Konzeption und Entwicklung eines webbasierten Audience Response Systems

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Moritz Beller

2048551

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Erstprüfer: Prof. Dr. Andreas Plaß

Zweitprüfer: Dipl.-Ing. Thorsten Wagener

Hamburg, 07. 04. 2016

Inhaltsverzeichnis

Abstract

This essay is about the conception and development of an audience response system based on web-technology which is deployed as software as a service (SaaS) application. The intention of this service is to improve the communicational possibilities between the teacher and students and to minimize effort for technical setup and staff training. At the very end the application will be evaluated in a field test to prove success or failure of the project.

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Konzeption und Entwicklung eines Softwaresystems, welches im Rahmen von Lehrveranstaltungen nach dem Software as a Service Modell eingesetzt werden kann. Die Absicht besteht darin, die Kommunikations- und Interaktionsmöglichkeiten zwischen dem Lehrenden und Studenten im Klassenraum zu verbessern und die dadurch entstehenden Aufwände möglichst gering zu halten. Nach der Umsetzung wird die Anwendung durch einen Praxistest evaluiert.

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Moritz Beller

1 Einleitung

1.1 Problemaufriss

Das digitale Zeitalter und insbesondere die rasante Entwicklung des Internets bergen enormes Potential für den Menschen und seine Umwelt. Alles ist vernetzt und wird in Echtzeit erfasst. Es gibt smarte Kühlschränke, an das Internet angeschlossene Toaster und intelligente Kochtöpfe, die unser Leben in gewisser Weise verbessern oder erleichtern. Die digitale Revolution dringt nach und nach auch in den Bildungsbereich ein. Digitale Medien werden vielfach als Ergänzung oder als Alternative zu konventionellen Medien eingesetzt. Doch das digitale Lernen, seine Möglichkeiten und Grenzen, werden noch erforscht. Allein der Einsatz modernster Technologie wird das Lehren oder Lernen nicht per se besser oder schlechter machen, gewiss aber verändern diese Technologien das Arbeiten an Schulen und Universitäten. Mittlerweile sind E-Learning Systeme wie Moodle, Mahara oder ILIAS an deutschen Hochschulen sehr weit verbreitet, es gibt sogar Onlineplattformen die tausende Kurse auf Universitätsniveau anbieten. Dennoch wird im Präsenzunterricht meist noch analog gelehrt und die E-Learning Plattformen dienen lediglich als Dokumentenablage.

Internetfähige Geräte wie Smartphones und Tablets sind in jedem Hörsaal zahlreich zu finden, sogar 88% der Jugendlichen zwischen 12 und 19 Jahren (JIM-Studie 2014) besitzen ein eigenes, internetfähgies Smartphone. Moderne Webtechnologien werden heute genutzt um Anwendungen zu erstellen, welche sich ohne Aufwand auf diversen Geräten aufrufen lassen, egal ob Smartphone, Tablet oder Desktop Computer.

1.2 Zielsetzung

Gerade weil so viele Geräte in den Bildungseinrichtungen mitgeführt werden, könnten diese Kapazitäten in Verbindung mit jenen jungen Technologien genutzt werden um den Hochschulalltag zu erleichtern. Ziel dieser Bachelorarbeit ist es, eine Anwendung zu entwerfen und umzusetzen die ein modernes Audience Response System darstellt. ARS haben einen weitgefassten Anwendungsbereich, in dieser Arbeit wird das Augenmerk allein auf den Einsatz im Bildungsbereich gerichtet.

1.3 Aufbau der Arbeit

Diese Arbeit beschäftigt sich zuerst mit den Hintergründen von Audience Response Systemen (Kapitel 2) und untersucht ob und unter welchen Voraussetzungen sie eine Bereicherung für den Unterricht darstellen können. Darauffolgend werden die weiteren Schritte, die zum erfolgreichen Einsatz der Anwendung notwendig sind, dargelegt:

1. In einer Anforderungsanalyse (Kapitel 3) sollen zunächst die erforderlichen Merkmale an das Audience-Response-System ermittelt werden, damit es im Unterricht einen möglichst hohen Nutzen für Lehrende und Studierende hat.

2. Daraufhin werden die technische Grundlagen erklärt, welche den anschließenden Ausarbeitungen zugrunde liegen (Kapitel 4).

3. Es soll ein Anwendungskonzept (Kapitel 5) erstellt werden, welches die Möglichkeiten der Umsetzung sowie die Konzeption des Designs beschreibt.

