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 [**Teevan:MobileFeedbackDuringPresentation**]. 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 [**Bry:Backstage**, **Cheng:TreebasedOnlinePresentations**, **Esponda:ElectronicVotingOnThe**
Inoue:RealTimeQuestionnaire, **Teevan:MobileFeedbackDuringPresentation**, **Triglianos:InteractiveWebPresentationsImpress**], 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 [**Inoue:RealTimeQuestionnaire**, **Triglianos:InteractiveWebPresentationsImp**
(in [**Inoue:RealTimeQuestionnaire**] 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 [**Triglianos:InteractiveWebPresentationsImpress**], 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 [**Farrell:TrialByTablet**]. Especially during presentations, mobile device usage is still perceived as rude and can be a source of distraction [**Bohmer:SmartphoneUseRude**, **Bajko:ComparativePerceptionSmartphoneMeeting**, **Kuznekoff:ImpactPhoneStudentLear** although other studies indicate that lecture-relevant phone use in classrooms can actually be beneficial [**Kuznekoff:MobilePhoneClassroomTwitter**] 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 [**Bohmer:SmartphoneUseRude**] and holds the potential of promoting participation and helping introverts overcome the hurdle of speaking out loud [**Bry:Backstage**]. 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 multi-media 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. [**Stefik:BeyondTheChalkboard**] already experimented with the use of personal computers in meeting rooms as early as 1987 and Myers et al. [**Myers:CollaborationPDAs**] 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

Figure 2.1: *i>clicker* devices, used in [**Chamillard:StudentResponseSystem**]. Image source [**iclicker**].

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 [**Inoue:RealTimeQuestionnaire**] or remote-controlling [**Chattopadhyay:OfficeSocialRemoteControl**], 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 [**Bry:Backstage**], 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 [**cuclickers:faq**] which can be used for tasks like taking attendance and voting [**Chamillard:StudentResponseSystem**]. Using these clicker systems has shown to "yield a strong and positive relationship with student learning" [**Chamillard:StudentResponseSystem**]. 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. [**Lindquist:ExploringMobilePhonesActiveLearning**] 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 [**Esponda:ElectronicVotingOnTheFly**] 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 [**Chamillard:StudentResponseSystem**], have also recognised the shortcommings of their hardware-approach and now build mobile apps for students' personal devices. Like[**Esponda:ElectronicVotingOnTheFly**], 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* [**Triglianos:InteractiveWebPresentationsImpress**] 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. [**Cheng:TreebasedOnlinePresentations**], 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.[**Bry:Backstage**] developed a backchannel specifically for large classrooms. Students can post

[1] http://www1.iclicker.com/

[2] http://impress.github.io/impress.js

[3] http://backstage.pms.ifi.lmu.de/

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 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 [**Lindquist:ExploringMobilePhonesActiveLearning**, **Triglianos:InteractiveWebPresentationsImpress**, **Bry:Backstage**], 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 [**Bohmer:SmartphoneUseRude**], 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* [**Teevan:MobileFeedbackDuringPresentation**], a piece of software

Figure 2.2: *Crowd Feedback* [**Teevan:MobileFeedbackDuringPresentation**] used during a presentation. The bar next to the slides shows one dot per participant in the meeting. The feedback dots fade out over time.

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 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* [**Chattopadhyay:OfficeSocialRemoteControl**], 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 [**Chattopadhyay:OfficeSocialRemoteControl**]. 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 [**Negulescu:TapSwipeMove**]. Another disadvantage is the overhead of

Figure 2.3: *Office Social* [**Chattopadhyay:OfficeSocialRemoteControl**]'s smartphone app interface. A preview of the slide is shown on top, followed by big buttons for navigating between slides.

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 [**Inoue:RealTimeQuestionnaire**]: 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 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 lis-

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

teners, 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.

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 5.2.4).

[6] http://prezi.com
[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 [**Bry:Backstage**] 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 [**Moore:ThreeTypesOfInteraction**]. Group-dynamics largly depend on the audience size and smaller groups usually perform better than big ones [**Phillips:GroupProblemSolving**]. 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 [**McClain:TypeOfPresentations**] 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" [**McClain:TypeOfPresentations**]. 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 [**Bry:Backstage**]. 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 [**Moore:ThreeTypesOfInteraction**]: 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 [**Wacker:PresenterExperience**]. 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 [**Chattopadhyay:OfficeSocialRemoteControl**], 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.

While using arrow-keys in a desktop environment feels natural to navigate between slides, the native equivalent on touch-devices are swipe gestures. These are more accurately and faster when operating a phone with one hand [**Lai:SingleHandedThumbInteraction**] and are less prone to error when used eye-free [**Negulescu:TapSwipeMove**] and should therefore be prefered over buttons, clustering the interface.

3.2.2 Following Slides

For members of the audience, one important feature, both in the context of re-visiting slides, to accommodate individual learning paces [**Cheng:TreebasedOnlinePresentation**
and even to give latecomers a chance to catch up with the presentation [**Chattopadhyay:OfficeSocialRemoteControl**], is to be able to independently follow the slides. This 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 [**Chattopadhyay:OfficeSocialRemoteControl**], 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

[**Lichtschlag:CanvasPresentationsInTheWild**] and have 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 [**Dieberger:NarrativeFlow**]. PowerPoint extensions enabeling advanced forms of navigation, as well as the presenter looking through slides before projecting them are discussed in [**Dieberger:NarrativeFlow**], [**Nelson:PalettePaperInterface**] and [**Signer:PaperPoint**], the latter, however, will not be part of our implementation.

