Mobile Device Usage in Interactive, Co-located Presentations

Iris M. Schaffer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2016

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 26, 2016

Iris M. Schaffer

Contents

Chapter 1

Introduction

1.1 Introduction

1.2 Goals

The main aim of the project and thesis is to explore different ways of incorporating mobile devices into presentations in business-settings efficiently and productively. This includes polls created by the speaker (both beforehand and on-the-fly), as covered by other studies, as well as continuous, spontaneous feedback as described in [41]. Additionally other types of annotations should be supported, namely textual comments, images, links and youtube-videos, which will be rendered accordingly on new presentation slides. I believe this approach has the potential of transforming presentations into an collaborative effort in which all meeting-participants are empowered to shape the progress and outcome of presentations. From a technological point of view, like most existing approaches [4, 9, 14, 19, 41, 42], a web application will be developed to make use of modern web technologies' quick prototyping capabilities and the web's cross-platform and cross-device nature. As *WebSockets* have successfully been deployed in the real-time features of other presentation tools [19, 42] (in [19] a difference in response time between 10 and 600 simultaneous users of under 150ms and a package loss of approximately 0% was reported), this technology will be used to communicate between speaker and audience. Like in [42], an existing web presentation library will be used to be able to concentrate on the collaborative features rather than building a cross-device presentation platform. Instead of *impress.js*, *reveal.js*[1] will be used for this purpose, as it already offers a few key features, namely focus on mobile devices, embedded videos, speaker-notes and a possibility to follow or control presentation from ones personal device.

[1] http://lab.hakim.se/reveal-js/

Chapter 2

Related Work

Mobile phones, tablets and laptops have become our every day companions. We take them with us wherever we go, may it be the classroom or meetings, lately they have even made an appearance in courtrooms [16]. Especially during presentations, mobile device usage is still perceived as rude and can be a source of distraction [2, 3, 25] although other studies indicate that lecture-relevant phone use in classrooms can actually be beneficial [24] for information-recall.

Instead of banning modern technologies, incorporating mobile devices into presentation workflows has proven to foster collaboration and connection between attendees in meetings [3] and holds the potential of promoting participation and helping introverts overcome the hurdle of speaking out loud [4]. The growing computing power as well as the ubiquitousness of mobile phones, tablets and laptops make them suitable candidates for giving instant feedback to speakers as well as voting and sharing relevant multimedia content on-the-fly. Resulting presentations provide more flexibility, a better understanding of the listeners' opinion and the possibility to close the gap between presenter and audience.

The idea of using electronic devices to foster group interaction in meetings and presentations is not new. Stefik et al. [40] already experimented with the use of personal computers in meeting rooms as early as 1987 and Myers et al. [32] developed a collaboration tool which could be used to annotate PowerPoint slides from PDAs in 1998. Since then, digital whiteboards, telepresence systems, productive multi-user web applications and other computer-aided collaboration tools have become a common sight and we choose to carry smart devices around wherever we go. Surprisingly little research, however, has covered the use of these mobile devices in the context of presentations. Most of these studies were conducted in the educational sector and usually aim at quizzing students, which is why an own sub section is dedicated to classroom related approaches. While most relevant research has concentrated on one aspect, such as real-time polling [19] or remote-

Figure 2.1: *i>clicker* devices, used in [7]. Image source [47].

controlling [8], no system known to us has combined as many interactive mechanisms in one application as ours and allowed seemless integration between them, which sets the present approach apart.

2.1 Classroom related

As growing class-sizes have caused student participation to sink drastically [4], researchers have tried to deploy mechanisms to make lectures more interactive and engaging. The first approaches in this field of student-response-systems (SRS) utilised so-called clickers (see figure 2.1) – remote-control-like devices, connected to a receiver station via radio frequency technology [51] which can be used for tasks like taking attendance and voting [7]. Using these clicker systems has shown to “yield a strong and positive relationship with student learning” [7]. However, the limitations of clickers – the need for proprietary hardware and the limited interface consisting only of a few buttons – lead researchers to experiment with personal mobile devices as input instead. In 2007 Lindquist et al. [29] presented a system integrated with the University of Washington’s Classroom Presenter software, which lets students submit answers to assignments and in-class quizzes via SMS and MMS or using their laptops. Although the mobile phone users struggled with the input of longer messages, they perceived the ubiquity and concenience of using a light-weight personal device as an advantage. Most students, however, were worried about the costs of using SMS or MMS as a requirement in class – a concern modern devices with internet access and cheap data plans dispel. The first of these web-based approaches were explored around the same time. Esponda [14] for example describes a system in which iPods and other wifi-enabled devices can be used to answer questions during class. What is interesting about her approach is not only the technology used, but also that questions do not have to be prepared in advance, but can also be created on-the-fly, using a pen-based tablet, resulting in more lively and spontaneous student-teacher-interaction. The creators behind *i>clicker*[1], the clicker system used in [7], have also recognised the shortcommings of their hardware-approach and now build mobile apps for students' personal devices. Like[14], their application makes it possible for lecturers to prepare quizzes beforehands or create polls on-the-fly to monitoring the students' knowledge, understanding and progress. Although also available as iOS and Android app, like most modern approaches, the i>clicker software also has a web version, making use of modern browsers' possibilities and the device-independence of the web as a platform. The tool *ASQ* [42] lets lecturers create HTML5 presentations with *impress.js*[2] which are then distributed to listeners via a link. Students follow the presentations on their mobile devices, and can submit questions connected to the current slide to the speaker. Quizzes (both open questions and multiple-choice) can be embedded in the slides by the teacher. These quizzes can either be graded automatically (for coding assignments and multiple-choice questions), corrected by teaching assistants or by the students in self or peer-assessment. While this project has put a lot of effort into the server-side and administration of slidesets, the present work concentrates more on the client-side and does not provide slide management tools. In contrast to our implementation, however, this approach lacks Another interesting approach is presented by Cheng et al. [9], who propose a system which generates HTML presentations from *Microsoft PowerPoint* slides and lets viewers add their own content (either additional material or questions) as vertical sub-slides. This way a tree-like structure is created in which teachers and students collaborate in interactive presentations. This architecture also inspired the sub-slide based presentation space deployed in this software.

Another popular application, with richer audience-speaker-interaction and an emphasis on listener-listener-interaction is *Backstage*[3]. As digital backchannels like Twitter can foster the sense of community within the audience, but are usually hard to follow for presenters, Bry et al.[4] developed a backchannel specifically for large classrooms. Students can post messages publicly and send private messages to their colleagues. These public posts can be up or down-voted, as well as marked as unrelated. Together with an ageing-algorithm, this community feedback is used to estimate a post's relevance. Important feedback is then presented to the lecturer, to allow him or her to get a better sense for the audiences' opinion and understanding. Additionally, small quizzes and polls serve as performance feedback to the teacher and students. Though one of the most mature systems studied for this thesis, having been developed specifically for classrooms, the use of the software in other scenarios is not ideal. Moreover, most of the features concentrate on listener-listener-interaction, while the present thesis focuses on

[1] http://www1.iclicker.com/
[2] http://impress.github.io/impress.js
[3] http://backstage.pms.ifi.lmu.de/

mechanisms strengthening the speaker-audience-interaction.

Like Backstage, most of these approaches sound promising but are tightly bound to an educational context. The project discussed in this thesis, however aims for a broader field of application and concentrates on business-settings.

2.2 Office environments

In contrast to classroom-related software, meeting-environments usually have an significantly lower amount of participants, as well as a smaller gap between the speaker and the audience. Another difference lies in the polling, surveying and quizzing functionality most of the presented projects offer: while these usually have only one correct answer in educational settings, to grade students [4, 29, 42], the goal in meeting environments is to make decisions and collect ideas, without judgment and often anonymously. The systems we want to quickly introduce all have a focus on mobile devices and their usage in meetings and office-related presentations and curiously were all developed by Microsoft Research: In [3], as well as examining the perception of smartphone use in meetings, the mobile application *Meetster* is presented. The study finds that although people primarily use their phones for meeting or work-related tasks, they tend to think their colleagues use theirs for private purposes. Unlike the present thesis, in which mobile devices should be used in the context of presentations, *Meetster* was developed to help getting to know other meeting attendees in a playful way. This changed the perception of using one's smartphone during the meeting and was described as "fostering social interactions". While the findings of the study conducted as part of this publication legitimise our approach, this thesis presents a more practical approach, more relevant to the presentation itself, instead of just connecting meeting attendees through a game.

A system concentrating more on presentations directly is *Crowd Feedback* [41], a piece of software which allows listeners to give a speaker continuous, real-time feedback, using their personal devices. A responsive web application with a like and dislike button controls the feedback-system. The participants' reactions are shown with a red (dislike) or green (like) dot for each attendee in a sidebar next to the presentation slides (see figure 2.3). An evaluation of the system showed that the participants felt more engaged with the presentation and connected to other listeners. Many users stated only having the possibility to like or dislike did not reflect enough options and that a button related to the speech pace might have helped. It was also noted that the sidebar was perceived as disturbing and made it harder to pay close attention to the presentation. This study and its conclusions have inspired the implementation of an instant feedback mechanism for listeners in the present work, however, instead of only having the binary like and

(a) (b)

Figure 2.2: *Crowd Feedback* [41] used during a presentation (a): The bar next to the slides shows one dot per participant in the meeting, the dots can be controlled with the app (b). The feedback dots fade out over time. Image source [41].

dislike, the reactions are based on emojis, allowing for more insightful and faceted feedback.

The third study concerns itself with the navigation through slides: *Office Social* [8], a PowerPoint plugin with companion smartphone app, allows presenters and listeners to navigate through PowerPoint slides using their mobile phones. Members of the audience can either browse the slides privately, or take over the control of the presented slides, allowing them to effictively steer the presentation or discussion. As in the present approach, Chattopadhyay et al.'s software allows members of the audience to review the slides privately, making it possible for latecomers to catch up and to generally estimate the length and direction of the talk [8]. However, as their interface focuses on the navigation between slides, the preview of the slides is fairly small. Our approach tries to focus on the content of the slide and instead of offering big buttons to navigate around, makes use of intuitive swipe gestures, which can potentially be used eye-free more easily [33]. Another disadvantage is the overhead of having to download a smartphone application before the start of a presentation, as well as the limitation of the application only being available for Windows Phones.

2.3 General presentations

Since lectures and meetings both are very specific forms of presentations, a few paragraphs should also be dedicated to general approaches in this third section. One publication, which concentrates on polls and their real-time evaluation and rendering is [19]: Inoue et al. present a system which distributes *Microsoft PowerPoint* presentations using modern web-technologies while making it possible to alter and update the slides in presentation mode. This way questionnaires can be answered and their results displayed

Figure 2.3: *Office Social* [8]'s smartphone app interface. A preview of the slide is shown on top, followed by big buttons for navigating between slides. In the left picture, the application is in *review* mode, where a local copy of the slides can be navigated through. By pressing a button in the interface the *interaction* mode is activated, allowing listeners to navigate through the master slides (right). Image source [8].

in real-time. Additionally, members of the audience can add annotations (both handwritten and digital) to slides. Although this approach seems very promising, pictures, videos and other types of media are ignored entirely. Moreover the interface seems too complicated for small devices and is therefore only usable on laptops and maybe tablets.

Two more products, though not subject to scientfic research and more commercial than the approaches presented so far, are *Mentimeter*[4] and *sli.do*[5]. Both tools are web applications with real-time polling support, usable in any presentation. Both systems work very similarly: listeners go to the respective website and enter a presentation code to then be connected to the live voting. A handy feature Mentimeter offers is to query the device's location to determine the right presentation. Sli.do on the other hand also supports questions from the audience, which can be up-voted by the listeners, making it easy for speakers and participants in podium-discussions to answer the most relevant questions. Moreover, additionally to multiple-choice polls, sli.do also supports open questions and ratings. While the creators of Mentimeter provide a PowerPoint plugin, sli.do is not directly linked to any presentations. However, the popular canvas-based presentation-tool prezi[6], offers seamless integration with the application. It is worth noting that prezi itself already offers mobile features out of the box: Presentations can be controlled remotely from the speaker's phone or tablet as well as be viewed and followed by members of the audience in real time, using a mobile application.

[4] http://www.mentimeter.com/
[5] http://sli.do
[6] http://prezi.com

More web-based presentation tools include *Google Slides*[7] and *PowerPoint Online*[8]. While PowerPoint Online seems to only offer a simplyfied version of the desktop application online, Google Slides also provides mobile features such as editing and authoring slides on phones or tablets and controlling them remotely.

