Freie Universität Berlin

CaSe - Categorical Scheduler

Jim Saringer (4515403)
jim.saringer@fu-berlin.de

A thesis presented for the degree of
Bachelor of Science

Advisors:
Prof. Dr. Agnès Voisard
Prof. Dr. Katinka Wolter

Freie Universität Berlin
Department of Computer Science & Mathematics
Institute for Computer Science
Research Group: Databases and Information Systems

September 13th, 2017
Berlin, Germany

Abstract

In the field of crisis management, a swift coordination of staff with special abilities as well as the determination of the required quantity and current availability of the respective professionals is necessary to solve arising problems. In particular, schedule coordination between different authorities that do not share a calendaring system is cumbersome and can be quite time consuming, depending on the number of participants. Within the scope of this Bachelor thesis an application is developed to support the personnel planning of a safety authority.

When an appointment is created, the system determines in advance how many people from previously defined categories are required in order to ensure that the appropriate number of qualified personnel appear at the event. Participants can be easily assigned to an event from an existing contact list. If individual invitees cannot attend the event, they can be easily replaced by other persons of the respective categories.

The contacts can be imported from common contact applications, such as *Google Contacts* or *Microsoft Outlook*, and each contact be assigned to freely definable categories.

In the context of this thesis a main focus is the development of user-friendly interfaces. The herefore applied method is *User-Centered Design* which allows a user-based analyze of software requirements. Based on the results a low-fidelity prototype is developed, in the following refined into a high-fidelity prototype and subsequently implemented.

Zusammenfassung

Im Bereich des Krisenmanagements ist eine rasche Koordination von Mitarbeitern mit besonderen Fähigkeiten sowie die Bestimmung der benötigten Teilnehmeranzahl und Verfügbarkeit zur Problemlösung notwendig. Insbesondere die Terminkoordination zwischen verschiedenen Behörden die nicht über einen gemeinsamen Terminkalender verfügen gestaltet sich umständlich und fordert abhängig von der Teilnehmeranzahl einen erheblichen Zeitaufwand. Im Rahmen dieser Bachelorarbeit wurde eine Anwendung zur Unterstützung der personellen Planung einer Sicherheitsbehörde entwickelt.

Um zu gewährleisten, dass bei einem Termin die gewünschte Anzahl von Personen aus verschiedenen Bereichen erscheinen, wird bei Erstellung eines Termins vorab festgelegt wie viele Personen aus vorher definierten Kategorien erforderlich sind. Dem Termin werden dann Teilnehmer aus einer vorhandenen Kontaktliste zugeordnet. Bei Absagen von einzelnen Teilnehmern können diese komfortabel durch andere Personen der jeweiligen Kategorien ersetzt werden.

Die Kontakte sollen aus gängigen Kontaktanwendungen wie *Google Kontakte* oder *Microsoft Outlook* importiert und mit frei definierbaren Kategorien verbunden werden können.

Der besondere Fokus dieser Arbeit liegt auf der Entwicklung benutzerfreundlicher Interfaces. Hierzu wird die Methode des *User-Centered Designs* verwendet um durch eine benutzernahe Bedürfnisanalyse Anforderungen an das System zu erheben. Im Anschluss daran erfolgt die Erstellung von hochgranularen Prototypen welche letztendlich in feingranularen Prototypen implementiert werden.

Table of Contents

List of Figures

Statutory declaration

I declare that I have developed and written the enclosed Bachelor thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Bachelor thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Berlin (Germany), 13th September 2017

Jim Saringer

Chapter 1

Introduction

The first section of this chapter describes the motivation for the thesis which is the development of a user-friendly software solution for coordination between different authorities and their personnel by applying *User-Centered Design* methods. In the second section, the outline of contribution contains an overview of work design concepts for handling the creation and maintenance of appointments with fixed locations, times and participants. In the last section of this chapter, a brief overview of the thesis structure is provided.

1.1 Motivation

Trying to find suitable dates and times for meetings within a single authority should be a minor issue when employees share the same software for arranging meetings. Usually all individual departments are running their own calendaring services, for example *Microsoft Outlook* or *Google Calendar*. But what happens when you want to invite people outside of your organisation or when the meeting organiser is outside of your organisation?

Example 1 *After the attack on the Berlin Christmas Market (On December 21, 2016), local authorities are developing preventive measures for a general overhaul of security. As a first result, on-call services are created where teams composed of individual staff of police, ambulance and firefighters are available in shifts around the clock. The personnel of the parties involved needs to be scheduled in a way that teams stand ready on a daily basis.*

An intuitive approach by sending emails to all the authorities involved in *Example 1* would result in an extensive workload. As described in the work of *Shakshuki et al.*, scheduling meetings by using email as the primary communication mechanism is considered "inefficient for several reasons"[3]. Firstly, the more the number of people that are supposed to participate in an event, the more email messages have to be exchanged. The meeting organizer has to keep track of the replies, deal with all the data to find the optimal date and time, and then email the date to the supposed participants.
Secondly, in case that attendees cannot attend a meeting and might be replaced (as mentioned in *Example 2*), then the rescheduling process will result into even more message exchange[3].

Example 2 *A security company is employed for the event protection of a concert. For a smooth run of the event they need to work hand in hand with the local authorities and the event organizer. This is only possible by building a steering group of authoritative decision makers. Decision makers of all the participating authorities need to meet in recurring appointments to properly set up the concert. The group meets twice a month over five months. Each appointment starts at 12 o'clock with changing locations. The number of persons of the steering group needs to be kept as low as possible, so that amicable decisions can be taken quickly. In case of not being able to attend a meeting, the respective participant needs to be replaced by another authoritative representative of the respective authority.*

Thus, to avoid overwhelming number of emails and massive message exchange, the objective of this thesis is the development of a system which provides an easy to use and user-friendly interface for keeping track of all the designated participants of an event. Additionally, it has to be ensured that representatives of the respective working field are available for the event by visualizing their consent and replacing them if necessary.

1.2 Outline of Contribution

Within the scope of this thesis a web-based service is developed which supports the coordination and personnel planing within and between authorities. With the help of a pleasant look and feel, i.e., a clean and easy to use user-interface, the main focus of the application is to ensure that qualified personnel from different authorities can participate in an event if necessary.

To fulfill the demand as good as possible the *User-Centered Design* method was applied. Initially software requirements are analyzed based on user surveys. Build on the results a low-fidelity prototype is developed and iteratively refined into a high-fidelity prototype.

When creating a new appointment the initiator is able to select from predefined categories which represent the different sectors where participants are needed from. When adding a category to an appointment the creator can set how many people belonging to this category are required for this meeting. Moreover the initiator selects particular users from his contact list to fill the required category slots. If supposed participants reject the appointment they can easily be replaced by other individuals of the respective category. Every user of the system can create an unlimited amount of categories and assign them to all the contacts from their contact list. While being able to add contacts to the list by hand, users should rather use a given import functionality to import their contacts from common contact applications, such as *Google Contacts* or *Microsoft Outlook*.

1.3 Structure of the Thesis

The first chapter serves as an introduction to the topic of the thesis. The introduction presents the motivation for the bachelor thesis, the outline of contribution and the structure of the thesis.

The second chapter comprises the background and the context of the thesis. Besides introducing necessary definitions, this chapter presents state of the art applications which are related to the work of this thesis.

Chapter three gives an overview of the User-Centered Design process and the methods used within its scope. When creating user-interfaces the UCD-approach aims for achieving the best possible user-experience by including potential users of the software in the design process.

In chapter four, the frontend design of the web application and the requirements for the backend are presented. For designing a user-friendly interface the UCD-framework is applied. Requirements for the backend crystallize as a result of the designing phase.

After the designing phase, chapter five describes the implementation of the web application. Firstly, the architecture is introduced, followed by a summary of used technologies. Finally, a detailed overview of the frontend and backend implementation is given highlighted by excerpts from the source

code.

In chapter six, the results of the thesis are summarized. Moreover the limitations of the current software are reviewed.

Chapter seven covers the conclusion of the work by personally evaluating the results of the work described in chapter six.

Finally, the last chapter includes an overview of referenced literature.

Chapter 2

Background

In the first section of this chapter, definitions are provided for important terms which are addressed throughout the thesis. Since the main focus of this work is the design and eventual implementation of an easy to use scheduling software, already existing state of the art software systems are considered in the related work section.

2.1 Definitions

JavaScript Object Notation (JSON) "JSON is designed to be a data exchange language which is human readable and easy for computers to parse and use."[1]

Java Persistence API (JPA) "The Java Persistence API is a Java standards-based solution for persistence. Persistence uses an object-relational-mapping approach to bridge the gap between an object-oriented model and a relational database."[2]

Mockup Mockups represent the visual design of a software and are usually drawn on paper. Low-fidelity mockups only display basic design solutions while high-fidelity mockups are used for displaying the complete functionality of the software. In summary, the fidelity of the mockup refers to "how closely it matches the look-and-feel of the final system"[34].

Open Source The term open-source is applied in software development for software with source code that anyone can inspect, modify, and enhance. This does not mean that the software can be used commercially, which is defined by the provided license of the software.

2.2 Related Work

The idea of creating a scheduling service in the context of this thesis emerged from a request by a French security authority which is looking for a software to support their personnel scheduling in the context of crisis management.

A general approach to the process of personnel scheduling was developed by *A.T. Ernst, H. Jiang, M. Krishnamoorthy and Sier*[5] in their work *Staff scheduling and rostering: A review of applications, methods and models.* According to the aforementioned work, a software for personnel scheduling is supposed to meet six different requirements. Due to the reason that it is not planned to develop a fully-featured staff scheduling service (rostering service), only two of the mentioned requirements are of interest for further research:

SR1: A functionality should be given to select fitting candidates and the needed amount to cover particular tasks like selecting meetings at specific times.

SR2: The system should provide the option to set the working fields for the individual personnel.

To avoid massive email traffic for scheduling personnel across organizational borders, various software can be utilized. The following sections briefly introduce state of the art applications which can be used by employees from different agencies in order to meet requirements *SR1* and *SR2*.

Poll-Based Scheduling Services

The vote for group meetings and meetings is rather done with a few short clicks via free online schedulers. One of the most popular and widely used services is *Doodle*. Appointments and surveys can be quickly created using the tool and sent to selected participants via e-mail. As an alternative serves *Dudle* which in contrary to *Doodle* puts a heavy focus on the privacy of its users.

Doodle

Doodle is a simple and popular online service for scheduling events. Polls do not only serve finding a common date between different participants but can be about literally anything. Doodle is an implementation of the approval voting mechanism, where candidates are the time slots and each responder approves a subset of the slots. The poll initiator is free to choose if the poll is hidden or open. While an open poll allows every participant to see what the other participants voted for the hidden poll only reveals to the initiator what the participants voted for. As shown in the work by *James Zou et al.*[4] responders are more likely to approve highly popular and unpopular time slots in open polls than in hidden polls. When the majority is voting for a particular event people tend to be compliant and therefore follow the majority. Moreover responders showed a higher availability in general in public polls which probably can be explained due to the participants being afraid of showing a lack of commitment if they only vote for few to none proposed events.

Dudle

Several security problems occur in the aforementioned application. For one thing it is not always desirable that all participants who participate in appointments are able to see the preferred dates of everyone else participating. Another problem is, that in some surveys one wants to be sure that nobody falsifies the survey for example, in which he treats the deadlines of another. In an attempt to fix those privacy issues the event scheduling application *Dudle*[8] was developed by *Benjamin Kellerman* at the *Technical University Dresden*. *Dudle* is applying a protocol for anonymous scheduling[14] which divides the voting process into three different steps:

1. First of all each participant of a vote needs to register on a key server for enrolling a cryptographic key.

2. Secondly the creation and setup of a vote where a fixed number of participants is set in order to create an anonymity set.

3. Lastly the vote itself. Each participant needs to enter his key for being able to vote. Participants preferences and names are not revealed to each other. The result of the vote will be shown after everybody voted.

In contrast to *Doodle*, *Dudle* is an open source software which allows its users to participate in the development of the program in order to improve the user-experience and general functionality. Moreover it doesn't apply *Google Analytics*, a software used for tracking and evaluating user behavior.

Skill-based Scheduling Service

Ensure(Enablement of Urban Citizen Support for Crisis Response)[7] is a project organized by *Frauenhoher FOKUS* with the contribution of the *Freie Universität Berlin* and *Technische Universiät Berlin* among other authorities[1]. The goal of the ENSURE project is the development of a concept for the recruitment of citizens to support local authorities in emergency situations. The basic idea of the project approach is that in the future the security and protection of the urban population in crisis situations must mobilize and organize the potential of the population for self-help and support of the local authorities. As a result of the current progress of the project a mobile application was built which allows to:

- Register volunteers and gather essential characteristics like capability, age or specific abilities.
- Receive alarms with a short introduction of the problem and having the opportunity of accepting or rejecting the assignment

Thanks to the developed mobile application authorities are able to create assignments via a web application which are focusing on people with specific professional or personal knowledge (e.g. concierge, security staff, first-aider). These assignments also allow a focus on sensitive sectors (e.g. retirement homes, kindergarten) due to the possible preselection of contemplable volunteers. At the time of writing the application can be downloaded but was only available due to a test run in 2016 and there are currently no assignments offered.

Calendaring Services

Regardless of how meetings are performed, calendars must be checked and available times synchronized. Calendaring services extend this approach with

[1]Deutsches Rotes Kreuz (DRK), Berliner Feuerwehr (BFw), Gesellschaft für Datenschutz und Datensicherheit e.V. (GDD), HFC Human-Factor-Consult GmbH (HFC)

the possibility of sharing calendars along employees. This way the most suitable days for arranging meetings are shown at a glance. The following section displays two software solutions for calendaring services by *Google* and *Microsoft*.

Google Calendar

Google Calendar is a free to use software for coordinating meetings and events, or scheduling work shifts. People invited to an event will receive a push notification if they are using a smart phone with an enabled Google account. When using Google Calendar, users are able to set up alarms for being reminded of upcoming appointments. Created calendars can be shared with other users and the creator of the calendar is able to configure access privileges. Moreover the calendar settings allow to automatically accept all the invitations received or alternatively only accept events when they are not causing conflicts with already existing events. Based on the calendar sharing mechanism the software also supports staff scheduling[6]. This can be done by initially creating a master calendar which is shared with the personal calendars of the employees. A personal calendar can be associated with a color in order to be able to see at a glance when shifts are taking place[6].

Outlook Calendar

In contrast to the *Google Calendar* the *Microsoft* software *Office Outlook* is running as a desktop application and allows its users to manage their events without an Internet connection. You only need Internet access when sharing calendars with other users. Just like the *Google Application* users can select a time on the calendar, create a meeting request, and select the people to invite. Contrary to Google's calendar Outlook helps you find the earliest time when all the invitees are free. This functionality is provided by showing availability indicators, with a range from poor to good depending on the schedules of the participants, when clicking on a day for a potential meeting in a monthly calendar. When you send the meeting request by email, the invitees receive the request in their inbox. When the invitees open the request, they can accept, tentatively accept, or decline a meeting. If a request conflicts with an item on the invitees' calendar, Outlook displays a notification. If allowed by the meeting organizer, invitees can propose an alternative meeting time. The organizer is always able to track who accepts or declines the requests or

who proposes another time for the meeting by opening the respective request. Outlook also permits users to drag and drop emails right from their inbox into the Outlook calendar which enables user to view the conversations that inspired the scheduling of an event.

Chapter 3

User-Centered Design

As the User-Centered Design (UCD) process and its related methods are addressed throughout the thesis, this chapter offers an introduction to the process. At the beginning, the term and its concept are explained and the following section introduces methods used for the practical implementation of the UCD. Due to the sheer range and complexity of methods only some exemplary methods are presented in this section.

3.1 What is User-Centered Design?

> "User-Centered Design (UCD) is a broad term to describe design processes in which end-users influence how a design takes shape."[17]

The term *User-Centered Design* was shaped for the first time in 1986[17] by Donald Norman and Stephen Draper in their co-authored publication *User-Centered System Design: New Perspectives on Human-Computer Interaction*. While the meaning of the term evolved over the years the *International Organization for Standardization* developed a UCD definition which is broadly accepted. The ISO 9241-210 (2010) standard – formerly known as ISO 13407 (1999) – "is a powerful tool to assure a human-centered design process. The comprehensive nature of these standards make them the most authoritative starting point for human-centered design education, training,and practice"[20].

This standard outlines UCD as a process for interactive system development with the basic idea of developing design solutions which then can be

evaluated by the users.