3.2.4 Audience Questions

Another 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 [**Esponda:ElectronicVotingOnTheFly**], 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 [**Esponda:ElectronicVotingOnTheFly**]. 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 [**Bry:Backstage**], 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 [**Teevan:MobileFeedbackDuringPresentation**]; as highlighted by Teevan et al., however, their simplistic approach of just offering *likes* and *dislikes* is not faceted enough to represet 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 [**Teevan:MobileFeedbackDuringPresentation**], 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

[**Cheng:TreebasedOnlinePresentations**] and text-based annotations [**Inoue:RealTimeQuestion**
Myers:CollaborationPDAs] 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 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.

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 possibility to shape presentation for audience	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; slide-content by adding related resources	Small groups

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 fea-

tures 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 4.4. 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

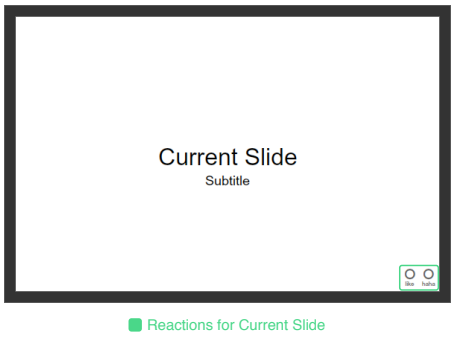


Figure 4.1: Wireframe of slide in projector mode, as seen on a projector. No visual controls are shown, only the current slide and listener reactions are displayed. The presentation progresses through the presenter mode's remote controlling feature.

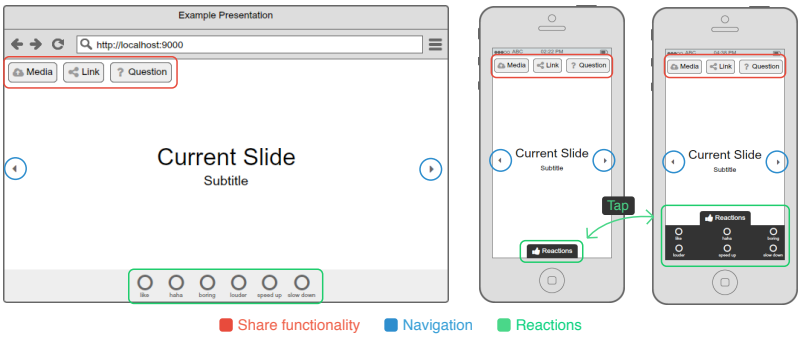


Figure 4.2: Wireframes of general interface in listener mode for mobile phones and desktops. Both offer buttons to share media, links and questions with the presenter, arrow buttons to navigate through the presentation and a possibility to react to the current slide. On mobile this feature is revealed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

interactions 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 [**GoogleMaterialDesignGuide**] "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 [**Wacker:PresenterExperience**];

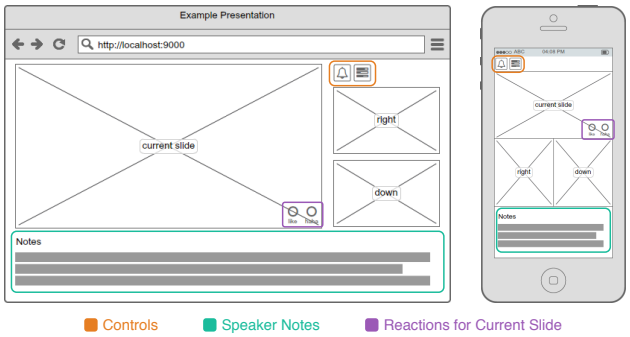


Figure 4.3: Wireframes of general interface in speaker mode for mobile phones and desktops. The interface consists of a preview of the current slide, the next main slide (*right*) and the next subslide (*down*), as well as showing presenter notes. It also offers buttons to toggle muting of incoming requests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

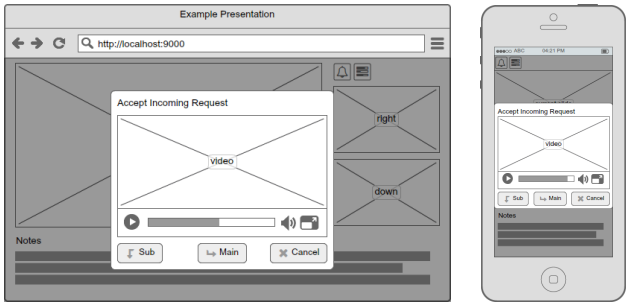


Figure 4.4: Wireframes of modal interface in speaker mode for mobile phones and desktops. The shown modal allows the presenter to add a listener-submitted video as a new main or subslide or dismiss the request. It pops up as soon as a listener wants to share content with the slide.

we therefore decided to design 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 **??**).

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 [**Lai:SingleHandedThumbInteraction**] and require less time looking at the screen on mobile devices [**Negulescu:TapSwipeMove**], 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

(a) (b) (c)

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

metaphors are used to symbolises their state (pressed, hovered, disabled), as shown in figure 4.7.

