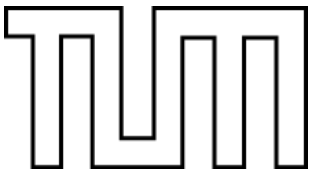


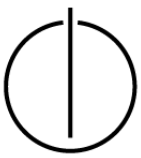
FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis

Implementation of Collaboration Features in CAD Software

Robert Gleixner



FAKULTÄT FÜR INFORMATIK

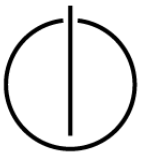
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis

Implementation of Collaboration Features in CAD Software

Integration von Kollaborationsfunktionalität in CAD-Anwendungen

Author:	Robert Gleixner
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M.Sc. Alexander Waldmann
Submission Date:	March 16, 2015



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, March 16, 2015 Robert Gleixner

Acknowledgments

I would like to express my special appreciation and thanks to my advisor Alexander Waldmann. Thank you for guiding me throughout the creation of the thesis. I wish to thank my contacts at Siemens, Dr. Manfred Langen and Mike Burgold, for taking the time to discuss the concepts of my thesis. Finally, i owe special thanks to my family and all my friends who gave me a lot of advice and encouragement. I would not have completed this work without them.

ABSTRACT

Urgent problems cost a lot of money if they are not addressed immediately. Question and answer systems are excellent platforms for getting solutions to problems. The thesis investigates whether the process of finding answers to a question can be improved with a deeper integration into the user's workspace. The ability to collect context information can be used to classify questions more accurately. Together with a social approach of tagging this is the foundation for a new question recommendation system. Recommending and actively pushing notification should lower the response time of user and lower the barrier of getting active at all. The proposed question and answer system is prototyped as a REST service and an integration plugin for the CAD tool SolidEdge. Through a comprehensive survey the usefulness of the system is evaluated.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

"An investment in knowledge always pays the best interest."

— Benjamin Franklin

The history of Knowledge Management (KM) dates back to the early/mid eighties [Wii97], where companies had to manage their knowledge manually due to the lack of technical solutions. Over the years knowledge had become one of the most important assets for companies and although the first tools for KM had arisen, no one really knew how to manage it. Intranet and wiki systems had proven to be suitable solutions to support the accumulation and sharing of knowledge. Nevertheless, access to static information is not sufficient, as there is tacit knowledge which requires face-to-face interaction [Swa+99].

Community-based platforms, which support asking questions and discussions, are an important source of information [NAA09]. Questions and Answers (Q&A) systems have their roots in *Answer Garden* [AM90], a system for collecting and retrieving already answered questions. It simplifies the access to information and eliminates the need for experts to answer the same questions over and over again. Recently, a lot of general Q&A platforms like *Quora* [14f] or *Yahoo Answers*[14o] have grown. These are not limited to a specific subject area and target the general public. However, the poor quality of questions and answers have led to much mischief [14n], where people ask silly questions or give useless answers. Google's Q&A platform *Google Answers* [CHK10] did not suffer from mischief as much as *Yahoo Answers*, because they have another business model. In their knowledge market, the questioner offers a bounty of at least $2 and the researchers need to complete an application process to be approved to be able to answer. This model has proven to be inefficient due to the low amount of contracted researchers and the platform was shut down in 2006 [14a].

Today, the trend is towards domain specific communities. One of the most successful platforms is *Stack Overflow*, "a question and answer site for professional and enthusiast

programmers" [14j]. According to Mamykina, Manoim, Mittal, et al. [Mam+11], it had a community of approximately 300,000 user and more then 7 million monthly visitors back in 2011. As of November, 2014, there are over 3.3 million registered users and about 8.4 million questions [14l; 14k]; the monthly visitors are already beyond 42 million [14m]. That huge growth alone justifies the raison d'être of Q&A systems for knowledge management. Nowadays, the existing solutions just need to be successfully integrated into the business process.

1.1. Problem Statement

Though the Q&A concept is great for accumulating knowledge and making people more productive, it has optimization potential. The study conducted by Lou, Fang, Lim, and Peng [Lou+13] "regards quantity and quality as two important, yet distinct, aspects of knowledge contribution". Therefore, the design of a Q&A system has to reflect about these two aspects carefully.

The system becomes more valuable the more questions are covered. Thus, the system is very dependent on an active community to increase the quantity of information. This implies to motivate people to ask questions and more important to take time to answer questions. The design of the platform has to accommodate to the human drives [RH08] and come up with solutions to increase user participation. A lot of big communities, not only limited to the KM area, use Gamification to increase social engagement [ZC11].

Another important factor of Q&A systems is the quality of the knowledge. Questions and answers need to be formulated precisely and preferable not overlap with other questions. Certain quality standards ensure the sustainability of the contained knowledge. Again Gamification elements can support quality management by using voting or reputation mechanisms. Although the quantity of information within the system is a positive characteristic in general, it can lead to a loss of trust in the platform when the quality is too low. [Lou+13]

Finally, the added value of a Q&A platform is revealed by the structure of and the accessibility to the knowledge. On large platforms, a humongous amount of questions is asked. Although there are active members in a community which are able to answer a specific questions, it might be buried under the flood of information. It is vital that questions and answers can be found by anyone easily.

1.2. Motivation

The main aspects of Q&A systems – namely quantity, quality and accessibility of the information – are coupled tightly. Nevertheless, as there are already plenty of concepts [Har+08; Lou+13; RH08; ZC11] for increasing the quality and quantity of Q&A sites, this thesis mainly focuses on the accessibility aspect. A service-oriented middleware system is proposed, which allows direct integration of Q&A functionality into end-user applications. The goal is to reduce answer times and thus increase the

users' productivity. The concept involves designing a sophisticated architecture, which can be extended with intelligent logic.

Questions can be enriched with meta data, such as tags, to increase accessibility. Context information from the user's working environment helps to automatically categorize questions. Through machine learning techniques the system is able to learn about the user's interests and suggests questions to him. This reduces the problem of finding adequate questions to answer. Additionally, the response time is minimized by using push notifications.

With the rising count of different platforms it has become hard to keep track of the information flow and process them efficiently. Another goal of the proposed system is to aggregate knowledge across different sources. A central place to search for answers makes finding them more efficient. A middleware-based approach is able to provide a consistent Application Programming Interface (API) to access multiple knowledge sources.

1.3. Structure of the Thesis

The thesis consists of two - a theoretical and a practical - parts. The theoretical part tries to answer the question how to improve the quality of the data in Q&A systems. Its findings are (partially) implemented in the practical part, which is the design of a new Q&A prototype and the integration into SolidEdge.

As the thesis is conducted in close collaboration with Siemens, there is a context, which influences mostly the practical parts of the thesis. Chapter 2 describes this context through requirements, use cases and mockups for the user interface.

In chapter 3 related work concerning Q&A systems is evaluated. It provides theoretical and psychological background to understand how users interact with Q&A systems and what problems need to be solved.

The main contribution of the thesis is formulated in chapter 4. The conceptional design of the solution introduces a recommendation system for questions, which aims at improving the answering process.

Chapter 5 describes the data and the architecture of the target system. Furthermore, the API and the functionality behind are defined.

All important implementation details are summarized in chapter 6. This includes a description of the used technologies and third party libraries as well as the structure of the project.

The evaluation of the thesis is conducted in chapter 7 followed by the conclusion in chapter 8.

CHAPTER 2

CONTEXT

One of the main ways of getting corporate advantage for a company is to optimize their Product Development Process (PDP). [IK07] The thesis is conducted in close collaboration with *Siemens* in order to improve the experience in their PDP. Design is a major part of that process and thus improved Computer Aided Design (CAD) tools can greatly increase the productivity. According to Alducin-Quintero, Contero, Martín-Gutiérrez, et al. [Ald+11] a CAD model is not only a 3D geometry representation, but also stores knowledge on how to build it. Capturing and sharing this complex information is also part of the development process. In a lot of activities within the process direct interaction is needed. Whether it is sharing information or asking for assistance, social support at tool level can help to optimize the process.

A product developed and used by *Siemens* is *SolidEdge*, a 3D CAD history based, parametric feature and synchronous technology solid modeling software. Figure 2.1 shows a screenshot of SolidEdge. Although it is a starter-tier CAD tool, it is a complete design environment and has a lot of features.

For new users it can be quite frustrating to get a task done. When the user has a problem, either with the tool itself, the model or with the underlying process, he needs to ask an expert. In most cases this will not happen face-to-face, but rather filing an issue with an online tracking system or posting a question to a board. Remote users might have trouble to answer an question, when they are not fully aware about the questioner's context. It costs a significant amount of time to formulate a question with all the required background information, e.g. steps to reproduce the issue or a screenshot of the error. The integration of social features might be a way to optimize the solution finding process to common problems.

SolidEdge does not have any social features that are targeted to increase the users productivity. This thesis is part of a larger research project for the integration of social features into CAD environments. It focuses on improving the problem solving process by assisting the user in asking and answering questions. The elimination of the need to use external platforms and formulate the context manually helps the user to get answers to his questions faster.

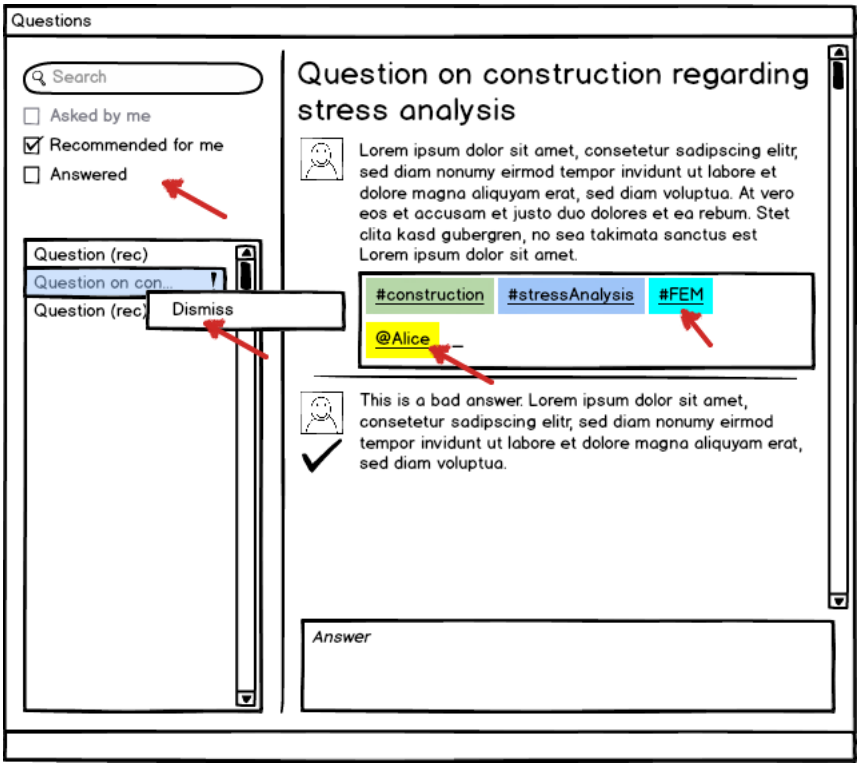


Figure 2.1.: Screenshot of the synchronous environment of SolidEdge.

The core idea besides using context information is a recommendation engine for questions. Users typically do not have enough time to constantly search for answerable questions. Thus, an urgent question may stay unanswered for a long time, although there are users, who are able to answer it. It is convenient to get notified when there is a question to answer available.

In this thesis a novel recommendation approach is developed and implemented as a Q&A service with integration into *SolidEdge*. This prototype addresses the lack of social integration in the target application. Together with Siemens, the following set of requirements, constraints, use cases and scenarios for the Q&A system have been identified.

2.1. Requirements

/ F 001 / *Core Q&A functionality*
The basic Q&A functionality contains the creation of questions and answers. A question is created by a user with a title and a body. Answers to a question only include a body and can be made by all users, including the one that asked the question. Furthermore, the questioner can mark the question as solved by accepting one of the answers. An accepted answer is highlighted in a special way and indicates the solution to the question.

/ F 002 / *Search for questions*
Users can search for questions by specifying multiple criteria. These criteria include free-text and tags. Furthermore, the user can filter questions that are asked or observed by him, or are recommended to him. The search yields a list of questions, which can be detailed to see all information including the answers.

/ F 003 / *Observe question*
A user can observe a question to express special interest in it. Usually, only the questioner is informed when a new answer is posted. When a user observes the question he is also notified about new answers. Observed questions must be easy findable using the search function.

/ F 004 / *Tagging and mentioning*
User can attach tags to questions in order to categorize them. A tag can either be a string describing the topic of a question or another user. The latter one expresses a direct relationship between the question and the user and hence is called a mention. Any user can attach an arbitrary amount of tags and mentions to a question. Tags are important for improving the search functionality and are the basis for the recommendation engine.

/ F 005 / *Profiling*
Every user has its own profile with personal information about him. This information must at least contain his name and a unique identifier. Additionally, the user's interests are saved in the profile; these can be set manually by himself and/or trained by the system. Through machine learning techniques the system is able to estimate the user's interests by examining his previous actions, especially his answered question. The interests are used together with tags in the recommendation engine to match questions and possible answerers.

/ F 006 / *Recommendation system*
In order to receive valuable answers as fast as possible, the system recommends an asked question to users, which are presumably able to answer the question. The mapping between questions and users is established by utilizing the tagging and profiling system. As tags represent the topic of a question, it can be matched against the interests profile of a user. With mentions one user can directly recommend the question to another user. Hence, the recommendation process consists of both machine learning and human interaction. Recommended questions must be explicitly presented to the user. If the user is not able to answer the question he can choose to add further tags or mentions. The recommender keeps up with his work until the question is marked as solved. The recommendation process is usually triggered by the creation of the question or when tags are added but it can also operate on schedule. This way, unanswered questions that receive no actions can be recommended to additional users.

/ F 007 / *Push notifications*
Users receive notifications of relevant events in order to reduce their response time. The push notification is displayed within the environment without requiring user interaction. The questioner and users observing the question need to be informed about answers. All users get a notification when there is a new recommended question for them to answer.

2.2. NF-Requirements

/ NF 001 / *Extensibility*
The architecture allows for an easy integration of additional knowledge sources for question and answers. Moreover additional systems can be plugged onto the middleware and post-process questions and answers. The recommendation engine is designed as an extension to the core middleware and can be easily replaced by another implementation.

/ NF 002 / *Scalability*
The core middleware and the attached systems need to scale individually through replication. This requires loosely coupled components, which can be distributed over different virtual or physical execution environments.

2.3. Constraints

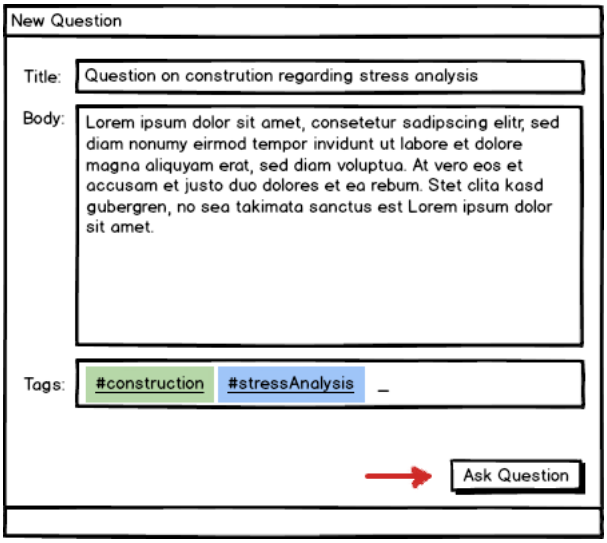
/ C 001 / *Service-orientation*
The Q&A system is implemented through a service-oriented middleware. The API needs to be consumable from different end-user applications.