3.2 User-Centered Design Life-Cycle

In march 2010 the *"International Organization for Standardization"* published the norm *ISO 9241-210*[11] which divides the user-centered design(UCD) process into the following four phases as shown in Figure 3.1:

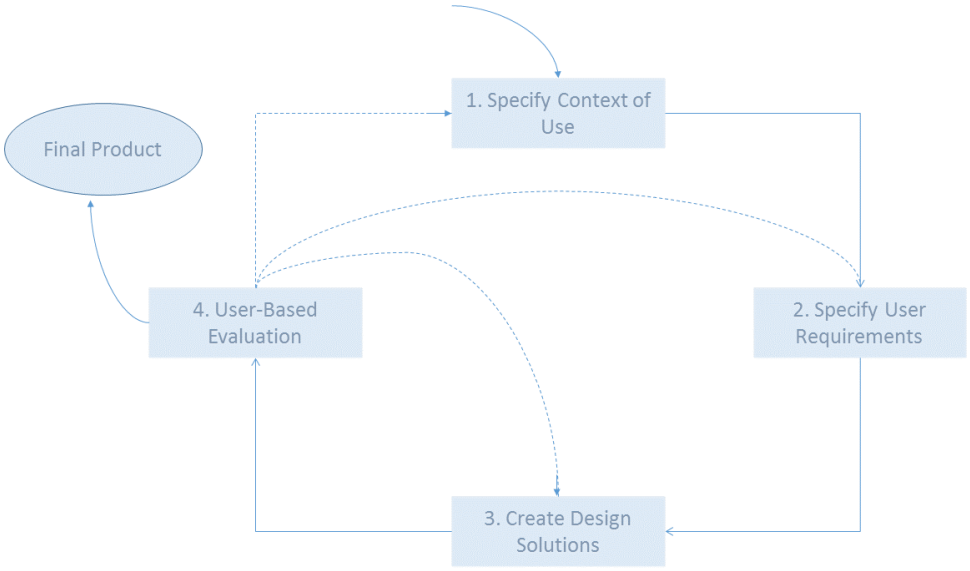

Figure 3.1: Phases of the User-Centered Design Process According to the *DIN EN ISO 9241-210*[11]

3.2.1 Specify User

"When a system or product is developed, it will be used within a certain context. It will be used by a user population with certain characteristics. They will have certain goals and wish to perform certain tasks."[21]

The first phase of the *UCD process* serves the collection of information about the future users. The practical implementation of this phase starts

with identifying the potential target groups of the upcoming software. Multiple target groups have varied points of views which need to be considered in order to create a design for reconciling their views as good as possible. The following methods are applied in the process of this phase:

Target Groups Defining target groups helps in the understanding of what the user needs. By being aware of for what group of users the application is developed, the developer is able to meet their wishes and interests in a better way.

Interviews "Interviewing is a commonly used technique where users, stakeholders and domain experts are asked questions by an interviewer in order to gain information about their needs or requirements in relation to the new system. Interviews are usually semi-structured based on a series of fixed questions with scope for the user to expand on their responses. Semi-structured interviewing is useful in situations where broad issues may be understood, but the range of respondents reactions to these issues is not fully known."[21]

Personas "Personas are a means of representing users' needs to the design team, by creating caricatures to represent the most important user groups. Each persona is given a name, personality and picture. They are particularly valuable when it is difficult to include user representatives in the design team. Each persona can be associated with one or more scenarios of use. Potential design solutions can then be evaluated against the needs of a particular persona and the tasks in the scenarios."[21]

3.2.2 Specify Context of Use

With the user information obtained in the first step the user requirements for the project are defined during this phase. These requirements then have to be implemented in the designing process. A general method for estimating the user requirements is the creation of *Scenarios of Use*:

Scenarios of Use "Scenarios give detailed realistic examples of how users may carry out their tasks in a specified context with the future system. The primary aim of scenario building is to provide examples of future use as an aid to understanding and clarifying user requirements and to provide a

basis for later usability testing. Scenarios encourage designers to consider the characteristics of the intended users, their tasks and their environment, and enable usability issues to be explored at a very early stage in the design process (before a commitment to code has been made)." [21]

3.2.3 Create Design Solutions

In this phase the first concepts and designs are elaborated based on the results of the preceding work. Designs are realized in form of *wireframes* and *mock-ups*. The resulting paper prototypes are used for developing first sketches as those allow to be quickly drawn and changed according to the users feedback in the evaluation process. The idea of this approach is to avoid the expansive process of correcting design faults in the later stages of the development cycle. An exemplary prototyping method is *paper prototyping*:

Paper Prototyping "Designers create a paper-based simulation of user-interface elements (menus, buttons, icons, windows, dialogue sequences, etc.) using paper, card, acetate and pens. When the paper prototype has been prepared, a member of the design team sits before a user and plays the computer by moving interface elements around in response to the users actions. The difficulties encountered by the user and their comments are recorded by an observer and/or on video or audio."[21]

3.2.4 User-based Evaluation

The evaluation of the software is based on the feedback of the potential users and realized by applying suitable methods of the user-centered design framework as described in *ISO 9241-210:2010*[11].
More precisely the created prototypes are iteratively evaluated with the users by applying the following *usability inspection methods*[13]:

Heuristic Evaluation The given heuristics, named below *H1* to *H10*, are general principles designed by Jakob Nielsen[33] for optimizing the interaction design of an user interface.

H1: **Visibility of system status** "The system should always keep users informed about what is going on, through appropriate feedback within reasonable time."

H2: **Match between system and the real world** "The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order."

H3: **User control and freedom** "Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo."

H4: **Consistency and standards** "Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions."

H5: **Error prevention** "Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action."

H6: **Recognition rather than recall** "Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate."

H7: **Flexibility and efficiency of use** "Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions."

H8: **Aesthetic and minimalist design** "Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility."

H9: **Help users recognize, diagnose, and recover from errors** "Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution."

H10: **Help and documentation** "Even though it is better if the system can be used without documentation, it may be necessary to provide help

and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large."

Think Aloud Method "Users employ a prototype as they work through task scenarios. They explain what they are doing by talking or thinking-aloud. This information is recorded and/or captured by an observer. The observer also prompts users when they are quiet and actively questions them with respect to their intentions and expectations"[21]

Chapter 4

Webservice Application

The first section of this chapter describes the application of the *User-Centered Design* method with the intent of creating a design for the user-interface. In the second section the requirements for the backend are introduced, which are derived from the design of the frontend.

4.1 Design and Evaluation of the Frontend

In order to design a user-friendly graphical interface the User-Centered Design framework is applied. The underlying idea of the framework is to let future users of the software participate in the designing process. Before a software can be designed and afterwards implemented its requirements need to be clearly defined. Due to the idea for creating a scheduling service being initiated by a security service, user research was focused on authorities. Thus first of all interviews were carried out with single representatives of the police, firefighters and ambulance service. During the interviews the interviewees were expected to answer the questions listed in the following:

1. What kind of scheduling software is required for coordination with other authorities?

2. At which points and in which situations is the functionality provided by common scheduling services not longer sufficient?

3. What are the essentials problem which need to be fixed by the new scheduling software?

4. What will be the difference between common existing solutions and the to be developed software?

5. In which way will the to be developed solution employed?

Three different people had been interviewed in order to get an overview of which requirements for the to be developed software need to be fulfilled. The in the following defined requirements are aggregating the survey data:

R1: The software is able to set up events at certain times and locations with designated participants.

R2: A functionality to check if participants are available or not. When participants reject invitations a functionality for replacing them with suitable participants needs to be provided.

R3: There need to be a convenient solution for participants to agree on a common date.

R4: Software users should be able to maintain a centralized pool of contacts. Contacts have to be assignable to freely definable categories.

R5: When creating an event the creator needs to be able to set a fixed number of participants and the number of participants per category.

R6: All events, regardless of being created by the user or being invited to by other users must be visible at a glance.

R7: An overview is required for maintaining contacts of the individual user.

R8: An overview is required for the maintenance of categories created by the individual user.

Furthermore the following technical requirements derived from the already acquired requirements:

R9: The software needs to be implemented by using Open Source software as there is no budget available for fee-based software.

R10: Popular programming patterns and languages should be applied to keep technical limitations and required knowledge for future work on the application at a low level.

4.1.1 Method: User-Centered Design

This section states the results of the first UCD activity. As stated in chapter three the foundation of any UCD process is to understand the intended users of the system, their environment of use and the tasks they are using the system for.

4.1.1.1 Target Groups

In principle the developed software is supposed to be used by governmental and non-governmental authorities. A focus on governmental authorities as primary target groups namely *Paramedics*, *Police* and *Firefighters* is set.

Paramedics The first group of users targeted by this application are the paramedic services, who have the responsibility to respond to medical emergencies as soon as possible. The paramedics can use the *CaSe* scheduler to coordinate between the paramedics who have to be on site, those who have to be on standby and those with a support role. In the event of a shortage of personnel, the paramedics also depend on volunteers. Thus, this application would facilitate coordination between different groups of paramedics responding to an emergency and in effectively planning and distributing their human resources.

Police The police department responds to different types of incidents, which can range from day to day problems to critical situations. Moreover, police stations are distributed across different locations in cities. Thus, prioritization and scheduling are very important requirements for their efficient functioning. Moreover, the police stations are distributed across different locations in cities. The *CaSe scheduler* can assist the Police department in collaborating and coordinating between their various units while responding to critical and non-critical situations.

Firefighters Another important group of potential target users are firefighters. Firefighters respond to different kinds of emergencies, ranging from preventing fire to educating the local community and conducting inspections for fire prevention. Scheduling is thus an important requirement as resources are limited and the response times have to be very low to avoid damage to life and property. This is particularly true for large dense urban areas.

4.1.1.2 Personas

Based on the user research, following three personas are created, which represent target users, including a name and picture, roles, goals and tasks.

John Gale Being a paramedic for 17 years John (Figure 4.1) became acting chief thanks to his many years of experience and wealth of education. Since he took over a management role he is not only responsible for his own team but also for the communication with other local services. As a result he spends his days more often in the office exchanging emails to arrange meetings instead of actively doing field work. He wishes there would be a more time saving approach for catching up with the people he needs to meet. Whenever there is some free time, John likes to spent it with his family and his son in particular.

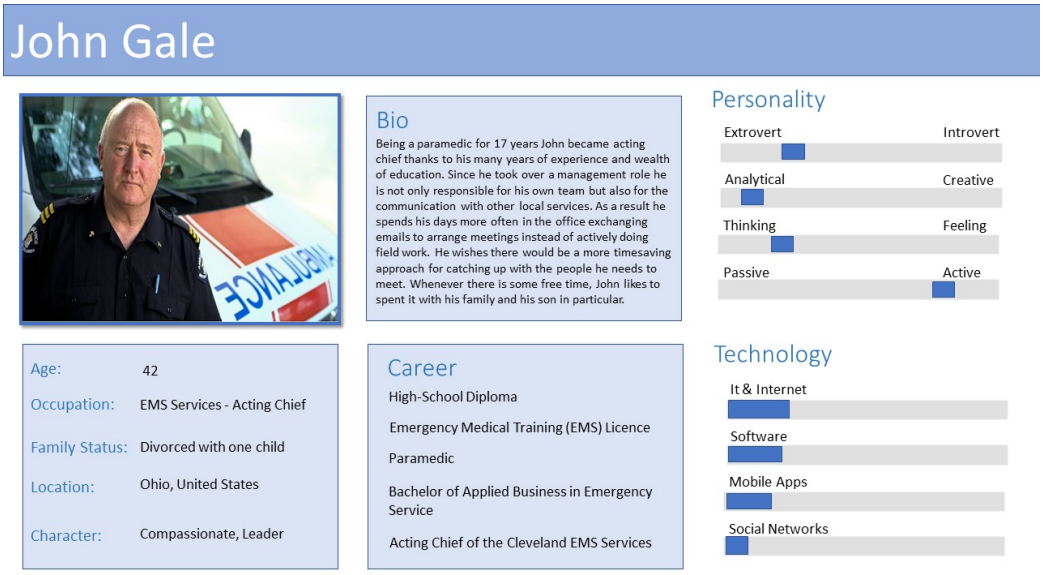

Figure 4.1: Persona John Gale

Sarah Smith Sarah (Figure 4.2) knew from a very young age that police work would be the career that she wanted to pursue. As an assistant crime analyst, she is supposed to work with different police agencies throughout the UK. Much to her regret, she observed that tackling organized crime

is often accentuated by a lack of communication between those agencies. Instead of sharing information for mutual benefits they are rather operating as rivals. In a preventive manner she started to organize regular meetings with representatives of the respective agencies. This comes at a price of her sometimes feeling like she is drowning in emails.

Figure 4.2: Persona Sarah Smith

Thomas Lang Even though his parents wanted Thomas (Figure 4.3) to study law he always preferred working with his hands instead of dryasdust studying from books. Consequently he applied for a job at the local fire department. For him it is exciting and challenging to arrive on scene and prepare the fire hose, throw ladders, and rescue people. Since becoming chief in command, he is responsible before the start of a mission to instruct the attacking team with exact details on how to approach the scene. He noticed that there are communication issues between the professional and volunteer firefighters. To improve the efficiency he wants to organize training events with the volunteers.

Figure 4.3: Persona Thomas Lang

4.1.1.3 User Scenarios

The following user scenarios describe situations in which the developed application is actually by the previously created personas to achieve specific goals. Creating scenarios helps to put an increased focus towards design efforts on the user's requirements, which are distinct from technical requirements. The scenarios depicted in the next paragraphs are quite abstract, as this lets them to be particularly useful for generating design ideas and for understanding the requirements of the system.

A Typical Day in John Gale's Life While being at his office John wants to organize a meeting for the upcoming football match at his local town stadium. In order to take precautionary measures he wants to terminate an appointment with the local police and security service of the stadium. By utilizing his new software he is proposing six different dates. On his way home he checks his smartphone application if the invitees already agreed on a date. He noticed that his contact person from the police is not available on any of the proposed dates. He quickly replace the invitee with another authoritative decision-maker of the local police force taken from his contact list. Only half an hour later he received a notification on his mobile phone

that the new invitee accepted four of the six proposed dates. Finally he found a date where authoritative figures from all the participating authorities are available and hence marks the date as an appointment which will take place.

At the Beginning of Sarah's Week Sarah's start in the week begins with organizing an upcoming meeting between officials of her police department and representatives of various government bodies. Apart from finding a proper location, this also involves providing refreshments and snacks. She is stressed out due to being supposed to find a new location for the meeting since the originally scheduled location isn't available anymore. Being on the phone all day long she finally found a suitable location for the event and reserved meetings rooms for several days. She is aware of the fact that the governmental representatives have tight appointment schedules. As a consequence proposed dates should include different day times as well. She is using *CaSe* to create various appointment proposals and invites the prospective participants. At the end of the week Sarah checks the responses for the proposed dates and saw that the invitees agreed upon a suitable date. Finally she uses an option in her software to set the proposed event as an active event so that the participants are informed that the event will take place.

A Thought Comes to Thomas Mind Over the last months Thomas noticed that the volunteer firefighters are not well enough prepared for performing their duties alongside the professional firefighters. In agreement with his colleagues he wants to offer training events where professional and volunteer staff are working hand in hand. He is aware of the fact that because of job demands and other factors affecting the volunteers, the training events should have flexible dates. In order to realize Thomas idea, his department provided him with a web service which supports him in the scheduling process. He creates an account at the service and imports his contacts from his private *Outlook* account. When the import is done he creates two categories called "professionals" and "volunteers". Hence he is able to quickly select between those two groups when creating invitations for training events. On the weekend Thomas wants to check if the volunteers agreed upon a date for training drills and if there is a professional firefighter available for that time. He turns on his computer, logs in to a service and sees a list of proposed dates for the volunteer training sessions. Thomas sees that for most of the

volunteers and his colleagues next Wednesday is suited best and hence marks the date as fixed.

4.1.1.4 Task Analysis

This section lists the tasks resulting from the given scenarios. Tasks that repeat themselves over the scenarios are only listed once.

Tasks of John Gale's Scenario Following tasks are taken from Gales Scenario:

Task 1: Users of the software need to be able to maintain a contact list.

T1: It is mandatory to have the option to propose more then a single date to invited contacts in order to agree on a common date.

T2: Invitees need to be able to receive invitations and to reject or accept the proposed dates.

T3: If needed, invited participants should easily be replaced with suitable candidates.

T4: If a proposed date is suitable for the invitees the event creator can mark the proposed date as an actual appointment for everyone to see.

Tasks of Sarah Smith's Scenario Following tasks are taken from Smiths Scenario:

T5: Proposed dates can exist of different times and locations.

Tasks of Thomas Lang's Scenario Following tasks are taken from Langss Scenario:

T6: Contacts can be sorted into different categories.

T7: Contacts can be imported from common contact applications.

4.1.2 1st Iteration

During the first iteration a top-level design of the user-interface is created based on the results of the preceding task analysis and subsequently evaluated with an usability expert. The first design is realized in form of low-fidelity mockups. The advantage of using low-fidelity mockups is that they can be easily adapted to the feedback received during the evaluation phase[22]. For the creation of the mockups the prototyping software *Balsamiq Mockups*[1] is utilized.

4.1.2.1 Prototyping

This section presents the first mockup drafts for the application. To meet the expectations of the preceding task analysis the first prototype exists of four different main views. These are the *Login View*, *Schedules View*, *Categories View* and the *Contacts View*.

Login View When opening the corresponding website of the web service in the browser the user is navigated to the *Login View* (Figure A.1). A login functionality is necessary so that user interaction as required by requirement *R1, R2, R3* and *R4* are possible.

The access to the actual service is restricted to registered users only. If the user doesn't have an user account yet he can click on "*No account yet?*" in order to create a new account.
With the user name and password entered, one is navigated to the schedules view and can freely navigate in the web service by clicking on the menu options on the menu left.

Schedules View After a successful login the *Schedules View* (Figure A.2) is the initial view which is presented. The calendar in the middle of the screen shows all important information at a glance. Namely events the logged-in user is invited to and personally-created events. With having events visible at a glance the requirements *R1, R2, R3* and *R6* are met.

A tab-bar on top of the calendar allows to either create a new event by clicking *"Create new schedule"* or to manage created events and react on invitations by clicking on *"My schedules"*.

[1] https://balsamiq.com/products/mockups/, accessed 15.08.2017

When clicking the button for creating a new schedule a new view is opened which separates the creation process into four steps. The functionality for adding a new event mets the requirement *R1* and *R5*.

The navigation from the first to the last step is realized by pressing the *"Next"* button in the bottom right corner of the context frame.