To conclude this chapter, a few words should also be said about the JavaScript presentation library *reveal.js*[9] and its accompanying visual edior *slides*[10]. Reveal.js offers features such as remote controlling slides for the speaker and following presentations on personal devices for members of the audience. However, the installation to achieve the latter so-called *multiplexing* functionality, is fairly complex and involves setting up a socket-io server, running the master-presentation statically and locally and uploading a client version of the presentation to a publicly accessible server. Reveal.js offers a reliable online presentation library and could have served as a starting-point for the project presented in this thesis. However, due to their closed environment, tightly coupled code and lacking support for extensibility, we decided to instead implement an own presentation library (see chapter 5, section 5.3.4).

[7] http://www.google.com/slides/about/
[8] http://office.live.com/start/PowerPoint.aspx
[9] http://lab.hakim.se/reveal-js/
[10] http://slides.com/

Chapter 3

Interactive Mechanisms

In a first step of identifying possible mechanisms which could make presentations more engaging and interactive, we analysed different types of presentations. There are several factors which determine these types, such as the size of the audience, the environment around and purpose of the presentation as well as the speaker and audience. In the following these factors will be described shortly to then present and discuss mechanisms these could profit from most.

3.1 Factors

3.1.1 Audience Size

One aspect which plays an important role in the type of presentation and thereby the interactive mechanisms applicable is the size of the audience. Different challenges present themselves depending on the amount of listeners: While there might be a debate between speaker and audience in small group sizes, it is hard for audience members to directly communicate with a speaker during conferences or in large lecture halls. Shy or introvert attendees might remain unheard [4] and only a usually randomly chosen subset of people get the opportunity to ask audience questions after talks in conferences. At the same time estimating the audience's knowledge and interest as well as the general mood gets increasingly difficult both for the presenter and attendee as the number of participants rises. Additionally to the interaction between speaker and audience, another important factor is listener-listener interaction [31]. Group-dynamics largly depend on the audience size and smaller groups usually perform better than big ones [35]. The general conclusion therefore is that big audiences struggle to connect and interact with the speaker and each other and interactive tools must aim to strengthen the bidirectional bond between presenter and listeners. In smaller groups, on the other hand, the focus should be put on supporting the already existing dialogue and exchange between all participants of the presentation. As peer-pressure might rise in smaller groups and the better listeners know each other, ways of anonymously contributing to the outcome or flow of a presentation become more important.

3.1.2 Presentation Environment

The environment of a presentation is described by all factors surrounding the presentation. One of them is the setting a talk is given in, in other words, if it is embedded in a meeting, a talk at a conference or a lecture at school or university. Other aspects worth considering are whether the audience is co-located or distributed and which technologies are available. As this work concerns itself only with mobile devices in the context of co-located presentations, difficulties added through remote presentations as well as missing technical equipment will be disregarded in this section. Instead, a closer look will be taken at the setting: In a lecture, it is desirable to measure the students' participation and engagement, as well as their understanding of a topic. In meetings, on the other hand, interactive mechanisms are more likely to aim for the promotion of collaboration between all participants. Conferences might search to foster the interactivity between attendees, to support networking. Instead of taking all possible scenarios into consideration, this work concentrates on business-related settings and explores mechanisms which foster collaboration.

Another part is the purpose of a presentation: McClain [30] identifies four major types: informational, motivational, pursuasive and sales. According to him, informational presentations search to educate the listeners, while motivational speeches try to inspire the audience to take action. Pursuasive talks usually present new ideas or directions and have the goal of making the listeners re-think old approaches and consider or even embrace new ones. Sales presentations, lastly, often use elements of the other three categories with the aim of "obtaining a decision at the presentation's end" [30]. While motivational, pursuasive and to some extend sales presentations often operate on an emotional level in the present moment, informational talks often include a way for listeners to re-visit the taught material through transcripts, lecture notes or handouts. Moreover, motivational, pursuasive and sales presentations focus on the goal of getting the audience to take action and therefore put more emphasise on the listeners than content-centric informational speeches. This creates two very distinctive needs for interactive mechanisms: on one side the ability for the audience to actively shape the path of the presentation, on the other hand the possibility to re-visit presentation slides (potentially including notes and additional material), after the end of a talk.

3.1.3 Speaker and Audience

The last factor taken into consideration in this chapter are the speaker and listeners themselves. Depending on the individual interest, but also character traits such as introversion, listeners will be more or less likely to engage in a presentation actively [4]. The inter-attendee relationship as well as the relationship between attendees and speaker also plays a role in which mechanisms are appreciated and which are not [31]: while it is common for listeners to jump into the role of the presenter in meetings with flat-hierarchies, the same behaviour is a rare sight in lectures or might even be deemed inappropriate or rude in more formal settings. When taking the speaker into consideration, the set of tools needed are more sophisticated than the ones necessary to only follow a presentation: Foremostly, speakers need a way of navigating through slide decks. It is desirable to have an overview of the entire presentation and while listeners only concentrate on the current slide, many speakers rely on notes or use timers, which also need to be placed in the interface. Moreover, the presenter's personal traits, experience and bluntly talent, play a central role in the successful deployment of interactive mechanisms: the flexibility, confidence and technological expertise of a presenter all determine how distracting or even stressful certain features are perceived as and whether a speaker is able to react to these spontaniously [43]. It is therefore crucial to give speakers the ability to turn said mechanisms on and off. An important challenge which also arises with this question is how to design these mechanisms in a way that is neither perceived as intrusive nor interrupting (this will be discussed in more detail in chapter 4). To summarise, when developing interaction tools, it is vital to take a participant's personality and their relationship to other ones into consideration. In the context of presentations, shy listeners should be given tools to make them heard; presenters need full control of the mechanisms provided.

3.2 Resulting Mechanisms

With these aspects and challenges in mind, a multitude of different mechanisms can be derived. Although many more are thinkable, this section concentrates on the ones implemented in course of the thesis project. However, we will try to point out other possible features and provide resources to projects focusing on these. One point to keep in mind is that not all of the presented mechanisms work equally well in every environment but instead have scenarios they are best suited for and others in which they are practically rendered redundant. The ideal settings and key advantages of each of these mechanisms are summarised in table 3.1.

3.2.1 Remote Control

One mechanism of special importance for speakers is the ability to control slides and navigate through them. As many presentations involve more than just one speaker and can profit from sharing control over slides with others [8], any amount of presenters should be able to be connected at any given point. Controlling should be possible from any personal device, may it be a laptop, tablet or mobile phone, giving the speakers maximal freedom. Suitable interaction metaphors therefore have to be found for different devices.

3.2.2 Following Slides

For members of the audience, one important feature is to be able to individually and independently navigate through and follow slides. This makes it possible to re-visiting slides after a presentation, accommodate individual learning paces [9] and even give late-comers a chance to catch up with the presentation [8]. Displaying and navigating through slides should again be possible on any personal device and focus on the slide content in a way that maintains the readibility of all text. The mechanism can be designed in many different ways and could even allow listeners to remote control the presentation [8], our implementation however only provides individual slide navigation on the personal device. Additionally, the progress of the presentation should always be synchronised with the individual devices, allowing listeners to truely follow along. This basic mechanism can be extended to offer features such as turning the synchronisation on and off (effectively allowing to navigate freely and jump back to the presenter's state) or to only allow listeners to see the last slide the presenter has already shown.

3.2.3 Paths

Also connected to navigation and following slides is the possibility to offer different paths through the presentation. Especially in informational talks these can account for different backgrounds and levels of knowledge in the audience, they however, also make it possible to get listeners more involved in shaping the presentation. Paths should both be accessible to each audience member individually (for further reference or to catch up on a topic), as well as on the projector (e.g. by polling, as discussed in the next subsection). The possibility to flexibly navigate through a presentation has proven to be one of the biggest advantages of canvas-based presentations [28] and has a wide field of application. The scenarios this thesis concentrates on are the following: On one hand, the paths can cover different levels of details (e.g. *overview*, *regular* and *detailed*), as well as providing a way of skipping certain slides without having to navigate through all of them (e.g. skipping the introduction). Another option would be to let the audience decide between entirely different topics, depending on their personal interest. While canvas-based presentation tools like Prezi innately offer this flexibilty, slide-based tools often only make this behaviour possible by manually skipping over slides, which can interrupt the flow of the presentation [11]. PowerPoint extensions enabeling advanced forms of navigation, as well as the presenter looking through slides before projecting them are discussed in [11], [34] and [39], the latter, however, will not be part of our implementation.

3.2.4 Audience Questions

A feature well-suited for informative talks, is the possibility for members of the audience to ask questions. Another scenarios are big crowds, in which it is hard to be heard as an individual. A tool specifically designed for such settings is sli.do, which was already introduced in chapter 2 section 2.3. More generally, such mechanism should enable members of the audience to submit questions for the presenter to answer. These questions should either be displayed directly, or collected for the presenter to go through at the end of the presentation, depending on their preference and flexibility. This mechanism also highly depends on the presentation environment: In a classroom, questions should be answered immediately, while conferences usually only allow them at the end of talks. Questions could moreover only be visible to the presenter, or every participant. Concentrating on business-related settings, we propose a question feature which allows audience members to submit questions at any point of the presentation. These should be accessible for every attendee, to spark others' interest and participation. From the presenter's point of view, questions should be displayable instantly, at the end of the talk or any time inbetween, leaving the decision when to react to questions to each individual speaker.

3.2.5 Polls

Another possibility to ask questions is polling. Although polls might also be generated by listeners, we propose a mechanism which lets the speaker create them. To give presenters more flexibility and because questions often only arise during talks [14], these surveys should be creatable in the preparation for a speech as well as on-the-fly, during presentations. This mechanism can help getting to know ones listeners (relationship between listeners and speaker), as well as estimate a crowd's mood (big audiences) and is especially useful in combination with paths. If supporting anonymous voting, relying on electronical aids instead of raising hands can also be benificial in smaller groups [14]. While a big number of different polling mechanisms are conceivable (open questions, ratings, multiple choice, as well as different ways of visualising the results), single choice voting and visualisation in a bar-chart serve as a starting point for our approach. Another detail lies in when the results are presented: they can either be rendered as soon as a user chooses his or her answer or only after everybody has given their votes. To summarise, the identified requirements for such mechanism are creation beforehands and during the presentation, real-time polling and data-visualisation as well as anonymity of the voting process.

3.2.6 Reactions

As described before, especially bigger crowds suffer from a lack of interaction possibilities between speaker and audience but also between members of the audience. While the latter is discussed in [4], the present work focuses on the interaction between speaker and listeners. Besides the difficulity of asking questions, which was already covered, the main problem for the presenter is to estimate the crowd's mood, which is why we suggest a mechanism that lets attendees send real-time feedback to the speaker. This functionality is based on [41]; as highlighted by Teevan et al., however, their simplistic approach of just offering *likes* and *dislikes* is not faceted enough to represent the full range of emotions listeners can feel during a presentation. It is therefore important to provide more detailed feedback. These reactions can either be displayed only to the speaker, or to the entire audience. The latter might distract listeners [41], however, also holds the potential to encourage others to also react to the current slide and strenghten listener-listener bonding. While this mechanism is expected to work well in bigger crowds, it will likely introduce an unnecessary technical burden to smaller groups, in which it is easier to estimate the attendees' mood.

3.2.7 Content Sharing

In contrast to live reactions and questions, content sharing is especially suited for smaller audiences. As discussed before, tools for smaller groups should strengthen the already possible dialogue between all participants. These scenarios make it possible for listeners to actively get involved in the presentation and not only shape the path through, but also the content of such. While adding subslides to a slide deck after a presentation [9] and text-based annotations [19, 32] during talks have already been discussed in previous work, to our knowledge, no other study has concerned itself with the possibility of adding listener-generated slides and multi-media content in live presentations. While being an exciting opportunity to explore a widely untouched research subject, this mechanism empowers listeners and transforms presentations entirely by combining classic slides with brainstroming-like interactions and related multi-media content. While the potential of this mechanism will be further discussed in chapter **??**, the requirements for this functionality should shortly be defined: It should be possible for any listener to add their own content to any slide. This content includes text, websites (per link), videos and uploaded images (e.g. taken with their personal

Table 3.1: Overview of resulting mechanisms, with their key advantages and optimal usage scenarios.