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 Reactions

Although binary digital reactions to a presentation are not an entirely new idea [**Teevan:MobileFeedbackDuringPresentation**], versatile feedback that goes beyond positive and negative, to our knowledge, has not yet been explored. From the evaluation in [**Teevan:MobileFeedbackDuringPresentation**] and [**Isaacs:InteractivePresentationsDistributedAudience**] and from observing presentations and meetings, a pool of possible reactions has been narrowed down to six – three emotions (approval, laughter, boredom) and three request types (louder, speed up, slow down) (see figure 4.6). The reason behind the missing disapproval is on one hand that test subjects in [**Teevan:MobileFeedbackDuringPresentation**] felt less comfortable giving negative feedback and the button was used less than the positive one and included reactions such as *boredom* or *speed up* and *slow down*, on the other side we hope this will encourage more elaborate feedback of disagreement using the content sharing functionality instead.

Since the presentation of feedback in [**Teevan:MobileFeedbackDuringPresentation**] 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 in the right corner of the projector, with a small badge symbolising how many people have sent this feedback for the current slide (see figure **??** (a)).

Another challenge with displaying non-binary feedback 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 communicating [**Instagramm:Emoji**, **Cappallo:EmojiVideoSearch**]. Instagram has found that almost half of its comments and captions nowadays include emoji. With Google [**Google:Emoji**] and Bing [**Bing:Emoji**] adding support for emoji-search and companies like Facebook [**Facebook:Reactions**] and GitHub [**Github:Reactions**] offering emoji-based reaction systems, it

Figure 4.6: All six possible reactions and their emoji (from left): *approval*, *laughter*, *boredom*, *louder*, *speed up* and *slow down*. The *speed up* emoji is in hover-state.

(a) (b)

Figure 4.7: Details of visualisation of reactions (a) with number of reactions as found in the presenter view and optionally the projector and (b) with hover status as found on listener interface.

is safe to assume the majority of digital natives is familiar with the concept and meaning of emojis. Although it would be possible to include a complete emoji-keyboard we feel it is easier for less technology-oriented users to offer only a sub-set including a short description of each reaction (see figure 4.6). However, this sub-set can easily be extended or overwritten by the presenter.

Another detail worth mentioning is the hover-effect of the emoji, which is of special importance on desktops (see figure **??** (b)).

4.5 Polls

4.6 Content Sharing

Chapter 5

Implementation

This chapter dives into the technical implementation details of the thesis project and gives an overview of the used technologies and explains why these were chosen over others. Like many other projects in this area [**Bry:Backstage**, **Cheng:TreebasedOnlinePresentations**, **Esponda:ElectronicVotingOnThe**
Inoue:RealTimeQuestionnaire, **Teevan:MobileFeedbackDuringPresentation**, **Triglianos:InteractiveWebPresentationsImpress**], the project is realised as a web application. 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, the advantages that come with this decision outweigh the disadvantages, as I like to think, for both users and developers: as nobody has to download any apps to their devices, it is easier to bring the audience to use the developed application [**Triglianos:InteractiveWebPresentationsImpress**]. The major advantages for developers on one hand is the ability to only concentrate 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 distributing them through the App Store or Play Store and without the need for users to manually update them. The chosen JavaScript library React, however, offers the possibility to cross-compile applications to different devices using React Native, which should make it fairly easy to port the existing web application in the future.

As it was important to achieve fast response times and because *WebSockets* have successfully been deployed in the real-time features of other presentation tools [**Inoue:RealTimeQuestionnaire**, **Triglianos:InteractiveWebPresentationsImpre**
this technology has also been used in this thesis project, to handle the communication between clients and server.

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 this community, all the libraries developed during this project are published as open-source on my GitHub profile[1] and freely available for anyone to use.

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 use an easily extensible presentation library to then build the mechanisms discussed in the previous chapter on top of it. As the focus was placed on the interaction possibilities between speaker and audience, the creation of the presentation for the speaker or the management of slides and presentations were out of scope. Therefore the server used for connecting different users to the presentation was kept as simple as possible, allowing any potential other developer to work with their own servers and technology stacks.

In total, a system with several ways of interacting with the presentation from mobile or desktop devices was created, putting emphasise on mobile-optimised views and navigation possibilities. This system includes synchronisation of navigation state and state changes between viewers and speaker, the possibility to add sub-slides during the presentation for the audience, a speaker-view showing the next slides and controls, real-time voting (both created on-the-fly and prepared beforehands) and the possibility to create different paths through the presentation. In the following the technologies used in the project will be analysed and described to then discuss implementation details, problems and solutions of the mentioned components.

5.2 Technologies

The project generally tries to follow 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 manager*[2](short *npm*) for managing dependencies and *Babel* to transpile to ECMAScript5. Additonally to relying on CSS3 features, this project also uses *Sass*[3] as a CSS pre-processor. Media-queries allow for mobile-friendly views.

On the front end, which this project focuses on, the JavaScript library *React* is the framework of choice, additionally applying the *reactive program-*

[1] https://github.com/irisSchaffer/
[2] https://www.npmjs.com/
[3] http://sass-lang.com/

ming paradigm using *RxJS* to allow for a simpler interface for event-driven operations. The communication between client and server is handled by *socket.io*[4]. This section tries to introduce the reader to the main technologies used to establish a base on which the following technical implementation details can be understood.