Firstly a name and a description need to be assigned to the event by filling the respective fields (Figure A.3).

Secondly when hitting the *"Next"* button a new view appears where any number of dates can be selected for the event by highlighting the fields in the displayed calendar (Figure A.4).

In the third view the user is able to add multiple time and location frames (Figure A.5) to every date which has been defined before by hitting the *"New"* button below the date headers.

The last view allows to add any number of participants to the event where the participants are separated in different categories (Figure A.6). This functionality is realized similar to the view before where the user added times and locations to the respective dates. By hitting the *"New"* button below the date headers a modal window is opened where the desired category and the required number of attendees from this category can be selected (*R5*). In contrast to the view before the headers do not only consist of the date, but also of the previously picked times and locations. For every pair of time and location exists another header. Finally the configured event can be created by clicking the *"Create schedule"* button in the bottom right corner and the user is navigated back to the initial *Schedules View*.

Besides creating a new schedule the user is able to overview his created schedules and invitations when clicking the *My schedules* button shown in figure A.7. This view fulfills the requisites of *R6*.

Categories View The *Categories Views* (Figure A.8) displays already created categories in a list frame left and a corresponding details frame right. This view allows to manage and overview categories (*R8*). When a category of the list is selected the detail frame provides information about the category. Namely the title, shortcut and description of the category. New categories can be created by clicking on the *"Add category"* button which is located on top of the categories list.

Contacts View The *Contacts View* (Figure A.9) is quite similar in pattern to the *Categories View*. Available contacts are shown in the contact list left while details of a highlighted contact are shown on the right hand side of the list. This view provides the functionality for maintaining a contact list and assigning contacts to categories as required by *R4* and *R7*. When a contact from the list is highlighted it can be assigned to a predefined category via the *"Set category"* button in the bottom left corner of the details frame. The contact information exists of first name, last name, age, sex, mobile number, home number and the email address. Besides adding a contact by clicking on the *"Add contact"* button it is also possible to import contacts from common contact application by clicking the buttons with the respective company logo. When the respective button is clicked a sub window is opened where the user can enter his credentials.

4.1.2.2 Evaluation

By taking into account the heuristics described in section 3.2.4 for the design of user-interface by *Jakob Nielsen* were the following deficiencies identified:

H1: Visibility of system status

When a new event was created by clicking the *"Create schedule"* button (Figure A.6) appropriate feedback must be provided so the user know the event was actually created. In the current design the user just gets relocated to the *schedules view* (Figure A.2) without any feedback.
Furthermore both success and error feedback is required for adding contacts and categories, importing contacts and assigning categories.

H2: Match between system and the real world

The buttons for accessing contacts from the individual contact services (Figure A.9) need to be named accordingly. By the icon alone a user might not know what functionality is provided by clicking the button and rather ignore it.

H3: User control and freedom

When a view listed on the main menu is clicked and a sub-view within is opened there is no possibility to return to the root view except for clicking

on the main menu again. Users might open views by mistake or out of curiosity. In the current prototype each view lacks the "undo" possibility. When a user feels stuck he might accidentally close the browser window or hit the back button of the browser which could result into unintended system behavior.
Furthermore the current navigation status is unclear to the user. When entering a view the main menu displayed left at every view must clearly display which is the currently active view which is not visible in the current prototype. Moreover when opening views which aren't displayed at the main menu, like adding or managing schedules, could result in the user losing track of his current position in relation to the rest of the site.

H4: Consistency and standards

The navigation alignment is inconsistent. When entering the *Schedules View* (Figure A.2) the user is supposed to click on a tab-bar on top of the context frame for further navigation. This contrasts to the interaction with the remaining views where the buttons are always at the bottom side of the respective context frame and confuses the user. Moreover when a new schedule is created the top space of the frame is used for showing the current progress of the creation process which adds more to the confusion.

H5: Error prevention

This heuristic couldn't be applied on a low-fidelity prototype.

H6: Recognition rather than recall

This heuristic couldn't be applied on a low-fidelity prototype.

Flexibility and Efficiency of Use (H7)

The current solution for assigning a category to an individual contact by highlighting the contact from the list and hitting the *Set category* (Figure A.9) button is regarded inefficient. It only supports single assignment instead of being able to select multiple contacts at once. For a more flexible approach and also in regard of the importance of this functionality a separate view needs to exists in the main menu in order to efficiently handle assignments. Similar to the functionality of assigning categories the import of contacts

is also considered a key functionality of the system and therefore must be available at a glance via the main menu.

H8: Aesthetic and minimalist design

This heuristic couldn't be applied on a low-fidelity prototype.

H9: Help users recognize, diagnose, and recover from erros

This heuristic couldn't be applied on a low-fidelity prototype.

H10: Help and documentation

This heuristic couldn't be applied on a low-fidelity prototype.

4.1.2.3 Findings

In the following all the findings from the individual views are summarized which are essential for the design of a new high fidelity prototype.

Login View No deficiencies were found for the *Login View*.

Schedules View To begin with the navigation controls for all sub-views should be located at the same position and not moved to new locations on different pages. When accessing sub-views within the main-views, back buttons are necessary in the navigation bar so that users can undo actions if wanted. Moreover users have to be aware of their current location when navigating trough the application. Therefore besides highlighting the current view in the main menu a breadcrumb navigation has to be added to each view and located standard conform at the top left corner of the context[23]. When an event is successfully created, feedback has to be provided in form a notification.

Categories View In order to easily assign categories to contacts a new view is created accessible from the main menu. To support multi-selection a drag and drop feature is applied where any number of users can be dragged and dropped on the category of choice. If a contact is dropped on a category tag, a notification for a successful assignment needs to be displayed.

Contacts View The import of contacts needs to be realized in a single view reachable from the main menu. To provide a better understanding links to the individual contact services have to be named accordingly as the icons could cause confusion. When the contact import is successful, appropriate feedback needs to be provided.

4.1.3 2nd Iteration

In the second iteration a high-fidelity prototype is created in consideration of the findings of the first iteration. The created prototype is fully interactive (all menus and links are clickable) in order to support usability-testing in the evaluation phase by applying the *Think-Aloud-Method*.

4.1.3.1 Prototyping

This section introduces the most significant functionalities of the newly created prototype. In total 34 different mockups are created in order to match the look-and-feel of the to be developed live system as good as possible. As only some of the mockups are shown in the appendix, the entire prototype can be viewed as a *PDF*-file which can be found on the attached CD. While the new prototype inherits a fully colored design and various improvements in the layout compared to the first prototype the following subsections focus on the core features introduced by the new prototype.

Schedules View Figure B.1 shows the initial view after the login. Navigation is now oriented to the bottom side for a consistent navigation layout throughout the application as required by *H4*. The application menu on the left consists of the three main views which were drawn up in the first iteration. These views now inherit sub views on the menu for a more flexible access to the covered functionalities (*H7*). Additionally a profile view (Figure B.2) is added to the menu which allows the user to share personal information with other users of the application.

When creating a new schedule the option is given to set this event as an recurring event as seen in Figure B.3. After filling out the forms and clicking next the user can add any number of dates via the displayed calendar (Figure B.4). In contrast to the first prototype a back button is added on the bottom left for a better user control (*H3*). Additionally the top bar now highlights

already passed views and allows backward navigation (H3) by simply clicking on the tabs.

The layout for adding time and location items is changed in the way that added items are now displayed from left to right instead of top to bottom (Figure B.5). Additionally the plus bottom is dynamically moving with the added items and its layout is of the same size as the item boxes for keeping the layout consistent (*H4*). When the plus button is clicked a modal window is opened shown in Figure B.7. Here a time range can be set with the provided fields. For setting up the location a map is placed below the input field which displays the entered location with a marker in order to support the user in the process of finding the right location.

For a consistent look-and-feel (H4) the layout for adding category items in the last view of the event creation process (Figure B.6) is build and sized in the same way as in the previous view. Clicking the plus button for adding categories opens a modal window (Figure B.8) where the number of participants and the category from a list of created categories can be selected.

Finally when the event is successfully created the user is navigated to the initial view from which the event creation process started and receives a notification that the operation was successful (Figure B.9).

Manage Schedules View While in the first prototype the functionality for managing created events and reacting to invitations is combined in a single view it is now split in two different views (Figure B.10) separated by a tab bar on top of the context layout. The separation of those views allows a more flexible approach to the provided functionalities as required by *H7*.

The initial active tab gives an overview of created events shown in a list left. By selecting an event its details are shown in the calendar on the right side. When a day is clicked in the calendar a modal window enables to invite contacts which is shown in Figure B.11. Names of the invited users are colored blue showing a pending status, green when they accepted the invitation and the names are removed if the invitation is rejected by the respective invitee. A colored feedback supports the visibility of the system status (*H1*).

Only when the required number of participants is met, as shown in the second appointment of the figure, can the respective appointment be set as active via the switch button far right.

When hitting the *Invitations* tab all the invitations are shown in a list

(Figure B.12) ordered by the time they are received. By clicking an invitation item from the list a dialog window opens where the user can accept or refuse the proposed dates (Figure B.13).

Assign Categories View The functionality for assigning categories to individual contacts is now provided as a separated view which can be seen in Figure B.14. Existing contacts are shown in a list and can be comfortably assigned to categories with the aid of a drag and drop feature for an efficient use of the assign feature (*H7*). When the mouse pointer hovers over a letter of the categories bar on top a pop-up opens showing all categories with the respective initial letter. Any number of contacts can be dragged from the contacts list and dropped on a category represented by the alphabet in the top bar.

Synchronize Contacts View As the import of contacts is another key feature of the system, it's functionality is now also provided in a standalone view (Figure B.15), as required by *H7*. The provided services exists of Google Contacts, Microsoft Outlook and the email-provider Yahoo. The *Synchronize* button is initially colored blue. When the button is clicked and the import is successful it is colored green, otherwise red.

4.1.3.2 Evaluation

For the evaluation of the second iteration the *Think-Aloud Method* is applied. Three test users have to test the prototype by performing the user stories (abbreviated "US" in the following) described below.

US1: Create a new account with the user name "user", email "user@gmail.com" and the password "password".

US2: Login to the application with the registered credentials.

US3: Create a category named "SAS".

US4: Import the contacts from your Google account.

US5: Assign the imported contacts to the category "SAS".

US6: Create a new event with the title "SAS Meeting". The event takes place on the 25th of August 2017. Moreover the event starts at 1 p.m.

at the location "Takustraße 7" and is supposed to last 2 hours. For the event are three participants of the category "SAS" required.

US7: Invite three contacts to the created event.

In the process of solving given user stories the test persons are told to verbalize their thoughts.

US1 - User Story 1 User 1 and 2 carried out the task without mentioning any problems. The third test user didn't reply while solving the task.

US2 - User Story 2 User 1 and 3 carried out the task without mentioning any problems . User 2 asked how he would get access to the application from the login screen (Figure B.16) if he forgets his password.

US3- User Story 3 All the test users carried out the task without mentioning any problems.

US4- User Story 4 All the test users carried out the task without mentioning any problems.

US5 - User Story 5 When assigning contacts (Figure B.14) to the created category "SAS" user 1 was confused about the column "Function" of the contacts table as he thought it would be the category the contacts were assigned to. User 2 carried out the task without mentioning any problems. User 3 mentioned that the column "Function" confused him as he thought the contacts which he imported were already automatically assigned to a category. When it was made clear that the column isn't related to the category, the user was wondering how he could sort the contacts by their assigned categories.

US6 - User Story 6 During the process of creating a new event User 1 suggested that an already available default category might be a good idea which holds all contacts of this user. User 2 was confused when being told to create a new event as he thought the button name *"Create schedule"* (Figure B.1) serves another purpose. As a result he switched between views multiples times before clicking the button. User 3 carried out the task without mentioning any problems.

US7 - User Story 7 All the test users carried out the task without mentioning any problems.

4.1.3.3 Findings

As a result of the *Think-Aloud-Method* the following deficiencies of the current design solution were discovered. If no deficiencies are found for a respective user story it is not mentioned below.

US1 - User Story 1 In the event that a user forgets his password there is no mechanism provided in the current prototype design to gain access to his account. As entering an email at the registration of a new account is mandatory, this address can be used for password recovery.

US5 - User Story 5 When assigning categories to contacts via drag-and-drop (Figure B.14) the column "Function" of the contact list may be confused with being the category the user is currently assigned to. The column provides additional information about the individual contact and should only be visible in the respective view for managing contacts (Figure B.17). Moreover users currently can't sort the contacts list by their assigned categories. This can be done by opening a pop-up window when clicking the initial letter of a category in the assign (Figure B.14) view and selecting the category of interest. Moreover the tags "General" and "Unassigned" in the top bar shouldn't both be underlined, only the active setting, as the user currently doesn't know which setting is active.

US6 - User Story 6 Due to "Schedule" being a broad term which can include an overview of many different events the button title "Create Schedule" isn't appropriate for adding new events and has to be replaced with a more distinct title. Moreover a default category should be available from the beginning which includes all contacts of the user.

4.2 Backend

In the following section the core tasks of the backend, derived from designing the frontend, are briefly introduced. The backend is implemented in *Java* and utilizing the *Spring Framework* for managing database access.

4.2.1 Authentication System

The user authentication is controlling who can access the application and what data is visible to the respective user. The system basics consists of:

- Registration: Creating a new user account for the application and store given credentials in the database.
- Login: Checking entered credentials with the respective database entry and grant or deny access to the application accordingly. Only content related to the particular user is shown.
- Password retrieval: Sending an email to the users registered email-account with instruction for resetting the password.

4.2.2 API Access For Importing Contacts

In order to import existing contacts from common mail providers like *Google* and *Outlook* the backend logic for accessing the respective API server needs to be implemented. Like the aforementioned companies most mail providers are using the *OAuth 2.0 Protocol*, which is widely accepted as industry standard, for verifying the identity of users sending API requests. The Java library *"ScribeJava"*[2] is applied for implementing the protocol. As described by *Suhas Pai et al.*[12] the protocol flow can be summed up in six steps:

1. "The client requests authorization from the resource owner regarding the usage of the resource owner's protected resource."
2. "The client receives an authorization grant which represents the authorization provided by the resource owner."
3. "The client requests the authorization server to provide it with an access token which can be used to access the protected resources. During this process, the client provides its client credentials and the authorization grant to authenticate with the authorization server."
4. "The authorization server confirms the validity of the client credentials and the authorization grant and provides the client with an access token."

[2]https://github.com/scribejava/scribejava accessed 14.08.2017

5. "The client requests the protected resources hosted at the resource server by producing the access token."

6. "The resource owner checks for the validity of the access token and if valid, it services the request."

Chapter 5

Implementation

At the beginning of this chapter the application architecture and applied design pattern are shown. This is followed by an introduction of the used technologies for the implementation separated by frontend and backend usage. The last section shows the actual implementation of the frontend and backend highlighted by relevant code snippets. The complete source code of the application is included on the attached DVD and available under *https://github.com/saringer/case-webservice*.

5.1 Architecture

In this section the general architecture of the application and the design pattern for the implementation is introduced.

5.1.1 Architectural Pattern

The architectural design is laid out as an layered architecture, containing a total of three layers as shown in Figure 5.1. Namely the *presentation layer*, the *domain layer* and the *data store layer*.
The *presentation layer* is providing the user-interface, which is built on top of the *domain layer* by using the open-source web framework *Vaadin V8.1* . Data for the user interface is provided by the *domain model*, which implements the required "business logic" and handles data access with services offered by the application framework *Spring*. Furthermore the *domain layer* defines the data model in form of *JPA Entities* which are implemented in

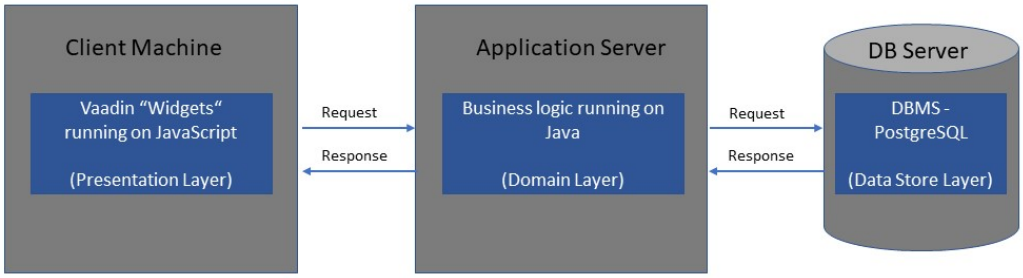

Figure 5.1: Application Architecture

form of *"Plain Old Java Objects" (POJOs).*

Beneath the *domain layer* lies the *data storage layer* which exists of a relational database. The database used in the context of this thesis is *PostgreSQL*[1]. The dependencies between the layers are restricted so that a higher layer may depend on a lower one, but never the other way around.

5.1.2 Design Pattern

As a design pattern the *Model-View-Presenter (MVP)* pattern is applied which "is one of the most common patterns in developing large applications with Vaadin"[29]. MVP divides the application in the model, the view and the presenter. Each view has a corresponding presenter which responds to

[1]https://www.postgresql.org/ accessed 01.09.2017

UI events and passes data to the view when changes to the model's state are detected. This approach aims for separating the business logic from the interface logic so that the code is easier to understand and maintain. While the original pattern doesn't allow any business logic to be implemented in the respective view[26], in the context of this thesis a variation is applied which is called the *Supervising Presenter*. In this variation the view is allowed to directly interact with the model to perform data binding which is supported by *Vaadin* components. As an advantage boilerplate code is reduced[27][29] because the presenter is not required for simple UI changes and data presentation.