4. Auf Basis dieses Konzepts soll die Anwendung implementiert werden. Die Erfahrungen aus der Umsetzung sollen dem Leser ein konkretes Bild über die Anwendung und den Entwicklungsprozess an sich geben (Kapitel 6).

5. Weiterhin soll die Anwendung im Hochschulbetrieb getestet und mithilfe von Nutzerfeedback evaluiert werden (Kapitel 7).

6. Diese Arbeit schließt mit einem Fazit (Kapitel 8). Hier sollen die Ergebnisse der Arbeit nocheinmal abschließend zusammengefasst werden und zur Bedeutung der Anwendung im Bezug auf die Lehre Stellung genommen werden.

2 Allgemeine Grundlagen zu Audience Response Systemen

Dieses Kapitel geht auf den Begriff des ARS ein und thematisiert die Geschichte und Entwicklung dieser Systeme. Abschließend wird auf die Vorteile sowie die Voraussetzungen, die bei der Nutzung von ARS in einer Bildungseinrichtung entstehen, eingegangen.

2.1 Begriff

Ein Audience Response System (ARS) ist ein System, dass die Rückmeldungen von einer Gruppe von Menschen entgegennimmt, auswertet und bereitstellt. So ermöglicht ein ARS dem Publikum - ähnlich wie bei der TV-Sendung „Wer wird Millionär?“ - an einem Abstimmungsverfahren des Referenten teilzunehmen. Dazu werden klassischerweise sogenannte „Clicker“ oder webbasierte Lösungen, die mit Hilfe von mobilen Endgeräten wie Smartphones oder Tablets arbeiten, verwendet. Das ARS wertet daraufhin die Antworten der Teilnehmer aus und stellt sie dem Referenten zur Verfügung.

2.2 Geschichte und Entwicklung

Die Geschichte von ARS reicht bis in die 1950er Jahre zurück als das Personal der US Air Force Geräte benutzte um Multiple-Choice Fragen, die in Schulungsfilme integriert waren, zu beantworten (Judson & Sawada 2006). In den 1960er Jahren wurden erstmals ARS für den öffentlichen Bildungsbereich gebaut und in Hörsäle der Stanford University und Cornell University installiert (Abrahamson 2006). Die frühen ARS waren fest verdrahtete elektronische Systeme (Voting machine 1973), die ein Strommessgerät als Abstimmungsanzeige nutzten. Allerdings wurden ARS erst in den späten 1990er Jahren, nachdem die Auswirkungen von ARS auf die schulischen Leistungen untersucht und das Konzept des Handlungsorientierten Unterrichts in den USA durch eine Studie (Bonwell & Eison 1991) popularisiert wurde, akzeptiert.

Üblicherweise werden bei einem klassischen Hardwaresystem Fernbedienungen im Publikum verteilt. Diese kommunizieren über Funk mit einem Empfänger der mit dem Computer des Referenten verbunden ist. Über eine ARS Software kann der Referent die Antworten auswerten und ggf. grafisch darstellen. Je nach Ausstattung des

Systems können die Fernbedienungen mit einfachen Ja/Nein Antwortmöglichkeiten, numerischen Tasten oder einer kleinen Tastatur bestückt sein.

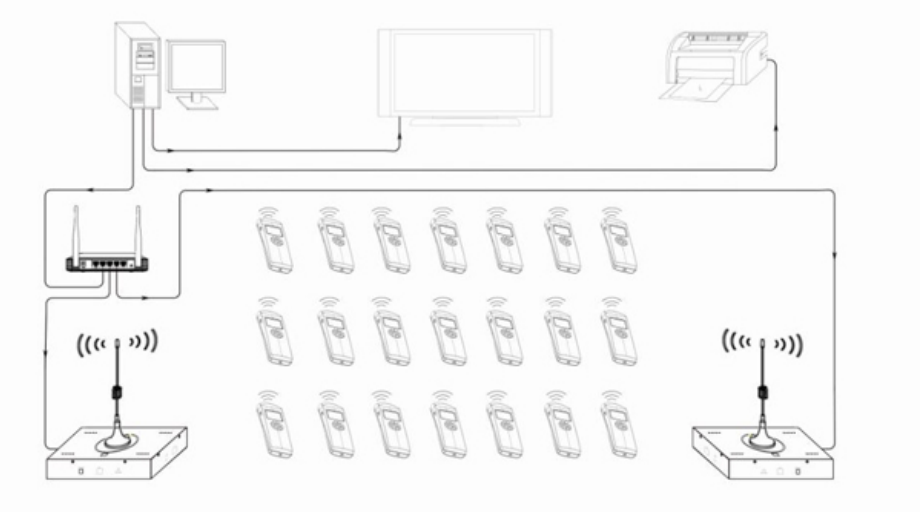

Abbildung 2.1: Wireless Voting System von Guangzhou Suntron Electronics Co., Ltd.