5.2.1 ECMAScript2015 and Babel[5]

JavaScript undoubtly is an integral part of front end web development and since the emergence of server-side JavaScript with Node.js[6] and its package manager npm has developed into a programming language widely used by web developers [**gpm-meta-transcompiler**]. Both PYPL[7] and TIOBE[8] programming language indices rank JavaScript among the top 10 programming languages (PYPL at 5, TIOBE at 7 at the time of writing) [**gpm-meta-transcompiler**]. 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[9] all three rank amongst the top 5 languages developers expressed an interest in developing with [**stackoverflow-developer-survey**].

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 [**ecma2015**], *Kangax' ECMAScript compatibility tables*[10] 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 [**es6**]. It also makes JavaScript easier to understand and safer to develop with the introduction of block-scoped variables (`let` and `const`) and finally offers native support of modules and promises [**es6**]. 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* EC-

[4] http://socket.io/
[5] https://babeljs.io/
[6] https://nodejs.org/en/
[7] http://pypl.github.io/PYPL.html
[8] http://www.tiobe.com/tiobe_index
[9] https://angularjs.org/
[10] https://kangax.github.io/compat-table/es6/

MAScript 2015 code to ECMAScript 5, which is exactly what Babel does. With almost 750.000 downloads in April 2016 [**npm-babel**] and companies like Facebook, Netflix, Mozilla, Yahoo or PayPal using this transpiler [**babel-users**], Babel is the de facto standard solution to transpile to ECMAScript 5 and was also chosen for this project.

5.2.2 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 [**reactive-programming-survey**]. The same is true for the server where the possibility to concurrently serve a multitude of different clients is crucial. In these cases JavaScript offers *event listeners* – functions called once a certain event happens. However, these event listeners or *asynchronous callback* [**reactive-programming-survey**], oftentimes execute more asynchronous code and in turn have to wait for another event, and another one, and another one..., which can result in something known and dreaded by most any JavaScript developer: *Callback Hell* (see programm 5.1).

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 [**reactive-vs-promises**]. 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 queued using the `then` keyword, to execute asynchronous code in a certain sequence (see programm 5.2).

However, promises can still create nested callbacks, especially when chaining promises that rely on other promises' resolution [**reactive-vs-promises**]. This is where *reactive programming* comes in: 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 this value are re-computed upon arrival of such a new value. To demonstrate this I would like to use Bainomugisha et al.'s illustrative example of a simple addition [**reactive-programming-survey**]:

```
1 let v1 = 1;
2 let v2 = 2;
3 let v3 = v1 + v2;
```

In sequentially executed code, `v3` will hold the value 3, no matter if or how `v1` or `v2` change. In reactive programming, however, `v3` will be re-computed as soon as either one of the values it depends on changes [**reactive-programming-survey**]. This way for a drag-and-drop feature, for example, the move of the mouse, continuously sending its location, could directly alter the position of an element in a page. JavaScript does not directly support reactive programming,

Program 5.1: *Callback Hell* – Nested callbacks in JavaScript. Simplified method taken from a previous project, which authenticates a user, creates a new google calendar for them and then saves the user to one's own database, to then redirect them. `{...}` is used to shorten the code, error-handling was also omitted in the example for simplicity.

```
router.get('/callback', function(req, res, next) {
  var code = req.query.code;
  var name = JSON.parse(req.query.state);

  // get token from oauth library
  oauth2Client.getToken(code, function (err, tokens) {
    // load configuration
    Configuration.findOne({}, function (err, configuration) {
      var calendar = google.calendar('v3');
      // save to google calendar
      calendar.calendars.insert({...}, function (err, cal) {
        var member = new Member({...});
        // save member to own database
        member.save(function(err, member) {
          return res.redirect(getRedirectionUrl(name) + '&success=true')
        });
      });
    });
  });
});
```

Program 5.2: *Promises* – Simple example of chaining ECMAScript 2015 promises with `then` and `catch`.

```
var promise = new Promise(function(resolve, reject) {
  asyncCall(function(error, data) {
    if (error) {
      reject(error); // reject the pending promise
    } else {
      resolve(data); // resolve the pending promise
    }
  })
})

promise
  .then(function(data) {
    // this is executed after asyncCall returns
    // other asyncronous calls can be placed here
  })
  .catch(function(error) {
    // this is executed if an error occurs somwehere along the way
  })
```

Program 5.3: *RxJS* – simplified example of the touch controls used to swipe to the next or previous slide. An Observable is created from the browser's `touchmove` event and is then transformed with `map` and `filter`, to in the end call the `navigate()` method with the direction the user swiped into.

```
1  this.moveObservable = Observable.fromEvent(document, 'touchmove')
2    // data: event object with array of touches
3    .filter(this.touchStarted) // only procede if touchstart is set to true
4    .map(this.toXY) // transform initial event data to latest touch's xy position
5    // data: {x: xPosition, y: yPosition}
6    .map(this.toDirection) // transform xy position to direction literal
7    // data: right/left/up/down
8    .do(this.resetTouchStart) // set this.touchStarted to false
9    // data: right/left/up/down
10   .subscribe(this.navigate) // call this.navigate() with direction data
```

but other, more functional languages, which can be transpiled to JavaScript, do. Another way of adding reactive programming concepts to JavaScript is using a library, such as *Bacon.js*[11] or the one chosen for this project, *ReactiveX*[12]. 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 simple creation of event streams (*Observables*) from browser events or promises directly and uses the same method names JavaScript developers are familiar with from array-methods, 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. These methods can be chained to sequentially alter a value (see programm 5.3).