5.2 Technologies

This section gives an overview of the significant technologies employed for the implementation of the frontend and backend.

5.2.1 Frontend

For the implementation of the frontend the open source framework *Vaadin* is used. *Vaadin*[2] provides a Java-to-Javascript compiler for providing a client-side engine and hence allows to create user-interfaces purely in Java or any other *JVM* compatible language. This has the advantage that no client side code in form of *Javascript* and *HTML* needs to be programmed and the frontend can easily be integrated with the backend. Moreover Java being the most popular programming language 2017[28], providing both frontend and backend in Java should simplify the access to the application for future extensions.
Though if necessary, *HTML* and *Javascript* components can be integrated in *vaadin*.

As *Vaadin* is featuring a server-side-architecture the entire application logic is located at the server-side. The server-side framework holds the UI logic and current state of every client which is connected to the server. Each server-side component (e.g. Button, TextField, Table) has a client-side counterpart, called *"widget"*. When an event like a button click occurs, the browser will asynchronously notify the server which handles the request and updates the UI.

[2]https://vaadin.com/home accessed 02.09.2017

5.2.2 Backend

For the implementation of the backend the technologies *Spring* and *Maven* are utilized.

Spring Framework

The Spring Framework[3] is an open source application framework, which provides various different modules and its core functionality of configuring and managing Java objects using the *Dependency Injection* design pattern. Simply put, "Spring handles the infrastructure so that the developer can focus on his application"[31]. For the implementation of the backend the modules *Spring Boot*, *Spring Data* and *Spring Security* are applied.

Spring Boot provides an embedded Tomcat server for the managed application and automates and simplifies the Spring configuration. The maxim of this module is that a developer can "just run"[32] his application.

Spring Data provides repository interfaces for simple CRUD (Create-Read-Update-Delete) operations on domain objects. The major advantage is that Spring Data automatically provides appropriate implementations of written interfaces at runtime.

Spring Security provides different tools for the implementation of the basic security for a web application. Notable the *Authentication Manager* and *Password Encoder* for secure storage of passwords in the database.

Maven

In order to resolve required dependecies for the application the build management tool *Maven*[4] is applied. The configuration of dependencies, other external modules, components, build order and required plug-ins is handled by an XML file.

5.3 Implementation of Frontend

"Vaadin Framework user interfaces are built hierarchically from components, so that the leaf components are contained within layout components and other component containers"[30].

[3]https://spring.io/ accessed 02.09.2017

[4]https://maven.apache.org/ accessed 02.09.2017

The user interface of the application is built hierarchically starting from the top with the *AppUI.class* which is extending the *Vaadin UI* component and called every time when a user establishes a connection to the service.

Listing 5.1: User validation

```
if (!accessControl.isUserSignedIn()) {
    setContent(new LoginForm(accessControl,
                    (LoginForm.LoginListener) () ->
                showMainView(), viewProvider, AppUI.this));
} else {
    showMainView();
  }
```

The above shown source code is called by the *AppUI.class* and navigates the user to the login component (*LoginForm.class*) of the application. Authentication logic is covered in the section *Implementation of Backend*. After a successful login process the user is redirected to the main view of the application. If the users credentials are already stored in the current browser session he can directly access the main view.

The main view of the application is implemented in the *MainScreen.class* which exists of two components, namely the main menu and the view container. The menu component implemented by the *Menu.class* provides methods for dynamically adding views and sub-views to the menu so that the user interface can be extended comfortably in the future with more views.

Listing 5.2: Vaadin Navigator

```
final Navigator navigator = new Navigator(ui, viewContainer);
```

The view container is a simple layout which is passed to the *Vaadin Navigator* (listing 5.2) and displays the content of the respective view clicked in the menu. Due to *Vaadin* applications usually running in a single page the navigator is always required for switching between views. Views managed by the navigator support bookmarking, forward navigation and backward navigation via the used browser by automatically adding a distinct URI fragment to each view.

On Figure 5.2 the menu and the view container are shown. The menu is located on the left and the rest of the space available in the page is consumed by the view container which currently displays the content of the view implementation for adding a new event.

Views which are handled by the navigator are classes which extend a *Vaadin Component* and implement the *Vaadin View* interface. In line with

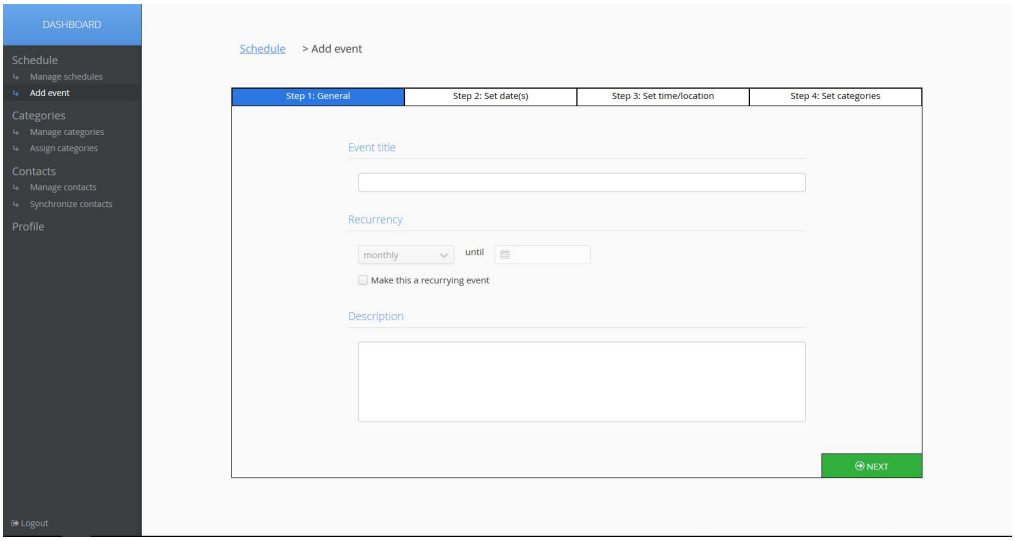

Figure 5.2: View for Adding a New Event

the *Supervising Presenter* pattern the view class itself inherits basic navigation logic (e.g. breadcrumbs), while more complex logic is handled in the respective *Presenter* which exists as a separate class for each view. All views provide the life-cycle methods *init()* and *enter()*.

Listing 5.3: Lifecycle methods of a view class

```
1     @PostConstruct
2     void init() {
3       addComponent(breadCrumbs);
4       ...
5       form.addComponent(calendarPanel);
6       form.addComponent(bottomNav);
7       addComponent(form);
8       ...
9       initPresenter(schedulerMainViewPresenter);
10        }
11
12    @Override
13    public void enter(ViewChangeListener.ViewChangeEvent event) {
14        ...
15        schedulerMainViewPresenter.loadInvitationsNotification();
16        schedulerMainViewPresenter.initCalendarEvents();
17    }
```

Whenever a new instance of a view class is created the first method called

is the *init()* method which is responsible for building the layout of the view as shown in the Listing 5.3 taken from the*SchedulerMainView* class. Furthermore the *@Postconstruct* annotation is used to ensure that dependency injection handled by the *Spring* framework has already been done at this point. The *enter()* method is called every time a user navigates to the respective view so that the presenter class gets notified to provide data for the UI components created beforehand in the *init* method (see listing 5.3).

The hierarchy of the view classes in regard to the beforehand developed prototype is shown in figure 5.3.

Functionality of the views has already been described in chapter four where the application is iteratively designed. Therefore the following subsections only briefly introduce the functionality and rather focus on presenting the significant *Vaadin UI Components* used in each view for the implementation of the frontend.

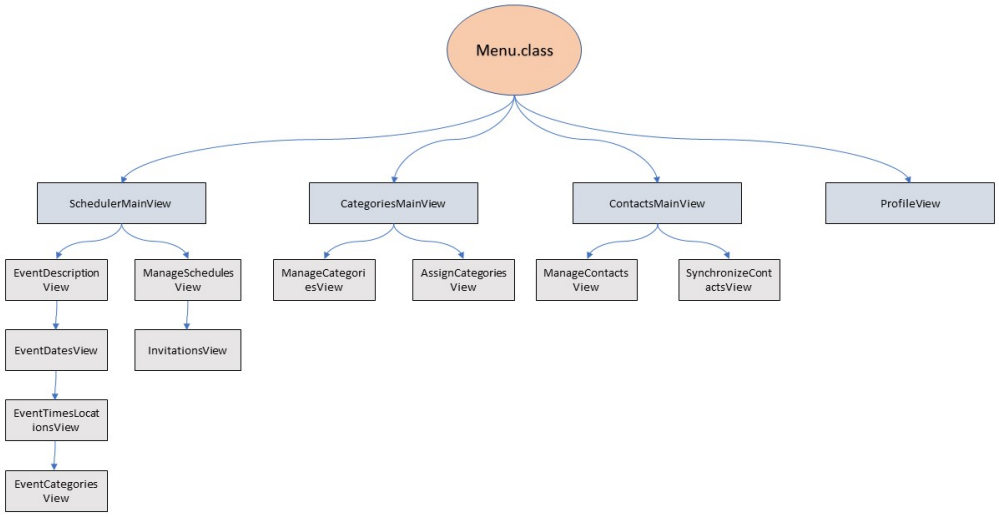

Figure 5.3: View Hierarchy

5.3.1 Class SchedulerMainView

This view is the first view displayed when a user passes the authentication and exists of a calendar component and a navigation bar below. Being the initial view the user gets an overview of the events he is related to at a glance. The shown calendar displays all the events where the user is invited to or which are created by himself. The calendar implementation is a custom component based on a *Vaadin* add-on taken from the add-on library[5] and extended by a simple navigation in the *CalendarComponent.class*.

A new event can be created by clicking the *"Add event"* button in the bottom navigation bar or from the main menu. This process consists of four steps, implemented by the following view classes:

1. *Class EventDescriptionView*

   Text input for setting up the event title and description is realized by a *TextField* and a *TextArea* component. If an event should be repeated monthly or weekly is set with a *ComboBox* component and the date till the recurrence is active is picked via a *DateField* component.

2. *Class EventDatesView* An instance of the *CalendarComponent.class* allows to select any number of dates by simply clicking on a date. Clicking an already selected date deselects it. Additionally the bottom navigation provides a button for resetting the selection.

3. *Class EventTimesLocationsView* Times and Locations can be added to the before selected dates by clicking the button with the plus icon below the date headers. When clicked a modal window is opened via the *addWindow()* method of the *Vaadin UI* as seen in listing 5.4.

Listing 5.4: Adding a sub-window to the current view

```
1 plusButton.addClickListener(new Button.ClickListener() {
2             @Override
3             public void buttonClick(Button.ClickEvent event) {
4                 Window setTimeLocationWindow =
5                 new SetTimeLocationWindow(day, itemLayout,
                      plusButtonLayout);
6                 UI.getCurrent().addWindow(setTimeLocationWindow
                      );
7             }
8         });
```

[5]https://vaadin.com/directory#!addon/calendar-add-on accessed 01.09.2017

The window is implemented as a custom component in the *SetTimeLocationWindow.class* which extends the *Window* component. The time range is set with two *TimeField* components which only allow valid times as input. The selection of a location exists of two components provided by *Vaadins* add-on library. Firstly the *LocationTextField*[6] component which features a geocoder from Google for providing auto-complete and suggestions when entering addresses. Secondly the *GoogleMap*[7] component for displaying the currently selected address.

4. *Class EventCategoriesView* This is the final view where the event can actually be created when required categories and the number of participants per category are set for all previously added times and locations. This is done similar to the procedure of adding times and locations via a *Window* component. For setting the number of participants the *Vaadin* add-on component *IntStepper*[8] is applied. For displaying the available categories the Vaadin *Grid* component is used which can be binded with the object class it it supposed to display.

Via the second button from the bottom navigation bar (*"Manage schedules"*) the user gets access to the maintenance views for created schedules and invitations. Those views are implemented by the following two view classes and are separated by a tab bar:

1. *Class ManageSchedulesView* The view exists of two horizontally aligned context frames. The frame on the left lists the created events by the time of their creation. When an item inside the list is clicked, all the dates which belong to the respective event are displayed in the *CalendarComponent* in the context frame right. By clicking a date displayed in the calendar a modal window is opened, as shown in Figure 5.4, which is implemented by the *SetParticipantsWindow* class.

2. *Class InvitationsView* Initially a list of event items is shown with the event name, description and date. When an item is clicked a modal window is opened *Class HandleInvitationsWindow* where times and location for the respective date are listed up and can be rejected or accepted.

[6]https://vaadin.com/directory#!addon/locationtextfield accessed 01.09.2017
[7]https://vaadin.com/directory#!addon/googlemaps-add-on accessed 01.09.2107
[8]https://vaadin.com/directory#!addon/stepper accessed 01.09.2017

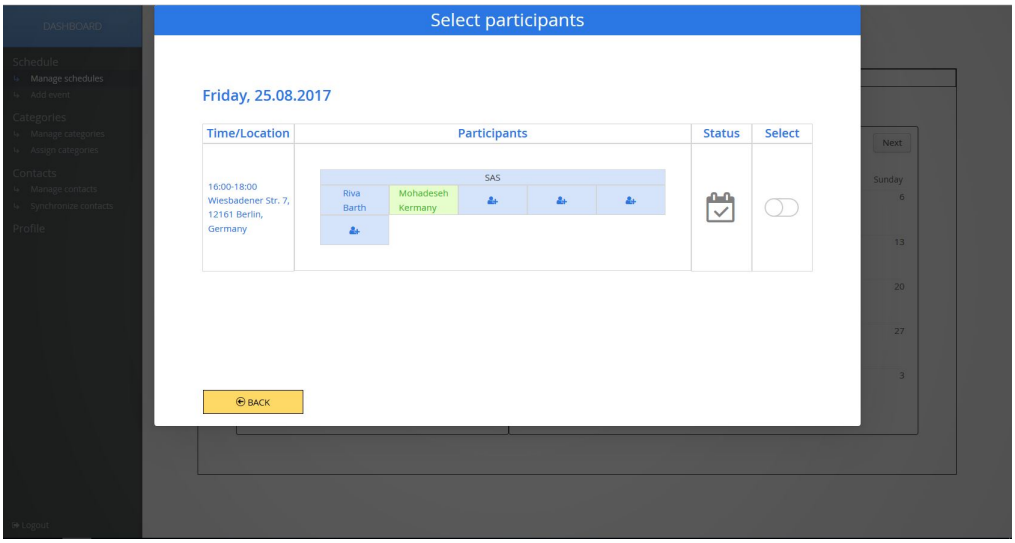

Figure 5.4: View which Allows to Add Participants for the Selected Date and Check on their Response

5.3.2 Class CategoriesMainView

This view serves as container view for accessing the following two view classes:

1. *Class AssignCategoriesView* In order to assign contacts to categories via drag-and-drop a *Grid* component was used for displaying draggable contact items in a list. For dragging multiple rows the selection mode of the *Grid* component has to be set appropriate as shown in line 1 of Listing 5.5. To make the component a drag source, the *GridDragSource* extension needs to be applied to the grid (line 2).

Listing 5.5: Grid Component with Draggable Rows

```
1  grid.setSelectionMode(Grid.SelectionMode.MULTI);
2  GridDragSource<Contact> dragSource = new GridDragSource<>(grid)
       ;
3          dragSource.addGridDragStartListener((GridDragStartEvent
               <Contact> event) ->                 {  ...
4              });
5
6          dragSource.addGridDragEndListener(event -> {
7              ...
8               }
9
10 });
```

The alphabetical stripe for representing the initial letters of the categories is implemented with Vaadin *Label* components. For each *Label* the content mode is set to *HTML* so that it was possible to add *Javascript* listeners for drop and click actions.

2. *Class ManageCategoriesView* This class mainly exists of a *Grid* component which shows all the categories in a list and various *TextArea* components for displaying and modifying the information of the selected category.

5.3.3 Class ContactsMainView

This view serves as container view for accessing the following two view classes:

1. *Class SnychronizeContactsView* The functionality for importing contacts is provided by the *OAuth2 Popup Add-on*[9] from the *Vaadin* add-on library. This add-on is based on the *ScribeJava*[10] library and is described in more detail in the section *5.4.2* of the frontend implementation. By clicking the *OauthPopUpButton* component provided by the add-on , a browser window is opened which redirects the user to the respective contact service.

2. *Class ManageContactsView* This view exists of the same component layout as the *ManageCategoriesView*. A *Grid* which shows all the con-

[9]https://vaadin.com/directory#!addon/oauth2-popup-add-on accessed 02.09.2017
[10]https://github.com/scribejava/scribejava accessed 02.09.2017

tacts in a list and *TextArea* components for displaying and modifying the information of the selected contact.

5.3.4 Class ProfileView

This views exists of the four sections *Personal Information*, *Contact Information* and *Organization* displayed in a *FormLayout* component. In order to modify the displayed fields the *Edit* button in the top right corner of the respective section layout needs to be clicked. By means of this view users can share personal information with other registered users which is accessible via the contacts overview provided by the *ManageContactsView* class.

5.4 Implementation of Backend

Below the implementation of core features provided by the backend are introduced.

5.4.1 Data Persistence and Access

The data models for the application are provided in form of *Java* objects which can directly be abstracted as database tables by utilizing the *Object-relational mapping* technique provided by *JPA*. The mapping of the Java classes and its member variables to a database table are implemented by adding annotations to the entity classes as seen in listing 5.6 for the *user* entity. In order to mark a class as an entity the *@Entity* annotation needs to be added on top of the class name which is shown in line 1. Annotations are also used for configuring relations to other data models as seen in line 11.