Mechanism	*Improvements*	*Ideal Scenario*
Remote Control	More flexibility for presenter(s)	Any, especially mutli-speaker presentations
Following Slides	Accounts for individual pace; can replace hand-outs	Any, especially informational presentations for later revision
Paths	Interactivity and flexibility	Any, especially informational
Audience Questions	Anonymity; possibility to be heard in big crowds	Big audiences; small groups for anonymity
Polls	Bond between speaker and audience by querying listeners' interest, mood and knowledge	Usage with paths; big audiences; small groups for anonymity
Reactions	Speaker-audience and listener-listener interaction	Big audiences
Content Sharing	Possibility to shape presentation for audience	Small groups

devices). Presenters should have a way of deciding whether to accept the contribution and if it should be added as a subslide or main slide. Moreover, this mechanism requires a lot of flexibility from the speaker, which is why it is important to allow them to turn off or silence the functionality, providing sensible fallbacks. While content sharing can transform a presentation into an interactive and collaborative effort in smaller groups, the functionality will likely lead to chaos in big groups without further interface changes.

Now that the implemented mechanisms are clarified and their requirements defined, the next chapter deals with the design and user experience of the application.

Chapter 4

Application Design

After defining the mechanisms which will be implemented, in a next step, the general application flow will be described, as well as offering insight into the user experience design of all parts of the application. What is important to note is that the design discussed here is just the default layout and can easily be changed and adapted by the presenter. All features identified in chapter 3 can be turned on or off, in the following it is assumed that all of them are enabled.

4.1 Application Flow

The flow and usage of the application is separated into two parts: the creation and authoring of the presentation and giving the presentation. As the technical details of how slide decks are composed are covered in chapter 5, this chapter focuses on the user interface and interaction design of the software from the speaker's and the audience's perspective, during the presentation.

The typical setup of an unveil presentation is as follows: We assume a presenter called Amy, who has already prepared her presentation and a listener called Greg who wants to follow the presentation from his smartphone. The slides are generally served from a server. This can either be a publicly accessible server or, if all participants are in the same network, locally from Amy's computer. We assume Amy is serving the slides from her laptop, which is connected to a projector. At the beginning of the presentation, Greg and all other listeners navigate their personal devices' browsers to the set up address (usually a combination of IP address and port). To make this step easier, Amy has put a QR code pointing to the address on the first slide and sent out an e-mail with the link to all participants before the start of the presentation.

The software supports three different modes out of the box: listener, speaker and projector mode. Depending on the mode, a certain set of features is activated, allowing Amy to have a different interface and more controls than Greg. Modes are activated via query parameters in the url: Amy navigates her laptop's browser to the url of the presentation and adds the query parameter `mode=projector`. On her smartphone, which she wants to use for remote controlling the presentation, the mode is set to `speaker`. If no query parameter is given, the application defaults to the listener mode, so Greg simply types in the address or follows the link in the url or QR code. Unveil generally offers a two-dimensional slide space, consisting of master slides (left to right) and subslides (top to bottom). Devices in speaker mode (in this example Amy's smartphone) can remote-control the presentation and navigate through said slides. All other devices (the laptop in projector mode and Greg's phone) are synchronised with the state of Amy's phone and automatically follow along in real-time.

4.2 General Interface

The general requirement for the interface of the application is to work in all three modes, on any device, from mobile phones to desktops and projectors. When in projector mode, only the content of the current slide, as well as listener reactions are shown (see figure 4.1). In listener mode, the interface is a lot richer and additionally features buttons for sharing media, links and asking questions, as well as six different reactions (see figure 4.2), which will be discussed in more detail in section 4.5. It also offers small arrow buttons, to navigate between slides. The speaker interface is the most intricate: Besides showing the current slide, we believe it should also include a preview of the upcoming slides in x (master slide) and y (subslide) direction, as well as speaker notes. Additionally to this interface, already familiar from PowerPoint or similar presentation software, buttons to mute incoming requests (media, link and questions) and to create new polls are provided (see figure 4.3). Since we expect presenters to switch between devices more often than listeners for more typing-intense tasks such as creating new polls, the mobile interface is as similar as possible to the desktop one and only re-arranges the displayed information to fit on smaller screens. The main difference between the mobile and desktop version of the listener interface is the design of the reactions: While desktop computers and tablets offer ample space for the placement of all six emoji, these are hidden behind a button in the mobile interface and only slide up upon a tap on said button.

4.3 General Interaction Principles

As far as the interaction design of the application is concerned, the main requirement technically is for all state changes to take immediate effect or in other words, for the software to work in real-time. This is true for interactions

Figure 4.1: Wireframe of slide in projector mode, as seen on a projector. No visual controls are shown, only the current slide and listener reactions are displayed. The presentation progresses through the presenter mode's remote controlling feature.

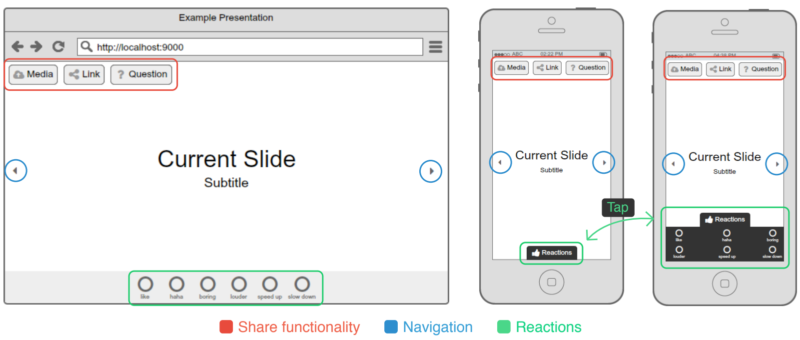


Figure 4.2: Wireframes of general interface in listener mode for mobile phones and desktops. Both offer buttons to share media, links and questions with the presenter, arrow buttons to navigate through the presentation and a possibility to react to the current slide. On mobile this feature is revealed with a tap on the *reaction* button, to not cluster the interface.

with the server as well as all internal state changes within the application. All transitions and animations last 200ms, a value which is both usable on mobile phones and desktops and, according to Google's Material Design Guide [18] "fast enough that it doesn't cause waiting, but slow enough that the transition can be understood". An easing curve with low outgoing and high incoming velocity is used.

The general aim for the interaction design of the application is to be as easy and intuitive to use as possible on any device, for both presenters and listeners. Especially the speaker's view has a lot of information to display and many ways of interacting with the interface. From the speaker's point of view, the main reason for negative presentation experiences stems from technical difficulties and problems [43]; we therefore decided to design

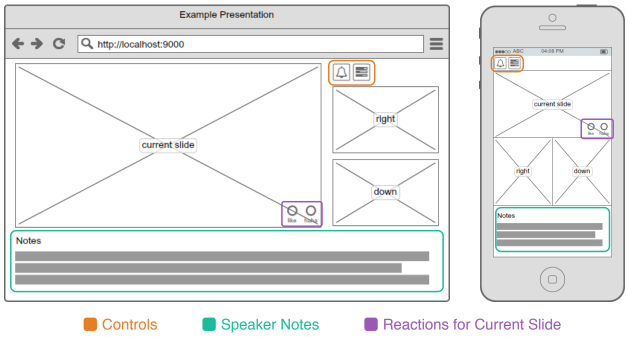


Figure 4.3: Wireframes of general interface in speaker mode for mobile phones and desktops. The interface consists of a preview of the current slide, the next main slide (*right*) and the next subslide (*down*), as well as showing presenter notes. It also offers buttons to toggle muting of incoming requests and creation of new polls.

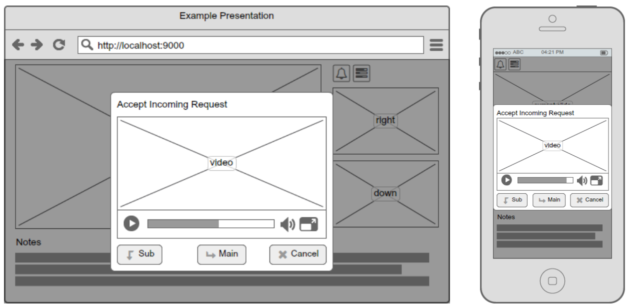


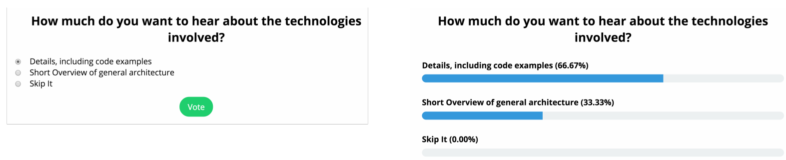
Figure 4.4: Wireframes of modal interface in speaker mode for mobile phones and desktops. The shown modal allows the presenter to add a listener-submitted video as a new main or subslide or dismiss the request. It pops up as soon as a listener wants to share content with the slide.

a presenter interface similar to the one already known from PowerPoint, Keynote, Google Slides or reveal.js (see figure 4.3) and employ familiar visual metaphors and interaction mechanisms such as buttons and modals (see 4.4).

Another important consideration when it comes to mobile and desktop environments is the question of supported inputs. While mouse and keys are a natural and intuitive way of navigating through desktop applications, swiping gestures are faster, more accurate [27] and require less time looking at the screen on mobile devices [33], making them the ideal candidate for the remote-controlling feature. For this reason, additionally to providing visual arrow-buttons for navigation, arrow-keys and swipe gestures are also supported. The interaction with buttons is controlled by mouse clicks or taps, respectively and common visual metaphors are used to symbolises

(a) (b) (c)

Figure 4.5: Button states, (a) normal, (b) hovered or active and (c) disabled.



(a) (b)

Figure 4.6: Details of polling interface for listeners before voting (a) and poll result after voting (b).

their state (pressed, hovered, disabled), as shown in figure 4.5.

Now that the general interaction principles are covered, a more detailed look is taken at the most interesting parts of the implemented features.

4.4 Polls

The first mechanism looked at in more detail is the polling feature. Polls can either be prepared before the start of the presentation or can dynamically be created during the presentation through the presenter interface. When Greg and the other listeners navigate through the presentation, they will already see the poll, they however cannot vote until Amy activates the voting by navigating to the slide with the poll. Polling again works in real-time: As soon as Greg chooses an option and presses the *Vote* button, the presenter, projector and all listener interfaces will be updated with his vote immediately. Every listener is only allowed to vote once per poll and the only currently supported poll type is single choice (see figure 4.6). As soon as Greg has voted, he is presented with the current results of the voting. These are displayed in horizontal bar charts (see figure 4.6), which are animated. This means, as soon as a new vote is registered, the bars will dynamically adjust their width as the percentages are updated. These results are available on the presenter and projector interfaces as well and also change in real-time. While the voting is enabled (by Amy navigating to the slide with the poll and until she navigates away from it again), all other navigation is frozen. This means, in this time, all listener navigation is disabled to ensure everybody is on the same slide and exercises their right to vote; this, however, is not mandatory and listeners can also choose not to vote.

To create a new poll, the presenter presses or clicks the *new poll* button

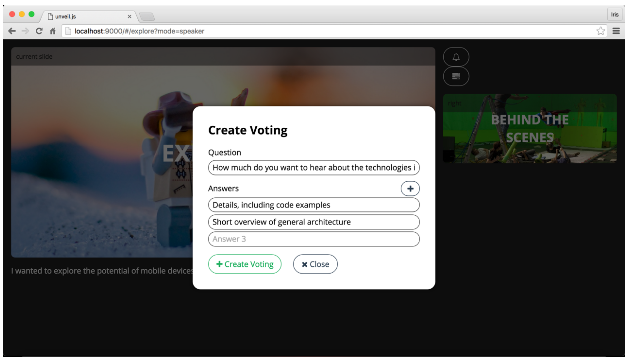


Figure 4.7: Poll creation modal in presenter mode on tablets, laptops and desktops.

in the controls section of the speaker interface, which opens a modal (see figure 4.7). In the beginning, a question and two answer fields are provided, more answers can be added using the + button. Once the question and answer options are entered, pressing the *Create Voting* button will generate a new poll and add it as the next main slide.

4.5 Reactions

Although binary digital reactions in presentations are not an entirely new idea [41], versatile feedback that goes beyond positive and negative, to our knowledge, has not yet been explored. The way this mechanism was implemented, our listener Greg can press a button to react to what the speaker Amy said or presented. The reactions are collected on a per-slide basis, allowing Amy to re-visit the slides afterwards, analysing which slides sparked most feedback. When Greg reacts to a slide, the feedback will immediately be displayed in the presenter view, as well as – if enabled – the projector interface.