/ C 002 / *Technology stack*
Node.js is used as primary technology for the middleware. This includes all required components except the integration into the end-user application. The data layer needs at least support for MongoDB.

2.4. Scenarios

Name: Ask question
Summary: John asks a technical question
Description: John is designing a driveshaft for a car with SolidEdge. He has a technical problem concerning the stress analysis of his draft and wants to ask a question via the SocialEdge plugin. He opens the dialog for asking a new question, which is sketched out in the mockup section. He fills in the title and body of his question and adds the two tags "'construction"' and "'stress analysis"'. The tags are added by writing in the tags box and starting with a hashtag. By pressing the "'Ask Question"' button in the dialog the question is sent and the dialog closes. Now, John waits until he gets notified that his question has been answered.

Mockup:

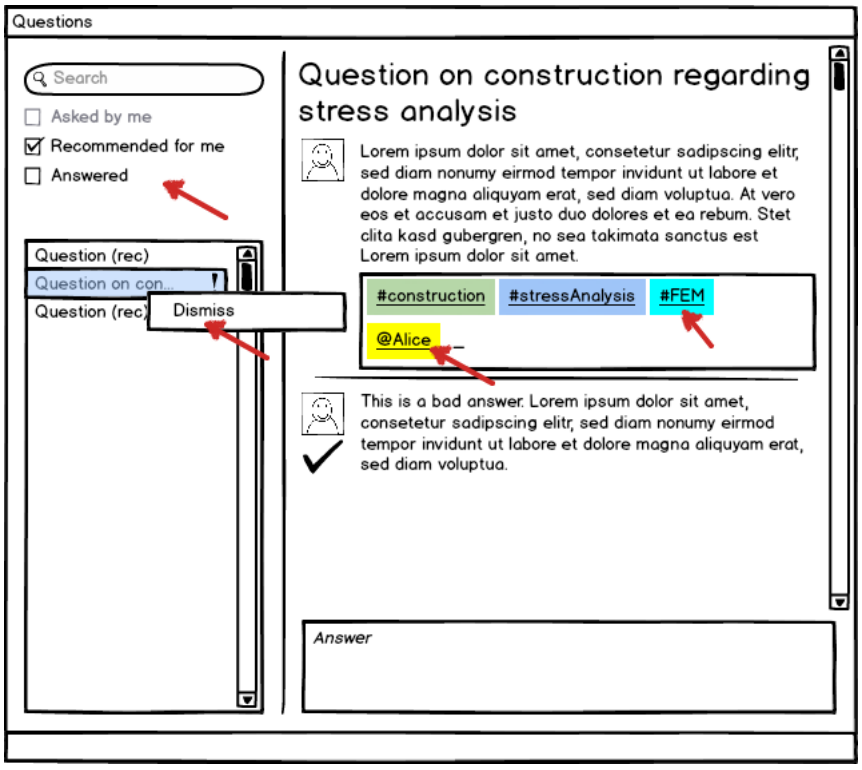


Name: Tag question

Summary: Raymond receives a recommended question and adds tags to it

Description: Raymond gets notified about being recommended for a question to answer. He opens the questions dialog (see mockup) and checks the "Recommended for me" box. A list of question titles appears and he clicks on one of his interest. The details of the question from John about stress analysis appear. Although Raymond has some technical knowledge about stress analysis he is not very familiar with FEM, which is required to solve the issue. He decides to add the tag "FEM" to the existing tags to narrow down the topic. The tag is added by writing in the tags box and starting with a hash sign. Furthermore, he associates Alice with the question because he knows that she is an expert in the FEM field. Mentioning people occurs by adding a tag, but starting with an "at" sign instead of a "hash" sign and typing the name of the user. After that he dismisses the question by right-clicking on it and selecting "Dismiss" from the context menu. The question vanishes from the list of recommended questions.

Mockup:

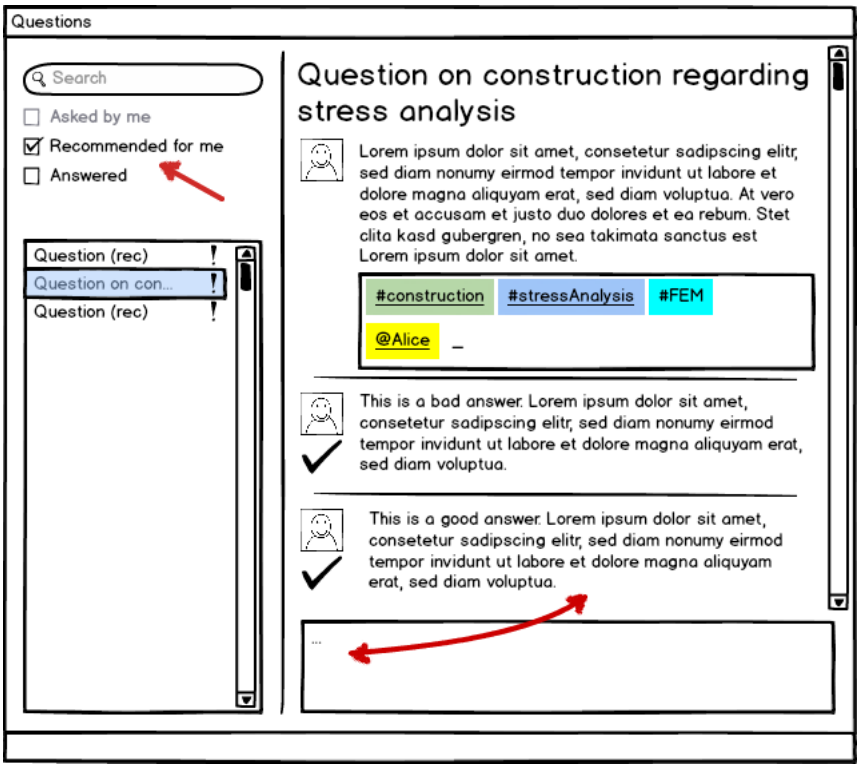


Name: Answer question

Summary: Alice receives a recommended question and answers it

Description: Alice gets notified about being recommended for a question to answer. She opens the question dialog (see mockup) and checks the "Recommended for me" box. A list of question titles appears and she clicks on one of her interest. By clicking on it, the details of the questions from John and an answer from Bob appear. Alice finds Bobs answer inaccurate and decides to write her own answer into the message field and hitting enter to send it. Her answer is immediately shown in the list of answers.

Mockup:

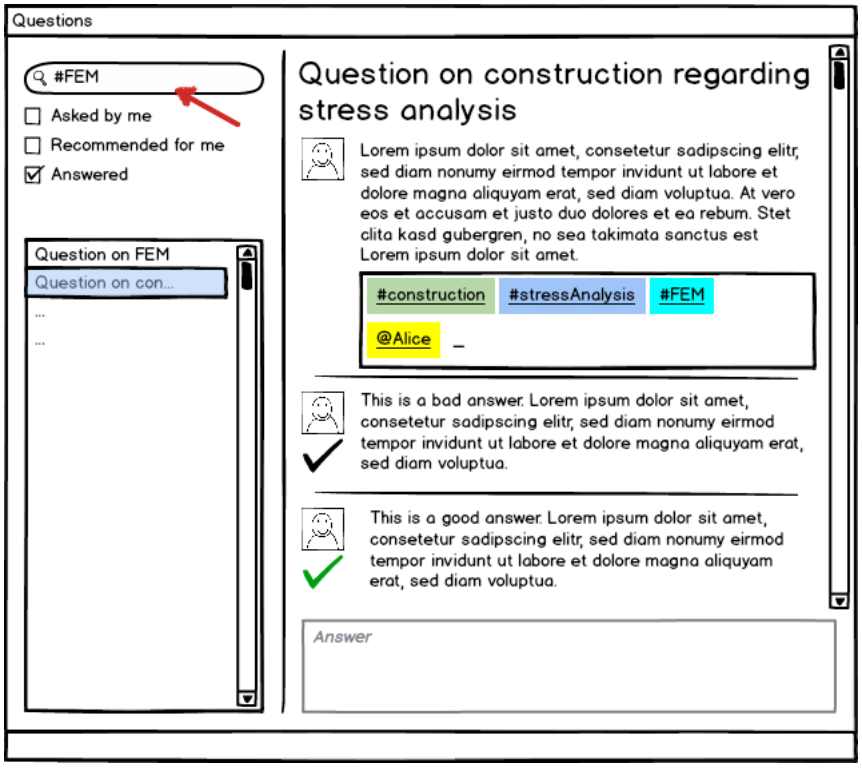


Name: Accept answer

Summary: John receives answers to his question and accepts the most helpful one

Description: John is waiting for answers to his questions. Finally, he checks the list of his asked questions by clicking the "'Asked questions"' button in the top navigation bar. A dialog appears where he sees a list with all his open questions. One question is marked as changed, so he clicks on it and the details show up. Two new answers from Alice and Bob are shown below his question. After reading and trying the proposed solutions he accepts Alice's answer, which helped him solving his problem, by clicking onto the accept button below the answer. The answer is marked as resolved and vanishes from the list of open questions.

Mockup:

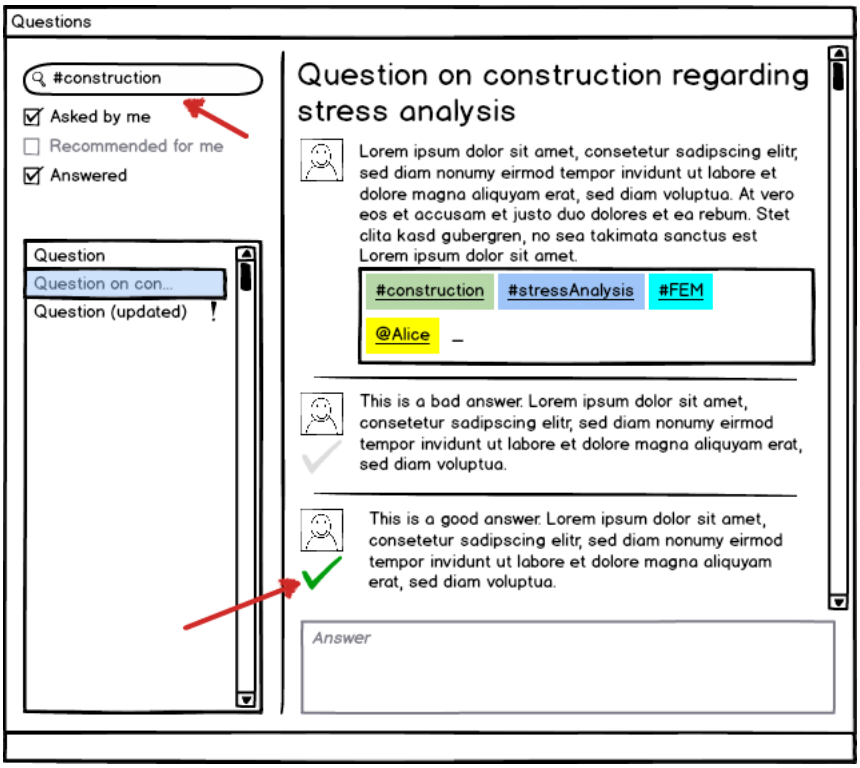


Name: Search question with answers

Summary: Fred has a question and looks for similar questions

Description: Fred has a problem with the FEM method. He knows that this is a common problem, which is already solved, but he does not know the exact solution. So he opens the questions dialog and searches for the FEM tag by entering it with a prefixed hash sign in the search box. Additionally he checks the "Answered" box to only receive questions that are answered. By clicking on the question title in the list of the search results, the details show up. Fred clicks through the list until he finds a working solution and closes the dialog.

Mockup:



2.5. Use Cases

Name: Authenticate client
Summary: The client connects to the middleware and authenticates the client
Pre-condition: The client is disconnected
Post-condition: The client is connected and authenticated
Flow:

1. The user starts the client (i.e. SolidEdge)
2. The client connects to the middleware and sends login credentials
3. The middleware evaluates the credentials and sends back an access token
4. The middleware starts pushing events to the user

Name: Ask question
Summary: The user asks a question
Pre-condition: The client is authenticated
Post-condition: The question with tags is available within the system
Flow:

1. The user asks the question
2. The client sends the question with context information to the middleware
3. The middleware saves the question

Name: Search question
Summary: The user searches a question
Pre-condition: The client is authenticated
Post-condition: A list of questions matching the search criteria is presented to the user
Flow:

1. The user fills in the criteria into the search field
2. The client sends the search criteria to the middleware
3. The middleware processes the request and sends the search results back to the client
4. The list of found questions is displayed

Name: Recommend question
Summary: The system recommends a question to experts
Pre-condition: A question is asked
Post-condition: The question is recommended for multiple users
Flow:

1. The middleware analyzes the questions and matches the question to experts
2. The question is marked as recommended for the experts
3. The experts are notified about the recommendation

Name: Answer question
Summary: The user answers a question
Pre-condition: The user has received a question, either through push or active search
Post-condition: The answer is pushed to the questioner
Flow:

1. The user answers the question
2. The client sends the answer to the middleware
3. The middleware saves the answer
4. The middleware initiates the push of the answer to the questioner

Name: Tag question or mention user
Summary: The user tags a question and/or mentions a user
Pre-condition: The user has received a question, either through push or active search
Post-condition: The question is tagged and is pushed to additional experts
Flow:

1. The user adds tags and/or mentions a user
2. The client sends the update to the middleware
3. The middleware saves the question
4. The middleware initiates the push of the question to new experts according to the updated tags/mentions

CHAPTER 3

RELATED WORK

This chapter summarizes related work which has influenced the design of the proposed middleware solution.

Various studies [NAA09; MTP10] deal with the importance of Q&A systems. In order to build a successful Q&A community, one must understand the process of asking questions and giving answers. D. R. Raban and Harper [RH08] summarize some intrinsic and extrinsic motivators for answering questions. Reputation and rewards are two powerful concepts to improve quantity and quality of quentions and answers. When people are looking for answers to a specific question they will typically either ask the question or search if it has already been asked. Hence, the two basic use cases for the Q&A system are derived thereof: asking questions and searching for answered questions.

Especially the search functionality requires more attention. In order to find suitable answers the question must be categorized properly. There is plenty of literature [GH06; Res+94; Mor08] that deals with mechanisms to improve the quality of stored data and increase the search effectiveness. Collaborative tagging proves to be very usable, as everybody would describe the same thing in a different way. When more than one user specify tags more terms are covered. On the one hand this enhances the chance to hit the right search term, on the other hand the topic of the question is narrowed down.