If a database connection is properly set up via the *application.properties* file for configuring *Spring Boot*, the framework automatically scans the project for entity classes at run time and create the respective database tables.

Listing 5.6: User Entity Class

```
@Entity
public class User implements Serializable {

    @Id
    @Column(name = "USER_ID")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstname;
    ...
    @OneToMany(...)
    @JoinTable(name = "USER_CATEGORY", ...)
    private List<Categories> categories;
    ...
```

The data access is handled via spring repositories which are provided by the *Spring-Data-JPA* module. In particular the *JpaRepository* interface class is utilized for performing *create*,*read*,*update* and *delete* operations on the data models. For each data model exists a repository class which extends the *JpaRepository* interface.

5.4.2 Contact Importer

The application supports the import of contacts from *Google Contacts*, *Microsoft Live* and *Yahoo*. For *API* access of the aforementioned services an authentication via the *Oauth2* protocol is required. As a prerequisite the application has to be registered at every single service in order to receive a *Client Id* and *Client Secret*. Furthermore a callback url had to be registered as well. The id and secret are used to identify the application when the application requests contact data. All the authentication codes required for the *Oauth2* process are stored in the *Constants* class.

The implementation of the process flow is provided by the *ScribeJava* library and abstracted by a simple service class as seen in Listing 5.7. The *clientId* (line 1) and *clientSecret* (line 2) are the credentials which are received as a result of registering the application. By passing a *scope* (line 3) the respective *API* gets to know which data needs to be accessed. Specifying the callback url (line 4) is mandatory, as this is the location where the user is redirected to after a successful authentication. Finally an *API* template is passed to the service class for providing *API* specific settings. While the *ScribeJava* library provides various templates from the start, a custom template for the *Yahoo* service is implemented (*YahooApi.class*) due to the

default template being outdated. When the authentication is successful, contact data is provided in *JSON*-format. For parsing the data the classes *GoogleContactsDBParser*, *MicrosoftContactsDBParser* and *YahooContacts-DBParser* were implemented.

Listing 5.7: Authentication Service

```
1 OAuth20Service service = new ServiceBuilder(clientId)
2                 .apiSecret(clientSecret)
3                 .scope("scope")
4                 .callback("callbackurl")
5                 .build(GoogleApi20.instance());
```

5.4.3 User Input Validation

In order to prevent that malformed data could be persisted, field validation is mandatory and is provided for every *Vaadin Component* of this application which allows user input.

Vaadin provides the *Binder* class in order to couple data objects with input fields and to add any number of validators to a field. Listing 5.8 shows the implementation of the *Binder* for the *RegistrationForm* class. The class allows the user to register a new account and exists of the three fields *username*, *password* and *email*. *Vaadin* provides basic validators like the *StringLengthValidator* (line 5, 11) and *EmailValidator* (line 14). Additionally custom validator classes can be added like the *IsAlphabeticalValidator* (line 8) by implementing the *Validator* interface from *Vaadin*.

Listing 5.8: User Input Validation with the Binder CLass

```
1  private Binder<Users> binder = new Binder<>();
2  ...
3  private void setUpValidationBinding() {
4        binder.forField(username)
5                .withValidator(new StringLengthValidator(
6                        "User name must be between 4 and 15
                             characters long",
7                        4, 15))
8                .withValidator(new IsAlphabeticalValidator())
9                .bind(Users::getUsername, Users::setUsername);
10       binder.forField(password)
11               .withValidator(new StringLengthValidator("Password
                     must be at least
                                                          6 characters long
                     ", 6, 20))
12               .bind(Users::getPassword, Users::setPassword);
13       binder.forField(email)
14               .withValidator(new EmailValidator("Not a valid email
                      address"))
15               .bind(Users::getEmail, Users::setEmail);
16   }
```

5.4.4 User Authentication

User authentication is necessary to ensure that users accessing the application only receive information related to them. For the implementation of the authentication service the default *AuthenticationManager* and *PasswordEncoder* components of the *Spring Security*[11] module are utilized.

Listing 5.9: Interface class for basic access control

```
1 public interface AccessControl {
2        public boolean signIn(String username, String password);
3        public boolean isUserSignedIn();
4        public String getUsername();
5 }
```

The Basic access control of the application is handled by three methods defined in the *AccessControl* interface which can be seen in the Listing 5.9 and are implemented in the class *BasicAccessControl*.

As already shown in Listing 5.1 the user has to pass the access control

[11] https://projects.spring.io/spring-security/ accesses 02.09.2017

when accessing the service and will be directed to the *LoginForm* component if the *isUserSignedIn()* method returns false.

Listing 5.10: Method handling the Login Process

```
1  if (!username.isEmpty() && !password.isEmpty()) {
2          ...
3      Authentication authentication = authManager.authenticate(request
           );
4      ...
5      if (authentication != null && authentication.isAuthenticated())
           {
6          AppUI app = (AppUI) UI.getCurrent();
7          app.setCurrentUsername(username);
8          parent.getLoginListener().loginSuccessful();
9                  parent.getAccessControl().signIn(username, password)
                       ;
10                 } else {
11               showNotification(new Notification("Login failed",
12               "Please check your username and password and try
                                 again.",
13                Notification.Type.HUMANIZED_MESSAGE));
14
15                }
16 }
```

The above code snippet shows the code which is executed when the login button is clicked. The significant part can be found in line 3 where the *authenticate()* method of the *AuthManager* class is called. The method retrieves the decoded password for the given user name from the database, encodes it and compares it with the given user name. Decoding and encoding of the password is handle by the *PasswordEncoder* component taken from the *Spring Security* module When the authentication is successful the *loginSuccessful()* method is called in line 9 which redirects the user to the *SchedulerMainView*.

When a user does not own an account yet he can reach the *RegistrationForm* from the *LoginForm* by clicking the *"No account yet?"* button.

Listing 5.11: Code Snippet for Handling the Persistence of a New User Entity

```
if ((usersRepository.findByEmail(email.getValue()) == null)
     && (usersRepository.findByUsername(userName.getValue()) == null
         )) {
                    Users newUser = new Users();
                    newUser.setUsername(userName.getValue());
                    newUser.setPassword(encodedPassword);
                    newUser.setEmail(email.getValue());
                    usersRepository.save(newUser);
}
```

When the fields for the user name, email and password are filled out and the *"Create"* button is clicked a new user entity is added to the database as shown in Listing 5.11. Persistence only happens provided that the given values passed the validation and neither user name nor email address are already registered.

Chapter 6

Results

In the first section of this chapter the results of the work are summarized. The following section gives an overview of the limitations of the work.

6.1 Summmary

In the process of this thesis potential users of the system are interviewed in order to create an initial set of requirements. Based on the identified requirements a first low-fidelity prototype of the application is created. This prototype is further refined by applying *Heuristic Evaluation*. Subsequently a high-fidelity prototype is designed with the intent of remedying the previously found deficiencies. This prototype is evaluated by applying the *Think-Aloud* method in order to uncover additional deficiencies. Then the system is implemented according to the final developed design. Following I consider if the set requirements are actually realized.

To begin with the developed application manages to meet all the requirements which are defined in chapter four. These requirements are implemented as follows:

Requirement 1 This requirement is implemented by a process which exists of four different views. The first view is the *EventDescriptionView* for setting up the title, description and recurrence if required. Appropriate dates are selected in the *EventDatesView* and times and locations are added via the *EventTimesLocationsView*. In the last view (*EventCategoriesView*) the user is able to restrict possible participants to selected categories.

Requirement 2 In order to check the availability of participants the *ManageSchedulesView* allows the user to invite participants in form of an invitation window which is opened when the respective event from the calendar is clicked. An invitee is represented in the invitation window with it's user name. A blue colored name means the response is still pending, a green colored name shows that the invitee accepted the proposed date and a red colored name displays that the invitee rejected the invitation. Participants can be simply replaced by clicking on the name of the invitee which is supposed to be replaced.

Requirement 3 The responses to the invitations which are visible via the *ManageSchedulesView* allow the creator of the event to comprehend which dates are most suitable.

Requirement 4 As a result of the first iteration of the designing phase the functionality for importing contacts is realized in a separated view (*SynchronizeContactsView*). Equally the functionally for assigning categories to contacts via drag and drop is provided as a separated view (*AssignCategoriesView*).

Requirement 5 This requirement is fulfilled as a part of the event creation process by the *EventCategoriesView* class where categories and required number of participants from this category are set.

Requirement 6 An overview of both, created events and invited events is provided by the *SchedulerMainView* which displays all the events in a calendar.

Requirement 7 An overview of contacts is given by the *ManageContactsView*. Moreover this view provides the functionality to add or delete contacts and modify information from existing contacts.

Requirement 8 A list of existing categories is shown in the *ManageCategoriesView*. This views provides the functionality for adding, deleting and modifying categories.

Requirement 9 This requirement is met due to all software used is open source and therefore can be used free of charge.

Requirement 10 As both the frontend and backend are developed in Java, which is one of the most popular programming language[28], technical limitations are kept low. Moreover to secure a clean code base for an easier access to the source code the *MVP Design Pattern* is applied to the application.

6.2 Limitations

The software which is developed in the context of this thesis has certain limitations. Although it was observed in the requirement analysis that a mobile application would be important for accessing and checking schedules along the way, it was not possible to implement the application for another platform due to the limited time.

Furthermore the process of creating appointments with participants covering various categories is only possible via all the users related to the appointment manually exchanging requests and responses. The application does not support preemptively scheduling, so that participants are be added automatically to appointments in regard to being needed or not. It was consciously decided to avoid tackling a problem for linear programming from the start of working on the thesis.

Chapter 7

Conclusion

This chapter serves as a conclusion of the work by evaluating the results which are described in the previous chapter.

By separating the event creation process into four steps a good usability is provided. Simply because this way the user is able to understand each view on its own without being supposed to look at other concerns, subsequently reducing the complexity. Providing the functionality of inviting participants and checking on their response in a single view is functional as this allows to easily replace the respective participants who rejected the invitation. A disadvantage of the current solution is that only the creator of the event is able to comprehend which date is most suitable for an appointment provided that several dates are proposed. The invitees only get informed when a proposed date is set as an event which will take place. Providing specific views for synchronizing contacts and assigning categories further decrease complexity and supports the understanding of the software. Due to using open source software, a popular programming language and providing a clean code base by applying the *MVP* pattern the software allows to be easily extended in future work.

Chapter 8

References

[1] N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta, *Comparison of JSON and XML data interchange formats: a case study*, 2009, Caine, Volume 2009, Pages 157–162

[2] Y. Bai, *Practical Database Programming with Java*, 2011, John Wiley & Sons, Page 555

[3] E. Shakshuki, H. Koo, D. Benoit, D. Silver, *A distributed multi-agent meeting scheduler*, 2007, Journal of Computer and System Sciences, Volume 74, Pages 279–296

[4] J. Zou, R. Meir, D. Parkes, *Approval Voting Behavior in Doodle Polls*, 2014

[5] A.T. Ernst, H. Jiang, M. Krishnamoorthy, D. Sier, *Staff scheduling and rostering: A review of applications, methods and models*, 2004, European Journal of Operational Research, Volume 153, Issue 1, Pages 3–27

[6] T. Feddern-Bekcan, *Google Calendar*, 2008, Journal of the Medical Library Association. 96.4, Page 394

[7] Bundesministerium für Bildung und Forschung, *Verbesserte Krisenbewältigung im urbanen Raum*, 2013, Forschung für die zivile Sicherheit Bekanntmachung: „Urbane Sicherheit“ - Brochure, [accessed July 13, 2017], Available goo.gl/dydarV

[8] B. Kellermann, *Dudle: mehrseitig sichere Web 2.0-Terminabstimmung*, 2011, Dissertation at the Technical University Dresden

[9] T. Fischer, D. Postert, *User Centered Design*, 2014, Web Style Guide, Volume 9

[10] J. Nielsen, *10 usability heuristics for user interface design*, 1995, Nielsen Norman Group 1.1

[11] Deutsches Institut für Normung, *Ergonomie der Mensch-System-Interaktion: Teil 210: Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme (ISO 9241-210:2010) : Deutsche Fassung EN ISO 9241-210:2010 : part 2010: Human-centred design for interactive systems (ISO 9241-210:2010)*, 2010, DIN Deutsches Institut für Normung

[12] S. Pai, Y. Sharma, S. Kumar, R. M Pai, S. Singh, *Formal Verification of OAuth 2.0 using Alloy Framework*, 2011, International Conference on Communication Systems and Network Technologies

[13] J. Nielsen, *Usability inspection methods*, 1994, Conference companion on Human factors in computing systems. ACM

[14] B. Kellermann, *Datenschutzfreundliche Terminplanung*, 2010, European Community's Seventh Framework Programm (FP7/2007–2013)

[15] B. Shneiderman, *Designing the user interface: Strategies for effective human-computer interaction. Reading*, 1988, MA: Addison Wesley Longman

[16] C. Braz, J.-M. Robert, *Security and usability: the case of the user authentication methods*, 2006, Proceedings of the 18th Conference on l'Interaction Homme-Machine. ACM

[17] C. Abras, D. Maloney-Krichmar, J. Preece, *User-Centered Design*, 2004, Encyclopedia of Human-Computer Interaction, Thousand Oaks: Sage Publications 37.4, Pages 445-456

[18] A. Charland, B. Leroux, *Mobile application development: web vs. native*, 2011, Communications of the ACM 54.5, Pages 49-53

[19] J. Nielsen, *Iterative User Interface Design*, 1993, IEEE Computer Vol. 26, No. 11, Pages 32-41

[20] N. Bevan, *International Standards for Usability Should Be More Widely Used*, 2009, Journal of usability studies, Vol. 4, Pages 106-113

[21] M. Maguire, *Methods to support human-centred design*, 2001, Int. J. Human-Computer Studies, Vol. 55, Pages 587-634

[22] D. Dzvonyar, S. Krusche, L. Alperowitz, *Real Projects with Informal Models*, 2014, EduSymp@ MoDELS. 2014, Pages 39-45.

[23] J. Nielsen, *Breadcrumb Navigation Increasingly Useful*, 2007, [accessed August 21, 2017], Available https://www.nngroup.com/articles/breadcrumb-navigation-useful/.

[24] K. Poslek, *UX Quick Tip: The proper way of handling notifications*, 2017, [accessed August 30, 2017], Available https://infinum.co/the-capsized-eight/ux-quick-tip-the-proper-way-of-handling-notifications

[25] Vaadin Documentation, *Advanced Application Architectures*, 2017, [accessed September 01, 2017], Available https://vaadin.com/docs/-/part/framework/advanced/advanced-architecture.html

[26] M. Rizwan Jameel Qureshi, *A Comparison of Model View Controller and Model View Presenter*, 2013, Science International, Vol. 25

[27] Microsoft Developer Network, *The Model-View-Presenter (MVP) Pattern*, 2017, [accessed August 29, 2017], Available https://msdn.microsoft.com/en-us/library/ff649571.aspx

[28] K. Iyer, *20 Best Programming Languages In 2017*, 2017, [accessed September 02, 2017], Available https://www.techworm.net/2017/03/top-20-popular-programming-languages-2017.html

[29] O. Damm, *Vaadin + CDI = MVP*, 2014, JavaSPEKTRUM 03/14, Pages 46-49

[30] Vaadin Documentation, *Building the UI*, 2017, [accessed August 26, 2017], Available https://vaadin.com/docs/-/part/framework/application/application-architecture.html

[31] R. Johnson, J. Hoeller, K. Donal, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B. Hale, A. Colyer, J. Lewis, C. Leau, M. Fisher, S. Brannen, R. Laddad, A. Poutsma, C. Beams, T. Abedrabbo, A. Clement, D. Syer, O. Gierke, R. Stoyanchev, P. Webb, *The spring framework–reference documentation*, 2004, Interface, Vol. 21

[32] Spring Boot Documentation, *Spring Boot*, 2017, [accessed August 29, 2017], Available https://projects.spring.io/spring-boot/

[33] J. Nielsen, *10 Usability Heuristics for User Interface Design*, 1995, [accessed September 01, 2017], Available https://www.nngroup.com/articles/ten-usability-heuristics/

[34] K. Pernice, *UX Prototypes: Low Fidelity vs. High Fidelity*, 2016, [accessed September 05, 2017], Available https://www.nngroup.com/articles/ux-prototype-hi-lo-fidelity/