Greg by default can choose from six pre-defined reactions: From the evaluation in [41] and [20] and from observing presentations and meetings, a pool of possible reactions has been narrowed down three emotions (approval, laughter, boredom) and three request types (louder, speed up, slow down) (see figure 4.8). The reason behind the missing disapproval on one hand is that test subjects in [41] felt less comfortable giving negative feedback, the button was generally used less than the positive one and it also included feedback such as *boredom* or *speed up* and *slow down*, which we have included own reactions for. On the other side, we hope this will encourage more elaborate feedback of disapproval using the content sharing functionality instead. This holds the potential of sparking a discussion instead of merely showing disagreement with the presented content.

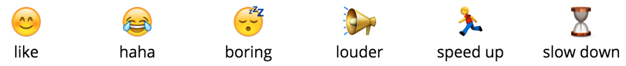
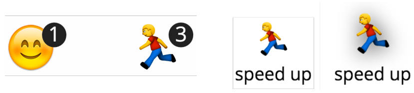


Figure 4.8: All six possible reactions and their emoji (from left): *approval*, *laughter*, *boredom*, *louder*, *speed up* and *slow down*.

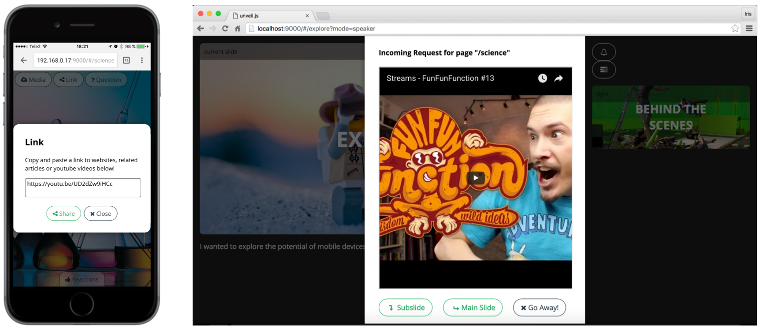


(a) (b)

Figure 4.9: Details of visualisation of reactions (a) with number of reactions as found in the presenter view and optionally the projector and (b) reaction in listener interface with (right) and without hover-state (left).

The biggest challenge with displaying non-binary reactions was to find an intuitive and familiar visualisation which would not take up too much space on smaller screens. Since the introduction of emoji on Apple's keyboard in 2011 and on Android's one in 2013, emoji have become a ubiquitous, language-independent means of communication [5, 12]. Instagram has found that almost half of its comments and captions nowadays include emoji [12] and with Google [38] and Bing [37] adding support for emoji-search and companies such as Facebook [23] and GitHub [22] offering emoji-based reaction systems, it is safe to assume the majority of regular internet users is familiar with the concept and meaning of emoji [6]. Although it would be possible to include a complete emoji-keyboard to allow for even more versatile feedback, we feel it is easier for less technology-oriented users to be provided only with a sub-set including a short description of each reaction (see figure 4.8). However, it is worth noting that this sub-set can easily be extended or overwritten by the presenter.

When it comes to displaying the reactions sent by the audience, as the presentation of feedback in [41] was perceived as a distraction from the presentation, the feedback mechanism proposed in the present work is either only shown to the presenter or displayed without any animations, in the lower-right corner of the projector interface, with only a small badge symbolising how many people have sent this reaction for the current slide (see figure 4.9 (a)). On the side of the listener, one detail worth mentioning is the styling of the hover-state. Especially on computers it is important to offer intuitive hover-states, to give the user a sense of what they are pointing at and if they can interact with the currently focused element [10]. The hover-state designed for this purpose includes a magnifying-effect as well as a light shadow around the emoji (see figure 4.9 (b)).



(a) (b)

Figure 4.10: Sharing of a youtube video. Request from listener on a phone (a) and accepting-modal in presenter-view on a computer (b).

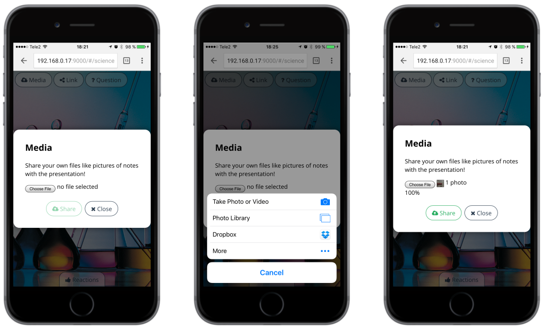
4.6 Content Sharing

The last big module developed for the present thesis is the content sharing functionality. At the time of writing, the software allows for sharing photos (via links or from the computer or phone), websites, youtube videos and free text (including questions). In an earlier iteration, only one *sharing* button was available in the interface, but feedback cycles with other web developers and designers have shown that it was not clear and intuitive enough which kind of content could be shared, which is why the functionality was now separated into three buttons: *Media*, *Link* and *Question*. Coming back to Greg and Amy, Amy has just talked about functional programming and Greg wants to share a related video, of his favourite youtube channel funfunfunction[1]. He opens the youtube app on his phone, searches for the video he wants to share and copies its link into the provided text field (see figure 4.10 (a)). When pressing the *Share* button, Amy will receive a pop-up in which she can review the request and then accept it as a new main or subslide or dismiss it (see figure 4.10 (b)). If she accepts, the video will be inserted in a new slide relative to the slide Greg sent the request from. Youtube videos and links to websites are embedded into the presentation, similarly to images and links to images, which are also directly included into the created slide. Text is simply displayed as a second-order heading.

What was important to be able to include personal notes, is the ability to take photos and upload them to the presentation. This is possible in the *Media* part of the content sharing feature (see figure 4.11) and makes use of the mobile operating system's native file uploads.

Since such mechanisms can be fairly stressful for the presenter [41, 43], we also built a feature to mute incoming requests. When muted, requests

[1] https://www.youtube.com/channel/UCO1cgjhGzsSYb1rsB4bFe4Q



(a) (b) (c)

Figure 4.11: Sharing of a picture from an iPhone's browser (Google Chrome on iOS 9.3.4) in listener mode. First, the listener opens the *Media* sharing feature (a). Then a file can be chosen from different sources, such as the camera roll, or a new picture can be taken (b). In a last step, the picture is uploaded to the application and can then be shared (c).

are automatically added as subslides to the slide the request was sent from, allowing all contributions to be collected without disturbing the flow of the presentation for the speaker [11].

Chapter 5

Implementation

This chapter dives into the technical implementation details, gives an overview of the used technologies and explains why these were chosen over others. Like many other projects in this area [4, 9, 14, 19, 41, 42], this project is realised using the web as a platform. This has many advantages, from modern web technologies' quick prototyping capabilities to the web's general cross-platform and cross-device nature, the project has benefitted from the dynamicity of the internet and the rapid evolvement of JavaScript over the past years. Although native sharing features of smartphones cannot be used due to the choice of platform, we believe, the merits that come with this decision outweigh the disadvantages for both users and developers. As no app has to be downloaded, it is easier to bring the audience to use the developed application [42]. The major advantage for developers on one hand is the ability to only focus on one platform instead of developing different applications for different operating systems, on the other hand the web is built for rapid prototyping as it is extremely easy and fast to roll out new updates without having to distribute them through the App Store or Play Store and without the need for users to manually update them. Since the JavaScript render layer this software was developed with also offers a library which can cross-compile JavaScript applications to different operating systems[1], the core code could potentially stay almost untouched, should the application be ported to other platforms in the future.

Finally, a few words should also be said about the distribution of this project. Without the vibrant open-source community, many of the frameworks and libraries used in this project would not exist. For this reason, and to give back to the community, all the libraries developed during this project have been published as open-source on GitHub[2] and are freely available for anyone to use. We concentrated on creating an extensible system for any developer to customiste, adapt, plug into and build interactive presen-

[1] https://facebook.github.io/react-native/
[2] https://github.com/irisSchaffer/

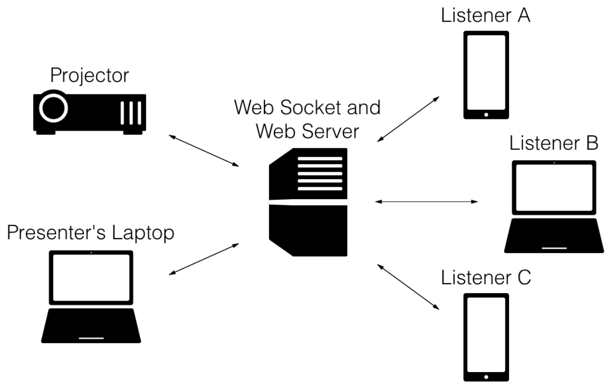


Figure 5.1: Example application setup. A server serves the presentation and connects all clients (presenter devices, listener devices and projector) through WebSockets.

tations with; the only requirement being basic HTML, CSS and JavaScript knowledge.

5.1 Project Scope

Before jumping into technical details, the scope of the project should be discussed. As the aim of the present work is to explore ways of incorporating mobile devices into presentation workflows, the goal of the project was to build the mechanisms described in the previous chapters on top of a JavaScript presentation library. As the focus was placed on the interaction possibilities between speaker and audience, the creation of the presentation (e.g. using a graphical user interface) or the management of slides and presentations were out of scope. As with most JavaScript presentation libraries, the resulting application is mainly aimed towards developers, both to extend the libraries further as well as to create presentations, as at least basic knowledge of HTML and CSS is necessary to build slides.

In total, a front end heavy system was created which features several ways of interacting with a presentation, both from mobile and desktop devices. Emphasise was put on mobile-optimised views and navigation possibilities. The end product consists of several, highly customisable libraries which can be combined to a presentation library which synchronises navigation state and state changes between listeners and speaker(s), includes three distinct interfaces for listeners, speakers and projects and offers the possibility to dynamically add content during the presentation. In the following, the general architecture and technologies used in the project will be analysed and described to then discuss implementation details, problems and solutions for the main features.

5.2 Server Architecture

In general, the application consists of two parts: the front end web application, run on every client, as well as a web server (see figure 5.1). As described in chapter 4, this server is either publicly-accessible or – if all listeners have access to the same network – run on any computer on the local network. While special attention was paid to the front end libraries developed, the server was kept as simple as possible, allowing any developer to work with their own servers and technology stacks. However, the limitation of having such a lightweight and *dumb* server, is that in the current iteration of the prototype all state changes are only persisted on the client-side. although good for testing purposes, as a reset is only a page-reload and clearing of local storage away, this means audience members joining the presentation after any additional slides were added, will not have the same state of the presentation.

The server's two only requirements are on one hand to serve the static web application to all clients, and on the other hand to connect all of these clients to enable the interaction and synchronisation between them using *WebSockets*. This technology was chosen as low response times for all network-requests were paramount and since the technology has already been successfully leveraged in the real-time features of other presentation tools [19, 42]. In our server implementation, the presentation is run from a *Node.js*[3] server, using *Express*[4] as a framework. WebSocket support is added using the popular library *socket.io*[5], which a few more words will be said on in section 5.6. To again emphasise how low the requirements for such a server are, a working example implementation can be found in program 5.1. Additionally to this, the server developed for this project also includes a `lastState`, which holds the last client state which is emitted whenever a new client joins. Once the connection between client and server is established, the server starts broadcasting all incoming requests from any client to all other ones (see figure 5.2). These requests are then processed on the clients locally, taking into consideration the mode they are currently in. Wherever possible, the clients optimistically update the interface instead of waiting for the response from the server, to make the application feel even faster.

5.3 Front End Technologies

The project generally follows modern best-practices in web development and utilises modern CSS3 and JavaScript features and frameworks. The software is written in ECMAScript2015, makes use of the *node package*

[3] https://nodejs.org/en/
[4] http://expressjs.com/
[5] http://socket.io/

Program 5.1: Very simple, possible implementation of a server running this project with Node.js and Express. [52] describes how wildcard support can be added to socket.io.

```
var express = require('express'); var app = express();
var server = require('http').createServer(app);
var io = require('socket.io')(server);

// directory 'client' will be served by server
app.use(express.static(__dirname + '/../client/'));

io.on('connection', function(socket) { // setting up socket io
  socket.on('*', function(event, data) {
    io.emit(event, data);
  });
});

server.listen(9000, function () {
  console.log('Unveil server listening on port 9000!');
});
```

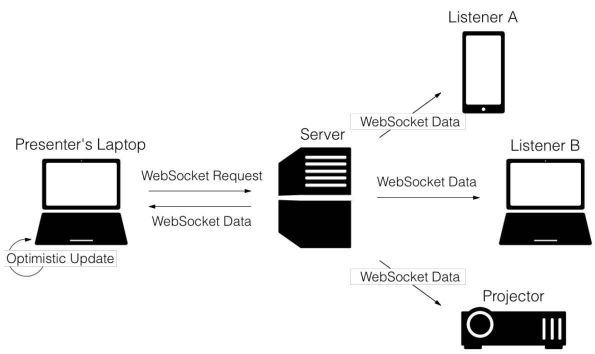


Figure 5.2: Client-server communication. The server receives the request from the client and forwards it to all clients.

manager[6] (short *npm*) for managing dependencies and *Babel* to transpile the code to ECMAScript 5. Additonally to relying on CSS3 features, this project also uses *Sass*[7] as a CSS pre-processor. The listener-interface was developed mobile-first and the speaker-view with mobile in mind; media-queries allow for these optimisations.