The typical way of asking a new question is to open a new thread in a Q&A system and wait for another user to answer it. This leads to duplicate questions and the information contained in the answers may be split and harder to find for other users. To avoid this, Jeon, Croft, and Lee [JCL05] propose to automatically retrieve answers by finding similar questions in the archives. Over time Q&A services build up large archives of questions and answers. When a user asks a question these archives can be searched and if this question has already been asked, the answers can immediately be retrieved. It reduces redundant data in the archives and eliminates the time lag until an answer is available. In contrast to the usual search paradigm, where the answers are searched through, the method in the paper proposes to look for semantic similarities between questions. It is however non-trivial to measure the distance of the meanings between

two questions. Exact the same question can be formulated using different words. The paper introduces a model to estimate the probabilities that one question semantically translates to another. In order to train the system existing similar question pairs need to be found. It is trivial to find lexicographically identical questions but semantic similar questions can be extracted by analyzing the answers. The idea is, when two answers are almost equal, then the corresponding questions are also likely to be equal. Results show that exploiting existing archives is a viable solution to find similarities between questions.

Design lessons [Mam+11] for a successful Q&A platform can be obtained from Stack Overflow. Their success is i.a. built upon the exploitation of human drives. By employing game mechanics, e.g. rating systems, competition is made productive and increases user participation. Up and down voting on questions and answers also assures quality and improves credibility of the community. Furthermore a feedback loop has been established and the users' wishes are incorporated quickly into the platform for assuring its quality and the satisfaction of dedicated users.

A low participation rate impedes the growth of community-based Q&A systems. In the paper by Guo, Xu, Bao, and Yu [Guo+08] the inefficiency of finding a suitable question for a user to answer is – besides lacking motivation – identified as one of the main causes. They tackle this problem by introducing a probabilistic model for user behavior in order to recommend questions to appropriate users. The model takes a question as input and maps it to an ordered list of persons which are able to give an answer in all likelihood. The key note consists of finding similarities between the new and previously solved questions as well as profiling users in order to build ranked lists of experts on various topics.

According to Adomavicius and Tuzhilin [AT05] a recommendation is based on the maximization of a utility function. A utility function outputs the value for a user and an item. Given an user, one want to recommend the item which yields the most utility. The utility function can be an arbitrary function, i.e. an profit function, but is usually based on user ratings. There are three major types of recommendation systems, which are content-based recommendations, collaborative recommendations and hybrid approaches. Content-based recommender systems recommend items based on a user's previously used items. In collaborative recommendations the items of other users, which share the same tastes are recommended. It is possible to combine both approaches, as they both are based on user history and the relevancy between users and items. Machine learning techniques are used to estimate the relevancy and can be trained with data from existing services.

In order to further decrease the time to find a solution for a problem task-specific search interfaces must be integrated into the development environment [Bra+10]. This has also the benefit of being able to exploit environmental context information to automatically categorize questions though machine learning [Seb02]. Scoble and Israel [SI14] describe the "Age of Context" in their eponymous book as an epoche driven by the *Five Forces*: mobile, social media, data, sensors and location. Location – as one

of the most important force – is vital for predicting the user's intentions and starting an appropriate action automatically. In the context of Q&A a question might yield different answers if the questioner's location is known. Nevertheless, location is tightly coupled to mobile solutions, as the physical location of a desktop computer does not yield as much useful information. For the integration of Q&A services into an end-user application a broader interpretation of location has to be found. Schmidt [SBG99] states that there "is more to context than location" and shows some examples using time as context information. To generalize the concept of context, they propose to see context information as a description of the situation and the environment a device or user is in. In a traditional working environment, information of the used software, e.g. name or version number, and his last actions can be leveraged.

A good way to maximize user satisfaction is using an User Experience (UX) driven design process. It aims on improving the usability of a system's interface by addressing all aspects of a system as perceived by users. An important part in the UX design process is the collection of usability metrics. In Albert and Tullis [AT13] a variety of UX design metrics is presented and discussed in a product and technology neutral way. A metric is defined as a way to reliable measure or evaluate phenomena; using the same set of measurements should yield comparable results. All UX metrics need to be observable and quantifiable, and represent aspects of user experience like effectiveness, efficiency or satisfaction. For the development of a prototype, it is important to establish a set of metrics against which future versions can be compared. It is the best way to see if the design is improving or not. There are essentially three types of UX metrics: performance metrics measure the user's behavior, self-reported metrics are based on what the user shares and behavioral or physiological metrics capture reactions of the human body as a result of the experience of the interaction with a user interface. According to the book, the most common method for collecting usability data is a lab test. Typically five to ten participants are needed; they need to conduct some product specific tasks and answer questions about them. The goal is to collect performance data such as task success, errors and efficiency.

CHAPTER 4

CONCEPTUAL DESIGN

This chapter describes the features of the solution in an abstract way and reasons about the design decision that were made.

4.1. Question and Answers

There is a basic set of functionality, which is present in all major Q&A community sites, like *StackOverflow* [14j], *Quora* [14f] or *Yahoo Answers* [14o]. Obviously, this includes the creation of questions and the possibility to find and answer them. In the prototype a question consists of a title and a body whereas an answer only has a body. The question can also be answered by the questioner. This is valid, because a self answered question also adds valuable information to the system and is better than deleting the question. Deleting questions and answers is also unsupported in order to preserve information; questions can only be edited by the creator. Otherwise, valuable information contained in the answers would get lost, when the corresponding question is deleted.

4.2. Search

Answering questions or finding information by looking for similar questions both require a flexible search functionality. Questions have a title upon which a full-text search can be performed. The questions in the systems are not organized in any type of hierarchy. There is no forum-like grouping of questions according to topics and subtopics. Instead every question has an arbitrary number of so-called tags attached to it. These keywords indicate the topic of a question. Figure 4.1 depicts the search process. The query contains a string that is searched for in all questions titles. To narrow down the search results of the full-text search, they are additionally filtered by a list of tags. A question matches the filter when it has at least one of the provided tags attached. This way a user willing to answer questions can enumerate all questions of interest by filtering an empty full-search text by a list of tags that are relevant to him.

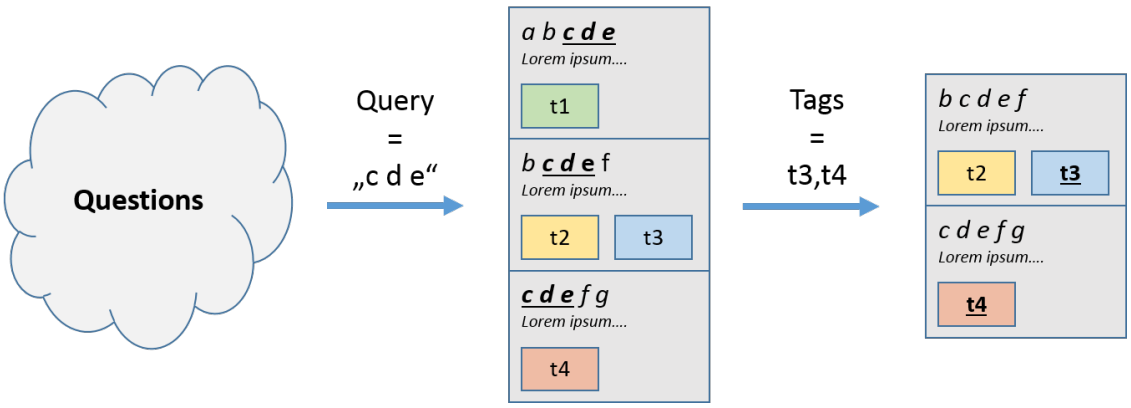


Figure 4.1.: The search process of the prototype. In the first step the query is searched within the question titles. The query needs to be included literally in the title. In the second step the resulting questions are filtered by tags. At least one tag must match in order to pass the filter.

4.3. Watch and Push

One goal of the thesis is to minimize the overall time a questioner needs to wait before he has the final solution to his question. A considerable amount of time is consumed for waiting. Besides the time a user needs to formulate an answer to a question, he first needs time to find an appropriate question; additional time passes until the questioner reads the answer. When a person asks a question, he usually does not wait until it is answered, but rather continues his business and periodically checks if an answer is available. To reduce this reaction time, the questioner needs to be informed about an available answer. The system sends out push notifications to relevant users, indicating that a relevant event occurred in the system. As not only the questioner himself, but also other users might have interest in the question and its answers, they need a way to express interest. Users can explicitly watch a question and will – like the questioner – get notified about relevant events, such as a new available answer. A *watch* is a relation between a user and a question.

4.4. Recommendation

The main contribution of the thesis is a unique recommendation process. The system actively pushes a question to users that interest profiles match the topic. The recommendation process is a recurring task, which users can influence by adding tags to a question or directly mentioning another user. Figure 4.2 shows the complete answering process, where a chain-of-responsibility for the question is established. When a user asks a question the system determines appropriate users and pushes the question to them. Users who cannot answer the question directly but have some insight in the topic can add appropriate tags to classify the topic. This makes it easier for another user to

find the question and it helps the system to find additional users, which the question can be pushed to. Furthermore if a user knows other users that might be able to answer the question, he can mention them in the question. Eventually, the question receives one or more answers, which are reported back to the questioner. The recommendation process stops when an answer is accepted. A *recommendation* is a relation between a user and a question.

A recommendation is issued automatically by the system. Figure 4.3 shows how a recommendation is issued by the system. The idea is to identify the topic of the question and match it with the interests of users. There are two – a manual and an automatic – approaches, which can also be used together. Both, the topic of questions and the interest of users can be represented by a statistical model. In their paper, Guo, Xu, Bao, and Yu [Guo+08] model them as distribution over terms. They suggest extracting the terms that describe the topic of a question using Natural Language Processing (NLP) and the user interest based on his answer history. By using this model, the probability for every user that he can answer the question can be estimated.

Of course these terms can be set manually. Every user is responsible for his interests by specifying terms in his interest profile. A numeric value is used to describe how deep his interest in the subject is. The topic of a question is derived from the tags attached to the question. The questioner as well as every other user can freely add tags – similar to collaborative tagging [GH06] in social bookmark sites.

4.5. Relations

Although, a user is notified about relevant events, he might miss them or need to search through them. An event log captures all occurring events in chronological order. The search functionality is extended by another option, where a relation between the user and the questions can be set as filter. Possible relations between users and questions are:

- **asked**: show only questions that are asked by the current user
- **watch**: show only questions that are watched by the current user
- **recommendation**: show only questions that are recommended for the current user

4.6. Contextual Information

Contextual awareness allows the system to take actions automatically based on the user's environment. Big data is an important aspect of context, as the system needs to learn about the user's behavior and more data enhances the training process. The prototype is meant to be integrated into end-user-application, specifically CAD tools like *SolidEdge*. Therefore, typical context information include information about:

- the software itself: e.g. name and version

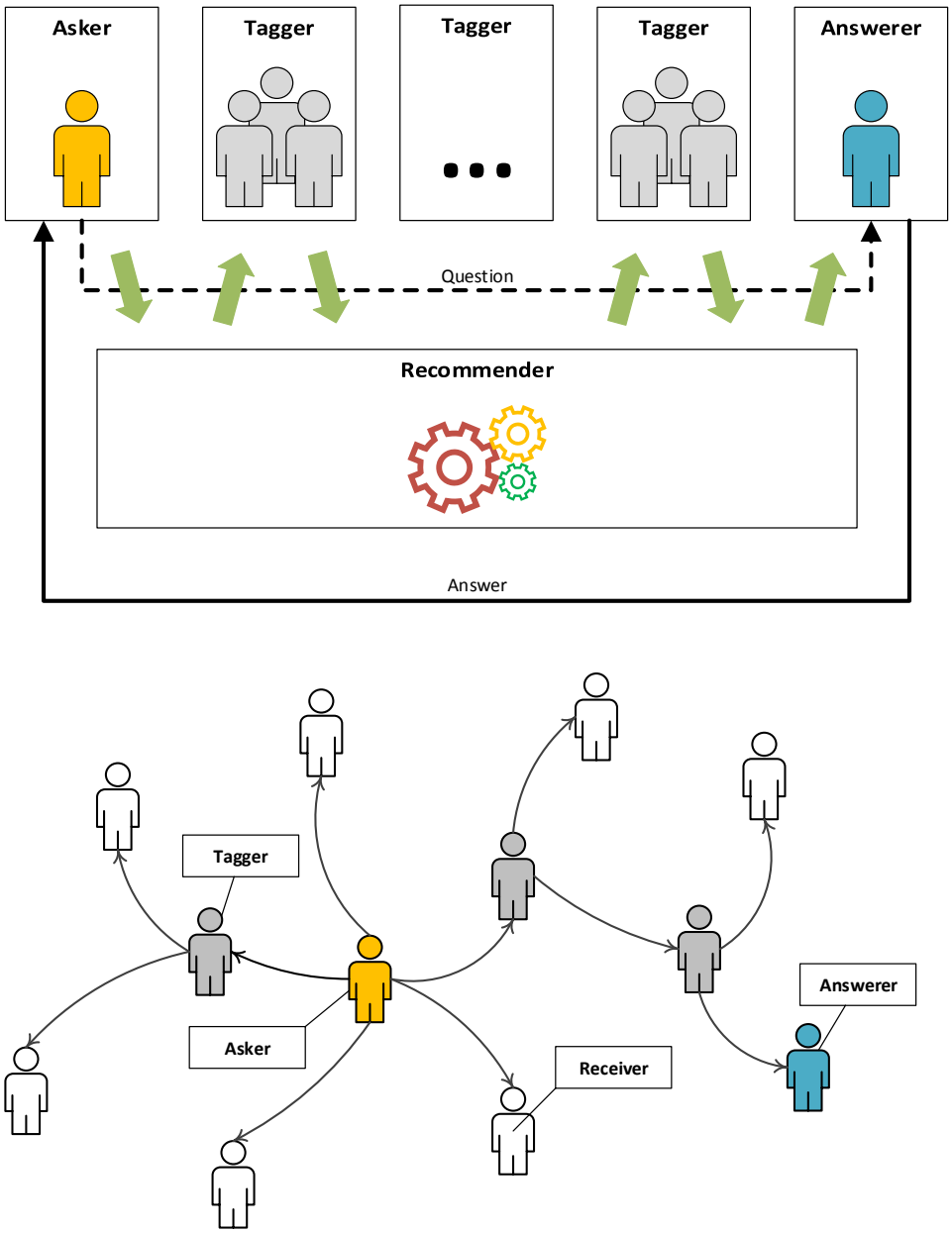


Figure 4.2.: Chain-of-responsibility for the question. The questioner asks a question which is pushed out to users until an answer is found. Every user adding tags or mentioning people causes the recommendation process to start. The bottom half of the picture shows a possible graph of the causal relationship on how the answer has been found.