Additionally to Observables, RxJs also knows *Subjects*, which combine both a source of events and a consumer of such. Subjects are Observables, but also Observers at the same time and can be used to broadcast values to several consumers [**rxjs-docu**].

5.2.3 React[13]

As this project concentrates on the front end, a mature JavaScript framework was searched for. After previous experience with the big and complex but slow AngularJS, and because of promising performance benchmarks [**react-benchmarks**] and simply to explore new JavaScript libraries, I decided to give React a try. Since Facebook started developing React in 2013, it has challenged existing approaches and set new standards in front end web

[11] https://baconjs.github.io/
[12] http://reactivex.io/
[13] https://facebook.github.io/react/index.html

development [**introduction-to-react**]. 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 by far in performance benchmarks [**react-benchmarks**]. Moreover, *React Native*[14] would make it possible to port the application to different mobile operation systems fairly easily.

The arguably most important method these re-usable, lightweight components implement is the `render` method, defining what HTML or JSX[15] should be rendered by the browser:

```
1 export default class HelloWorld extends Component {
2   render() {
3     return (<h1>Hello World!</h1>)
4   }
5 }
```

The created Component can then be rendered into the virtual React DOM, JSX makes it possible to simply use the name of the component to create it:

```
1 ReactDOM.render(
2   <HelloWorld />,
3   document.getElementsByTagName('body')[0]
4 )
```

This would simply put an `h1` element with the text `Hello World!` into the `body` element of the HTML page. Additionally to the `render` method, components also have a *state* and *properties* (`props`), through which they can communicate with other components and maintain their internal state. `props` are passed in to the component during creation, in JSX this can be achieved by simply passing them in as XML attributes:

```
1   <HelloWorld greeting="Hi" />
```

This could in turn be used in the `HelloWorld` component's `render` method:

```
1   render() {
2     return (<h1>{this.props.greeting} World!</h1>);
3   }
```

The children of a component are also available through `this.props.children`. The `state` variable, on the other hand, is responsible for handling internal updates e.g. through user interaction [**react-docu**]. To alter the state, the method `setState` can be used, which will cause an update and re-rendering of the component. So if in the above example, the word "World" should be changed to something else by the user, a text field with an event handler can be added inside the component (see program 5.4). Now, whenever a

[14] https://facebook.github.io/react-native/
[15] https://facebook.github.io/jsx/

Program 5.4: Example code snippet using state in a React component. Whenever the text input changes (i.e. a user types something), the `update()` method will be called and set the state and cause a re-render.

```
1 // set default state and props...
2
3 componentWillMount() {
4   // create observable from change event on input
5   this.observable = Observable.fromEvent('change', this.refs.input)
6     .pluck('target', 'value') // extract input text
7     .subscribe(this.update) // call update with value
8 }
9
10 componentWillUnmount() { this.observable.unsubscribe() }
11
12 update(text) { this.setState({name: text}) }
13
14 render = () => (
15   <div>
16     <h1>{this.props.greeting} {this.state.name}!</h1>
17     <input value={this.state.name} ref="input" />
18   </div>
19 )
```

user changes the text in the `input` field, the RxJS Observable will receive a new value (a change event). The `value` of the field is extracted on line 6 of program 5.4 and used to then update the state in the `update` method, which is implicitly called with the value passed through the Observable chain.

The two methods `componentWillMount`, `componentWillUnmount` as well as `componentDidMount`, `shouldComponentUpdate`, `componentWillUpdate`, `componentDidUpdate` and `componentWillReceiveProps` are *lifecycle methods*, called whenever the component is created, updated or destroyed.

As an end note on React, and a transition to the core presentation library, I want to add that React components can be nested, which is at the core of the project developed for the present thesis. To make it as easy as possible for other developers to use the created libraries, a presentation is built just as a usual HTML page, using JSX to reference the custom React components, as will be shown later this chapter in program 5.9.

5.2.4 unveil.js[16]

As a presentation layer, this project uses the open-source JavaScript library *unveil.js*, which was developed by Leandro Ostera and myself in the beginning of the project and which I extended and adapted to my needs in an

[16] https://github.com/ostera/unveil.js

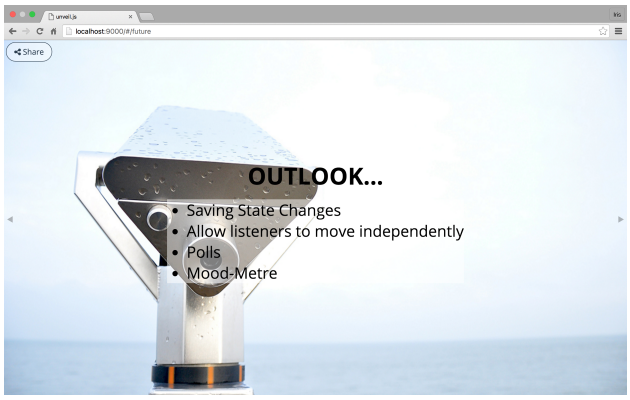


Figure 5.1: Screenshot of an example slide with unveil.js, using interactive extensions discussed in section 5.6.