The JavaScript library *React* serves as the render framework of choice, additionally applying the *reactive programming* paradigm using *RxJS* to cre-

[6] https://www.npmjs.com/
[7] http://sass-lang.com/

ate a simpler interface for event-driven and asynchronous operations. These technologies will now be introduced to the reader shortly, to establish the knowledge-base necessary to understand the then following technical implementation details.

5.3.1 ECMAScript2015 and Babel[8]

JavaScript undoubtly is an integral part of front end web development and since the emergence of server-side JavaScript with Node.js and its package manager npm, has developed into a programming language widely used by web developers [26]. Both *PYPL*[9] and *TIOBE*[10] programming language indices rank JavaScript among the top 10 programming languages (PYPL at 5, TIOBE at 7 at the time of writing) [26]. Stack Overflow's 2015 Developer Survey even places JavaScript as the number 1, most-used programming language with 54.4% and JavaScript, Node.js and *AngularJS*[11] all three rank amongst the top 5 languages developers expressed an interest in developing with [50].

However, like any front end technology, JavaScript suffers from slow end user adoption, as a multitude of browser versions exist for different devices and operating systems and many people still do not auto-update their browsers. Another factor is the time it takes for browser-vendors to implement new ECMAScript standards (the standard behind JavaScript) and roll out said updates. This is exactly what is happening with the new ECMAScript standard, ECMA-262, commonly known as ECMAScript 2015 or ES6: Although the General Assembly has adopted the new standard in June 2015 [13], *Kangax' ECMAScript compatibility tables*[12] still show a fairly low level of support, especially among mobile browsers. ES6 makes JavaScript easier and more efficient to write by providing new semantics for default values, arrow-functions, template-literals, the spread operator or object destructuring [48]. It also makes JavaScript safer to develop with and easier to understand with the introduction of block-scoped variables (`let` and `const`) and finally offers native support of modules and promises [48]. As these features are all included in the new ECMAScript standard, it is safe to assume browser-vendors will implement them in the near future. Until then, developers who want to already make use of them, can *transpile* ECMAScript 2015 code to ECMAScript 5, which is exactly what Babel does. With almost 750.000 downloads in April 2016 [44] and companies such as Facebook, Netflix, Mozilla, Yahoo or PayPal using this transpiler [45], Babel is the de facto standard in transpiling to ECMAScript 5 and was also chosen for this

[8] https://babeljs.io/
[9] http://pypl.github.io/PYPL.html
[10] http://www.tiobe.com/tiobe_index
[11] https://angularjs.org/
[12] https://kangax.github.io/compat-table/es6/

Program 5.2: *Callback Hell* – Nested asynchronous callbacks to create a file upload.

```
onFileChange(event) {
  fileReader.readAsDataURL(event.file, (content, error) => {
    uploadFile(content, (response, error) => {
      this.refs.modal.close(() => {
        updateSuccessMessage(response)
      })
    })
  })
}
```

project.

5.3.2 Reactive Programming

Another problem with JavaScript, although integral part of the reason for its high popularity, is its asynchronous nature. Especially when working with highly interactive parts, the prime example being user interfaces, sequential programming quickly gets too inflexible to handle complex, event-driven applications [1]. The same is true for the server where the possibility to concurrently serve a multitude of different clients is paramount. In these cases JavaScript offers *asynchronous callbacks*. These, however, oftentimes execute more asynchronous code and in turn have to wait for another callback to fire, and another one, and another one..., which can result in something known and dreaded by most any JavaScript developer: *Callback Hell* (see programm 5.2). Since this project has to fulfill many asynchronous tasks and is heavy on the user-interface and thereby JavaScript's event system, it was crucial to find a way of handling this code gracefully. In the following, different ways of handling asynchronous programming are described, the example of an imaginary file upload will be used to demonstrate the differences on code-level:

1. A file is chosen by the user
2. The file is read by `FileReader`
3. The file is uploaded
4. The file upload modal is closed with an animation
5. As soon as the animation is over, a success message is rendered

Different approaches have been employed to lower the hurdle of writing asynchronous code, one of them being *promises*: A promise is a value, yet to be computed [21]. A promise can be a) pending (if it has not been assigned a value yet), b) resolved (if it has been assigned a value) or c) rejected (if an error occurred). With ECMAScript 2015 promises, these objects can then be

Program 5.3: *Promises* – File upload example using ECMAScript 2015 promises.

```
onFileChange(event) {
  fileReader
    .readAsDataURL(event.file)
    .then(uploadFile)
    .then((response) => {
      this.refs.modal.close(() => promise.resolve(response))
    })
    .then(updateSuccessMessage(response))
}
```

queued using the `then` keyword, to execute asynchronous code in a certain sequence (see programm 5.3).
However, promises can still create nested callbacks, especially when chaining promises that rely on other promises' resolution [21]. This is where *reactive programming* shines: The reactive programming paradigm works with streams of events, in which every event is handled as a new value and all other parts depending on that value are re-computed upon arrival of such new value. Bainomugisha et al. use the illustrative example of a simple addition to demonstrate this [1]: In sequential programming, c the expression $c = a + b$ with $a = 1$ and $b = 2$ will always be 3, until assigned a different value. With reactive programming, however, should a or b change, the value of c is automatically re-computed. JavaScript does not directly support reactive programming, but more functional languages like *Elm*[13] which can be transpiled to JavaScript, do. Another way of adding reactive programming concepts to JavaScript is using a library, such as *Bacon.js*[14] or the one chosen for this project, *ReactiveX*[15]. ReactiveX provides libraries for a multitude of different programming languages, C, C++, Java and of course JavaScript among them. The latter, called *The Reactive Extensions for JavaScript* or short *RxJs*, allows for the processing of event streams (*Observables*) as if they were simple JavaScript arrays. Instead of writing sequential code, method after method, if asynchronous or not, is applied to every element in the incoming event stream, using the array-methods such as, most notably and well-known, `map` (to apply a method to every element in the incoming stream) and `filter` (to only let a subset of events pass) (see programm 5.4).

Additionally to Observables, RxJs also knows *Subjects*, which combine both a source of events and a consumer of such. Subjects are Observables

[13]urlhttp://elm-lang.org/
[14]https://baconjs.github.io/
[15]http://reactivex.io/

Program 5.4: *RxJS* – File upload example with reactive programming in RxJS.

```
Observable.fromEvent('change', fileInput)
  .pluck('file') // pluck event.file
  .map(fileReader.readAsDataURL)
  .map(uploadFile)
  .do(() => this.refs.modal.close)
  .subscribe(updateSuccessMessage)
```

but at the same time also Observers and can be used to broadcast values to several consumers [36].

5.3.3 React[16]

As this project focuses on the front end, a mature JavaScript library for front end rendering was needed. After previous experience with the big and complex but slow AngularJS, and because of promising performance benchmarks [49] and simply to explore new JavaScript libraries, React was chosen for the rendering layer of this application. Since Facebook started developing React in 2013, it has challenged existing approaches and set new standards in front end web development [17]. Instead of creating an entire MVC framework for the front end, React really concentrates on the view by offering a way of creating independent, lightweight view components. This gives React the huge advantage of beating other front end frameworks in performance benchmarks by far [49]. Moreover, *React Native*[17], which uses the same component-based system, makes it possible to port applications to different mobile operation systems.

To define how each of these re-usable, lightweight components is displayed, they implement a `render` method, returning JSX[18]. The communication with other components happens through *properties* (`props`), which are passed into the component as XML attributes. Their own internal state, which can be manipulated e.g. through user interactions, is maintained in the `state` member [15]. Every state (internal) or property (external) change causes a re-render of the component, an example `HelloWorld` component can be found in program 5.5. These updates, as well as construction and destruction of components are handled in *lifecycle methods* [15].

As an end note on React, and a transition to the core presentation library, it should be added that React components can be nested arbitrarily deep, effectively creating semantic XML syntax which is directly linked to

[16]https://facebook.github.io/react/index.html
[17]https://facebook.github.io/react-native/
[18]https://facebook.github.io/jsx/

Program 5.5: Example code snippet using properties and state in a React component. Whenever the text input changes (i.e. a user types something), the state will be updated and the component re-rendered. The component can be used in other components as <HelloWorld greeting="Hi"/>.

```
export default class HelloWorld extends Component {
  static propTypes = { greeting : PropTypes.string }
  static defaultProps = { greeting : 'Hello' }

  constructor(props) {
    super(props)
    this.state = { name : 'World' }
  }

  const render = () => (
    <div>
      <h1>{this.props.greeting} {this.state.name}!</h1>
      <input value={this.state.name} onChange={(name) => this.setState({
     name })} />
    </div>
  )
}
```

the rendering of the components. To make it as easy as possible for other developers to use the created libraries and components, presentations are built just as a usual HTML page, using these semantic XML tags, as will be shown in chapter **??**.

5.3.4 unveil.js[19]

Although not initially planned, due to several shortcomings of other presentation libraries, this project builds upon the open-source JavaScript library *unveil.js*, which we developed prior to this project and extended and adapted in an own fork[20] during the project. While other alternatives, such as *impress.js*[21] or the popular *reveal.js*[22], exist, extensive research showed that neither of the two libraries offers the flexibility necessary to easily implement the discussed interactive mechanisms. They both were not built unleashing modern web technologies' full potential and practically consist of one big file of JavaScript, handling all functionality. We therefore decided to build our own presentation platform. Screenshots of an unveil.js presentation can be found in figure 5.3. Generally, unveil.js, like impress.js and reveal.js, is a library with which online presentations can be built. All three do not require

[19] https://github.com/ostera/unveil.js
[20] https://github.com/irisSchaffer/unveil.js
[21] http://impress.github.io/impress.js
[22] http://lab.hakim.se/reveal-js/

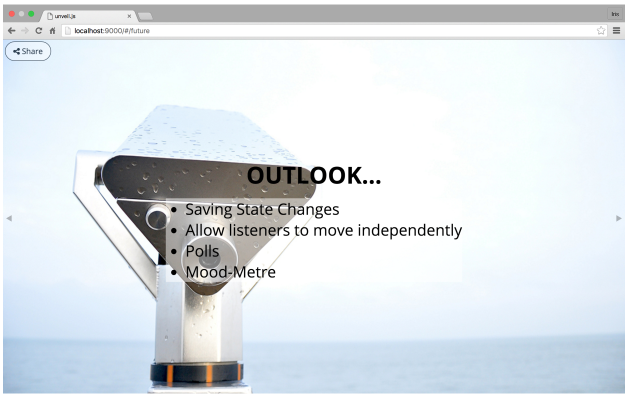


Figure 5.3: Screenshot of an example slide with unveil.js, using interactive extensions discussed in section 5.7.

a web server and can therefore be statically, and also locally served. Instead of building the presentation using a graphical user interface, all slides, transitions and styling are defined in HTML and CSS, meaning the presenter has to be familiar with basic front end web development techniques. Thanks to the use of React, unveil.js, in contrast to reveal.js and impress.js, however, does not depend on the usage of class names to identify the slide structure, but instead can make full use of semantically-named components, such as `<Slide/>` or `<Notes/>`. An example presentation can be found in program 5.9.

Like reveal.js, unveil.js operates on a 2-dimensional slide-space: Every slide can have a next and previous slide in x (main slide), as well as in y-direction (sub-slide). To generate the y axis, slides are nested in other slides. To be able to identify slides from a URL, slides have an optional unique name as well as an index in the slide-tree. Both the index and the name can be used to link to a certain slide, making it possible to share not only a whole presentation but also a certain slide.

The core of unveil.js is `UnveilApp`, which all slides and sub-slides are nested in. This component configures and sets up the entire application based on optional configuration passed in as properties. There are a few concepts we introduced jn unveil.js to allow for maximal extensibility and adaptability, namely *presenters*, *controls* and *modes*:

- **Presenters** define the way the current slide is rendered, e.g. where to display controls, if to show notes, or whether to rencer upcoming slides.
- **Controls** control a part of the application. One example would be the navigating from one slide to the next using the arrow keys on the keyboard.
- **Modes** are what allow a speaker to have a different presenter and con-

trols from an audience member. Each mode defines its own presenter and set of controls, the mode is determined by the url query parameter `mode`.