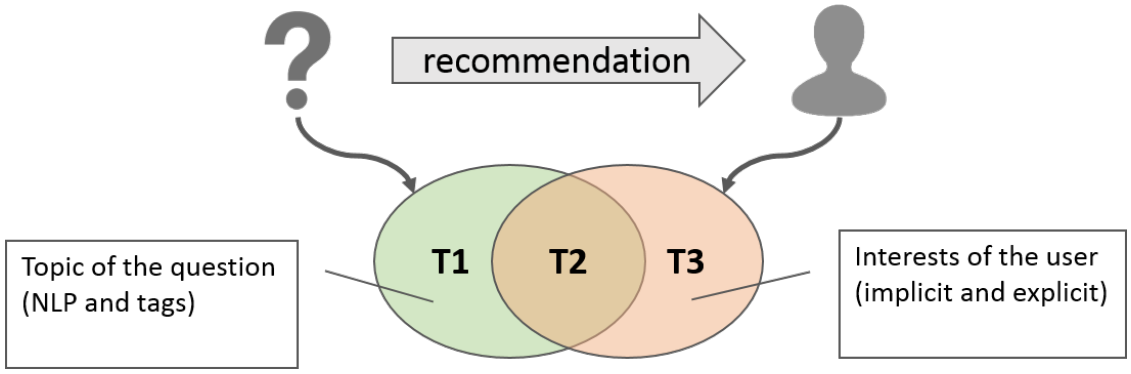


Figure 4.3.: Automatic recommendation of a question for a user. The topic of a question is matched with the interests of a user. Both can be declared explicitly (by adding tags to questions and specifying interests in the profile) or implicitly (by using NLP on the question and analyzing the user's answer history).

- the environment: e.g. opened view within the software and last action taken
- the document: e.g. type and path

The context information is collected every time a question is asked. A use case for using context information is to influence the recommendation process. A user asking a question in the context of a specific tool, might not have a technical issue rather than a problem with the handling of the tool itself. It makes sense to prefer pushing the question to users that operate out of a similar context.

4.7. User Management

The account concept ensures that every actor in the system is identifiable. Every user needs to register with the system by supplying authentication information and profile details. For a prototype, registering with name, email and password is sufficient. The name will be displayed in the interface, but it is not unique and cannot be used for authentication. The email address is unique for each user and is in combination with a password used for login. Only users which are logged in can interact with the rest of the system (all but register and login).

CHAPTER 5

SOFTWARE DESIGN

This chapter details the conceptual design of the previous chapter by describing the data and the architecture of the target system.

5.1. Data Design

This section describes the data model of the Q&A service. Figure 5.1 is a graphical representation of the main model in UML. The model is mainly built around the two entities User and Question. Every entity in the model is extended by two attributes `createdAt` and `updatedAt`, which contain the timestamp from the creation and the last update of the instance, respectively. These two attributes are omitted in the diagram for simplicity reasons.

A user consists of a `name` and the login credentials, `username` and a `password`. All other entities are aggregated to User, because all instances are created in the context of a particular user and therefore "owned" by him. A user also composed of instances of Interest with the attribute `term`, which defines a single topic the user is interested in.

A question is represented by instances of Question, which have a `title` and a `body`. Furthermore a question can have multiple answers and tags, represented by Answer and Tag. An answer is consists only of a `body`. There is an additional relation, accepted, which optionally links a question to one of its composed answers, when it is marked as the accepted answer.

Tag represents a tag of a question and has a sole attribute `term`. User mentions are represented by Mention, which has no attribute and only acts as a tertiary relation between and a question and a user in addition to the owning user. An instance means that the owner mentioned the `target` in the `question`.

A custom association of the owning user and a question with a certain `type` can be instantiated in the Relation entity. The meaning of the association is dependent of the type. Setting the type to *recommendation* defines a recommended question for the user. Setting the type to *watch* indicates interest of the user in the question.

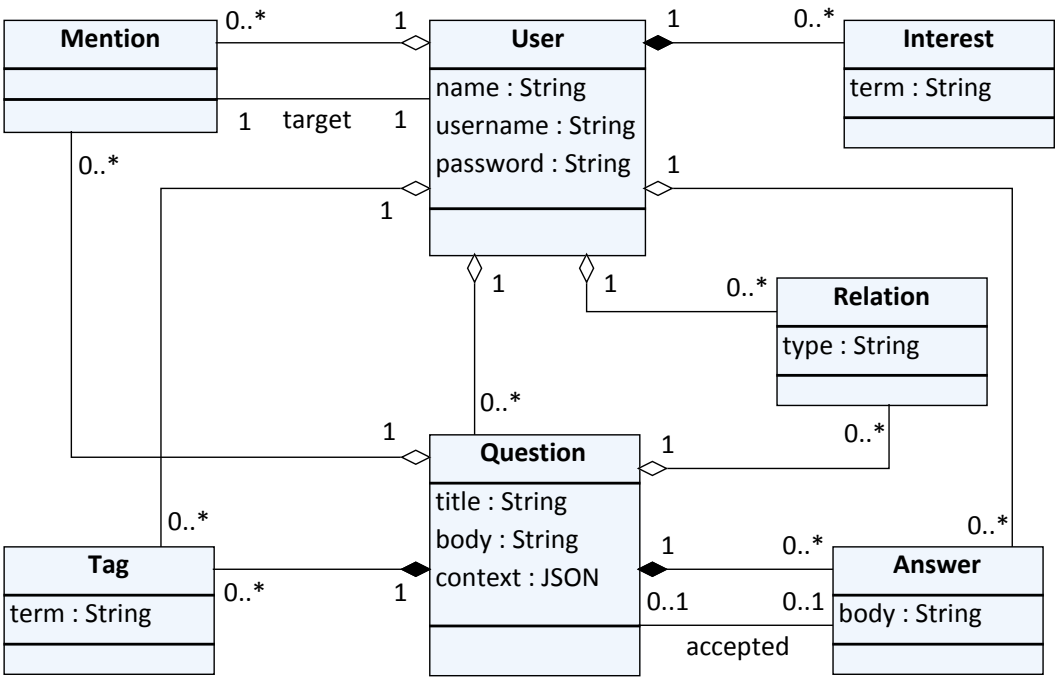


Figure 5.1.: Data model of the QnAaaS back-end. Every model instance has a reference to the "owning" user, that created to instance.

5.2. System Architecture

Since the goal is to integrate the functionality into existing end user applications through plug-ins, a classical client-server-architecture is used. The plug-in communicates with the back-end through a simple Representational State Transfer (REST) API. REST is chosen, because it is very lightweight and a de-facto standard in web service development. This way, the API can be easily integrated into other applications.

Basically, the system is organized in a 3-tier-architecture with a message bus system. Figure 5.2 details the architecture at component level. The client only communicates directly with the application layer of the core middleware which consists of the `QuestionProvider` and the `PushService`.

The `QuestionProvider` contains the main business logic for all the Q&A functionality and exposes the API of the service. It authenticates the user by using an external `AuthService` and checks if he has sufficient permissions for the request. It uses the `PushService` to register a user for push notifications and delegates model operations to the `AccessLayer`.

The `PushService` is responsible for delivering push notifications and has three different interfaces. One public for the `Client` to receive messages, one internal for associating

users and clients, which is used by the `QAProvider`, and one, which is available to external components for issuing push notifications. The push service addresses devices instead of users in order to remain compatible with existing push service, which are required if a notification needs to be delivered to a mobile device. The big services, Apple Push Notification Service (APNs) and Google Cloud Messaging (GCM) for iOS and Android, respectively, only distinct between devices for sending push notifications. The client obtains the id for his device and issues a request to the `QuestionProvider` to setup the push notification for the user. That way, the push service does not need to care about authentication details and only needs to maintain a mapping between users and devices.

The data tier consists of a thin abstraction layer, the `AccessLayer`, and the actual database. It allows to aggregate data across multiple external knowledge sources. A `KnowledgeSource` can for example be a connector to an existing Q&A service like *StackOverflow* [14j]. The access layer exposes CRUD operations on all entities of the model defined in section 5.1. Furthermore, the access layer acts as a message bus where all model operations are broadcasted as events. External services can listen to these events in order to perform additional tasks.

The `Recommender` and the `Watcher` are two external services, that listen to the events of the access layer and make use of the interfaces exposed by the core middleware. The watcher is responsible for sending question updates to interested users, i.e. the creator and users, who watch the question. The recommender is responsible for recommending questions to users and notifying users about it. Both use the push component of the core service to deliver the notifications.

Figure 5.3 is a deployment diagram, showing the partitioning of the components. The application layer, the data layer, the watcher and the recommender are designed (but not required) to reside on different machines. Furthermore, the components can be individually scaled though replication.

5.3. Procedural Flow

This section details the steps necessary to fulfill the two main use cases, getting an answer to a question and retrieving existing information. The first use case can be split up in three, so, together there are four steps:

- asking the question,
- adding tags (or mentions) to the question,
- answering the question and
- searching for the question.

These four steps are tasks initiated by a user and the first three cause the system to react by starting the recommendation process or sending push messages. The following

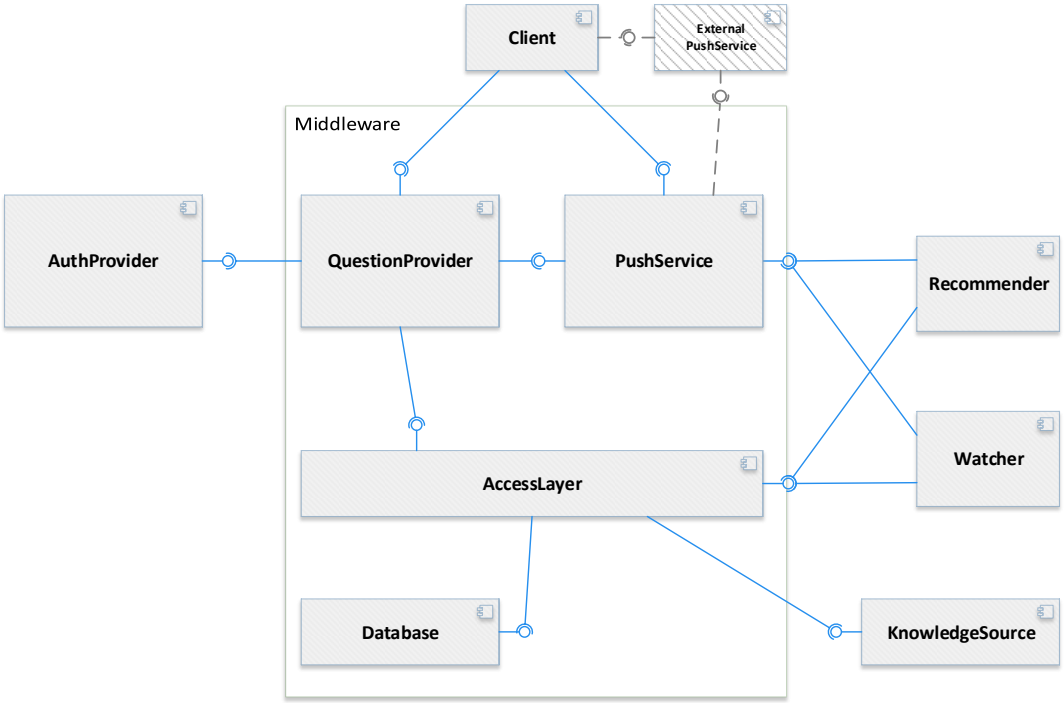


Figure 5.2.: Architectural overview containing the main components and connectors of the Q&A system.

sequence diagrams illustrate the process of each step and the interaction between the involved components.

Figure 5.4 shows the process of asking a question. The `Client` sends the question to the `QuestionProvider` in the backend. The provider is charge to store the using the access layer and returning the created question to the client. Furthermore, the `Recommender` is notified about the new question and starts the recommendation process. Every time a question is recommended it is passed on to the `PushService`, which is responsible to deliver the message to the client of the receiving user.

The process of adding a tag to a question is detailed in figure 5.5. The `Client` sends the tag request to the `QuestionProvider`, which creates the tag and sends back the results. The questioner and watchers are notified about the change by the `PushService`. The rest of the process is exactly like when a new question is asked. The `Recommender` is notified, gets active and pushes out notifications to recommended users. Mentioning a user is omitted in the diagrams since it is essentially the same process as tagging with the difference, that exactly one user receives a recommendation.

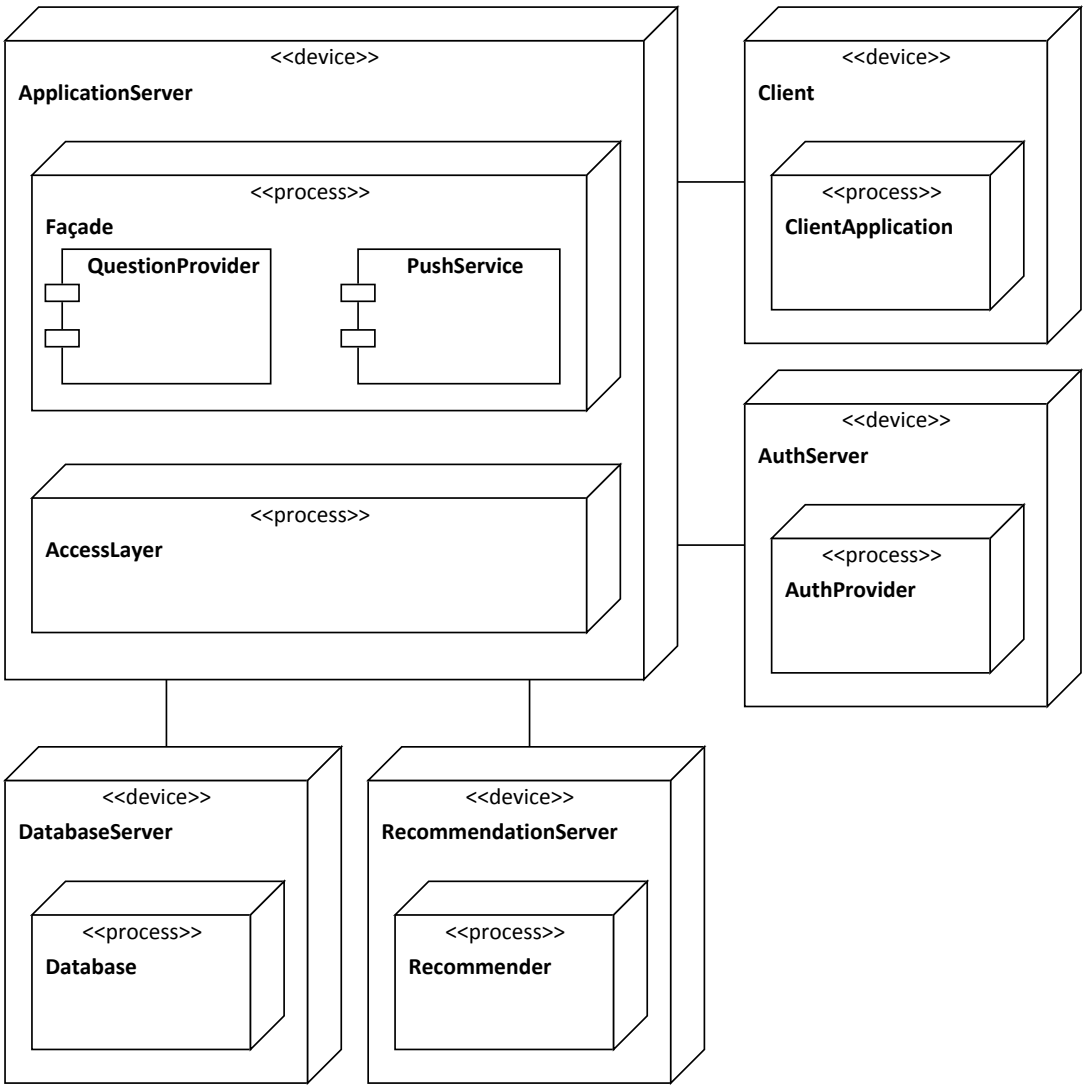


Figure 5.3.: Deployment diagram showing the physical distribution of the components.

The answering process is illuminated in figure 5.6. The `QuestionProvider` creates the answer with the data layer and responds to the `Client`. Again, questioner and watchers are notified via the `PushService`.

Finally, figure 5.7 details the search use case. The `QuestionProvider` evaluates the query from the `Client` and fetches the data using the `AccessLayer`. Data filtered data is afterwards returned to the client.

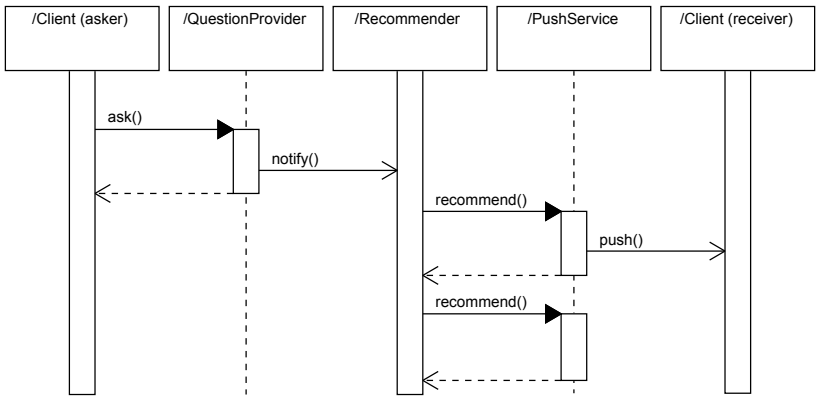


Figure 5.4.: Sequence diagram showing the process of asking a question.

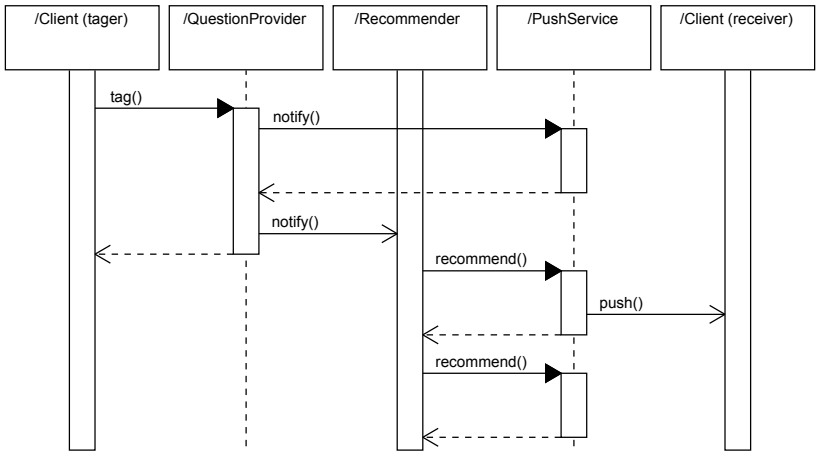


Figure 5.5.: Sequence diagram showing the process of tagging a question.

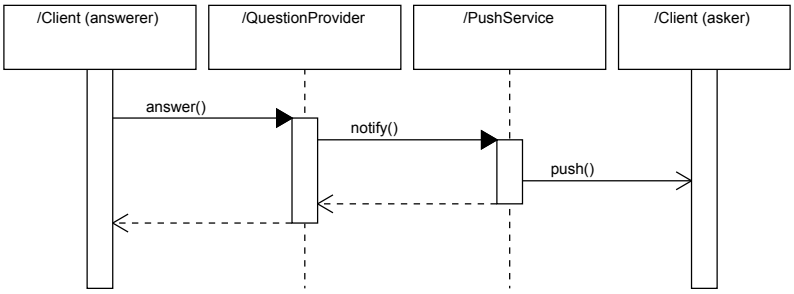


Figure 5.6.: Sequence diagram showing the process of answering a question.

5.4. API Design

The API of the core middleware exposed to client is REST conform. REST is an architectural style, which consists of best practices for scalable web services [FT00] and

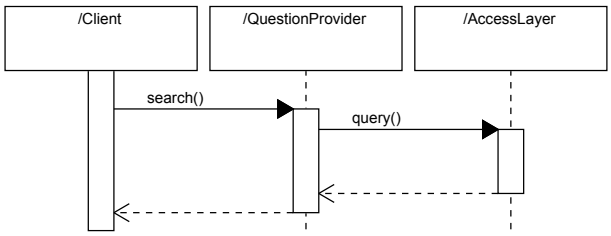


Figure 5.7.: Sequence diagram showing the process of searching for question.

widely adopted for designing web services.

A key principle of this architectural style is stateless communication. Every request to an endpoint is independent, has no side-effects and thus always yields the same result. The user has no session in the system and every request needs to be authenticated separately. For this reason a token based authentication mechanism is chosen. On successful login, an access token is generated and transferred to the client. This token uniquely identifies the user and needs to be attached to every request.

Every action in the system is identified by a Uniform Resource Identifier (URI), but there is no definite common naming convention. The API of the prototype follows the convention of the *Sails.js* [14h] framework, whenever it makes sense. It basically maps the verbs *POST*, *GET*, *PUT* and *DELETE* to CRUD operations. The structure of a request is:

[verb] [protocol]://[host]:[port]/[route]?[parameters]

The protocol, host and port part of the request address the system as a whole and therefore never change (in a simple setup). The parameters are a list of key-value-pairs passed to action. Consequently, the verb and the route part uniquely identify the action. The basic mapping between the action and the CRUD operations are defined in the table below. The controller is a part of the Model View Controller (MVC) pattern and usually operates on the eponymous database model. Every model element has a unique identifier *id*.

Verb	**Route**	$\longrightarrow$	**Operation**	**Description**
POST	/[controller]	$\longrightarrow$	*CREATE*	Create a new model element
GET	/[controller]	$\longrightarrow$	*READ*	List all model elements
GET	/[controller]/[id]	$\longrightarrow$	*READ*	Return the specified element
PUT	/[controller]/[id]	$\longrightarrow$	*UPDATE*	Update the specified element
DELETE	/[controller]/[id]	$\longrightarrow$	*DELETE*	Update the specified element

Table 5.1.: The standard mapping between REST endpoints and CRUD operations.

Complexer operations need either be spitted up in smaller ones or - if atomicity is preferred - be mapped to special actions. The prototype uses following forms:

[verb] /[controller]/[action]

[verb] /[controller]/[id]/[action]

The verb depends on the actions' characteristics and tries to follow the REST principles:

- *GET* is used when the action only returns data. Firing the same request twice in order will yield the same results.
- *PUT* is used when the action modifies data but is idempotent. Firing the same request a second time in order has no effect.
- *POST* is used when the action creates new data or has side-effects. Firing the same request twice will have some impact.
- *DELETE* is used when the actions destroys data and has irreversible effects on the resource. Firing the same request a second time in order has no effect.

The API documentation in appendix A contains a list of all exceptions to the standard mapping.

CHAPTER 6

IMPLEMENTATION

The implementation of the prototype is split into two parts:

- the backend service, *QnAaaS* (Questions and Answers as a Service), and
- the plugin for the SolidEdge integration, *SocialEdge*.

6.1. QnAaaS

6.1.1. Used Software

As required by the constraints, the backend service builds upon Node.js. Sails.js is a popular MVC framework for Node.js. It features automatic generation of REST APIs based on data models and is therefore a perfect fit for rapid prototyping. These models also allow to define the data structure in a database-agnostic way and the Object Relational Mapper (ORM) already has many adapters to support different database implementations, including MongoDB. The ORM has support for associations between models, even across different database implementations. Sails.js is built on top the famous express.js framework and every middleware for express.js can be simply reused in Sails.js. It also has excellent real-time support through deep integration of *socket.io* [14i], a websocket abstraction library. The SocialEdge plugin makes heavy use of *socket.io* for the push notification feature.

Messaging is used in *QnAaaS* for the communication between the core middleware and the plugins. The core middleware emits important events as messages and external plugins listen to them. The message broker RabbitMQ [14g] is an open source implementation of the Advanced Message Queuing Protocol (AMQP). It is written in *Erlang* and designed to built scalable messaging systems. Messages are stored in queues and buffered until they are delivered. Different routing methods allow for broadcasting or directed distribution of messages. This is used to implement a message bus architecture. The loose coupling allows to add plugins and remove them without the need to

restart the core. It is even possible to scale all parts of the system individually through replication.

Passport [14e] provides "simple, unobtrusive authentication for Node.js". It supports over 140 authentication strategies, like different OpenID and OAuth providers, Lightweight Directory Access Protocol (LDAP) and Active Directory, or just simple login with username and password. It offers a modular approach to authentication, so-called strategies. The prototype uses a combination of the *local* authentication strategy with username and password and *bearer* authentication strategy. The *local* authentication is used to generate a token, with which the user signs his requests. The *bearer* strategy finds the user associated with the token and authenticates the requests. No session is required on the backend; every request is independent.

6.1.2. Data Layer

The data layer consists of the Sails.js models, which are an exact transformation of the general data model defined in section 5.1, and the messaging system. The RabbitMQ based messaging system is depicted in figure 6.1. *QnAaaS* requires two types of communication: unicast for request and broadcast for events. This is achieved with two different exchanges. An exchange is virtual target for messages with a specific routing method. Queues can be bound to it and receive messages according to the routing method and the used binding key. The *fanout* exchange replicates a message to all bound queues, while the *topic* exchange forwards the message only to those queues that binding keys match the routing key of the message.

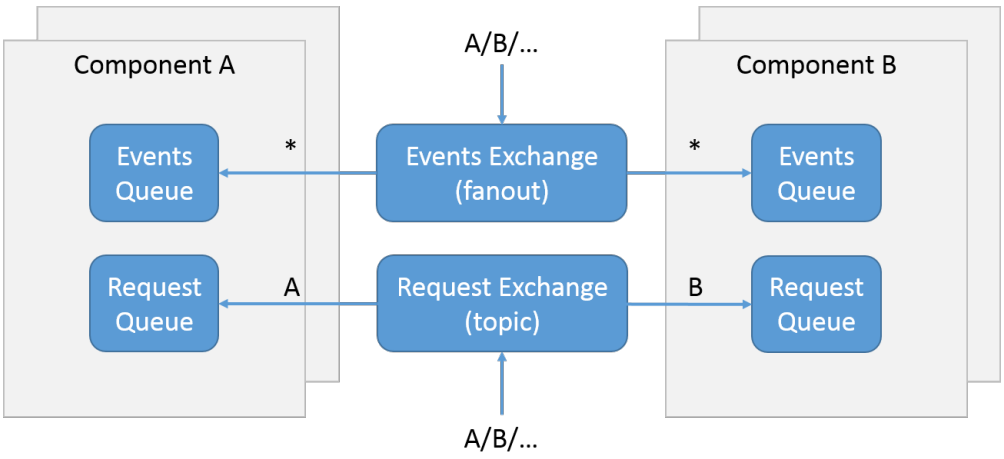


Figure 6.1.: The messaging system of the data layer uses RabbitMQ exchanges. Every component can bind to the two exchanges in order to receive messages and also enqueue messages to them.

The data layer uses the *fanout* exchange for events; whenever a model instance is created or updated it sends a message to this queue containing the model name, the

action and the data. The recommender and the watch component each bind a queue to the events exchange to observe model changes and react to them. The request exchange is used to make requests to a specific component. Every component that wishes to take requests binds a queue to the request exchange using the component name as binding key. The push component binds to the request exchange in order to take push requests from other components. The recommender and the watch component use the exchange to enqueue push requests while the push controller of the core middleware uses it to register the client to receive push notifications. The data layer listenes to directed requests under the routing key **access**. It expects a model name, an action name and action parameters as arguments, which are passed to the corresponding ORM action. RabbitMQ has two special header fields **replyTo**, which is the queue name where the response of the request is send to, and **correlationId**, which can be chosen freely and is also set in the response.

6.1.3. Application Layer

The application layer mainly consists of controllers together with route configuration and policies. The controllers implement the actions defined in chapter 5.4; the route configuration is responsible for the mapping between an URI and an action. Policies are functions which can be composed and bound to a route. They run before the action and are used to authenticate and do access checks. Except for user creation and login every request runs through the *bearer* policy, which checks if the request is signed with a valid access token and fetches the according user. If no or an invalid token is supplied, the request is aborted and the Hypertext Transfer Protocol (HTTP) status code [14b] 403 (forbidden) is returned. The generic *simpleRest* policy is another standard policy for all controllers which are backed by a model. It modifies the body of all *POST* requests so that the user field matches the current user. Every model has a attribute *user* which identifies the owning user. For *PUT* and *DELETE* requests it checks if the addressed instance is owned by the requesting user or denies the request with 403. The policy ensures nobody can create model instances on behalf of another user or make modifications to an unowned resource. An exception is the route which leads to the accept answer question. This route needs a special policy for access control, as not the owner of the answer but rather the owner of the related question has the authority to accept an answer. Watching a question, registering for push messages and logout are actions, which do not need a policy for access control; they only require a valid access token.

Another part of the application layer is the push component. It hooks into on connect event of Socket.IO to associate the client's id with the socket object. The socket object is necessary to send a message to it. In order to send a message to a user, the user must be associated with the client' id. This is realized via the push controller's only action: subscribe. The client calls this endpoint with the id as parameter and from now on the push component has a mapping from a user to one or more socket objects. It is possible for a user to receive pushes to multiple devices. The push component listens to directed

requests from RabbitMQ under the routing key **push**. A push request consists of the user's id, an event name and arbitrary data. The event name and the data is passed to Socket.IO

6.1.4. Watcher and Recommender

The *watcher* is a plugin listening to answer creation events from the access layer. Whenever a new answer is created, a push request is issued to the owner of the answered question. The push message uses the event name **answer** and sends the question, the answer and the current date/time as payload.

The *recommender* listens to create and update events of questions. This includes also the modification of related tags and mentions. Then the recommendation process starts, where the tags are matched against the users' interests. The resulting users together with the users that are directly mentioned are notified via push messages. The event name **recommendation** is used and the question and the current date/time are transmitted as payload.

6.2. SocialEdge

SocialEdge is the name of the plugin, which integrates the Q&A features into SolidEdge. It is written in C# using .NET Framework technology. To create a plugin, the *Solidedge.Community* package must be installed via the package manager. The main file of the plugin needs to extend the base class *SolidEdgeAddIn*. The plugin API allows to register custom controls in the ribbon bar and the side bar of SolidEdge. There are six different views in the plugin, the

- *Login* view, where the user logs into the backend, the
- *Profile* view, where the user manages his interests, the
- *Search* view, where questions are searched and filtered, the
- *Detail* view, where one question with answers is shown in detail, the
- *Ask* view, where a new question is asked, and the
- *Events* view, which lists all recent incoming push notifications.

The ribbon bar shown in figure 6.2 is the entry point for any activity. It has buttons to open all views except the detail view, which opens when detailing a list entry in the search or event view, or after a new question is asked.

SolidEdge provides a second interface for plugins to register UI components. Every plugin can register a tab in the edge view. SocialEdge uses this tab to directly integrate the search view, because it has the most important functionality. A screenshot of the integration with the edge-bar can be seen in figure 6.9.