All links were last followed on 13/09/2017

Appendices

Appendix A

Prototypes of 1st Iteration

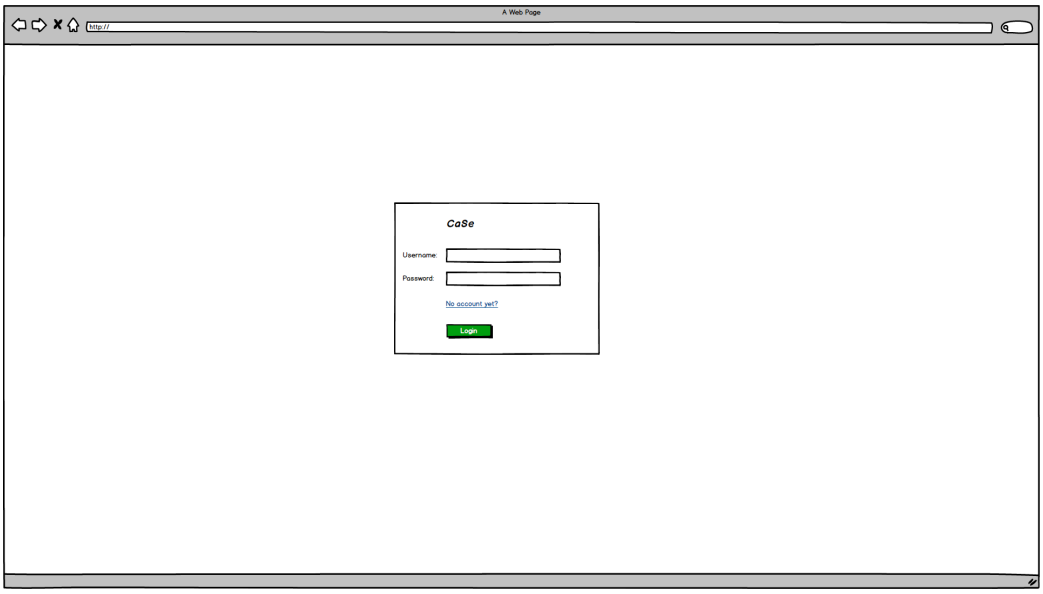

Figure A.1: Login View of the Web Service

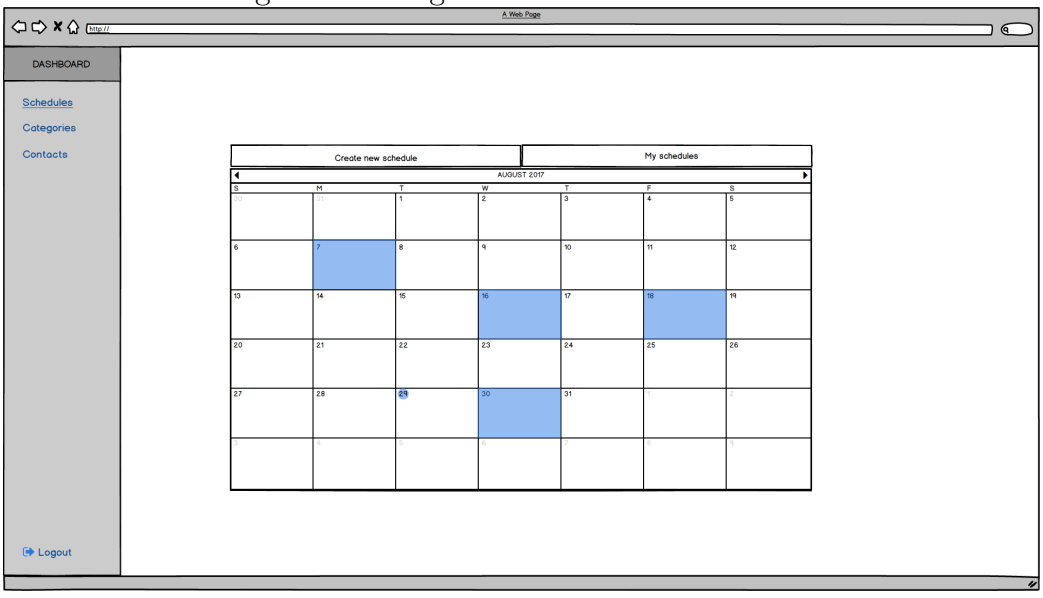

Figure A.2: Initial View after Successful Login

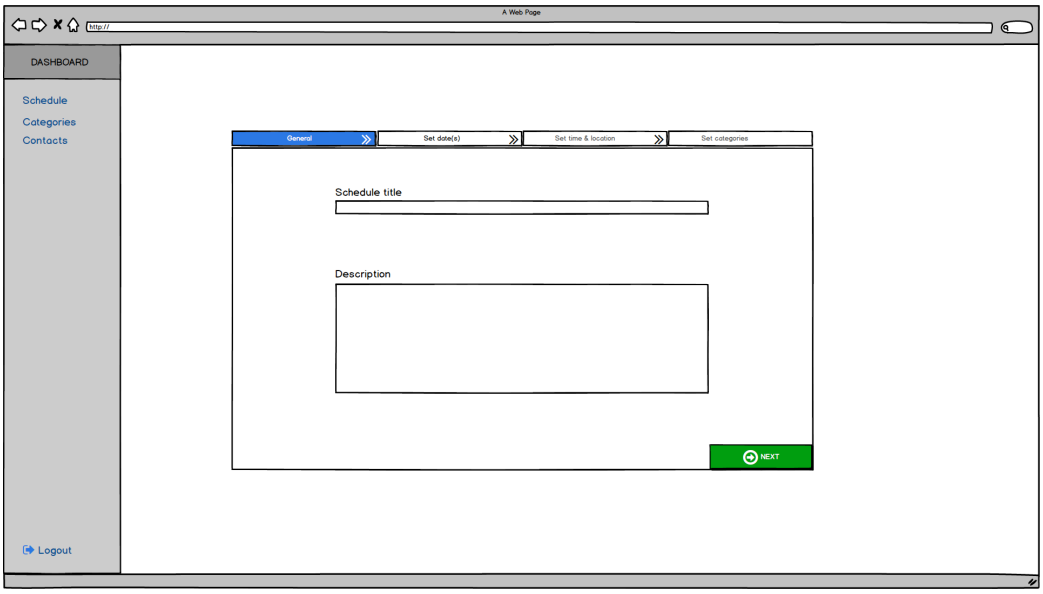

Figure A.3: Setting up Description and Title of a New Event

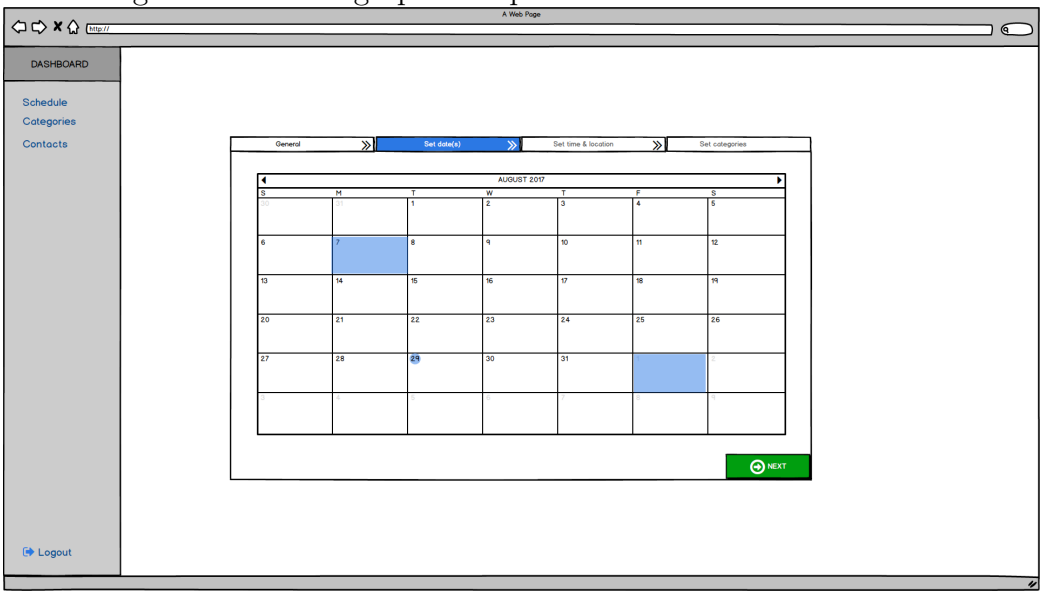

Figure A.4: View for Selecting Dates for the Event

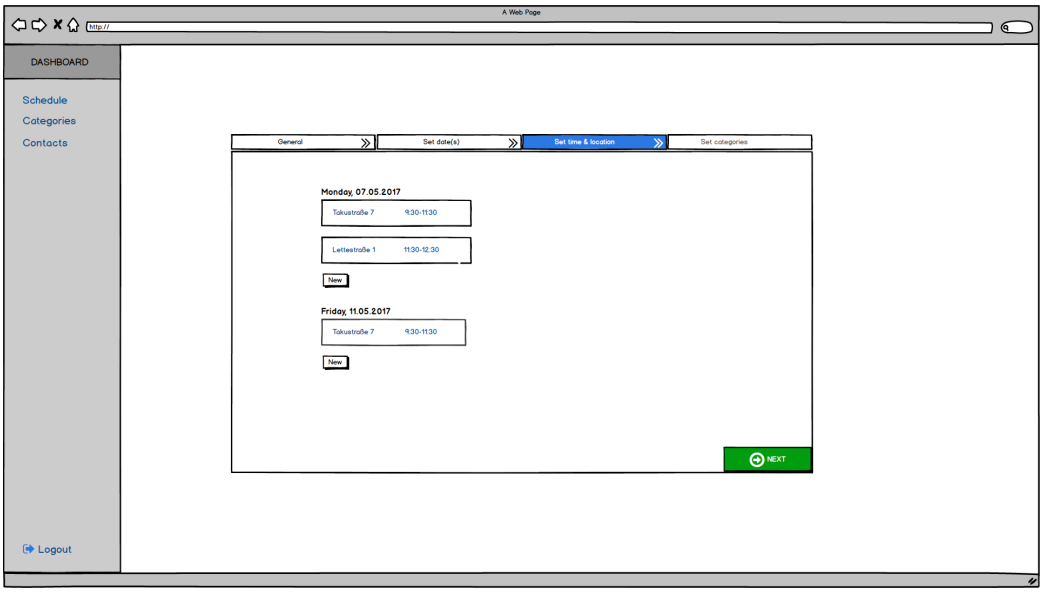

Figure A.5: View for Adding Time and Location to the Selected Dates

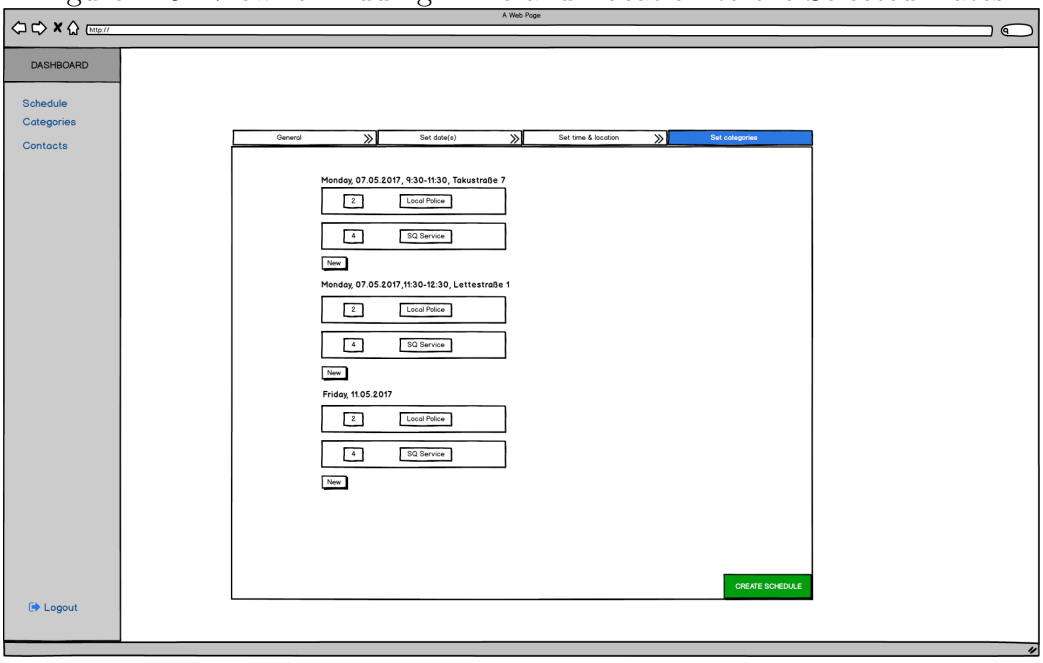

Figure A.6: View for Determining Categories and Required Numbers of Participants

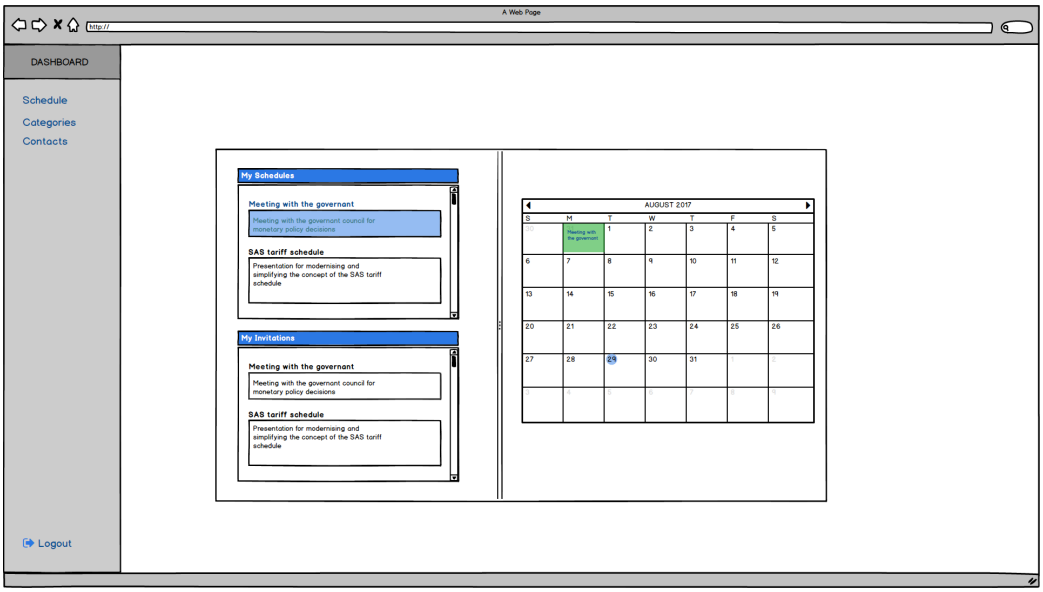

Figure A.7: View for Managing Created Events and Reacting to Invitations

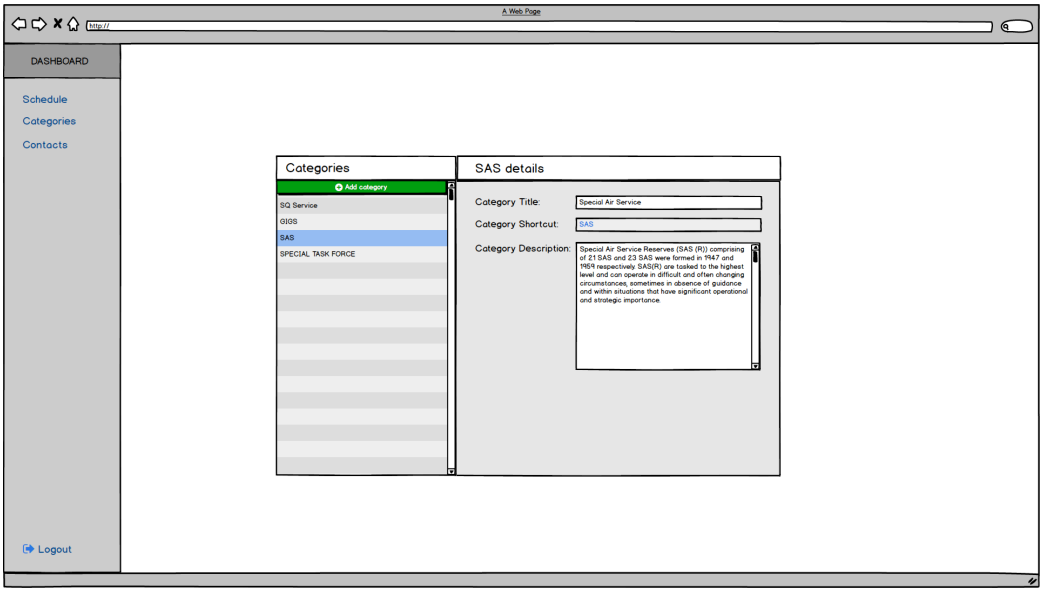