This allows anybody using unveil.js to define or overwrite modes, presenters and controls and thereby extend the base library as they wish. A few of these are already defined in the base library, namely a default `Presenter`, `UIControls` to navigate using buttons, `KeyControls` to navigate using the keyboard and `TouchControls` to navigate with swipe-gestures on touch screens. In section 5.8, modes for the audience (*default*), the speaker (*speaker*) and for use on the projection device (*projector*) will be introduced.

For these controls and the entire presentation to be navigatable, `UnveilApp` is responsible for the creation of two very important classes: `Router` and `Navigator`. These can be defined outside and passed into `UnveilApp` as properties, allowing other developers to customise their navigation logic. The `Router` is the class handling everything connected to the current url. It receives the slide-tree and can compute the indices of a slide by its name and vice versa. Whenever the browser history changes, the router finds the corresponding slide-indices, computes an array of possible directions to go into from this slide and propagates the event to `UnveilApp`, which can then re-render the application. `Navigator`, in turn, receives these directions and is responsible for the mapping of directions (*left*, *right*, *up*, *down*) to slide-indices. Controls know the navigator and can push new directions to the navigator subject, thus starting the navigation process described in detail in figure 5.4.

5.4 Project Structure

As the puropose of this project was not only to experiment with different ways of interacting with presentations using mobile devices, but also to create something worthwhile and contribute back to the vibrant open-source community, the project is enitrely open-source and separated into different repositories. These can be installed using npm, therefore allowing developers to rely only on the parts they really need.

Extended unveil.js: As discussed in section 5.3.4, the project is based on the library unveil.js. During the development of the project, certain parts of the base library were improved to allow for even easier extensibility and new presentation logic was added. This happened in a fork of the original library, which will be examined in section 5.5.

Network Synchronisation Layer: The first library of direct importance for the interaction between speaker and audience through personal devices

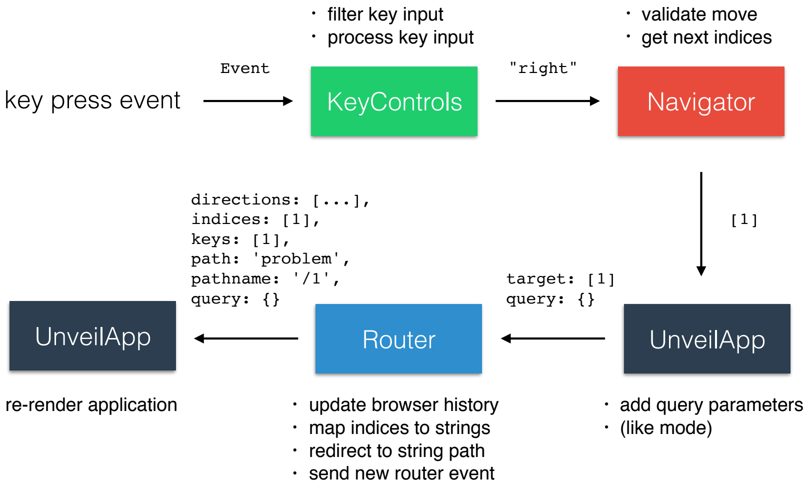


Figure 5.4: Navigation pipeline from user's key press to re-render of the presentation. The monospaced text next to the arrows symbolises the data transmitted. `KeyControls` listen for key events and process them, to then send a navigation request to go *right* to the `Navigator`. This component then maps the direction to the next slide's indices (1). `UnveilApp` then adds other information necessary for the `Router`, which then is responsible for updating the browser history, mapping the indices back to a human readable url and sending out a new router event. In the end `UnveilApp` receives this event and re-renders the presentation.

is *unveil-network-sync*[23]. This rather small library relies on unveil.js and is responsible for connecting the client and the server through web sockets and enables the synchronisation of the current slide displayed between speaker, audience and projector. The implementation of the features will be discussed in detail in section 5.6.

Interactive Extension: As the name already suggests, this library is at the core of the present thesis: It includes a dedicated presenter for the speaker, implements the insertion of additional slides and subslides and by that allows the audience to share content with the presentation. Reactions, the voting mechanism, as well as the creation of new votings on-the-fly, also live within this library. The repository, called *unveil-interactive*[24] relies on unveil-network-sync for the socket-interaction and will be covered in section 5.7 of this chapter.

[23] https://github.com/irisSchaffer/unveil-network-sync
[24] https://github.com/irisSchaffer/unveil-interactive

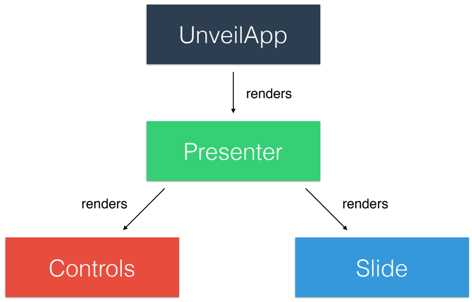


Figure 5.5: Overview over the render-flow of the application in the extended version of unveil.js. `UnveilApp` renders the presenter, which then takes care of rendering the (current) slide and all controls.

Server and Example Presentation: The last repository connected to this thesis is *unveil-client-server*[25], which includes a simple server as well as a real-world example of a presentation. In this chapter, one section was already dedicated to the server (5.2), as well as to the example application (5.8), to separate concerns a bit more clearly and be able to conclude with a demonstration of how all parts discussed earlier play together and can be used by other developers.

5.5 Extended unveil.js

The biggest adaptions and additions were necessary in the main component `UnveilApp`. A state subject was added to allow all components in the presentation to interact with the application. This subject receives an event with type and data and depending on this type starts a certain state change. Two of these state events are the `state/navigation:enable` and `state/navigation:disable` events, which set a state-variable `navigatable` to true or false. This variable is used in the controls to determine navigatability, therefore making it possible to keep the audience locked to a slide, e.g. during votings. To make it possible for the audience to add subslides, as well as to dynamically add votings on-the-fly, another event is `state/slide:add`. It includes what slide to add (content), how (subslide or main slide) and where (under or after which slide). On occurence of this event, the slide-tree has to be re-built, the router and navigator re-started and the whole presentation re-rendered, which the library also had to be prepared for.

Another adaption in `UnveilApp` is the introduction of the `context` object: Additionally to state and properties, there is a third way of communicating between components in React, called *context*. Instead of having to pass properties from one nested component to the other, every child com-

[25] https://github.com/irisSchaffer/unveil-client-server

ponent can access the context of its parents. The navigator, needed in the controls, was formerly passed from `UnveilApp` to the controls through several layers. Using context, `UnveilApp` now defines a number of different variables which are available through context, including current slide and router state, navigator, mode and the state subject discussed in the last paragraph. This makes it easy for new controls and presenters to access the data they need without other layers knowing about them or having to define them. This adaption was partly due to a change in the render hierarchy: Formerly, `UnveilApp` itself rendered the presenter (which rendered the current slide) and the controls. However, the presenter needs to be able to also control the rendering of controls (see figure 5.5), adding another layer between the rendering of controls and `UnveilApp`.

Another part that was added to the unveil.js base library is the `Notes` component. It allows adding speaker notes to each slide (see program **??**). These, however, are not rendered by the slide, but by the presenter, as will be shown in section 5.7. One more important new feature is the possibility to configure the next slide in a certain direction (left/right/up/down) and therefore allow for jumping into different branches of the presentation, thus making a presentation even more interactive. The following code, for example

```
<Slide name="start" left={[0]}>
  ...
</Slide>
```

means a navigation *left* (left arrow key pressed, swipe left etc.) will not go to the previous slide defined in the slide-tree, but rather jump to the first slide (of index 0).

5.6 Network Synchronisation Layer

As mentioned before, the network synchronisation layer is responsible for the communcation between server and client using web sockets. These are created with socket.io, a library which also provides fallbacks for browsers that do not support web sockets yet. However, this library also has a few drawbacks, especially when it comes to corporate networks. As Rob Britton describes in [46], socket.io seems to have problems getting through firewalls and can be blocked by some anti virus software. Because mobile browser support is essential for this project, I decided to still use this library.

The setup of the socket is simple: one helper function, called `createSocket` is called in the main entry point of the application to configure which server to connect to:

```
import { createSocket } from 'unveil-network-sync'
createSocket('46.101.166.172:9000')
```

Program 5.6: Shortened version of `NavigationReceiver`. First the inherited context properties are set up, then an observable waiting for `state:change` events from the socket is created. If the incoming request is not the currently displayed slide, the navigator will be pushed a new value.

```
// imports...

export default class NavigationReceiver extends React.Component {
  static contextTypes = {
    navigator:   React.PropTypes.object.isRequired,
    routerState: React.PropTypes.object.isRequired
  }

  componentDidMount () {
    this.observable = Observable.fromEvent(SocketIO, 'state:change')
      .filter((e) => !this.context.routerState.indices.equals(e))
      .subscribe(this.props.navigator.next)
  }
  ...
}
```

This function creates the socket and returns it as a singleton, so every component uses the same connection. To make importing even easier, there is another helper, called `SocketIO` which dan be importet directly and internally calls `createSocket` to retrieve the singleton:

```
import { SocketIO } from 'unveil-network-sync'
```

These sockets can then be used to listen to events or to emit them (see program 5.6) Like the state subject events, the socket.io events used in this library follow the naming convention of scoping the object targeted in by the event separated by slashes, followed by a colon and the name of the action, e.g. `state:change` or `state/slide/voting:start`.

The second responisbilty of this library is synchronising the navigation state of the presentation between speaker and audience. To do this, two controls, `NavigationSender` and `NavigationReceiver` were implemented. As the names already say, the sender broadcasts the state update, while the receiver is waiting for state updates and starts the navigation process. The latter is used in all modes (default, speaker and projector), whereas the sender is only added to the speaker mode. To make sure the sender does not end up in an infinite loop of sending and receiving its own state changes, the last received state is stored and only navigation events going to a different slide are processed further.

This mechanism, though relatively simple, already enables the audience to follow the presentation, the speaker to use his/her phone as a remote control and any number of projectors to be controlled by the speaker.

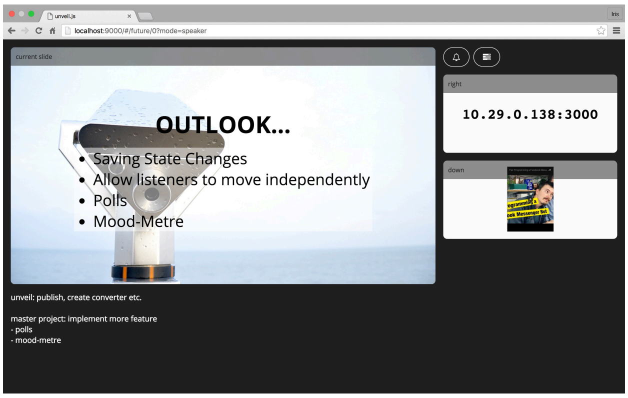


Figure 5.6: Screenshot of the speaker presenter with the current main slide, the upcoming slide to the right, available actions (muting and adding votings) and speaker notes.

5.7 Interactive Extension

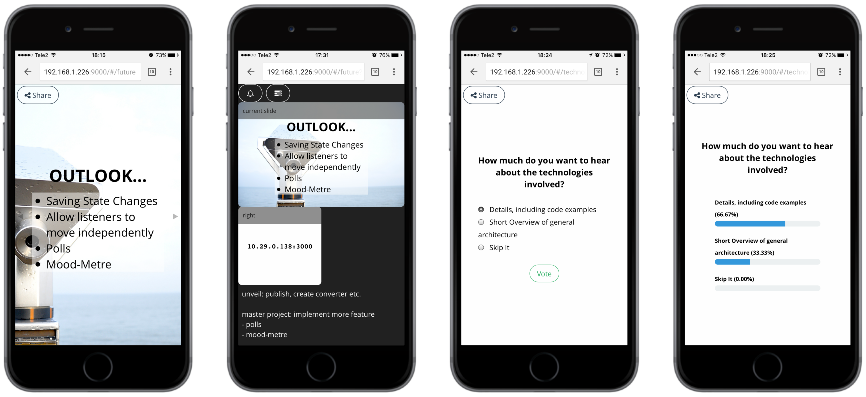
The interactive library includes several parts which will be discussed here: a speaker presenter (section 5.7.1) as well as different components connected to sharing media (section 5.7.2) and voting (section 5.7.3). Additionally controls handling the `state:initial` event were implemented to redirect new listeners to the current slide in the presentation.