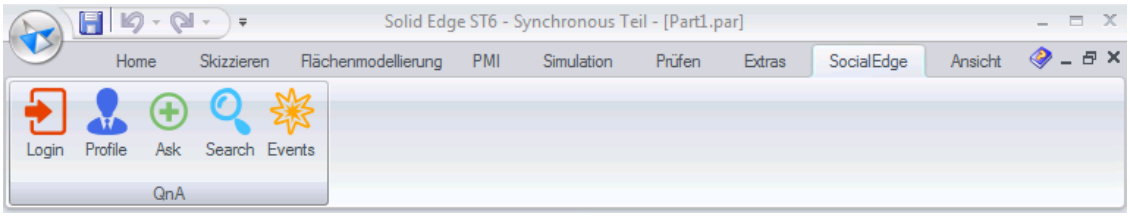


Figure 6.2.: Screenshot of the top ribbon bar of SocialEdge.

For displaying events SocialEdge creates a custom tray icon in Windows. Clicking on the icon will open the events view of SocialEdge. Furthermore, a popup notification is shown, whenever an important event occurs.

All views are designed using WPF, which favors using the MVVM pattern. They are described in XAML, a markup language, and each has its own view model attached to it. View models contain all the dynamic properties and actions, which are needed in the view. All view models communicate with the actual model to synchronize data. The model defines the mapping from actions to the REST endpoints and abstracts the transport protocol. SocialEdge uses Socket.IO not only for real-time messaging, but also as transport for the communication with the REST endpoints.

6.2.1. Views and View Models

Every view in SocialEdge is backed by its own view model, which defines all dynamic properties and actions. A view model is responsible to react to changes in the model and update the view properly. Furthermore, whenever the user makes an action, like clicking a button, the corresponding handler in the view model is executed. It is responsible to call the appropriate model function, process the results and refresh the view.

The login view in 6.3 is opened by pressing the login button in the top ribbon. The user must enter the url of the backend service as well as his credentials. By clicking the Login button the credentials are authenticated against the backend. On success the url and credentials are stored in application data and are used to automatically login the user when SolidEdge starts.

The profile view in figure 6.4 shows a list of the users' interests. New interests are added by entering the term and the skill level and pressing the Add button. By selecting an interest and clicking on Remove, it is deleted.

The search view seen in figure 6.5 is splitted in the filter part on the left and the results part on the right. In order to search for a title the first input is used. It can be left blank for matching all questions. Additionally, a comma-separated list of tags to filter can be entered in the second input field. The options below allow to filter for a certain relation between the user and the question. Only matching questions are shown in the list in the right. Double-clicking an item opens the question in the detail view.

A question can be asked using the ask view shown in figure 6.6. Title and body are entered in the text fields on the top. Below there are two lists with an autocomplete field

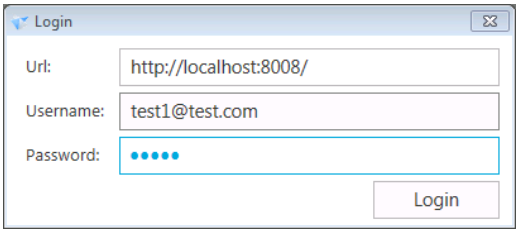


Figure 6.3.: Screenshot of the login view in SolidEdge.

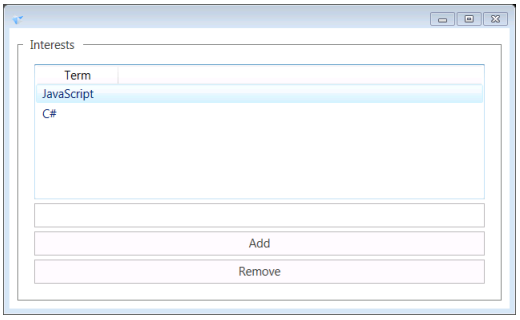


Figure 6.4.: Screenshot of the profile view in SolidEdge.

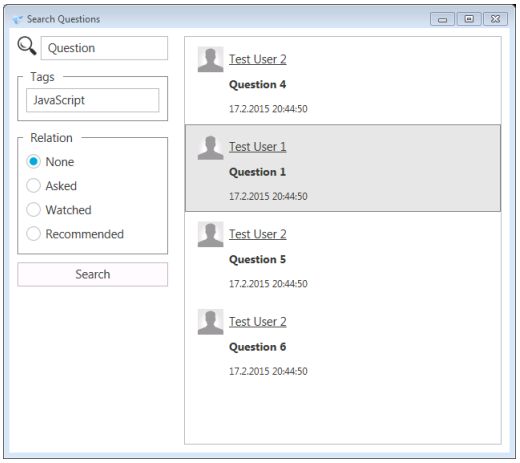


Figure 6.5.: Screenshot of the search view in SolidEdge.

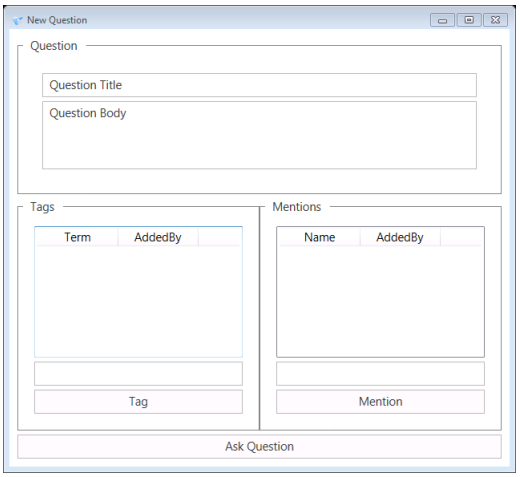


Figure 6.6.: Screenshot of the ask view in SolidEdge.

for adding tags and mentions. By clicking the Answer button the answer together with the tags and mentions are created at once. Afterwards the detail view for the created question is opened.

Figure 6.7 shows the detail view. The label in the top show the questioner, the title and the body of the questions. On the right there are toggle buttons for watching and editing the question. Only the questioner can edit the question, whereupon the labels turn into edit fields. By pressing the edit button again, the question is updated.

The two lists below, show the tags and mentions and by whom they were added. Every user can add new tags or mention users by using the autocomplete field and pressing the appropriate button. The tags and mentions are created immediately.

The bottom part is the answer section, where a answer can be created using the input field and the Answer button. All answers with their creators are shown in the list below. Only the questioner has the ability to accept an answer with the Accept button. It is possible to correct the accepted answer by accepting another one.

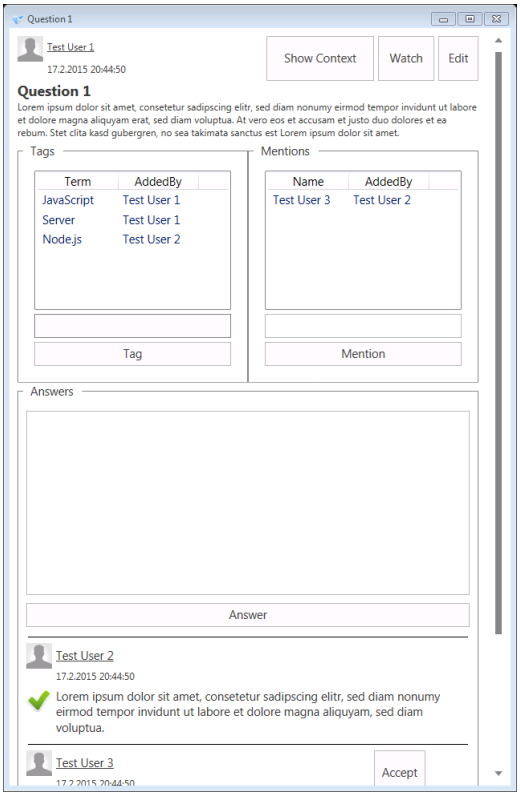


Figure 6.7.: Screenshot of the detail view in SolidEdge.

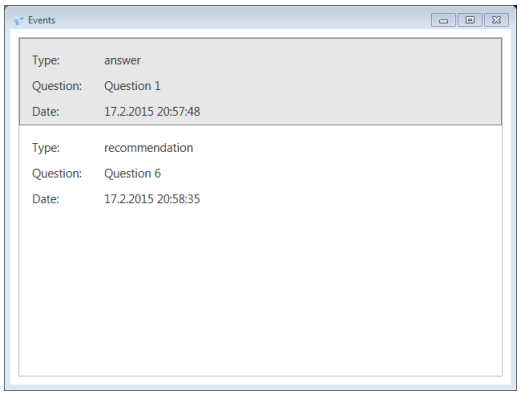


Figure 6.8.: Screenshot of the events view in SolidEdge.

All events which are shown for five seconds in the popup notification of the Windows tray icon are also collected in the events view shown in figure 6.8. A double-click opens the detail view of the associated question.

6.2.2. Model

The model consists of two abstraction layers which are implemented in the classes *Client* and *Model*.

The lower abstraction layer is called client and implements the data transmission. It exports only low level methods for sending requests to named endpoints. The implementation in *SocialEdge* uses *socket.io* to make requests to the sails backend, but can easily be swapped out to use HTTP for example. Furthermore the client handles authentication by storing the access token from the login process and attaching it to every subsequent request.

The second abstraction is the actual model and uses the client to make requests to the backend. The model exports functions for manipulating the data model and is used by all the viewmodels. The functions contain the actual mapping to the REST endpoints and transform the data to be compliant with the API.

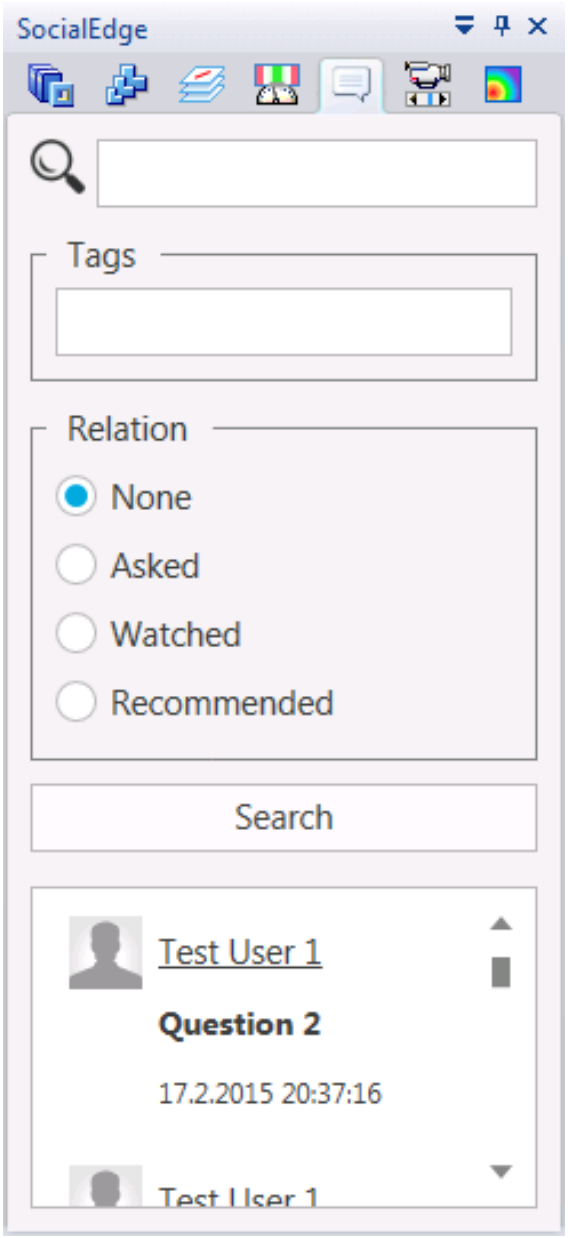


Figure 6.9.: Screenshot of the edgebar of SolidEdge containing the search view integration.

CHAPTER 7

EVALUATION

The evaluation of the prototype focuses on the system's usability and the usefulness of the Q&A functionality. According to Perlman Perlman [Per95], "Questionnaires have long been used to evaluate user interfaces". The questionnaire of the thesis' evaluation is based on "USE Questionnaire: Usefulness, Satisfaction, and Ease of use" Lund [Lun01]. The participants are asked thirty general questions about usefulness, ease of use, ease of learning and satisfaction. In order to be able to answer an questionnaire, the participants needs to get familiar with the system. Therefore, after a quick introduction, he is asked to solve some tasks using the prototype. These tasks cover all basic use cases of the system.

Albert and Tullis [AT13] cover in their book the essential statistical tools to evaluate the results of the questionnaire. Furthermore, they describe basic usability metrics, which can be taken by the interviewer, while the participant solves his tasks. The most common metrics include task success, completion time and committed errors, but also feedback of the user on positive and negative aspects of the system or the idea in general.

Five participants, who have either used Q&A systems before or have some experience with CAD applications, took part in the survey. The survey is attached in appendix B.

7.1. Survey

The first part of the survey covers basic information about the participant. This includes his age and gender as well as his self-estimated skill level with CAD systems, SolidEdge, Enterprise Social Software (ESS) and Q&A systems. The skill level as well as all other items in the third part of the survey follow the Likert scale [Lik32] from *strongly disagree* (1) to *strongly agree* (7). Although, the data is ordinal, the items and answer possibilities are constructed in a way, that they are considered as equidistant. It is therefore common to use the scale as an interval scale.

The seconds part consists of eight tasks, which the participant must solve. These tasks are:

1. Registration,

2. Login,
3. Profile Management,
4. Asking Questions and Tagging,
5. Watching Questions,
6. Answering Questions,
7. Editing Questions and
8. Finding Recommendations.

Each task has a setup phase, where the user fulfills eventual preconditions. The second phase consists of one or more objectives the user must fulfill.

The third part of the survey utilizes the items of the USE questionnaire, which contains thirty questions spread over the four categories *usefullness*, *ease of use*, *ease of leaning* and *satisfaction*. Additionally the three most positive and negatives aspects are written down as free text.

7.2. Procedures

At first, the participant gets a quick introduction to SocialEdge and the Q&A system by the interviewer. Afterwards, the participant knows all relevant information on how the systems works as well as the basic usage of the environment. After collecting their demographic data by filling out the first part of the survey, they are assigned eight tasks about SocialEdge. These tasks cover the basic use cases and involve utilizing all user interface elements. The participant is asked to read the instructions and then finish the task on his own, while giving verbal feedback. The interviewer watches, collects metrics about each task and makes notes about the participants' thinking-aloud process. When the user is stuck or is about to make a wrong move, the interviewer gives him a hint. The metrics include the task success, the completion time and the errors the user made. The task success can either be total, partial or none, depending on whether the participant completed the task without making mistakes, with minor mistakes or failed completely. A minor mistake is for example when the user uses the wrong controls, because he has a different opinion on how the system should work. A major mistake is when the user is totally unable to solve the task without help. After finishing their tasks, the participants fill out the third part of the survey, which is a questionnaire about the usability of the system.

7.3. Results

All the results of the survey including the notes of the interviewer are consolidated on the result sheet in appendix C. It contains the values of the questionnaire as well as the

completion times and task success for every user. The textual fields like remarks of the interviewer and input from the participants are preprocessed; similar statements are grouped and only mentioned once.