own fork[17] during the project. This fork will be covered in section 5.4 of this chapter, until then, a short overview of the different parts of unveil.js is given and key concepts of the library are discussed. Screenshots of what unveil.js looks like can be found in figure 5.1. Generally, unveil.js operates on a 2-dimensional slide-space: Every slide can have a next and previous slide in x, as well as in y-direction. To generate the y axis, slides can be nested in other slides. These slides have an optional unique name as well as an index in the slide-tree, which they are identified by. As shown in program 5.9, slides are created inside the component `UnveilApp`, the core of unveil.js. This component configures and sets up the entire application based on optional configuration passed in as properties. There are a few concepts unveil.js is built around to allow for extensibility and configurability, namely *presenters*, *controls* and *modes*:

- **Presenters** define the way slides are rendered, e.g. show notes, upcoming slides or hide them.
- **Controls** control a part of the application, e.g. navigating from one slide to the next using the arrow keys on the keyboard.
- **Modes** are what allows a speaker to have a different presenter and controls 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 new modes, presenters and controls and thereby extend the base library as they wish. A few of these are already defined, 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

[17] https://github.com/irisSchaffer/unveil.js

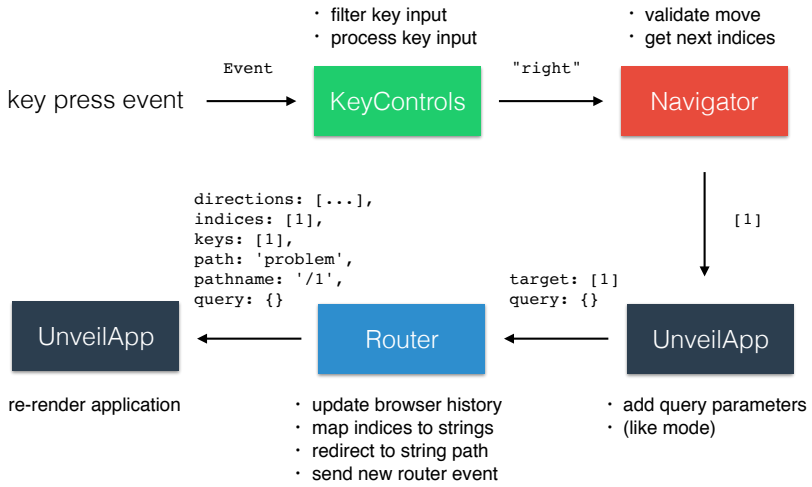


Figure 5.2: Navigation pipeline from user's key press to re-render of the presentation. The monospaced text next to the arrows symbolises the data transmitted. `KeyControls` listen for key events and process them, to then send a navigation request to go *right* to the `Navigator`. This component then 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.

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 users 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 5.2.

5.3 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, which are all available on GitHub. 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 5.2.4, the project is based on the library unveil.js. During the development of the project, certain parts of the base library did not offer the flexibility needed for easy extensibility and so several parts were adapted and new presentation logic was added. This happened in a fork of the original library, which will be examined in section 5.4.

Network Synchronisation Layer: The first library of direct importance for the interaction between speaker and audience through personal devices is *unveil-network-sync*[18]. 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 5.5.

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. The voting mechanism, as well as the creation of new votings on-the-fly, also live within this library. The repository, called *unveil-interactive*[19] relies on unveil-network-sync for the socket-interaction. The interactive extension will be covered in section 5.6 of this chapter.

Server and Example Presentation: The last repository connected to this thesis is *unveil-client-server*[20], which includes a simple server as well as a real-world example of a presentation, which was used in the intermediate thesis project presentation as part of the Interactive Media course IM690, on the 2nd of February, 2016. In this chapter, a whole section was dedicated to the server (5.7), as well as to the example application (5.8), to separate

[18] https://github.com/irisSchaffer/unveil-network-sync
[19] https://github.com/irisSchaffer/unveil-interactive
[20] https://github.com/irisSchaffer/unveil-client-server

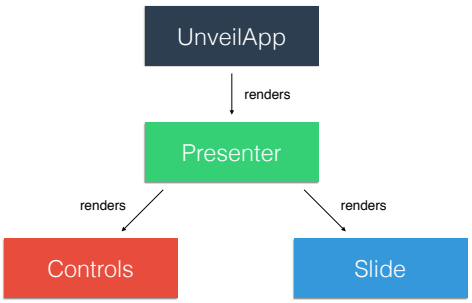


Figure 5.3: Overview over the render-flow of the application in the extended version of unveil.js. `UnveilApp` renders the presenter, which then takes care of rendering the (current) slide and all controls.

concerns a bit more clearly and be able to conclude with a demonstration of how all parts discussed earlier play together in a final presentation.

5.4 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 component 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 5.3), 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 5.6. 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

```
1  <Slide name="start" left={[0]}>
2    ...
3  </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).

5.5 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 [**socketio-problems**], 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:

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

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:

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

Program 5.5: 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.