5.7.1 Speaker Presenter

The speaker presenter, like the normal presenter, is responsible for rendering controls and slides. In speaker mode, where this presenter is used, notes as well as the upcoming slides (right and down) are also shown (see figure 5.6). This means the presenter has to find these next slides, using the router's information about the navigatable directions, and render them in a designated area. To make the presenter view usable on mobile devices, special attention was paid to mobile stylesheets (see figure 5.7 (b)). This ensures that everything is big enough to be readable and all buttons are clickable.

5.7.2 Media

Another responsibility of the interactive extension is the possibility for audience members to share content with the presentation. For this to work, three different controls were created: `MediaSender`, `MediaReceiver` and `MediaAcceptor`. The sender is used in the default mode so users can share their content (see figure 5.8 (a)), the acceptor is enabled in speaker mode, to accept or reject incoming media (see figure 5.8 (b)) and the receiver in the end handles the creation of a new slide if the content was accepted and



(a) (b) (c) (d)

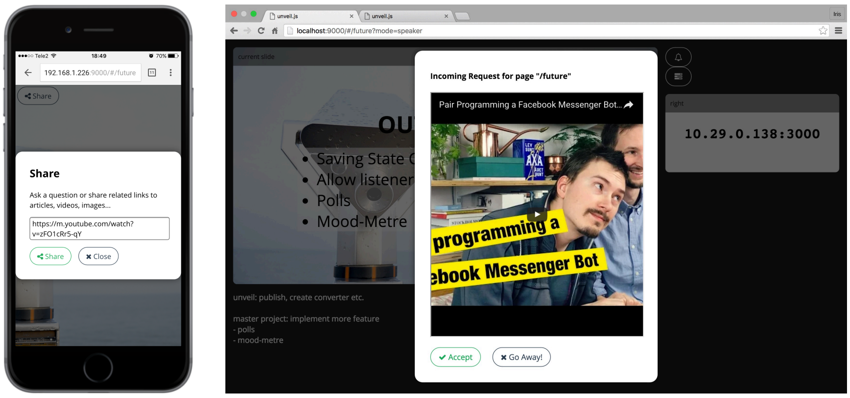
Figure 5.7: Mobile view of a presentation slide in (a) default mode and (b) speaker mode as well as (c) voting view before voting and (d) voting view after voting. Mind the share button in the left upper corner in (a) as well as the buttons to mute content requests and create votings in (b).

is therefore necessary in all modes. As incoming content requests could disrupt the presentation flow and distract the speaker, an option to mute the requests was built into the application. If the *do not disturb* mode is turned on, slides will silently be added as subslides, without causing the acceptor modal to open. This way the audience' additions can be re-visited after the end of the presentation. Generally, this feature can be used to either post a link to an interesting picture, website or even youtube video, or also as text input, for example to add a comment or a question regarding a certain slide. This works through the introduction of the presentation components `Media` and `IFrame`, which, depending on the shared content, render an image-tag, blockquote or IFrame. The differentiation of these is carried out with regular expressions, as the following to check if the content is the link to an image:

```
isImg (str) {
  let imgRegex = new RegExp(/\.(jpe?g|png|gif|bmp)$/i)
  return imgRegex.test(str)
}
```

As far as the implementation of the controls is concerned, these ones are the first ones discussed in the present thesis makeing use of the `render()` method. It is used to output the *share* button in the left upper corner in default mode (see figure 5.7 (a)) as well as the share modal opened when clicking on said button (see figure 5.8 (a)). The same happens in the `MediaAcceptor`, which uses the `render()` method to display the *mute* button shown in figure 5.6 as well as the modal for accepting media (see figure 5.8 (b)).

These are also the first controls using state: in the sender a click on the



(a) (b)

Figure 5.8: Screenshots of sharing content. (a) media sender on mobile phone shares youtube link (b) speaker receives content request.

share button sets the state variable `sharingMode` to `true` and through that enables the rendering of the modal. In the acceptor an array of `requests` is filled as new content is shared and emptied again, as the speaker accepts or denies them.

On event level, the socket events `state/slide/add:accept` and `state/slide:add` are included in the process of sharing new content, in the end an unveil state event of type `state/slide:add` lets `UnveilApp` create the new slide and add it to the slide tree. The whole flow is outlined in details in figure 5.9.

5.7.3 Voting

The last group of components connected to the interactive extension covered in this chapter allow speakers to create votings (both during in the preparation of the presentation and on-the-fly as shown in figure 5.10) and members of the audience to vote (see figure 5.7 (c) and (d)).

The main presentational component involved in the voting process is `Voting`, which keeps track of the current voting scores and has a `Question` and a number of `Answer` components as children (see program 5.7). `Voting` also remembers if an audience member has already voted and if so, displays `Result` components. In speaker and presenter mode only these results are shown.

The audience can start voting as soon as the speaker navigates to the slide with the voting, until then the vote button is disabled. Once the voting has started, the possibility for all audience members to navigate to a different slide is disabled. This happens in the `VotingNavigatableSetter`, which is only installed for default mode. The voting start event, as well as the

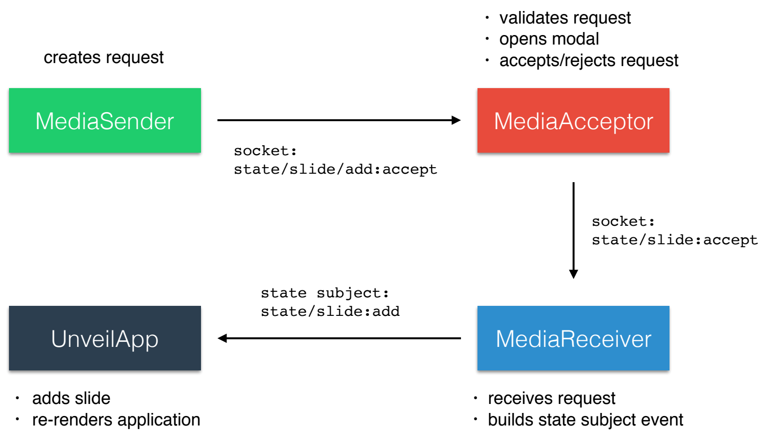


Figure 5.9: Flow of adding content, monospaced text symbolises type and name of events. First the `MediaSender` of the default mode sends a request, which the speaker mode's `MediaAcceptor` listens to. If the request is accepted by the speaker or requests are muted, another socket event is broadcast, which the `MediaReceiver` waits for. This component is enabled in all modes and emits the state subject event to add a new slide, which `UnveilApp` reacts to.

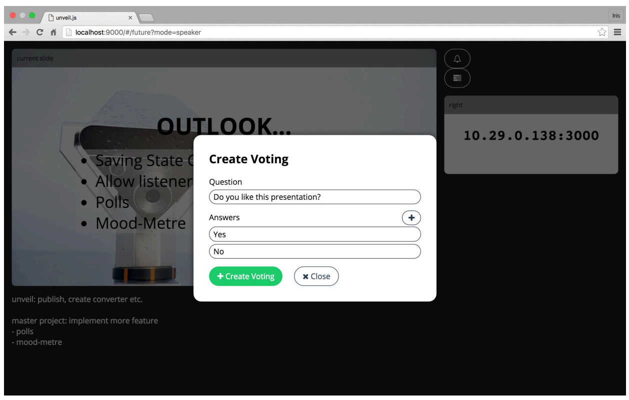


Figure 5.10: Screenshot of the desktop speaker mode, creating a new voting, on-the-fly, during a presentation.

Program 5.7: Example code for preparing a slide with voting.

```
<Voting name="like">
  <Question>Do you like these slides?</Question>
  <Answer value="yes">Yes</Answer>
  <Answer value="no">No</Answer>
</Voting>
```

Program 5.8: Mode definition for setting up an unveil.js presentation. Speaker and projector modes are omitted to keep the example short but follow the same pattern as the default mode.

```
const modes = {
  default: {
    controls : [
      KeyControls, TouchControls, UIControls,
      NavigationReceiver,
      MediaSender, MediaReceiver,
      VotingNavigatableSetter, VotingReceiver
    ],
    presenter: Presenter
  },
  speaker: {...},
  projector: {...}
};
```

voting end event are broadcast by the `VotingController`, which checks the speaker's current slide for the occurence of a `Voting`.

Once the voting has started, internally, the `Voting` component remembers which answer the user has clicked in the `answer` state variable. Once the submit button is pressed, a `state/slide/voting:answer` event is fired and broadcast to all clients, which update their internal voting results. Because the results of the voting should be available throughout the whole presentation and not be reset when leaving the slide, `Voting` also handles the communication with local storage, to store current results.

5.8 Example Application

To conclude this chapter, I want to show how all the discussed libraries in the end can be put together to a presentation. The whole presentation can be found in the unveil-client-server repository.

The first step to use unveil.js and its extensions is to start a new project, require the necessary npm packages and set up an `index.html` page which includes the necessary stylesheets and scripts. The entry point for these scripts is the file `index.js`, in which the whole presentation is set up and unveil.js is configured by setting up its modes (see program 5.8). As explained before, slides are defined in HTML using the components unveil.js and its extensions provide. An example of this is shown in program 5.9.

The styling of the presentation is handled by CSS. The `Slide` component automatically adds the name of a slide as its id, allowing to efficiently style slide by slide, as well as applying styles for all slides at once using the `.slide` class. Program 5.10 shows an example of how to style slides.

Program 5.9: Creation of presentation using modes from program 5.8. Sets up two slides as an example. The DOM will be attached to the element of id `unveil` in the base HTML document.

```
ReactDOM.render((
  <UnveilApp modes={modes}>
    <Slide name="start">
      <h1>Unveil</h1>
      <h2>a meta presentation</h2>
    </Slide>
    <Slide name="problem">
      <img src="./img/problem.png" />
      <Notes>someone always wanted to show something from their device</
    Notes>
    </Slide>
    ...
  </UnveilApp>
), document.getElementById('unveil'));
```

Program 5.10: Example styling for slides using Sass. In this particular piece of code, the font family of all slides is set and a background image is added to the start slide.

```
.slide
  font-family: 'Open Sans'
#start
  background-image: url('../img/explore.jpg')
```

References

Literature

[1] Engineer Bainomugisha et al. "A Survey on Reactive Programming". *ACM Comput. Surv.* 45.4 (Aug. 2013), 52:1–52:34. URL: http://doi.acm.org/10.1145/2501654.2501666 (cit. on pp. 30, 31).

[2] R. Bajko and D. Fels. "A comparative analysis of meeting participant perception and use of smartphones and other mobile devices during meetings". In: *Professional Communication Conference (IPCC), 2013 IEEE International*. July 2013, pp. 1–6 (cit. on p. 2).

[3] Matthias Böhmer, T. Scott Saponas, and Jaime Teevan. "Smartphone Use Does Not Have to Be Rude: Making Phones a Collaborative Presence in Meetings". In: *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services*. MobileHCI '13. Munich, Germany: ACM, 2013, pp. 342–351. URL: http://doi.acm.org/10.1145/2493190.2493237 (cit. on pp. 2, 5).

[4] François Bry, Vera Gehlen-Baum, and Alexander Pohl. "Promoting awareness and participation in large class lectures: The digital backchannel backstage". *Proceedings of the IADIS International Conference e-Society 2011* (2011), pp. 27–34 (cit. on pp. 1–5, 9, 11, 14, 25).

[5] Spencer Cappallo, Thomas Mensink, and Cees G.M. Snoek. "Image2Emoji: Zero-shot Emoji Prediction for Visual Media". In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM '15. Brisbane, Australia: ACM, 2015, pp. 1311–1314. URL: http://doi.acm.org/10.1145/2733373.2806335 (cit. on p. 22).

[6] Spencer Cappallo, Thomas Mensink, and Cees G.M. Snoek. "Query-by-Emoji Video Search". In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM '15. Brisbane, Australia: ACM, 2015, pp. 735–736. URL: http://doi.acm.org/10.1145/2733373.2807961 (cit. on p. 22).

[7] Albert T. Chamillard. "Using a Student Response System in CS1 and CS2". In: *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*. SIGCSE '11. Dallas, TX, USA: ACM, 2011, pp. 299–304. URL: http://doi.acm.org/10.1145/1953163.1953253 (cit. on pp. 3, 4).

[8] Debaleena Chattopadhyay et al. "Office Social: Presentation Interactivity for Nearby Devices". In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. Santa Clara, California, USA: ACM, 2016, pp. 2487–2491. URL: http://doi.acm.org/10.1145/2858036.2858337 (cit. on pp. 3, 6, 7, 12).

[9] Yufeng Cheng, Jiayu Sun, and Junfeng Hu. "TOPT: A Tree-based Online Presentation Tool". In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '15. Vilnius, Lithuania: ACM, 2015, pp. 342–342. URL: http://doi.acm.org/10.1145/2729094.2754851 (cit. on pp. 1, 4, 12, 14, 25).

[10] Creative Bloq Staff. "Hover is dead, long live hover" (Apr. 15, 2013). URL: http://www.creativebloq.com/design/hover-dead-long-live-hover-4132957 (visited on 08/21/2016) (cit. on p. 22).

[11] Andreas Dieberger, Cameron Miner, and Dulce Ponceleon. "Supporting Narrative Flow in Presentation Software". In: *CHI '01 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '01. Seattle, Washington: ACM, 2001, pp. 137–138. URL: http://doi.acm.org/10.1145/634067.634151 (cit. on pp. 13, 24).

[12] Thomas Dimson. "Emojineering Part 1: Machine Learning for Emoji Trends" (Apr. 2015). URL: http://instagram-engineering.tumblr.com/post/117889701472/emojineering-part-1-machine-learning-for-emoji (visited on 08/17/2016) (cit. on p. 22).

[13] Ecma International. *ECMAScript 2015 Language Specification*. 6th ed. June 2015. URL: http://www.ecma-international.org/ecma-262/6.0/ (cit. on p. 29).

[14] Margarita Esponda. "Electronic Voting On-the-fly with Mobile Devices". In: *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*. ITiCSE '08. Madrid, Spain: ACM, 2008, pp. 93–97. URL: http://doi.acm.org/10.1145/1384271.1384298 (cit. on pp. 1, 3, 4, 13, 25).

[15] Facebook Inc. Official React Documentation. URL: http://facebook.github.io/react/docs (visited on 04/13/2016) (cit. on p. 32).

[16] Graham Farrell et al. "Trial by Tablet: User Evaluation of the Digital Courtroom". In: *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*. OzCHI '13. Adelaide, Australia: ACM, 2013, pp. 325–328. URL: http://doi.acm.org/10.1145/2541016.2541068 (cit. on p. 2).

[17] Cory Gackenheimer. *Introduction to React*. Apress, 2015. URL: https://books.google.se/books?id=NZCKCgAAQBAJ (cit. on p. 32).

[18] Google. *Motion – Duration and easing*. URL: https://material.google.com/motion/duration-easing.html (visited on 07/26/2016) (cit. on p. 18).

[19] Ryota Inoue et al. "Visualizing Real-Time Questionnaire Results to Promote Participation in Interactive Presentations". In: *Advanced Applied Informatics (IIAIAAI), 2014 IIAI 3rd International Conference on*. Aug. 2014, pp. 64–69 (cit. on pp. 1, 2, 6, 14, 25, 27).

[20] Ellen A. Isaacs, Trevor Morris, and Thomas K. Rodriguez. "A Forum for Supporting Interactive Presentations to Distributed Audiences". In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. CSCW '94. Chapel Hill, North Carolina, USA: ACM, 1994, pp. 405–416. URL: http://doi.acm.org/10.1145/192844.193060 (cit. on p. 21).

[21] Kennedy Kambona, Elisa Gonzalez Boix, and Wolfgang De Meuter. "An Evaluation of Reactive Programming and Promises for Structuring Collaborative Web Applications". In: *Proceedings of the 7th Workshop on Dynamic Languages and Applications*. DYLA '13. Montpellier, France: ACM, 2013, 3:1–3:9. URL: http://doi.acm.org/10.1145/2489798.2489802 (cit. on pp. 30, 31).

[22] Sammi Krug. "Add Reactions to Pull Requests, Issues, and Comments" (Mar. 10, 2016). URL: https://github.com/blog/2119-add-reactions-to-pull-requests-issues-and-comments (visited on 08/17/2016) (cit. on p. 22).

[23] Sammi Krug. "Reactions Now Available Globally" (Feb. 24, 2016). URL: http://newsroom.fb.com/news/2016/02/reactions-now-available-globally/ (visited on 08/17/2016) (cit. on p. 22).

[24] Jeffrey H. Kuznekoff, Stevie Munz, and Scott Titsworth. "Mobile Phones in the Classroom: Examining the Effects of Texting, Twitter, and Message Content on Student Learning". *Communication Education* 64.3 (2015), pp. 344–365. eprint: http://dx.doi.org/10.1080/03634523.2015.1038727. URL: http://dx.doi.org/10.1080/03634523.2015.1038727 (cit. on p. 2).

[25] Jeffrey H. Kuznekoff and Scott Titsworth. “The Impact of Mobile Phone Usage on Student Learning”. *Communication Education* 62.3 (2013), pp. 233–252. eprint: http://dx.doi.org/10.1080/03634523.2013.767917. URL: http://dx.doi.org/10.1080/03634523.2013.767917 (cit. on p. 2).

[26] Kyriakos-Ioannis D. Kyriakou, Ioannis K. Chaniotis, and Nikolaos D. Tselikas. “The GPM meta-transcompiler: Harmonizing JavaScript-oriented Web development with the upcoming ECMAScript 6 x201C;Harmony x201D; specification”. In: *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. Jan. 2015, pp. 176–181 (cit. on p. 29).

[27] Jianwei Lai and Dongsong Zhang. “A Study of Direction’s Impact on Single-handed Thumb Interaction with Touch-screen Mobile Phones”. In: *CHI ’14 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’14. Toronto, Ontario, Canada: ACM, 2014, pp. 2311–2316. URL: http://doi.acm.org/10.1145/2559206.2581154 (cit. on p. 19).

[28] Leonhard Lichtschlag et al. “Canvas Presentations in the Wild”. In: *CHI ’12 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’12. Austin, Texas, USA: ACM, 2012, pp. 537–540. URL: http://doi.acm.org/10.1145/2212776.2212828 (cit. on p. 12).

[29] David Lindquist et al. “Exploring the Potential of Mobile Phones for Active Learning in the Classroom”. In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’07. Covington, Kentucky, USA: ACM, 2007, pp. 384–388. URL: http://doi.acm.org/10.1145/1227310.1227445 (cit. on pp. 3, 5).

[30] Gary R. McClain. *Presentations: Proven Techniques for Creating Presentations That Get Results*. Adams Media, 2007. URL: https://books.google.se/books?id=bBnb6zFw7pEC (cit. on p. 10).

[31] Michael G. Moore. “Editorial: Three types of interaction”. *American Journal of Distance Education* 3.2 (1989), pp. 1–7. URL: http://dx.doi.org/10.1080/08923648909526659 (cit. on pp. 9, 11).

[32] Brad A. Myers, Herb Stiel, and Robert Gargiulo. “Collaboration Using Multiple PDAs Connected to a PC”. In: *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*. CSCW ’98. Seattle, Washington, USA: ACM, 1998, pp. 285–294. URL: http://doi.acm.org/10.1145/289444.289503 (cit. on pp. 2, 14).

[33] Matei Negulescu et al. “Tap, Swipe, or Move: Attentional Demands for Distracted Smartphone Input”. In: *Proceedings of the International Working Conference on Advanced Visual Interfaces*. AVI ’12. Capri

Island, Italy: ACM, 2012, pp. 173–180. URL: http://doi.acm.org/10.1145/2254556.2254589 (cit. on pp. 6, 19).

[34] Les Nelson et al. "Palette: A Paper Interface for Giving Presentations". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '99. Pittsburgh, Pennsylvania, USA: ACM, 1999, pp. 354–361. URL: http://doi.acm.org/10.1145/302979.303109 (cit. on p. 13).

[35] Gerald M. Phillips, Gerald M. Santoro, and Scott A. Kuehn. "Computer: The use of computer-mediated communication in training students in group problem-solving and decision-making techniques". *American Journal of Distance Education* 2.1 (1988), pp. 38–51. URL: http://dx.doi.org/10.1080/08923648809526607 (cit. on p. 9).

[36] ReactiveX. RxJS Documentation on Github. URL: https://github.com/Reactive-Extensions/RxJS (visited on 04/12/2016) (cit. on p. 32).

[37] Nick Roberts. "Do You Speak Emoji? Bing Does" (Oct. 27, 2014). URL: https://blogs.bing.com/search/2014/10/27/do-you-speak-emoji-bing-does/ (visited on 08/17/2016) (cit. on p. 22).

[38] Barry Schwartz. "Google now also allows you to search using emoji characters" (May 18, 2016). URL: http://searchengineland.com/google-now-also-allows-search-using-emoji-characters-249802 (visited on 08/17/2016) (cit. on p. 22).

[39] Beat Signer and Moira C. Norrie. "PaperPoint: A Paper-based Presentation and Interactive Paper Prototyping Tool". In: *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*. TEI '07. Baton Rouge, Louisiana: ACM, 2007, pp. 57–64. URL: http://doi.acm.org/10.1145/1226969.1226981 (cit. on p. 13).

[40] Mark Stefik et al. "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings". *Commun. ACM* 30.1 (Jan. 1987), pp. 32–47. URL: http://doi.acm.org/10.1145/7885.7887 (cit. on p. 2).

[41] Jaime Teevan et al. "Displaying Mobile Feedback During a Presentation". In: *Proceedings of the 14th International Conference on Human-computer Interaction with Mobile Devices and Services*. MobileHCI '12. San Francisco, California, USA: ACM, 2012, pp. 379–382. URL: http://doi.acm.org/10.1145/2371574.2371633 (cit. on pp. 1, 5, 6, 14, 21–23, 25).

[42] Vasileios Triglianos and Cesare Pautasso. "ASQ: Interactive Web Presentations for Hybrid MOOCs". In: *Proceedings of the 22Nd International Conference on World Wide Web*. WWW '13 Companion. Rio de Janeiro, Brazil: International World Wide Web Conferences Steer-

ing Committee, 2013, pp. 209–210. URL: http://dl.acm.org/citation.cfm?id=2487788.2487894 (cit. on pp. 1, 4, 5, 25, 27).

[43] Philipp Wacker. *How Does It Feel? : Presenter Experience and Evaluation While Using Canvas Presentation Tools.* MS. Aachen, Techn. Hochsch., Masterarbeit, 2014. Aachen, 2014. URL: http://publications.rwth-aachen.de/record/461192 (cit. on pp. 11, 18, 23).

Online sources

[44] NPM's babel package page. URL: https://www.npmjs.com/package/babel (visited on 04/21/2016) (cit. on p. 29).

[45] Babel. *See who's using Babel · Babel.* URL: https://babeljs.io/users/ (visited on 04/11/2016) (cit. on p. 29).

[46] Rob Britton. *Why We Ditched Socket.IO.* Dec. 10, 2013. URL: https://www.robbritton.com/2013/12/10/why-we-ditched-socket-io (visited on 04/11/2016) (cit. on p. 38).

[47] i>clicker. *About i>clicker.* i>clicker website, About i>clicker page. URL: http://www1.iclicker.com/about-i-clicker/ (visited on 07/04/2016) (cit. on p. 3).

[48] Lars Kappert. *ECMAScript 6 (ES6): What's New In The Next Version Of JavaScript.* Oct. 28, 2015. URL: https://www.smashingmagazine.com/2015/10/es6-whats-new-next-version-javascript/ (visited on 04/08/2016) (cit. on p. 29).

[49] Sebastián Peyrott. *More Benchmarks: Virtual DOM vs Angular 1 2 vs Mithril.js vs cito.js vs The Rest.* Jan. 12, 2016. URL: https://auth0.com/blog/2016/01/11/updated-and-improved-more-benchmarks-virtual-dom-vs-angular-12-vs-mithril-js-vs-the-rest (visited on 04/11/2016) (cit. on p. 32).

[50] Stack Overflow. *2015 Developer Survey.* 2015. URL: http://stackoverflow.com/research/developer-survey-2015 (visited on 04/11/2016) (cit. on p. 29).

[51] University of Colorado Boulder. *CUClickers / iclickers - FAQ.* URL: http://www.colorado.edu/oit/services/teaching-learning-tools/cuclickers/faq (visited on 01/07/2016) (cit. on p. 3).

[52] René Verheij. Answer to Stack Overflow Question "Socket.io Client: respond to all events with one handler?" Nov. 27, 2015. URL: http://stackoverflow.com/a/33960032/3340229 (visited on 04/20/2016) (cit. on p. 28).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —

Breite = 100 mm
Höhe = 50 mm

— Diese Seite nach dem Druck entfernen! —