Figure A.8: View for Managing Categories

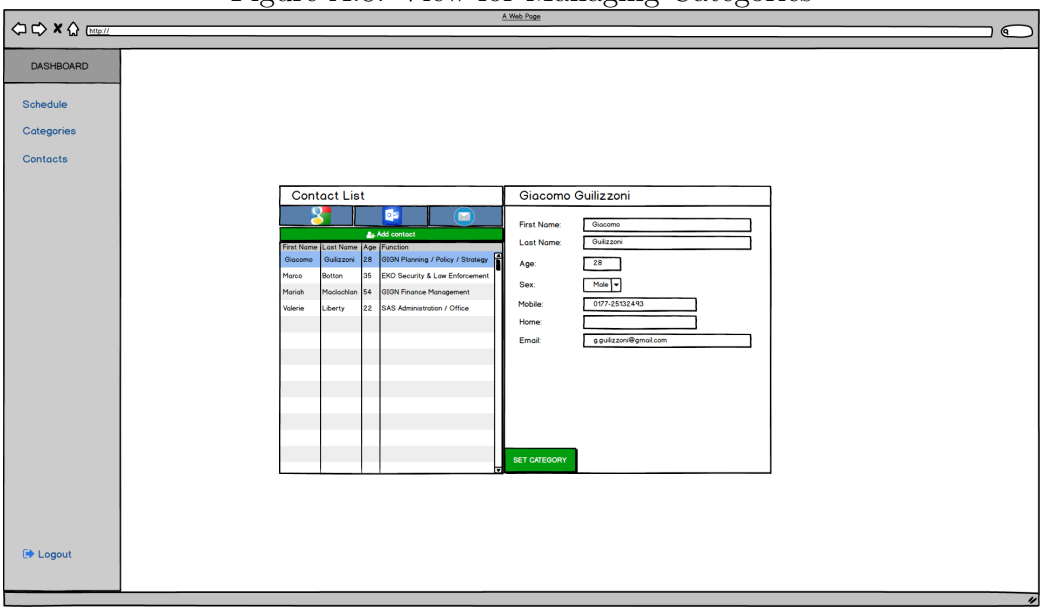

Figure A.9: View for Managing Contacts

Appendix B

Prototypes of 2nd Iteration

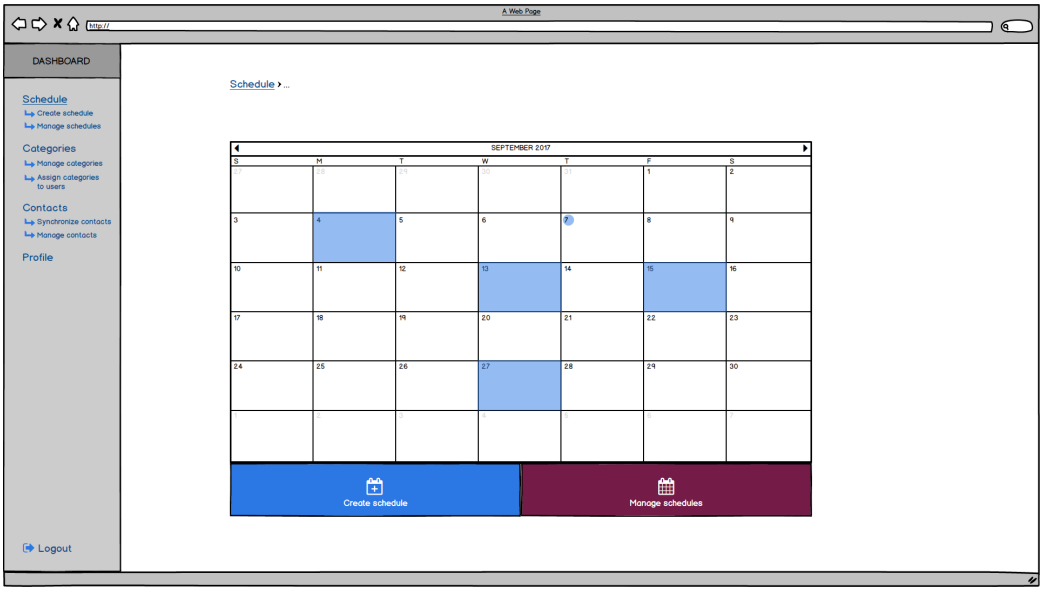

Figure B.1: Initial View after Successful Login

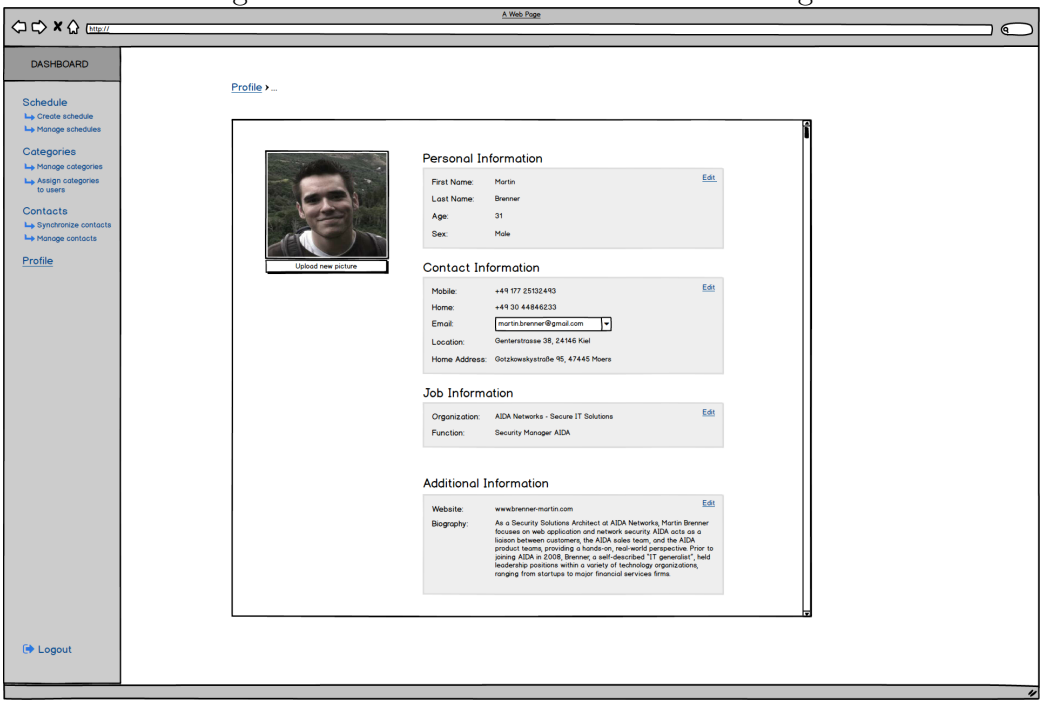

Figure B.2: Editable User Profile

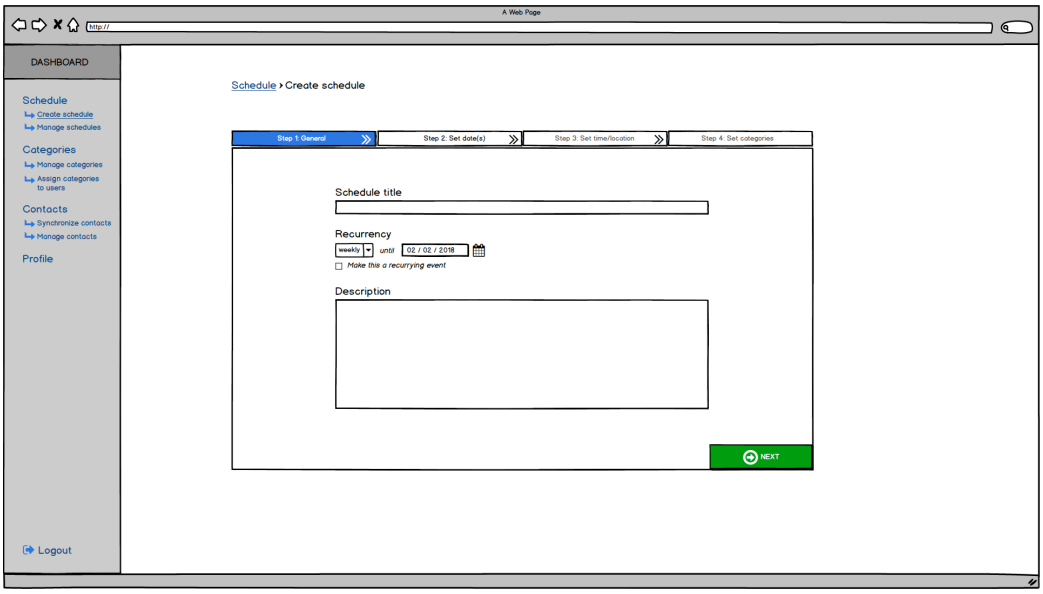

Figure B.3: View for Setting the Event Details

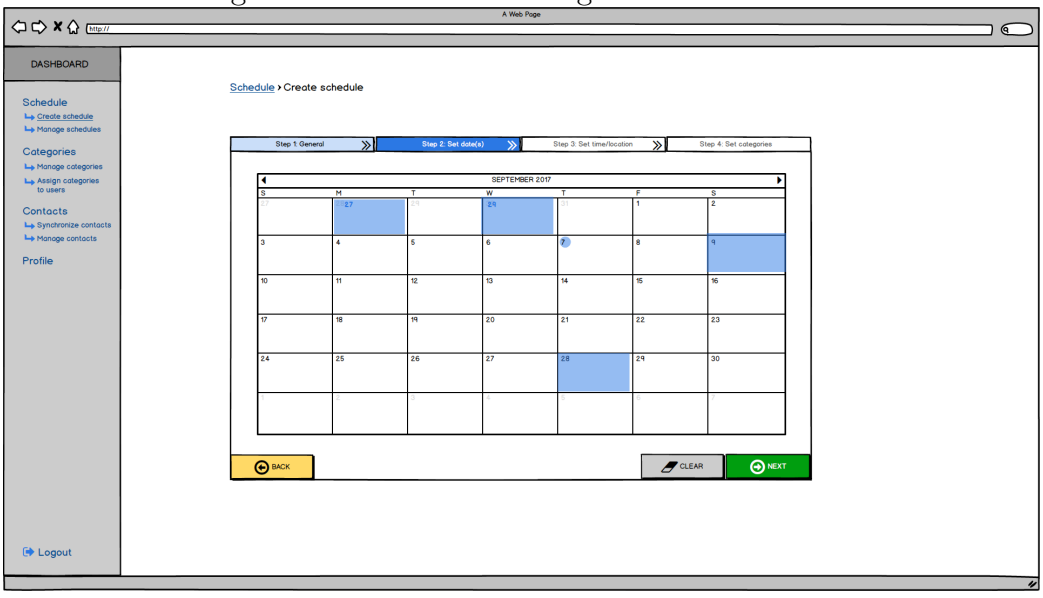

Figure B.4: View for Selecting Dates

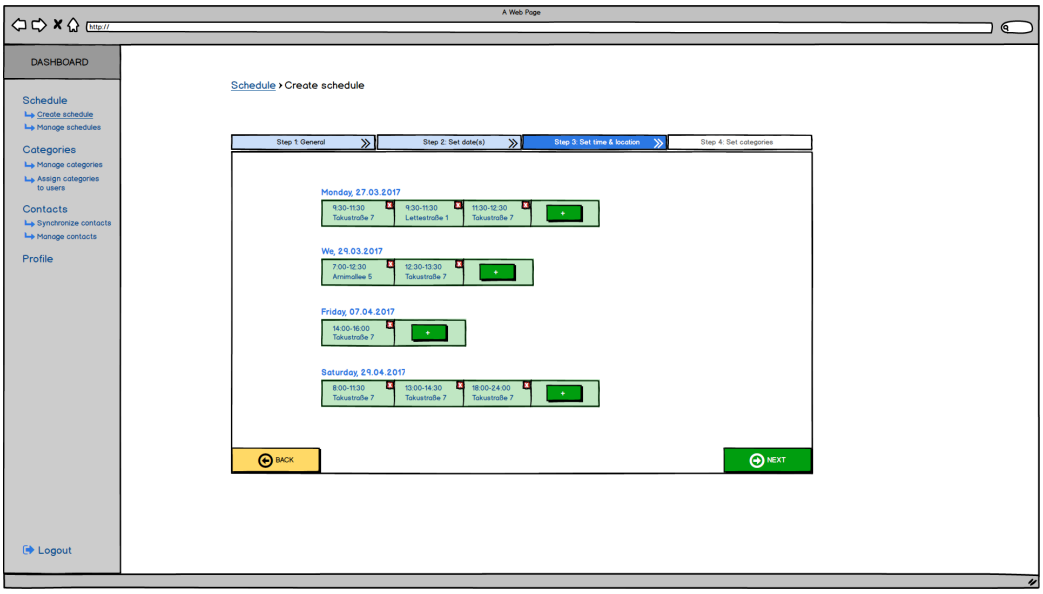

Figure B.5: View for Setting Times and Locations

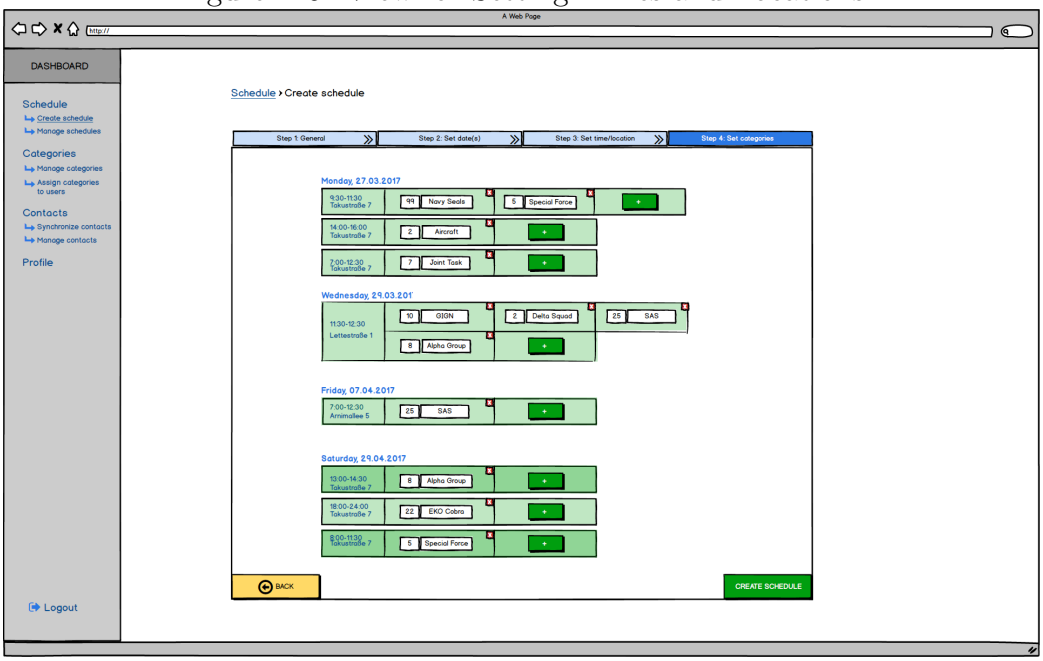

Figure B.6: View for Setting Categories and Number of Participants

Figure B.7: Modal Window for Adding a Time and a Location

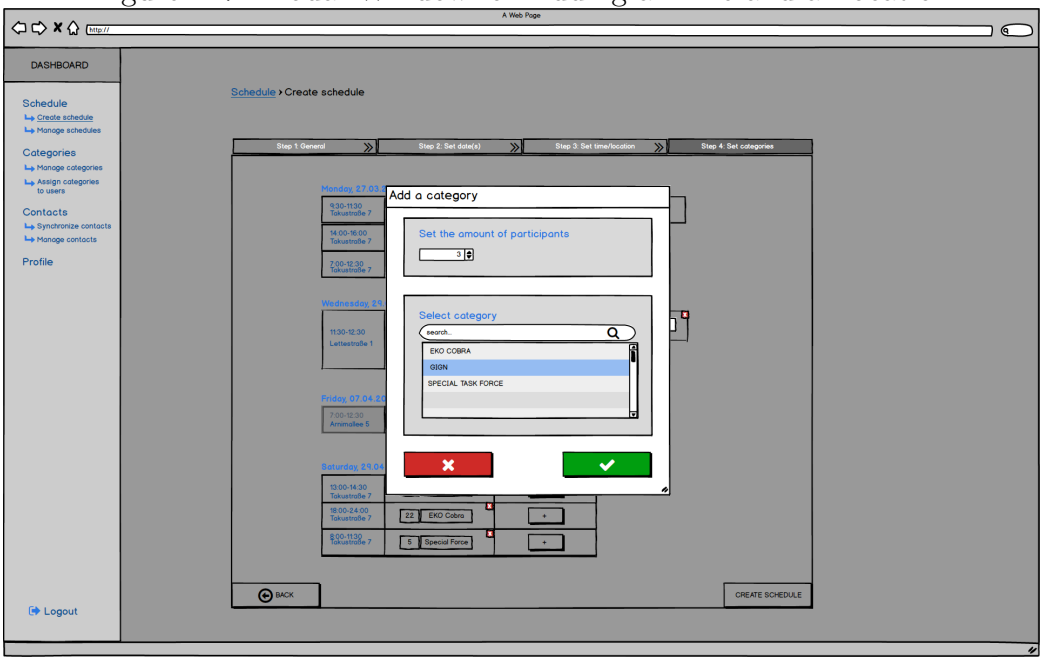

Figure B.8: Modal Window for Adding a Number of Participants and the Required Category

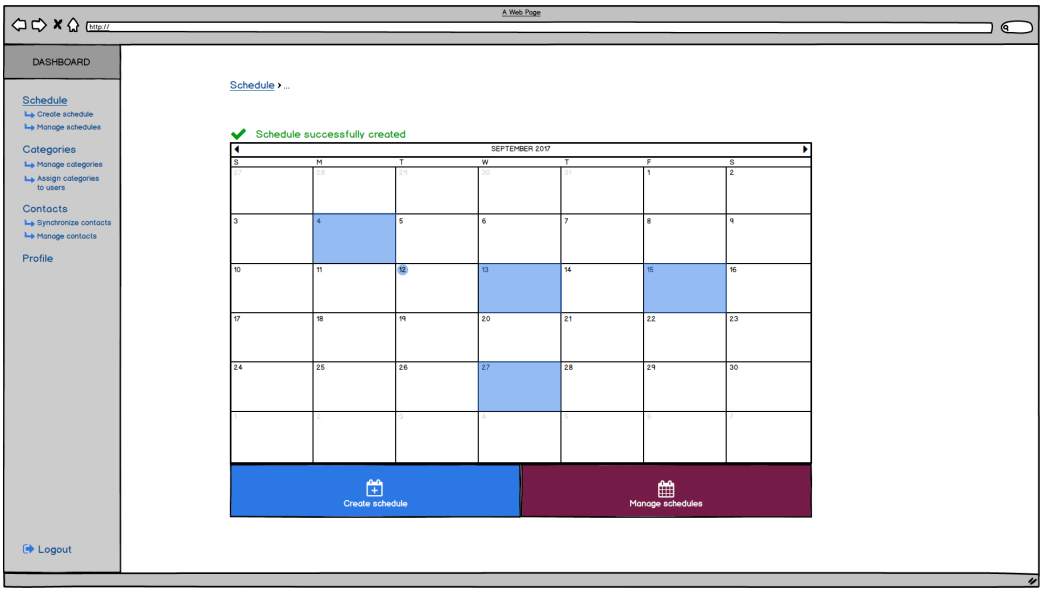

Figure B.9: Success Feedback after Creating an Event

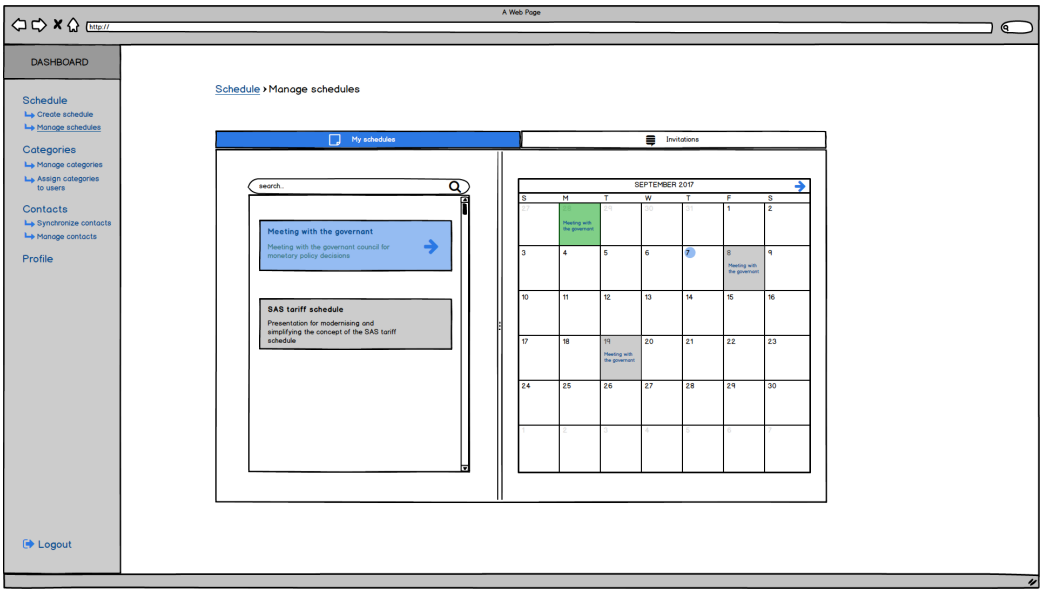

Figure B.10: View for Managing Created Schedules

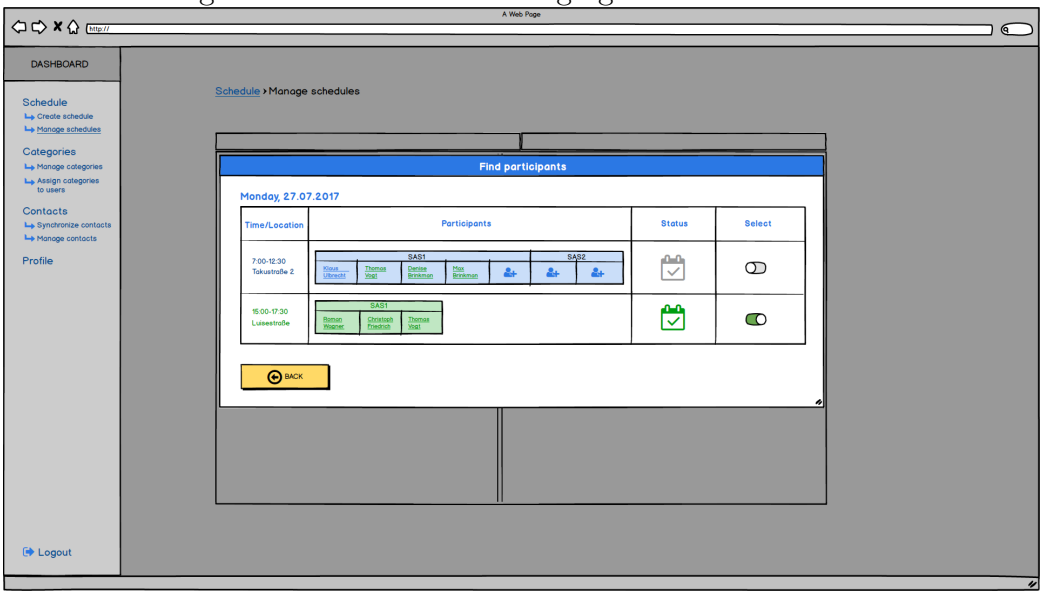

Figure B.11: Modal Window for Finding Participants for a Created Event

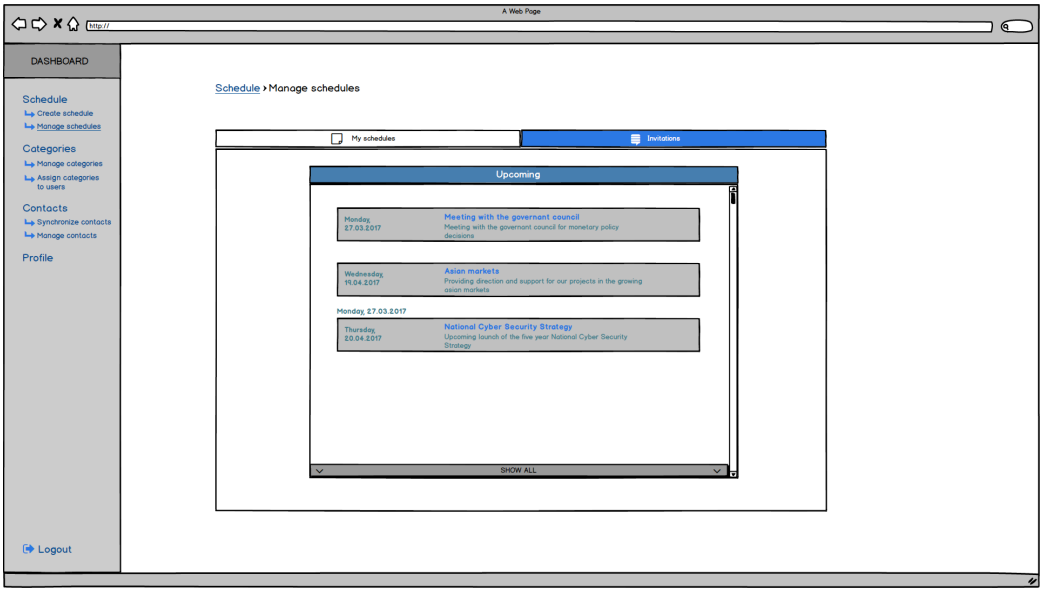

Figure B.12: View Showing Invitations from Other Users

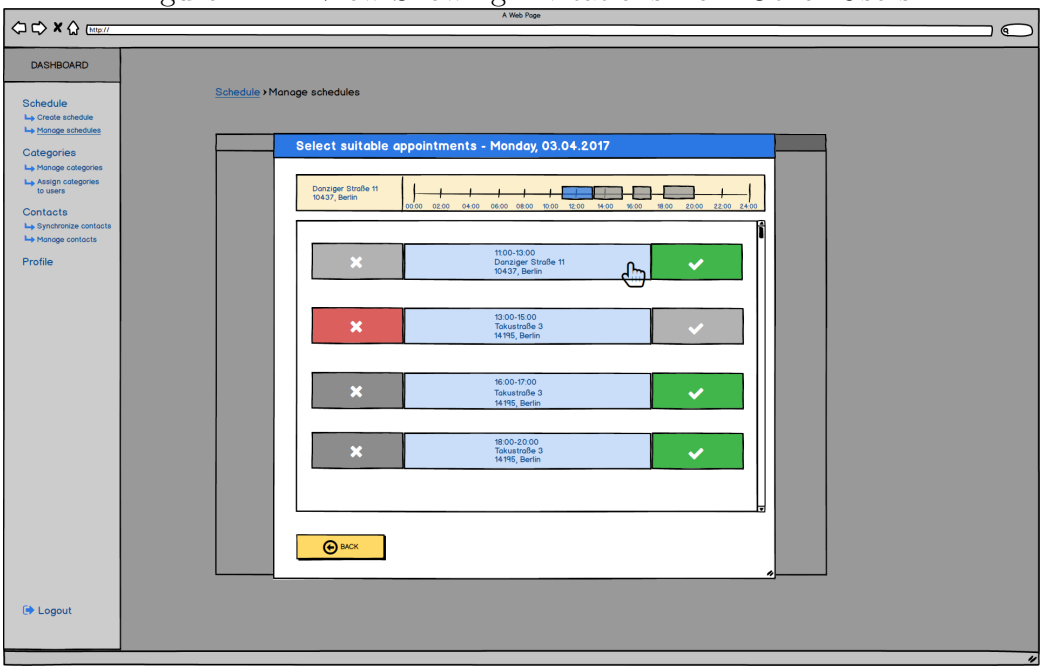

Figure B.13: Modal Window for Accepting or Rejecting Invitations

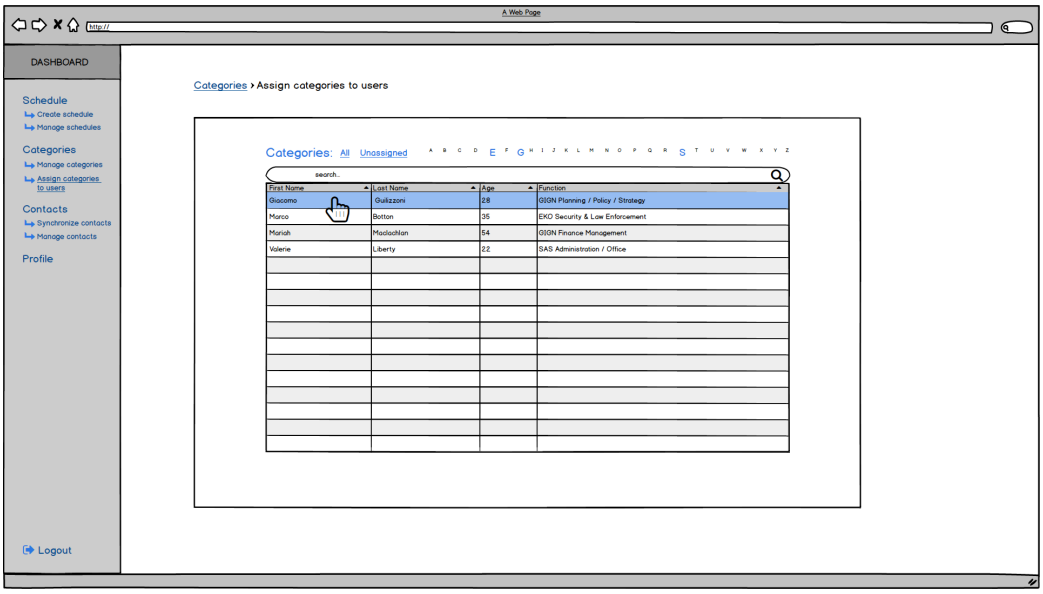

Figure B.14: View for Assigning Categories

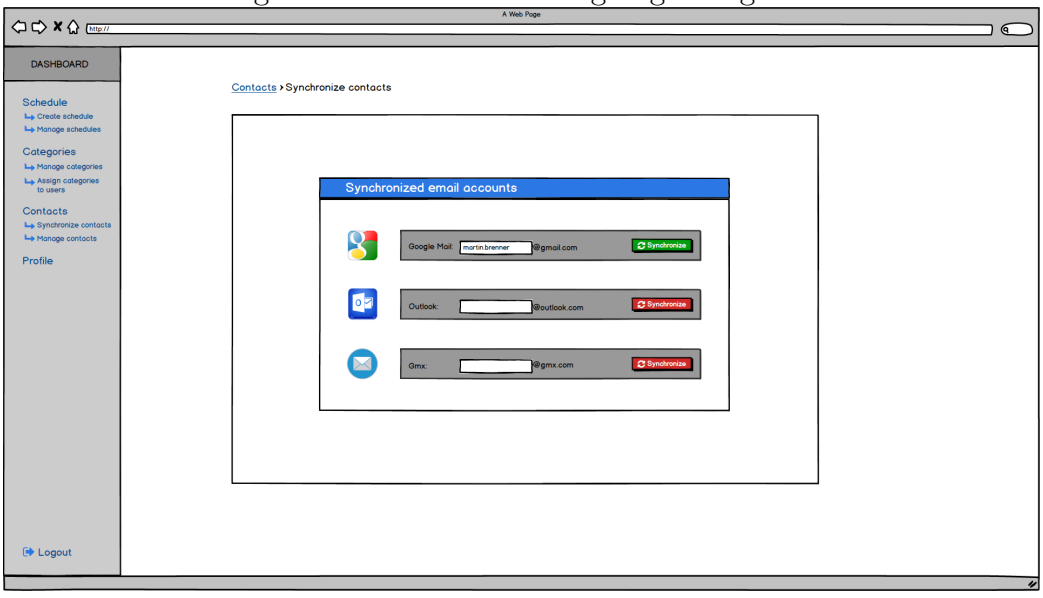

Figure B.15: View for the Import of Contacts

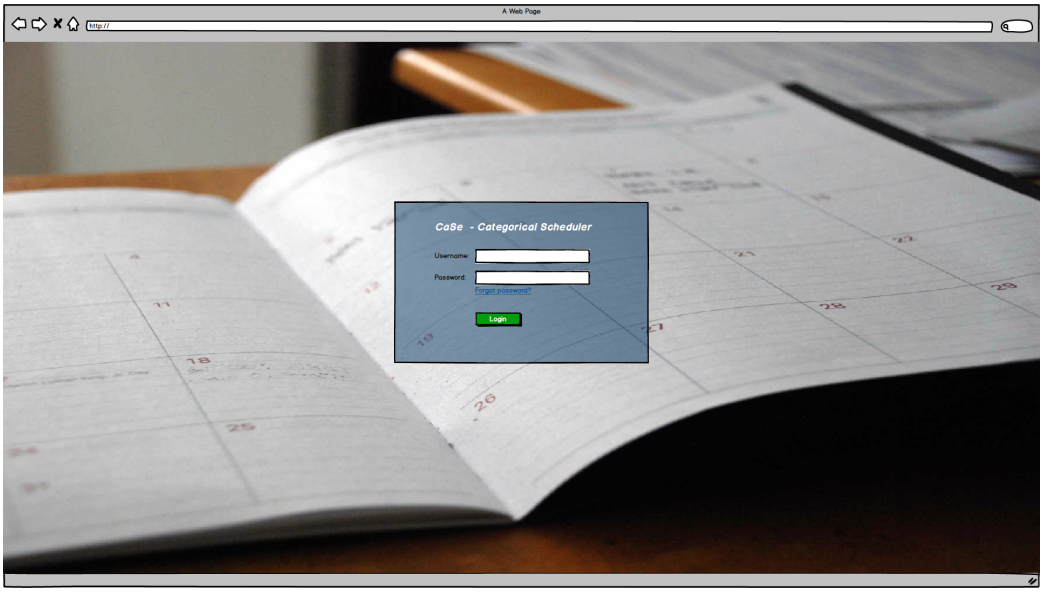

Figure B.16: Login View of the Web Service

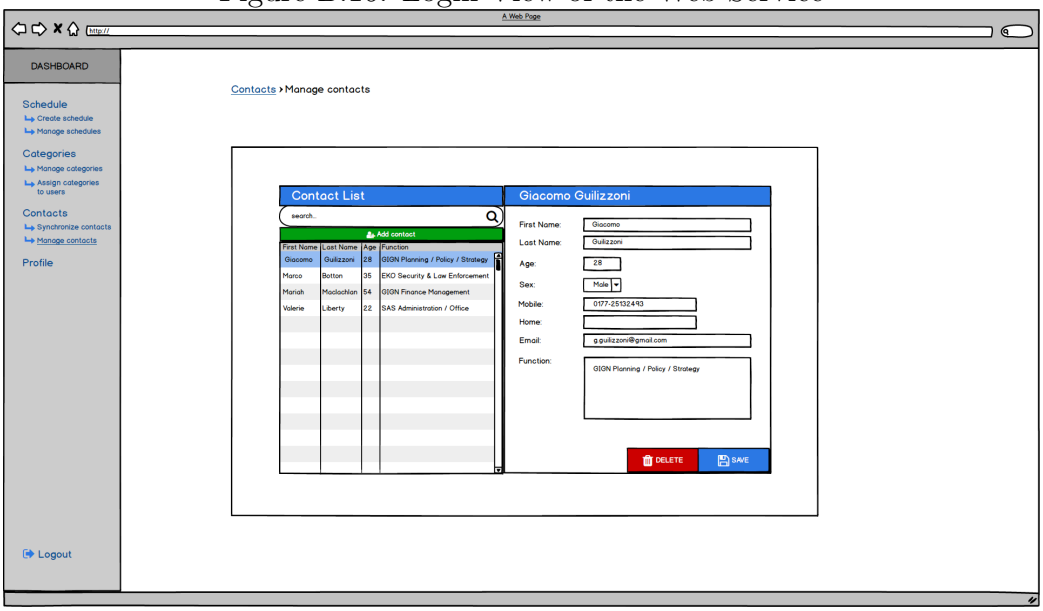

Figure B.17: View for Managing Contacts