```
1  // imports...
2
3  export default class NavigationReceiver extends React.Component {
4    static contextTypes = {
5      navigator:   React.PropTypes.object.isRequired,
6      routerState: React.PropTypes.object.isRequired
7    }
8
9    componentDidMount () {
10     this.observable = Observable.fromEvent(SocketIO, 'state:change')
11       .filter((e) => !this.context.routerState.indices.equals(e))
12       .subscribe(this.props.navigator.next)
13   }
14   ...
15 }
```

These sockets can then be used to listen to events or to emit them (see program 5.5) 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.

5.6 Interactive Extension

The interactive library includes several parts which will be discussed here: a speaker presenter (section 5.6.1) as well as different components connected to sharing media (section 5.6.2) and voting (section 5.6.3). Additionally

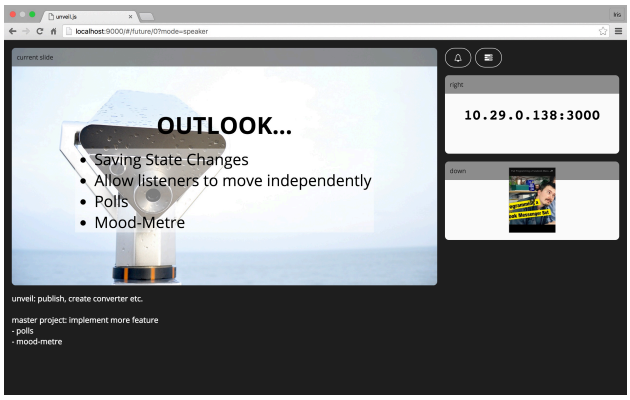


Figure 5.4: Screenshot of the speaker presenter with the current main slide, the upcoming slide to the right, available actions (muting and adding votings) and speaker notes.

controls handling the `state:initial` event were implemented to redirect new listeners to the current slide in the presentation.

5.6.1 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 5.4). 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 5.5 (b)). This ensures that everything is big enough to be readable and all buttons are clickable.

5.6.2 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 5.6 (a)), the acceptor is enabled in speaker mode, to accept or reject incoming media (see figure 5.6 (b)) and the receiver in the end handles the creation of a new slide if the content was accepted and 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

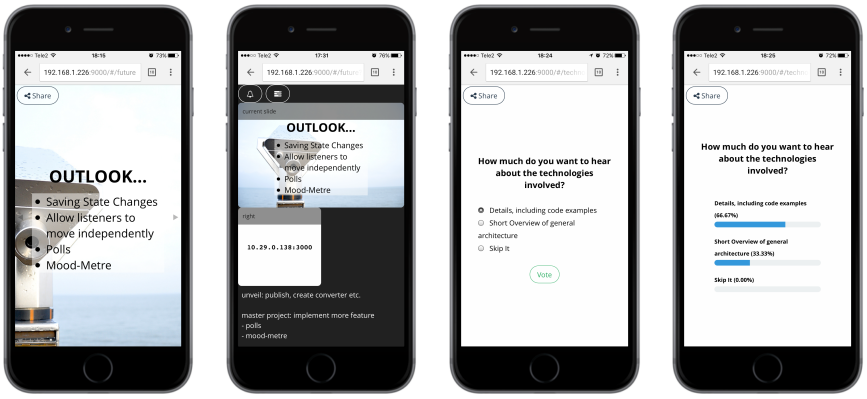


Figure 5.5: Mobile view of a presentation slide in (a) default mode and (b) speaker mode as well as (c) voting view before voting and (d) voting view after voting. Mind the share button in the left upper corner in (a) as well as the buttons to mute content requests and create votings in (b).

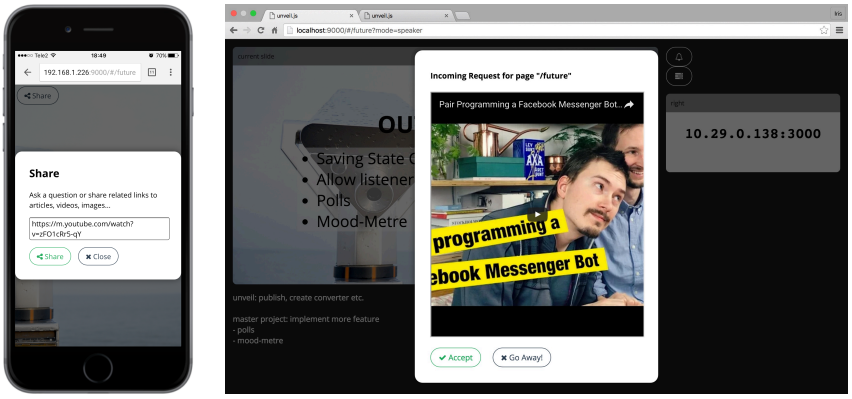
end of the presentation. Generally, this feature can be used to either post a link to an interesting picture, website or even youtube video, or also as text input, for example to add a comment or a question regarding a certain slide. This works through the introduction of the presentation components `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:

```
1 isImg (str) {
2   let imgRegex = new RegExp(/\.(jpe?g|png|gif|bmp)$/i)
3   return imgRegex.test(str)
4 }
```

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 5.5 (a)) as well as the share modal opened when clicking on said button (see figure 5.6 (a)). The same happens in the `MediaAcceptor`, which uses the `render()` method to display the *mute* button shown in figure 5.4 as well as the modal for accepting media (see figure 5.6 (b)).

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



(a) (b)

Figure 5.6: Screenshots of sharing content. (a) media sender on mobile phone shares youtube link (b) speaker receives content request.

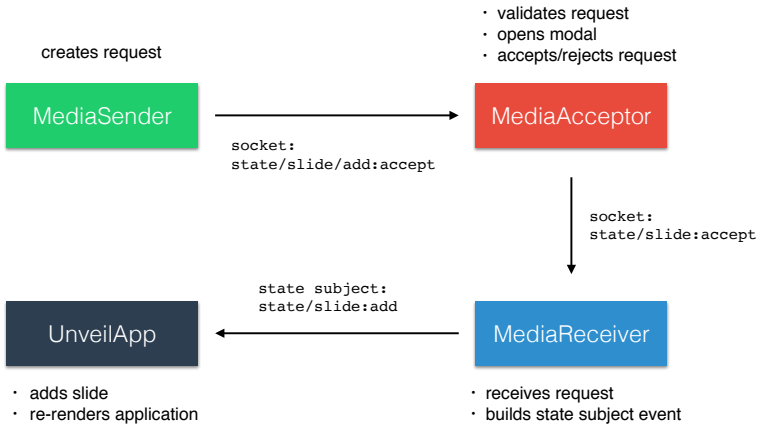


Figure 5.7: Flow of adding content, monospaced text symbolises type and name of events. First the `MediaSender` of the default mode sends a request, which the speaker mode's `MediaAcceptor` listens to. If the request is accepted by the speaker or requests are muted, another socket event is broadcast, 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.

`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 5.7.

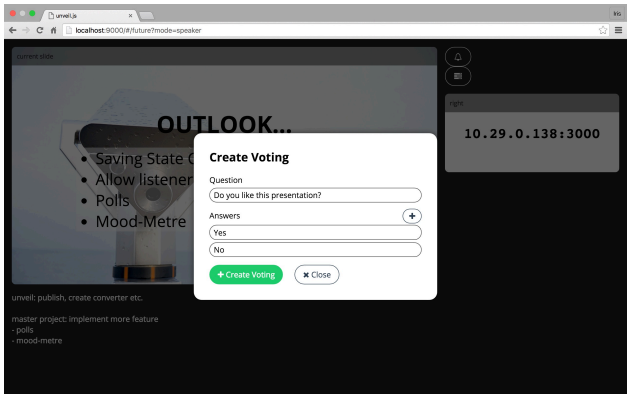


Figure 5.8: Screenshot of the desktop speaker mode, creating a new voting, on-the-fly, during a presentation.

Program 5.6: Example code for preparing a slide with voting.

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

5.6.3 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 5.8) and members of the audience to vote (see figure 5.5 (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 5.6). `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 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 remem-

bers 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.

5.7 Server

As shortly outlined before, the focus of this project was never the server, but instead the client. For this reason the server was kept as lightweight and *dumb* as possible, only handling the most important communication. In reality, it acts as a proxy and broadcaster between the clients and serves the JavaScript and HTML files. This has the advantage that other developers can use their own servers and technology stacks without relying on the architecture chosen for the example presentation. Literally the only requirement for the server is to support Web Sockets. However, the disadvantage is that currently all state changes are only persisted on the client-side. This is good for testing, as a reset is only a page-reload and clearing of local storage away, but means audience members joining the presentation after any additional slides were added, will not have the same state of the presentation.

The server developed for the project presentation of the Interactive Media course IM690 runs on Node.js, using Express[21] as a framework. To again emphasise how low the requirements for such a server are, a working example implementation can be found in program 5.7. Additionally to this code, the server in the unveil-client-server repository also includes a `lastState` variable, in which the last navigation state is stored. Every time a new client connects, this state is then emitted with a `state:initial` event. Moreover, socket.io does not currently support wildcards in events, which is why it was necessary to add the code snippet discussed in [**socket-io-wildcards**].

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 ex-

[21] http://expressjs.com/

Program 5.7: Very simple possible implementation of a server running the thesis project with Node.js and Express. [**socket-io-wildcards**] 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!');
});
```

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: {...}
};
```

plained 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'));
```

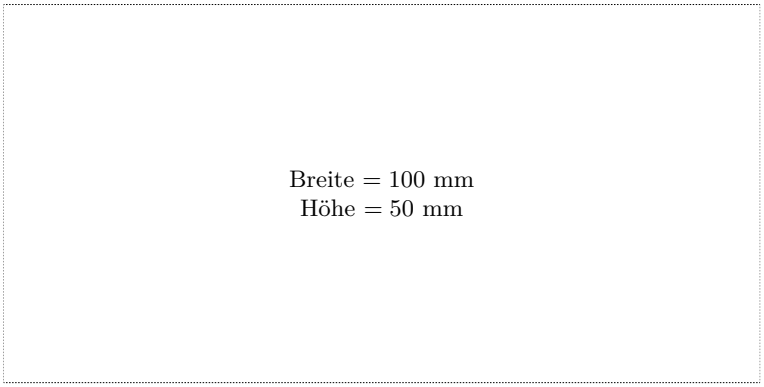
Program 5.10: Example styling for slides using Sass. In this particular piece of code, the font family of all slides is set and a background image is added to the start slide.

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

References

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —