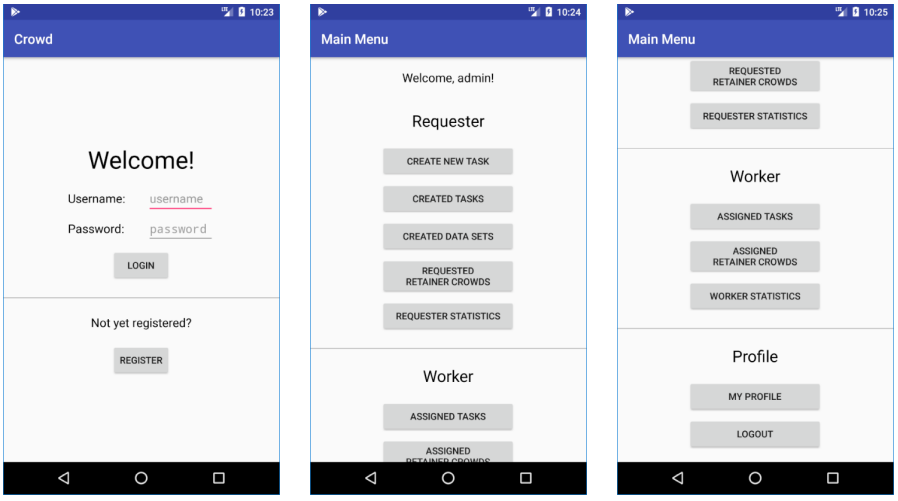

Fig. 21. Login page and main menu of the Android client

Users can create tasks and request retainer crowds. Previously uploaded data set via the web client application mentioned in section 4.5.1 can be inspected in a separate menu and used during task creation.

A dedicated menu lists all the tasks created by the user and provides a few options to interact with them. Users can change the state of a task, which also includes activating a task as described in section 4.4.4. Additionally, retrieved results can be obtained and the task can be cloned or deleted.

When a task is activated and distributed to workers, their Android device with an installed client application will receive a push notification. In the worker section, a list of all assigned tasks is available, with the option to work on a task and submit the result. Figure 22 shows screenshots of the menus for browsing all assigned tasks and working on a task.

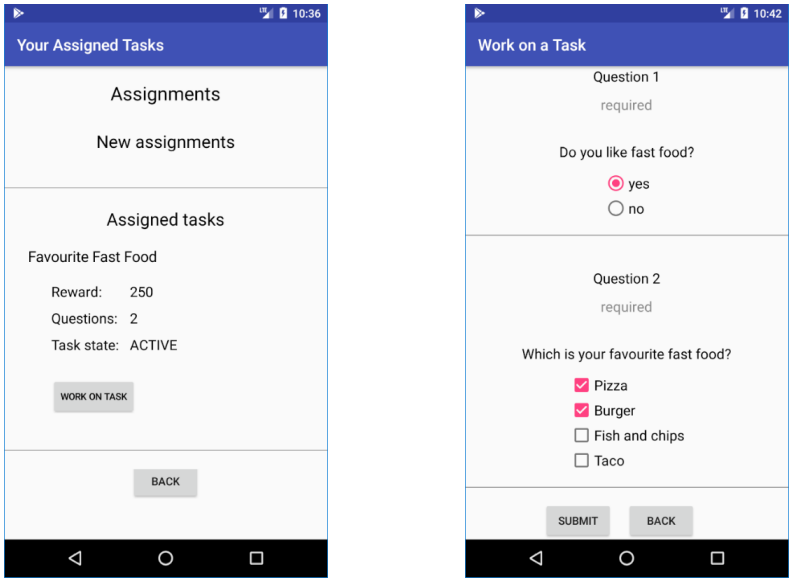

Fig. 22. Browsing assigned tasks and working on a task in the Android client

Fig. 23. Inspecting the retrieved results of a requested task in the Android client

As soon as task results are submitted to the server, the requester can see the currently retrieved results for each task in a separate menu. This menu presents the statistical results with the option to view the details of each received single result. The different result views are explained in section 4.4.5. An example of the result overview is shown in the screenshot presented in figure 23.

5 Evaluation

The evaluation has been performed in consideration of two different aspects, namely quality and quantity. The quality evaluation targets the user experience and was done with an evaluation survey. The quantity aspect describes the system from a more technical point of view and presents measurements of the crowdsourcing platform during operation.

5.1 Quality

The first part of the user quality survey focuses on the usability of the application [6][13], giving an overview on the design of the client applications, which are described in section 4.5. Additional questions target the integration of the crowdsourcing features within the system and the tester's belief that the system can actually fulfill the functional requirements stated in section 1.2.

The survey has been answered by 10 different people, which have been given access to the prototype system with an installation of the Android application. They also received a user manual on how to operate the system with a suggestive to-do list to get familiar with all the main features. The user manual is attached to this thesis and can be found in appendix A.

The original evaluation survey as presented to the test subjects can be found in appendix B.

5.1.1 System Usability Scale Questions 1 to 10 are statements that concern the user's experience with the interface. Each question offers five answer options, agreeing and disagreeing on a scale from 1 to 5 with the statement. The statements are taken from the system usability scale (SUS) [6].

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Figure 24 illustrates the result from questions 1 to 10. The answers have been quite homogeneously, showing that all users felt comfortable using the developed prototype. This is quite a positive outcome to notice, given the fact that nobody of the testers have had worked with a crowdsourcing system before. The highest variation is spotted for question 1, which can be explained by the fact that some people simply do not desire to participate in crowdsourcing, for example due to their lack of time or interest.

System Usability Scale Score The SUS score is a simple way of rating a system's usability based on the questions of the system usability scale. The score can range from 0 to 100, which refer to the worst and best possible usability respectively. The developed crowdsourcing prototype achieved an average SUS score of 93.6, which represents an excellent usability [3].

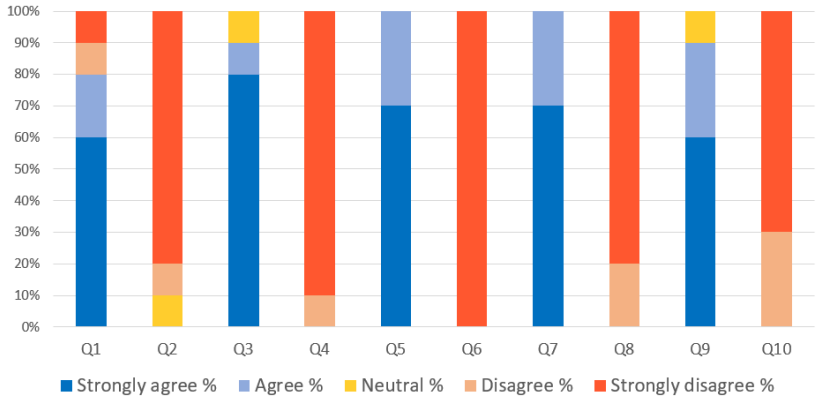

Fig. 24. Evaluation survey result of questions 1 to 10 about the user experience

5.1.2 Integration of Crowdsourcing Features The last four questions deal with the presentation and integration of crowdsourcing specific topics.

Question 11 let the test subjects rate the system's usability on a more fine-grained level.

11. How did you experience the user interface of the mobile application?
    - Highly user-friendly
    - User-friendly
    - Little user-friendly
    - Not user-friendly

As shown in figure 25, the answers for question 11 correlate to the observed result from the system usability scale questions. Only one person did not vote for the application being "highly user-friendly" or "user-friendly". The most common improvement suggestion was the unnecessary scrolling in the main menu, as all menu options are listed right below each other.

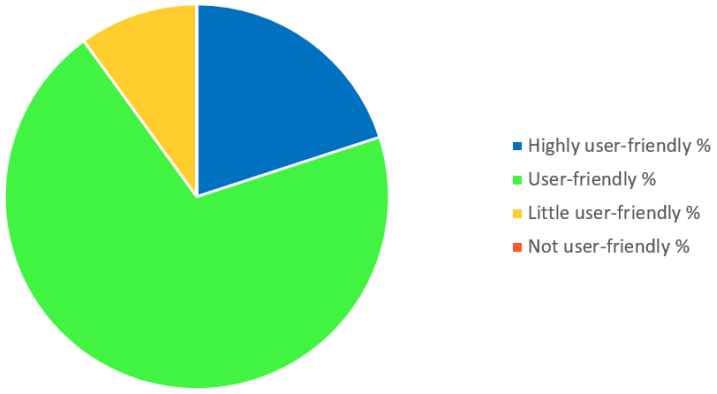

Fig. 25. Evaluation survey result of question 11

Question 12 targeted the integration and presentation of crowdsourcing features. As the test persons had no prior experience with crowdsourcing, the provided user manual introduced some of the available features to a certain extent already.

12. Are all provided features presented easily understandable or did you have problems figuring out the purpose of certain options?
    - I had no problems; all features were presented easily understandable.
    - I had troubles understanding some features.
    - I did not understand most features.

Diagram 26 illustrates that the majority of people found the presentation of the crowdsourcing features good enough for them being easily understandable.

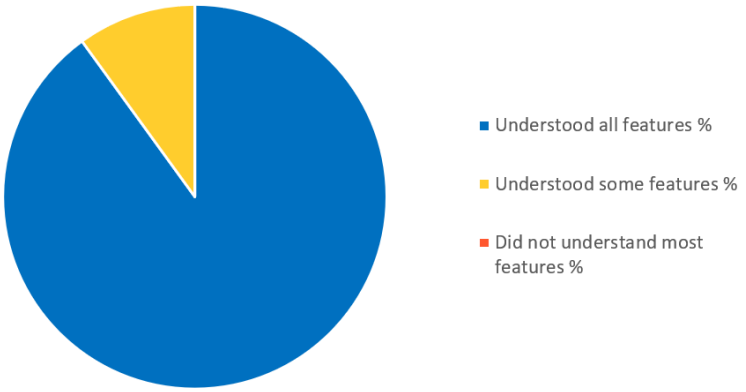

Fig. 26. Evaluation survey result of question 12

The test persons were also confronted with the crowdsourcing system's requirement of delivering results on demand and in real-time. After testing the application for a short amount of time and getting familiar with the features, the users should have an opinion whether the platform is capable of fulfilling those needs.

13. In your opinion, is the system able to deliver results on demand and in real-time?
    - Yes, the system fulfills those criteria.
    - The system fulfills those criteria to a certain extent but needs improvement.
    - No, the system is not able to satisfy those needs.

As figure 27 shows, all users were confident that the application fulfills the requirements.

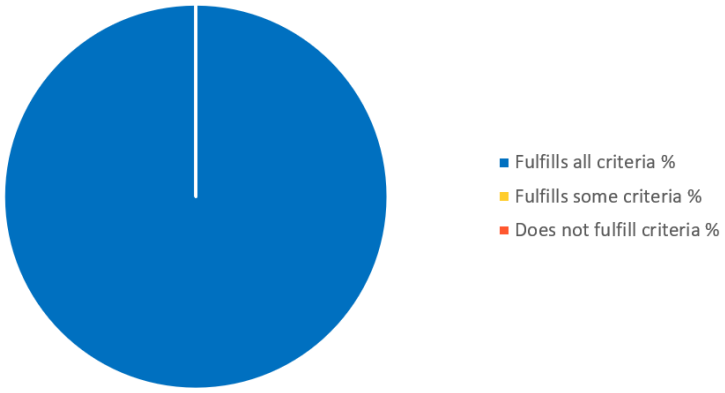

Fig. 27. Evaluation survey result of question 13

The last question was about the presentation of the received task results. The platform is able to provide an aggregated statistical view on the results, which is described in section 4.4.5. Additionally, an anonymous view on each single result is also available. The combination of those two views allows the user to quickly interpret the result in various levels of detail.

14. How useful is the presentation of the task results?
    - Very useful
    - Useful
    - Not useful

As presented in diagram 28, all testers found the presentation at least useful, with the majority considering it a very useful solution. Although the comments

suggested to further improve the result view by providing even more information with possible diagrams and graphs, the overall reception was quite positive.

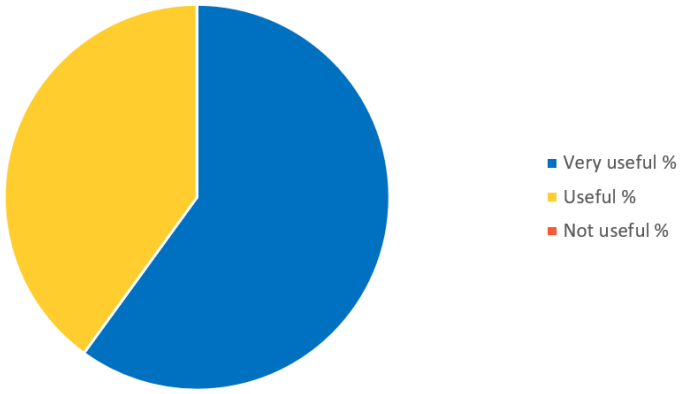

Fig. 28. Evaluation survey result of question 14

5.2 Quantity

The second aspect of the evaluation interprets some technical measurements. The goal of those measurements is to detect whether the system can fulfill the functional requirements in an appropriate amount of time, not being a bottleneck of the overall user experience.

5.2.1 Result Validation Duration One of the main critical aspects of the real-time requirement is the validation of results. Task results uploaded by the crowd should take as little time as possible, minimizing the delay until the requester receives an outcome of his task.

In average, the validation of a task with 100 questions of varying complexity took 51 milliseconds. This is a very good result, meaning that the duration of the result validation does not play any considerable role at all.

5.2.2 Task Distribution Load Test Another critical step during the whole task workflow is the distribution of tasks. For this measurement, a load test has been performed that performed a task distribution to a total of 116 users.

The measured duration included the determination of the crowd workers and the creation of every task assignment. The average duration of the whole task distribution was 2884 milliseconds. In comparison to the load test with lots of assignments, a smaller distribution to only 5 different users took in average 339 milliseconds.

The task assignment process could definitely be improved, but overall it is taking an acceptable amount of time. Users will not suffer too hard from the short waiting time until their task is distributed to the desired amount of workers.

6 Conclusion

The goal of this project was to create a working prototype of a crowdsourcing system fulfilling the requirements specified in section 1.2. In order to achieve this, some state-of-the-art approaches have been reviewed and implemented.

While developing the crowdsourcing platform, the questions presented in section 1.3 should be answered. The following sections deal with those questions and summarize the discovered findings.

6.1 Result

The main question was about the design of a real-time on demand crowdsourcing framework for multimodal process. As the resulting prototype is able to fulfill the specified functional requirements, the chosen architecture and design can be seen as a successful solution for this problem. An in-depth explanation of the architectural decisions and the design can be found in section 3 and 4 respectively.

One requirement of the prototype was to offer the functionality of a framework, meaning that it should be easily extensible for other developers. By choosing to implement the platform as a web service, this functionality is implicitly given. It is easily possible to extend the features by implementing a new service and offering additional URL mappings via separate controllers. The prototype already offers services for the core feature set. Also there are base classes in place that help implementing new components.

6.2 Limitations

While investigating different state-of-the-art approaches to various challenges in crowdsourcing, it was pretty obvious that some techniques target to optimize or improve a very specific topic or aspect of the field. As the scope of this project was only to develop a prototype supporting basic functionality, it was not feasible to implement some approaches, because they would have required a much more complicated feature set.

6.3 Evaluation

The evaluation process in terms of quality was performed with the system usability scale [6] and a related method for interpreting the resulting score [3]. While this approach was able to quickly deliver results, it also turned out to be really simple and straight-forward to perform. The downside of this evaluation method is that it only provides a generic overview on a system's usability. That

means that the scale exclusively shows if there exist any usability problems and not what problems there are exactly. Even John Brooke himself called it only a "quick and dirty" solution [6]. In order to overcome this downside, a few additional questions have been added to the qualitative evaluation survey, asking for improvement suggestions and targeting the crowdsourcing features explicitly. With this extension, a really short questionnaire was enough to retrieve quite useful results.

On the other hand, the evaluation in terms of quantity was purely based on technical measurements. As the key aspect of the application is to deliver results in real-time, the main focus of those measurements was the computational duration of possible bottlenecks in the workflow for retrieving results. Additionally, as a crowdsourcing system requires a lot of users to work out, some time measurements under load have been performed. This evaluation strategy was also easy to perform and clearly pointed out improvement possibilities from a technical point of view.

7 Future Work

As specified in the functional requirements in section 1.2, the goal of this project was to develop a prototype that implements the basic features of a crowdsourcing platform. Additionally, some state of the art concepts should be integrated to further improve the performance of the system. However, there still are many improvement possibilities, regarding the addition of new features as well as some technical adaptions to enhance the quality of the software.

7.1 Utilization of Mobile Device Services and Technologies

Most modern mobile devices come with a rich set of services that provide useful functionality. For this project, mobile devices have been mainly used to increase the users' availability. However, with the integration of existing services and technologies, the product's capabilities could be seriously enhanced.

7.1.1 GPS Service All location features of the current prototype implementation are purely text based. Making the platform aware of GPS data would be a major improvement of the whole application. The integration of location information not only allows mobile devices to make use of their location tracking services, which would lead to a much better user experience by automatically detecting and submitting the user's location. It could also enable the easy integration of location based third party services like maps.

7.1.2 Calendar Synchronization One more great opportunity is the synchronization of the device's or third party calendars with the user information. The platform would be able to automatically determine the users' availability, ultimately leading to more fine-grained task assignments and therefore an improved overall performance.

7.1.3 Wearable Technologies Another step would be the combination of mobile devices with wearable technologies such as smart watches in order to further improve availability and response time of crowd workers. The wide spread of wearable applications is yet to come and the hardware itself comes with certain restrictions, namely limited hardware power and user interface constraints. Therefore it would need a prior analysis of how to design a supportive wearable application that further improves the system's results.

7.2 Task Extensions

Currently there are only simple text or multi-media based tasks supported. As described in 2.4.1, there exists a variety of other task concepts. The platform could be extended to support other task types, such as location tagging, uploading pictures, ratings or even very specific ones like defining schedules for opening hours of stores and restaurants.

7.3 Reputation System

In general, it is desired that reliable workers also get to work on tasks in the future. Therefore it makes sense to implement a reputation system that keeps track of each worker's reliability. During the task assignment phase, the platform could then prioritize workers with a higher reliability. As the framework already stores the amount of finished, missed and expired tasks, those statistics could be reused to determine a reliability score. Additionally, the detection of quickly accepted tasks is already implemented, so the approach could be even more refined to also take the speed of a worker into account.

7.4 Final Product Completeness

The prototype implementation is not yet ready to actually being used as a complete product. Some aspects of the platform are only implemented in a very basic way. There are specific features that need to be extended and improved to satisfy the needs of a production ready system.

7.4.1 Score System Currently the users can only track their work by a simple score system that only counts the rewarded and spent points. While this is useful to monitor the own status of productivity, it lacks some additional value to motivate users for a continuous participation in the crowdsourcing system. There are a few possible extensions to overcome this drawback.

Payment System The obvious solution for motivating people to work on something is to pay them. The score system could be extended to an actual payment system that maps the rewarded points to earned money. This requires of course a few other improvements, especially from a security perspective. Various payment methods could be offered, not only to pay workers, but also to allow a requester to buy points for his credit balance.

Leaderboard As described in section 2.4.1, a good approach for improving the participation in crowdsourcing applications is the integration of gamification concepts [23]. The introduction of a global leaderboard, maybe even in combination with other elements like achievements, can help keeping up the motivation for crowd workers.

7.4.2 User Verification The prototype implementation allows anyone to register with hardly any input validation and no user verification at all. In order to guarantee the valid existence of a user, the registration process can be refined. Possible verification methods include the widely used e-mail verification or the integration of social media platforms.

7.4.3 User Interface Improvements An evaluation of the user experience, presented in section 5, resulted in an already excellent usability. However, there are still improvement possibilities that can help avoiding small annoyances.

The most requested change concerned the structure of the Android application's main menu. It is possible to introduce a role based interface, only showing menu options of the current role. This separation of requester and worker role would lead to a much cleaner user interface, making the work with a mobile device more appealing. Additionally, the input of certain data could definitely be improved.

Furthermore, it is still possible to provide more information in the client application. The presentation of the already available information can also be enhanced. The introduction of additional diagrams and dashboards would allow a better overview, leading to an easier understanding and interpretation of the data.

References

1. Google Firebase documentation. Website, https://firebase.google.com/docs/, last visit on 2018-05-02
2. Alex, B., Taylor, L., Winch, R., Hillert, G.: Spring Security Reference 4.1.3 RELEASE. Website (2015), https://docs.spring.io/spring-security/site/docs/4.1.3.RELEASE/reference/pdf/spring-security-reference.pdf, last visit on 2018-04-11
3. Bangor, A.: Determining What Individual SUS Scores Mean : Adding an Adjective Rating Scale (2009)
4. Bernstein, M.S.: Crowd-powered interfaces. In: UIST (2010)
5. Bernstein, M.S., Brandt, J., Miller, R.C., Karger, D.R.: Crowds in Two Seconds: Enabling Realtime Crowd-powered Interfaces. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology. pp. 33–42. UIST '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2047196.2047201
6. Brooke, J.: SUS: A quick and dirty usability scale 189 (11 1995)
7. Bruno, A., Backes, J.: Comparison between CSV parsers. Website (Feb 2018), https://github.com/uniVocity/csv-parsers-comparison, last visit on 2018-05-02
8. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall PTR Upper Saddle River, NJ, USA (2004)
9. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis (2000), aAI9980887
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
11. Gyorodi, C., Gyorodi, R., Pecherle, G., Olah, A.: A comparative study: MongoDB vs. MySQL. 2015 13th International Conference on Engineering of Modern Electric Systems (EMES) pp. 1–6 (2015)
12. Hadano, M., Nakatsuji, M., Toda, H., Koike, Y.: Assigning Tasks to Workers by Referring to Their Schedules in Mobile Crowdsourcing. In: HCOMP (2015)
13. Hearst, M.A.: Search User Interfaces. Cambridge University Press, New York, NY, USA, 1st edn. (2009)
14. Hwang, H.: Moral Reminder as a Way to Improve Worker Performance on Amazon Mechanical Turk. In: HCOMP (2015)
15. Johnson, R., Hoeller, J., Donald, K., et al.: Spring Framework Reference Documentation 4.3.1 RELEASE. Website (2015), https://docs.spring.io/spring/docs/4.3.1.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf, last visit on 2018-04-11
16. The JSON Data Interchange Format. Tech. Rep. Standard ECMA-404 1st Edition / October 2013, ECMA (Oct 2013), http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, last visit on 2018-04-07
17. Lasecki, W.S., Rello, L., Bigham, J.P.: Measuring text simplification with the crowd. In: W4A (2015)
18. Lasecki, W.S., Weingard, L., Ferguson, G., Bigham, J.P.: Finding action dependencies using the crowd. In: K-CAP (2013)
19. Lasecki, W.S., Weingard, L., Ferguson, G., Bigham, J.P.: Finding dependencies between actions using the crowd. In: CHI (2014)

20. Leff, A., Rayfield, J.T.: Web-Application Development Using the Model/View/-Controller Design Pattern. In: EDOC (2001)
21. Malaya, V.: SQL vs. NoSQL. Website (2013), http://www.sql-vs-nosql.blogspot.co.at/2013/11/indexes-comparison-mongodb-vs-mssqlserver.html, last visit on 2018-04-09
22. Mondal, A., Raravi, G., Chugh, A., Mukherjee, T.: LoRUS: A Mobile Crowdsourcing System for Efficiently Retrieving the Top-k Relevant Users in a Spatial Window. In: HCOMP (2015)
23. Morschheuser, B., Hamari, J., Koivisto, J.: Gamification in Crowdsourcing: A Review. In: HICSS (01 2016)
24. Nandan, N.: SimplyCity - A Simple Mobile Crowdsourcing Platform for Emerging Cities. In: HCOMP (2015)
25. Raj, R.: Spring MVC Request Life Cycle. Website (May 2014), https://justforchangesake.wordpress.com/2014/05/07/spring-mvc-request-life-cycle/, last visit on 2018-04-10
26. Saloranta, T., Brown, P.: Jackson project repository on GitHub. Website (Sep 2016), https://github.com/FasterXML/jackson, last visit on 2018-05-02
27. Smith, G., Conway, S.: opencsv documentation. Website (Nov 2017), http://opencsv.sourceforge.net, last visit on 2018-05-02
28. Sweet, R.E.: The Mesa programming environment. SIGPLAN Notices 20, 216–229 (1985)
29. Vazirabadkar, K.M., Gadre, S.S., Sebastian, R., Dwivedi, N.: Crowdsourcing Based on GPS. In: HCOMP (2015)

A User Manual

The following user manual has been provided to all testers which were involved in the evaluation phase of the project. It gives a short overview about the topic and describes some major use cases and functionalities of the system. Additionally, there is a to-do list appended, providing a suggestive list of actions to perform to discover the application's features.

INTRODUCTION

CROWDSOURCING

Crowdsourcing is the concept of outsourcing smaller subtasks to a group of voluntary workers over the internet.

Most existing crowdsourcing platforms come with certain limitations. Several tasks often require the crowd workers to be available at a specific time or location. When working from a desktop computer it may not be very appropriate to provide solutions to these problems.

Because of the wide spread of mobile devices, a mobile application for crowdsourcing could provide results from any place at any time.

PLATFORM

The system consists of a server back end and a mobile front end application.

Following main characteristics define the functionality of the crowdsourcing platform:

- Registered users can either create tasks ("requester") or work on tasks that are assigned to them ("worker").
- Every user can choose whether he wants to be active as a worker and therefore receive tasks. This option can be changed in the profile settings.
- Users are rewarded with points for finishing tasks.
- Users are charged with points for each received result of a created task or for reserving retainer crowds.
- Users can upload CSV files that contain referenceable, reusable data and predefined question definitions.

USER INSTRUCTIONS

REQUESTER – CREATING TASKS

A task consists of title, description, the reward to give each worker and a list of questions.
Images, audio and video files can be referenced for the whole task or a specific question.
Furthermore, assignment conditions can be defined in order to only target specific workers.
Additionally, a previously uploaded data CSV file can be referenced to provide data values and predefined questions.

Here is a list of all **task properties**:

PROPERTY	DESCRIPTION
Title	The title of the task.
Description	A more detailed description.
Reward	The reward a worker is paid for completing this task. Must be in between 0 and 1000.
Maximum workers	The maximum amount of results that will be received. Must be over 3. The default value is 3.
Start time	The planned start time for this task. The task will automatically be activated and assigned to workers at this time.
End time	The planned end time for this task. The task will automatically be finished at this time.
Task media	A list of attached media resources. These will be displayed before the questions. Each media consists of the URL to the source, a description and the media type.
Task data	The referenced data CSV file. A more detailed description can be found below.

Here is a list of all **assignment condition properties**:

PROPERTY	DESCRIPTION
Gender	A worker must be of this gender to be suitable.
Minimum age	The required minimum age in years.
Maximum age	The required maximum age in years.
Country	A worker must be in this country to be suitable.
City	A worker must be in this city to be suitable. A country must be set when setting a city.
Language	A worker must speak this language in order to be suitable.

Here is a list of all **question properties**:

PROPERTY	DESCRIPTION
Question type	The type of the question. Can be either "single choice", "multiple choice", "dropdown" or "open".
Question text	The question itself.
Answers	A list of possible answer options. "Open" questions ignore the answer options, as the worker is expected to type in a custom text.
Required	This option defines whether it should be mandatory to answer this question.
Question media	A list of attached media resources. These will be displayed after the question. Each media consists of the URL to the source, a description and the media type.

REQUESTER – TASK STATES

Tasks are initially in the "created" state and not distributed to any worker.
The distribution only starts as soon as the task is "activated".
Active tasks can either be "paused" or "finished".
Only active tasks can receive new results from workers.

REQUESTER – TASK DATA

Task data has two different purposes:

1. Reusable data values can be defined.
   The data values can be referenced in the task description, question texts and answer options.
   References are done with the mark-up `[data:0]`, where the number is the index of the data value.
2. Questions can be defined.
   This is useful for creating quickly many questions in the CSV file and not having to build them manually via the mobile application.

Task data can be either uploaded using the web client (accessible via any browser), or manually created using the mobile application (data values only).

When uploading a CSV file, following format is required:

```
data value 0|data value 1|data value 2
question type|required|question text|answer option 1|answer option 2
```

Example:

```
sky|fire
SINGLE_CHOICE|true|"Is 'blue' matching the topic '[data:0]'?"|yes|no
SINGLE_CHOICE|true|"Is 'blue' matching the topic '[data:1]'?"|yes|no
OPEN|false|Please add any comments about this survey.
```

Entries need to be separated by the '|' character.
More complex strings can be written using the '"' character.
The first line is reserved for all the data values.
From the second line on, questions can be defined.

For questions, the first entry must be the question type
(**SINGLE_CHOICE**, **MULTIPLE_CHOICE**, **DROPDOWN**, **OPEN**).
When choosing **OPEN**, the answer options for this question are ignored.

The second entry is the flag whether an answer to this question is required and must be a bool value (**true**, **false**).

Task data can be uploaded by visiting the "Crowdsourcing Web Client":
http://131.130.122.222:8080/crowdsourcing/

To do so you simply have to log in and then select "Upload a file". Afterwards you should be able to see your uploaded data set in the mobile application's main menu under "Created data sets".

REQUESTER – TASK CLONING

Tasks can be cloned with a different data set.
Every other property will be taken over from the original task.

Cloned tasks are created just like any other task and need to be activated again.

WORKER – PROFILE INFORMATION

In order to be suitable for more tasks, workers can specify information in their user profile.

They can also add availabilities to improve their chances of being assigned to a task, because the system will assume a shorter response time.

WORKER – WORKING ON TASKS

When a task is assigned to a worker, a push notification is sent to the mobile device.

A worker can either accept or decline the assignment – or start working on the task right away.

If the worker does not react on an assignment and the task is finished, his assignment will be counted as a "missed task".

If the worker accepts an assignment and the task is finished before he completed the task, his assignment will be counted as an "expired task".

Once the task is completed, the worker will be rewarded with the defined score points.

TO-DO LIST

A SUGGESTIVE LIST OF ACTIONS TO PERFORM

In order to provide a reasonable feedback and allow a proper evaluation of the system, here is a list of tasks that should be accomplished.

TASK DESCRIPTION	PRIORITY	DONE
Register as an active worker	high	☐
Log into the system	high	☐
Visit your user profile and check your information	medium	☐
Add some availability information	medium	☐
Check your initial requester and worker statistics	low	☐
Create your own task • feel free to attach video or audio data to task or questions • do not be too strict on the assignment conditions, as the group of testers is not very large	high	☐
Activate your created task • this step is required in order to distribute your task	high	☐
Upload a CSV with task data via the web client • make sure to try defining data and questions within the file	medium	☐
Create another task with the use of your uploaded CSV data	medium	☐
Work on tasks that are assigned to you	high	☐
Check your received results	medium	☐
Check your requester and worker statistics	medium	☐
Update your profile	low	☐

B Evaluation Survey

The following evaluation survey has been handed to all testers which were involved in the evaluation phase of the project. It includes basic questions about the user interface and more complex ones about more detailed aspects of the application. The evaluation itself is described in detail in section 5.

Evaluation in terms of quality of a real-time on demand crowdsourcing platform

1. **I think that I would like to use this system frequently.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

2. **I found the system unnecessarily complex.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

3. **I thought the system was easy to use.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

4. **I think that I would need the support of a technical person to be able to use this system.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

5. **I found the various functions in this system were well integrated.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

6. **I thought there was too much inconsistency in this system.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

7. **I would imagine that most people would learn to use this system very quickly.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

8. **I found the system very cumbersome to use.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

9. **I felt very confident using the system.**
   - ○ Strongly agree
   - ○ Agree
   - ○ Neutral
   - ○ Disagree
   - ○ Strongly disagree

10. I needed to learn a lot of things before I could get going with this system.

- ○ Strongly agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly disagree

---

11. How did you experience the user interface of the mobile application?

- ○ Highly user-friendly
- ○ User-friendly
- ○ Little user-friendly
- ○ Not user-friendly

How could the user experience be improved?

---

12. Are all provided features presented easily understandable or did you have problems figuring out the purpose of certain options?

- ○ I had no problems; all features were presented easily understandable.
- ○ I had troubles understanding some features.
- ○ I did not understand most features.

How could the features be presented in a more understandable way?

---

13. In your opinion, is the system able to deliver results on demand and in real-time?

- ○ Yes, the system fulfills those criteria.
- ○ The system fulfills those criteria to a certain extent but needs improvement.
- ○ No, the system is not able to satisfy those needs.

What improvements are necessary in order to provide on demand and real-time functionality?

---

14. How useful is the presentation of the task results?

- ○ Very useful
- ○ Useful
- ○ Not useful

How could the presentation of results be improved?

C Source Code

The source code can be found on the enclosed CD. It includes the Java server back end as well as an example client software in form of an Android app. Additionally to the source code, it also provides a back end web application archive (compiled with Java 8) and an Android package (built for Android 7.1), both ready for installation.

The back end can be installed by deploying the attached `crowdsourcing.war` archive on a Java application server. Once installed, the web client is accessible via the URL `/crowdsourcing`.

The attached `crowdsourcing.apk` is the package for the native Android client application. Before installing it on an Android device, the server address of the installed back end should be configured in the configuration file `config.properties` located in the package's `assets` directory.