The main metric used for evaluating the thesis is usability. Usability cannot be measured directly as it is individually perceived by a user. The survey contains 30 items from the USE questionnaire which are divided in four usability aspects: usefulness (1-8), ease of use (9-19), ease of leaning (20-23) and satisfaction (24-30). Figure 7.1 shows the average rating of each user for the usability aspects. All categories have strictly good ratings; no user rated any category below 4, which is the middle of the rating scale. Also the individual values do not differ much from each other. Ease of use, usefulness and satisfaction score very similar with average values of 5.0, 5.0 and 4.8. The ease of leaning has the best score with a mean value of 6.0.

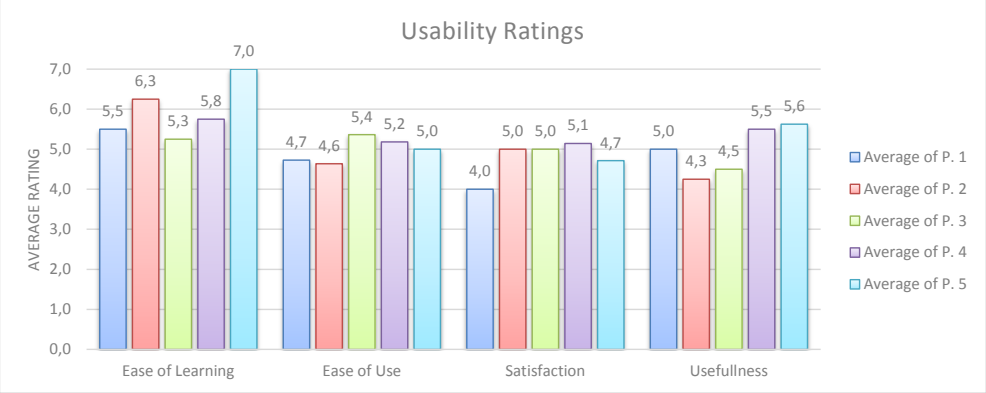


Figure 7.1.: Average scores of the four usability aspects from each participant. The average score can range from 1 (worst) to 7 (best).

The average times to complete the tasks are shown in figure 7.2. The average is computed using two datasets. The first contains all participants' completion times and the second uses only the data of those participants that have an average self estimated skill level of 2.0 or higher. The completion time reduces drastically, if the users have some basic experience with CAD or Q&A systems. Probably, this is related to the general

usage of computers in the daily life of the participants, as most concepts in handling the prototype are very generic. This metric gains significance in future design iterations or in the development of new integrations as it only useful when compared to something.

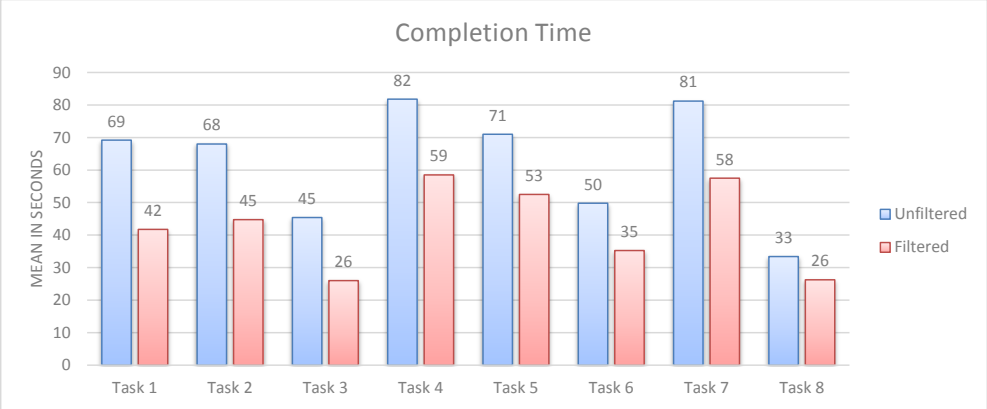


Figure 7.2.: Average completion time for each task of the survey measured in seconds. The red bars show the average completion time for each task using filtered data from only those participants that have an average skill level of 2.0 or higher. The blue bars are using unfiltered data.

The task success is a simple metric to evaluate whether the current user interface is designed good enough to fulfill the systems use cases. Figure 7.3 illustrates the success of the participants in solving the eight tasks. While most of the tasks are completed successfully by the majority or even all users, there are some issues with task five and seven. The error details and remarks indicate, that opening the detail view after searching and can be designed more intuitively. The last step of editing a question requires to untoggle the edit button, which also needs a significant time for the participants to figure out how it works.

The verbal and written feedback from the participants are mostly about the same aspects. There are some minor usability issues, especially the low contrast between the controls and the wish for more feedback on actions. However, the participants like the idea of recommending questions and find the system very useful. The handling of the

Figure 7.3.: Distribution of the task success. Each user can totally (T), partially (P) or not (N) succeed in finishing the task. Partial success is when the user makes some mistakes but easily recovers or when he just needs minor hints from the interviewer. Making an irreversible error or being completely stuck results in task failure.

prototype is mostly intuitive, fast and easy to learn. The ideas for improvement contain amongst others email notification and support for completing actions with the keyboard (i.e. using the enter button instead of clicking a button using the mouse).

CHAPTER 8

CONCLUSION AND FUTURE WORK

The thesis is focused on deeper integration of social services into end user applications and raising contextual awareness. A question and answer service was chosen, because it is a very common social service in corporate environments. Furthermore, it increases productivity through a simple way of managing knowledge and reveals an important case for using context information. Questions are always embedded into a certain context and people need to know it in order to provide a satisfying answer. A lot of information can be automatically collected from the user's working environment to make question and answer system smarter.

Through a social tagging system combined with the power of automatic data analysis a new recommendation engine for question and answer systems has been designed. Similar to recommender systems in e-commerce, an inverse way on how the information finds the target is established. And used together with push notifications users do not need to explicitly search for questions which they can answer, the questions magically find the users. This does not only save time but also sways the users to get active and answer questions.

For the prototypical implementation, the question and answer service was designed from scratch and implemented using Node.js. The service is build for and integrated into the CAD application SolidEdge, although the concept is in no way limited to it or the construction domain in general.

The thesis was evaluated by conducting a survey consisting of eight tasks the user must solve using the prototype followed by a usability questionnaire. Five user participated in the survey and gave important feedback to the idea and the design of the prototype. Although the feedback was very positive, usability can and must be improved because it is especially important for social applications.

Additionally, the recommendation concept for question and answer system described in the thesis is subject to future work. The user currently has no possibility to influence how many recommendations he receives. A high amount of notifications is likely to annoy the user and can yield a contra-productive effect. Furthermore, the recommendation engine can be supplemented with machine-learning techniques. The prototype only uses

self-created profiles and matches them with tags added to the questions by hand. The profiling of users as well as analyzing the topic of questions can additionally be deducted automatically based on previous user actions and natural language processing.

APPENDIX A

API DOCUMENTATION

The following list documents all REST actions which differ from the standard mapping.

Question Controller

Route: **GET /question/**
Summary: Search for questions.
Parameters:

Name	Type	Description
query	string	The search string
tags	string[]	A list of tags to filter for (OR)
relation	string[]	A list of relations to filter for (AND)

Route: **POST /question/**
Summary: Create a question.
Parameters:

Name	Type	Description
title	string	
body	string	
tags	string[]	A list of tag terms to be attached to the question.
mentions	integer[]	A list of user ids to be mentioned in the question.

Route: **POST /question/{id}/tag**
Summary: Add a list of tags and/or mentions to the question.
Parameters:

Name	Type	Description
tags	string[]	A list of tag terms to be attached to the question.
mentions	integer[]	A list of user ids to be mentioned in the question.

Route: **PUT /question/{id}/watch**
Summary: *Un*Watch a question.
Parameters:

Name	Type	Description
id	integer	
active	boolean	true to watch / false to unwatch question

Answer Controller

Route: **PUT /answer/{id}/accept**
Summary: Accept an answer.
Parameters: —

User Controller

The creation of a user is unprotected; no login is required to use this action.

Route: **GET /user/me**
Summary: Get current user.
Parameters: —

Auth Controller

The auth controller does not have an underlying model and thus no default actions. The login of a user is unprotected; no login is required to use this action.

Route: **POST /auth/login**
Summary: Sign the user in.
Parameters:

Name	Type	Description
username	string	The user's username
password	string	The user's password

Route: **POST /auth/logout**
Summary: Sign the user out
Parameters: —

Push Controller

The push controller does not have an underlying model and thus no default actions.

Route: **PUT /push/subscribe**
Summary: Register the client to receive push notifications.
Parameters:

Name	Type	Description
token	string	The client's id (socket id)

APPENDIX B

SURVEY

This is the survey used for the evaluation of the thesis.

Usability Survey

QnAaaS-SolidEdge-Integration

Introduction

Thank you for participating in the survey for the evalution integration of QnAaaS, a question and answer system integrated into the CAD application SolidEdge. The survey has three parts.

The first part is basic information about yourself and your skills. Rate every statement depending if you:

> Strongly disagree (1), Mostly disagreee (2), Slightly disagree (3), Neither disagree nor agree (4), Slightly agree (5), Mostly agree (6), Strongly agree (7).

In the second part you have to interact with the system. There are eight tasks on the following sheets, which you need to process. Every task consists of one or more setup steps. Feel free to ask the interviewer if something is unclear. Once the setup is finished and the interviever gives the go signal, the actual tasks begins and ends when all objectives are done. Please try to fulfill the objects without asking for help and give feedback about your thoughts. The interviever will assist you with tips if you are stuck or making errors.

The third part is a questionnaire about the usability of the software. Again, rate every statement from Strongly disagree (1) to Strongly agree (7).

Basic Information

Please fill out the following information:

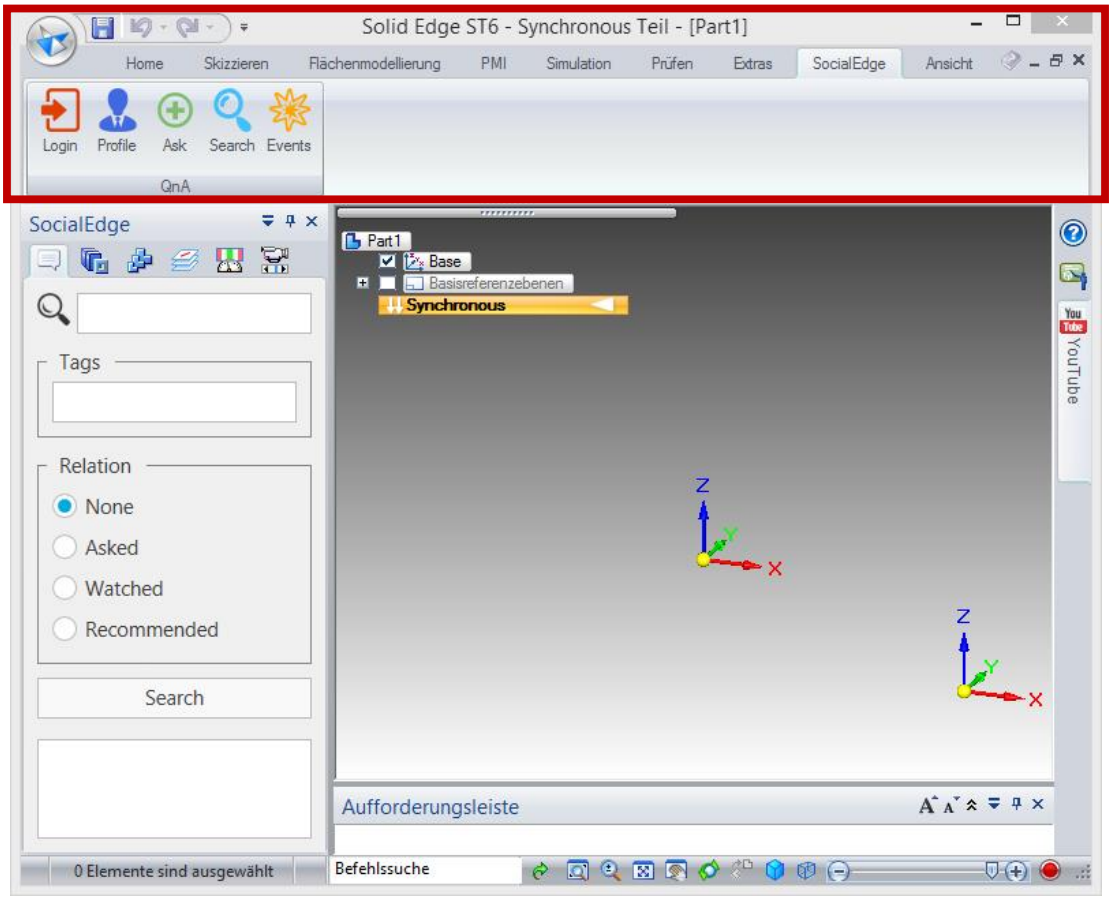
DEMOGRAPHICS	
Gender	
Age	

Please rate the following statements using the scale from *Strongly Disagree* (1) to *Strongly Agree* (7):

#	SKILL LEVEL	1	2	3	4	5	6	7
A	I have experience with CAD software.							
B	I have experience with SolidEdge.							
C	I have experience with enterprise social software.							
D	I have experience with question and answer systems.							

Tasks

The screenshot below shows the synchronous environment of SolidEdge. This is the entry point for the tasks 2-8. The red marked area is the ribbon of the Q&A integration, which contains all controls required for the following tasks.



1. **Registration**

    a. **Setup**

    - Navigate to http://localhost:8008/ with your browser.

    b. **Objectives**

    - Register new user using
        - Name: John Doe
        - Email: test@test.com
        - Password: test1234

2. **Login**

    a. **Setup**

        - Open SolidEdge synchronous environment.

    b. **Objectives**

        - Login using
            - Url: http://localhost:8008/
            - Username: test@test.com
            - Password: test1234

3. **Profile Management**

    a. **Setup**

        - Open SolidEdge synchronous environment.

    b. **Objectives**

        - Add a new interest in „JavaScript“ with skill level 4.

4. **Asking**

    a. **Setup**

        - Open SolidEdge synchronous environment.

    b. **Objectives**

        - Ask a question with:
            - Title: Q1
            - Body: Lorem ipsum…
            - Tags: „.NET“ und „C#“

5. **Watching**

    a. **Setup**

        - Open SolidEdge synchronous environment.

    b. **Objectives**

        - Find the question with title „Question 1“
        - Enable „Watch“ on the question

6. **Answering**

    a. **Setup**

        - Open SolidEdge synchronous environment.

    b. **Objectives**

        - Find an arbitrary question containing the tag „JavaScript“
        - Answer the question with „A1“

7. **Editing**

    a. **Setup**

        - Open SolidEdge synchronous environment.

    b. **Objectives**

        - Find your own question
        - Edit the title to „Q1a“

8. **Recommendations**

    a. **Setup**

        - Open SolidEdge synchronous environment.
        - Find the question with title „Question 2“
        - Mention yourself („John Doe“)
        - Close Window

    b. **Objectives**

        - Find a recommended question
        - Add Tag „T1“

Results

Filled out by the interviewer.

#	**Task Completion** *Total / Partial / None*	**Completion Time** *In seconds*	**Errors** *Description*	**Remarks** *Think-aloud protocol*
1				
2				
3				
4				
5				
6				
7				
8				

USE Questionnaire

Please rate the following statements using the scale from *Strongly Disagree* (1) to *Strongly Agree* (7):

#	USEFULLNESS	1	2	3	4	5	6	7
01	It helps me be more effective.							
02	It helps me be more productive.							
03	It is useful.							
04	It gives me more control over the activities in my life.							
05	It makes the things I want to accomplish easier to get done.							
06	It saves me time when I use it.							
07	It meets my needs.							
08	It does everything I would expect it to do.							

#	EASE OF USE	1	2	3	4	5	6	7
09	It is easy to use.							
10	It is simple to use.							
11	It is user friendly.							
12	It requires the fewest steps possible to accomplish my tasks.							
13	It is flexible.							
14	Using it is effortless.							
15	I can use it without written instructions.							
16	I don't notice any inconsistencies as I use it.							
17	Both occasional and regular users would like it.							
18	I can recover from mistakes quickly and easily.							
19	I can use it successfully every time.							

#	EASE OF LEARNING	1	2	3	4	5	6	7
20	I learned to use it quickly.							
21	I easily remember how to use it.							
22	It is easy to learn to use it.							
23	I quickly became skillful with it.							

#	SATISFACTION	1	2	3	4	5	6	7
24	I am satisfied with it.							
25	I would recommend it to a friend.							
26	It is fun to use.							
27	It works the way I want it to work.							
28	It is wonderful.							
29	I feel I need to have it.							
30	It is pleasant to use.							

Positive Aspects

-
-
-
-
-

Negative Aspects

-
-
-
-
-

Ideas for Improvement

-
-
-
-
-

APPENDIX C

SURVEY RESULT

This is the result sheet of the evaluation survey.

Questionnaire						
Gender	M	M	M	M	M	
Age	27	29	21	60	30	
#A	1	2	5	2	1	
#B	1	1	1	1	2	
#C	4	5	2	1	6	
#D	6	3	4	2	4	
#01	5	6	5	6	6	
#02	5	5	5	6	6	
#03	6	7	5	6	7	
#04	4	2	4	4	2	
#05	6	4	4	6	6	
#06	7	4	6	5	7	
#07	4	2	4	5	7	
#08	3	4	3	6	4	
#09	5	6	6	6	4	
#10	6	5	6	5	4	
#11	3	6	5	5	2	
#12	4	5	4	5	4	
#13	6	3	6	6	6	
#14	6	4	5	5	6	
#15	5	6	4	6	7	
#16	3	2	5	5	6	
#17	4	5	6	4	6	
#18	5	5	6	5	4	
#19	5	4	6	5	6	
#20	6	7	6	5	7	
#21	6	6	5	6	7	
#22	5	6	5	7	7	
#23	5	6	5	5	7	
#24	4	6	5	5	4	
#25	4	6	6	5	7	
#26	2	4	4	4	1	
#27	3	5	5	6	6	
#28	4	5	5	5	4	
#29	6	4	5	6	7	
#30	5	5	5	5	4	
Skill	3,0	2,8	3,0	1,5	3,3	**2,7**
Usefullness	5,0	4,3	4,5	5,5	5,6	**5,0**
Ease of Use	4,7	4,6	5,4	5,2	5,0	**5,0**
Ease of L.	5,5	6,3	5,3	5,8	7,0	**6,0**
Satisfaction	4,0	5,0	5,0	5,1	4,7	**4,8**
Overall	4,4	4,6	4,6	4,6	5,1	**4,7**

Completion Times								
Task	1	2	3	4	5	6	7	8
P. 1	24	50	22	41	75	22	57	17
P. 2	33	31	35	62	32	45	35	23
P. 3	32	56	25	70	41	47	90	19
P. 4	179	161	123	175	145	108	176	62
P. 5	78	42	22	61	62	27	48	46
Mean	69,2	68	45,4	81,8	71	49,8	81,2	33,4

Task Success								
Task	1	2	3	4	5	6	7	8
P. 1	T	T	T	T	P	T	T	T
P. 2	T	T	T	T	T	T	P	T
P. 3	T	T	T	T	T	P	P	T
P. 4	T	T	T	P	N	T	P	T
P. 5	T	T	T	T	P	T	T	T

Task	Errors
1	
2	
3	
4	Multiple tags added
5	Open Details failed
6	Tag entered in title field
7	Finish Editing failed
8	

Task	Remarks
1	
2	Url inconvenient / No Feedback
3	Low contrast
4	
5	Filter options unclear / Focus input
6	
7	Show asked questions in profile
8	

General Feedback

Positive Aspects

- Good Idea
- Pretty Icons
- Autocomplete
- Easy to use
- Fast / Responsive
- Easy to learn

Negative Aspects

- Usability improvable
- Low visual contrast
- No Feedback

Ideas for Improvement

- Notification by Email
- More colors
- Enter button to finish action
- Autofocus on search field

GLOSSARY

.NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows..

Active Directory is a directory service developed by Microsoft..

Android is a mobile operating system based on the Linux kernel and currently developed by Google..

CRUD stands for the basic database operations **C**reate, **R**ead, **U**pdate and **D**elete..

express.js is a rapid application framework for Node.js.

Gamification denotes the usage of game mechanics in non-game contexts to increase user engagement. Common elements are i.a. virtual currency, experience points, badges and leader boards. [ZC11].

iOS iOS is a mobile operating system developed by Apple Inc. and distributed exclusively for Apple hardware..

MongoDB is a cross-platform document-oriented database following the NoSQL approach. In constrast to traditional table-based rational databases it stores data in JSON-like documents with a dynamic structure. [14c].

MVVM is an architectural pattern for software development..

Node.js is a server-side platform for running networking applications. It is based on JavaScript and provides an event-driven architecture with non-blocking I/O operations. [14d].

OAuth is an open standard for authorization..

OpenID is an open standard for authentication..

RabbitMQ is a request broker implementing the Advanced Message Queuing Protocol..

Sails.js is a rapid application framework for Node.js.

Socket.IO is a JavaScript library for realtime web applications. It enables realtime, bi-directional communication between web clients and server..

WPF is a graphical subsystem for rendering user interfaces in Windows-based applications by Microsoft..

XAML is a declarative XML-based language developed by Microsoft that is used for initializing structured values and objects..

ACRONYMS

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

APNs Apple Push Notification Service.

CAD Computer Aided Design.

ESS Enterprise Social Software.

FEM Finite Element Method.

GCM Google Cloud Messaging.

HTTP Hypertext Transfer Protocol.

KM Knowledge Management.

LDAP Lightweight Directory Access Protocol.

MVC Model View Controller.

NLP Natural Language Processing.

ORM Object Relational Mapper.

PDP Product Development Process.

Q&A Questions and Answers.

REST Representational State Transfer.

TUM Technische Universität München.

URI Uniform Resource Identifier.

UX User Experience.

BIBLIOGRAPHY

[14a] Oct. 2014. URL: http://googleblog.blogspot.de/2006/11/adieu-to-google-answers.html.

[14b] *HTTP Status Code*. Oct. 2014. URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

[14c] *MongoDB*. Oct. 2014. URL: http://www.mongodb.org/.

[14d] *Node.js*. Oct. 2014. URL: http://nodejs.org/about/.

[14e] *Passport*. Oct. 2014. URL: http://passportjs.org/.

[14f] *Quora*. Oct. 2014. URL: https://answers.yahoo.com/.

[14g] *RabbitMQ*. Oct. 2014. URL: http://www.rabbitmq.com/.

[14h] *Sails.js*. Oct. 2014. URL: http://sailsjs.org/.

[14i] *socket.io*. Oct. 2014. URL: http://socket.io/.

[14j] *StackOverflow*. Oct. 2014. URL: http://stackoverflow.com/.

[14k] *StackOverflow question statistics*. Oct. 2014. URL: https://stackoverflow.com/questions.

[14l] *StackOverflow user statistics*. Oct. 2014. URL: https://stackoverflow.com/users.

[14m] *StackOverflow visitor analysis*. Oct. 2014. URL: https://www.quantcast.com/stackoverflow.com.

[14n] *Strange questions on Yahoo Answers*. Oct. 2014. URL: http://www.pcworld.com/article/184999/strange_questions_yahoo_answers.html.

[14o] *Yahoo Answers*. Oct. 2014. URL: https://www.quora.com/.

[Ald+11] G. Alducin-Quintero, M. Contero, J. Martín-Gutiérrez, D. Guerra-Zubiaga, and M. Johnson. "Productivity Improvement by Using Social-Annotations about Design Intent in CAD Modelling Process." English. In: *Online Communities and Social Computing*. Ed. by A. Ozok and P. Zaphiris. Vol. 6778. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 153–161. ISBN: 978-3-642-21795-1. DOI: 10.1007/978-3-642-21796-8_16.

[AM90] M. S. Ackerman and T. W. Malone. "Answer Garden: A Tool for Growing Organizational Memory." In: *Proceedings of the ACM SIGOIS and IEEE CS TC-OA Conference on Office Information Systems*. COCS '90. Cambridge, Massachusetts, USA: ACM, 1990, pp. 31–39. ISBN: 0-89791-358-2. DOI: 10.1145/91474.91485.

[AT05] G. Adomavicius and A. Tuzhilin. "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions." In: *Knowledge and Data Engineering, IEEE Transactions on* 17.6 (June 2005), pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99.

[AT13] W. Albert and T. Tullis. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Interactive Technologies. Elsevier Science, 2013. ISBN: 9780124157927.

[Bra+10] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer. "Example-centric Programming: Integrating Web Search into the Development Environment." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. Atlanta, Georgia, USA: ACM, 2010, pp. 513–522. ISBN: 978-1-60558-929-9. DOI: 10.1145/1753326.1753402.

[CHK10] Y. CHEN, T.-H. HO, and Y.-M. KIM. "Knowledge Market Design: A Field Experiment at Google Answers." In: *Journal of Public Economic Theory* 12.4 (2010), pp. 641–664. ISSN: 1467-9779. DOI: 10.1111/j.1467-9779.2010.01468.x.

[FT00] R. T. Fielding and R. N. Taylor. "Principled Design of the Modern Web Architecture." In: *Proceedings of the 22Nd International Conference on Software Engineering*. ICSE '00. Limerick, Ireland: ACM, 2000, pp. 407–416. ISBN: 1-58113-206-9. DOI: 10.1145/337180.337228.

[GH06] S. A. Golder and B. A. Huberman. "Usage patterns of collaborative tagging systems." In: *Journal of Information Science* 32.2 (2006), pp. 198–208. DOI: 10.1177/0165551506062337. eprint: http://jis.sagepub.com/content/32/2/198.full.pdf+html.

[Guo+08] J. Guo, S. Xu, S. Bao, and Y. Yu. "Tapping on the Potential of Q&a Community by Recommending Answer Providers." In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. CIKM '08. Napa Valley, California, USA: ACM, 2008, pp. 921–930. ISBN: 978-1-59593-991-3. DOI: 10.1145/1458082.1458204.

[Har+08] F. M. Harper, D. Raban, S. Rafaeli, and J. A. Konstan. "Predictors of Answer Quality in Online Q&Amp;A Sites." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: ACM, 2008, pp. 865–874. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357191.

[IK07] U. Ibusuki and P. C. Kaminski. "Product development process with focus on value engineering and target-costing: A case study in an automotive company." In: *International Journal of Production Economics* 105.2 (2007). Scheduling in batch-processing industries and supply chains, pp. 459–474. ISSN: 0925-5273. DOI: http://dx.doi.org/10.1016/j.ijpe.2005.08.009.

[JCL05] J. Jeon, W. B. Croft, and J. H. Lee. "Finding Similar Questions in Large Question and Answer Archives." In: *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*. CIKM '05. Bremen, Germany: ACM, 2005, pp. 84–90. ISBN: 1-59593-140-6. DOI: 10.1145/1099554.1099572.

[Lik32] R. Likert. "A technique for the measurement of attitudes." In: *Archives of Psychology* 22.140 (1932), pp. 1–55.

[Lou+13] J. Lou, Y. Fang, K. H. Lim, and J. Z. Peng. "Contributing high quantity and quality knowledge to online Q&A communities." In: *Journal of the American Society for Information Science and Technology* 64.2 (2013), pp. 356–371. ISSN: 1532-2890. DOI: 10.1002/asi.22750.

[Lun01] A. Lund. *Measuring Usability with the USE Questionnaire. STC Usability SIG Newsletter, 8:2*. 2001.

[Mam+11] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. "Design Lessons from the Fastest Q&a Site in the West." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 2857–2866. ISBN: 978-1-4503-0228-9. DOI: 10.1145/1978942.1979366.

[Mor08] P. J. Morrison. "Tagging and searching: Search retrieval effectiveness of folksonomies on the World Wide Web." In: *Information Processing & Management* 44.4 (2008), pp. 1562–1579. ISSN: 0306-4573. DOI: http://dx.doi.org/10.1016/j.ipm.2007.12.010.

[MTP10] M. R. Morris, J. Teevan, and K. Panovich. "What Do People Ask Their Social Networks, and Why?: A Survey Study of Status Message Q&a Behavior." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '10. Atlanta, Georgia, USA: ACM, 2010, pp. 1739–1748. ISBN: 978-1-60558-929-9. DOI: 10.1145/1753326.1753587.

[NAA09] K. K. Nam, M. S. Ackerman, and L. A. Adamic. "Questions in, Knowledge in?: A Study of Naver's Question Answering Community." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 779–788. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518821.

[Per95] G. Perlman. "Practical Usability Evaluation." In: *Conference Companion on Human Factors in Computing Systems*. CHI '95. Denver, Colorado, USA: ACM, 1995, pp. 369–370. ISBN: 0-89791-755-3. DOI: 10.1145/223355.223726.

[Res+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. "GroupLens: An Open Architecture for Collaborative Filtering of Netnews." In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. CSCW '94. Chapel Hill, North Carolina, USA: ACM, 1994, pp. 175–186. ISBN: 0-89791-689-1. DOI: 10.1145/192844.192905.

[RH08] D. R. Raban and F. M. Harper. *Motivations for Answering Questions Online Abstract*. 2008.

[SBG99] A. Schmidt, M. Beigl, and H.-W. Gellersen. "There is more to context than location." In: *Computers & Graphics* 23.6 (1999), pp. 893–901. ISSN: 0097-8493. DOI: http://dx.doi.org/10.1016/S0097-8493(99)00120-X.

[Seb02] F. Sebastiani. "Machine Learning in Automated Text Categorization." In: *ACM Comput. Surv.* 34.1 (Mar. 2002), pp. 1–47. ISSN: 0360-0300. DOI: 10.1145/505282.505283.

[SI14] R. Scoble and S. Israel. *Age of Context*. Patrick Brewster Press, 2014. ISBN: 978-1-4923-4843-6.

[Swa+99] J. Swan, S. Newell, H. Scarbrough, and D. Hislop. "Knowledge management and innovation: networks and networking." In: *Journal of Knowledge Management* 3.4 (1999), pp. 262–275. DOI: 10.1108/13673279910304014. eprint: http://www.emeraldinsight.com/doi/pdf/10.1108/13673279910304014.

[Wii97] K. M. Wiig. "Knowledge management: Where did it come from and where will it go?" In: *Expert Systems with Applications* 13.1 (1997). Knowledge management, pp. 1–14. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/S0957-4174(97)00018-3.

[ZC11] G. Zichermann and C. Cunningham. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. 2011.