Softwarebasierte Lösungen bieten den Vorteil, dass diese die bereits vorhandene Hardware (Smartphone, Tablet, Laptop) nutzen und sich dadurch die Anschaffungskosten - zumindest für die Hardware - erübrigen. Softwarebasierte Lösungen nutzen weiterhin die bereits bestehende Infrastruktur von WiFi-Anbindungen bzw. die weitreichende Flächenabdeckung des Mobilen Internets. Softwarelösungen werden meist als Webanwendung oder native App des jeweiligen Betriebsystems angeboten. Letzteres hat den Nachteil dass die App im Vorfeld über einen App-Store verteilt und von den Benutzern heruntergeladen, installiert und eingerichtet werden muss.

Heutzutage gibt es eine breite Palette von Software- und Hardwarelösungen, jüngst werden auch Soziale Netzwerken als ARS auf Konferenzen genutzt[1].

2.3 Vorteile

Diese Audience Response Systeme können vor allem im Bildungsbereich den Präsenzunterricht in vielen Aspekten bereichern:

1. **Verbesserte Aufmerksamkeit**: Es ist bekannt, dass die Aufmerksamkeitsspanne von Menschen rund 20 Minuten beträgt (Dukette & Cornish 2009). In Anbetracht der Tatsache, dass die typische Dauer einer Vorlesung 90 Minuten beträgt, lässt es sich nicht vermeiden dass Studierende nicht alles aus dieser Zeit behalten können. Eine Möglichkeit um die Aufmerksamkeit der Studierenden zurückzuholen wäre zum Beispiel über ein ARS alle 20 Minuten Fragen

[1] https://confwall.com/, zuletzt abgerufen am 30.03.2016

zu stellen. Der Erfolg dieser Methode wurde von vielen Studien bestätigt: Studenten sind aufmerksamer wenn ein ARS während der Vorlesung genutzt wird (Bergtrom 2006, Caldwell 2007, Draper & Brown 2004).

2. **Anonymität**: Studierende können auf ARS Fragen antworten ohne von ihren Mitstudierenden, den Tutoren oder dem Professor verurteilt zu werden. Anonymität ermöglicht es jedem Studierenden ein aktives Mitglied in der Vorlesung zu sein und erlaubt so eine zwanglose Beteiligung am Lernprozess (Banks 2006).

3. **Erhöhte Teilnahme**: Studien belegen dass Studenten ein höheres interesse oder eine größere Bereitschaft zur Teilnahme zeigen wenn ein ARS in der Vorlesung eingesetzt wird (Bergtrom 2006, Simpson & Oliver 2007, Vand Dijk, Van Den Berg & Van Keulen 2001).

4. **Interaktiver Unterricht**: Durch ein ARS wird ein aktives Lernen bei den Studierenden begünstigt. Was für für mehr Diskussionen und Wissensaustausch untereinander sorgt(Bergtrom 2006, Caldwell 2007). Falls das Feedback der Studierenden Unklarheiten oder Missverständnisse aufzeigt, kann der Lehrende darauf eingehen. Dies ändert maßgeblich das Wesen einer Vorlesung vom statischen Frontalunterricht hin zum dynamischen und interaktiven Lernerlebnis.

5. **Unmittelbares Feedback**: In einer herkömmlichen Vorlesung kann Feedback zum Beispiel durch Handzeichen oder Aufrufen von Freiwilligen erfolgen. Jedoch haben diese Methoden bedeutende Nachteile: Studierende neigen dazu Antworten von anderen zu kopieren oder haben nicht die Selbstsicherheit die Hand zu heben. Das erschwert es ein korrektes Empfinden über das Sachverständis der Gruppe zu erlangen. Darüber hinaus sind die Daten verloren wenn die Hände wieder sinken. Ein ARS verlangt von den Studierenden, dass sie sich selbst mit der Frage auseinandersetzen während die Anonymität gewahrt wird. Außerdem hilft ein ARS den Feedbackprozess schnell und effektiv durchzuführen und die Antworten automatisiert zusammenzufassen.

2.4 Voraussetzungen für einen effektiven Einsatz

Neben diesen vielseitigen Vorzügen der ARS bestehen jedoch auch einige Schwierigkeiten bei der Verwendung eines ARS, welche dem Einsatz entgegen stehen können.

Technische Hürden Das größte Problem sind die technischen Hürden die bei der Wahl und Konfiguration eines ARS auftreten können. Vor der Nutzung muss möglicherweise Soft- und/oder Hardware installiert werden und die Lehrenden sowie Studierenden müssen im Umgang mit einem ARS geschult werden. Weiterhin müssen die Geräte regelmäßig gewartet, repariert oder instandgesetzt werden. Deswegen soll der Aufwand für die Einrichtung, Wartung und Schulung eines ARS so gering wie möglich ausfallen um Reibungsverluste zu vermeiden.

Budget Hardwarebasierte ARS sind kostspielig und können schwer in Großveranstaltungen eingesetzt werden, da die Anzahl der „Clicker“begrenzt ist, sodass nicht jeder Studierende Zugriff auf einen „Clicker“hat. Deswegen werden Softwarebasierte Lösungen immer beliebter, vor allem weil diese auf die bereits vorhandenen Ressourcen der Studierenden zurückgreifen. Diese müssen nicht von der Bildungseinrichtung erstanden oder gewartet werden und ermöglichen so einen kostengünstigen Einsatz in großen Hörsälen.

Richtige Fragestellung Einen guten Fragenkatalog zu entwickeln ist eine anspruchsvolle Aufgabe und kann sehr zeitintensiv sein. Mitunter können falsche Fragestellungen zu falschen Schlüssen führen und den Erfolg der Vorlesung gefährden.

3 Anforderungsanalyse

Ziel dieses Kapitels ist es die Anforderungen an das System zu ermitteln und zu prüfen. Auf Basis der Grundlagen aus dem vorherigen Kapitel sowie den dort herausgearbeiteten Vorteilen und Schwierigkeiten der ARS können Anforderungen abgeleitet werden, die die Eigenschaften des Systems und die Interaktion des Benutzers mit dem System beschreiben. Es ist eine Vielzahl von Nutzungsszenarien von ARS im Hochschulumfeld denkbar, jedoch wird in diesem Kapitel der Fokus auf die Nutzung in der Vorlesung beschränkt. Eine Analyse von bereits bestehenden Systemen schließt dieses Kapitel ab.

3.1 Anforderungen an die Anwendung

In einem ersten Schritt sollen nun also Anforderungen an die Anwendung definiert werden.

3.1.1 Hauptziel

Das System soll einem Vortragenden die Möglichkeit geben seine Hörer ohne großen Aufwand in den Vortrag miteinzubeziehen.

3.1.2 Anwendungsfall Dozent

Der Dozent startet die Anwendung und erzeugt mit einem Klick eine neue Session, welche als „virtueller Klassenraum“ in dem die beiden Parteien anonym miteinander kommunizieren können fungiert. In diesem neu geschaffenen Raum kann der Dozent nun Fragen erstellen. Diesen Schritt kann der Dozent bereits vor der eigentlichen Vorlesung gehen um Fragen vorzubereiten. Folgende Fragetypen sind vorgesehen:

- **Wahr/Falsch**: Eine Frage kann nur mit Wahr oder Falsch beantwortet werden
- **Multiple choice**: Der Dozent gibt die Antwortmöglichkeiten vor, davon kann der Student eine auswählen.
- **Kurze Antwort**: Der Hörer selbst kann eine Antwort in ein Freitextfeld eingeben.

- **Gruppenorganisation**: Der Hörer gibt seinen Namen in ein Textfeld ein. Die Mechanik ist identisch mit der Kurzantwort, lediglich in der Auswertung unterscheiden sich die Fragetypen.

Der Dozent soll mit diesen Fragetypen Meinungsumfragen und Wissensabfragen durchführen können um die Bereitschaft am aktiven Lernen zu fördern. Weiterhin sollen diese Fragetypen die Organisation der Gruppe vereinfachen und die Diskussionsbereitschaft der Hörer durch kreative Anteilnahme verbessern.

Die Auswertung der Antworten hängt vom Fragetyp ab: Bei **Multiple Choice** und **Wahr/Falsch** soll das Ergebnis der Umfrage einerseits als Diagramm dargestellt werden, damit der Dozent einen Überblick über das Verhältnis der abgegebenen Antworten bekommt. Andererseits soll in einer Legende die absolute Anzahl an abgegebenen Stimmen pro Antwort dargestellt werden. Die Mechanik der Fragetypen **Kurzantwort** und **Gruppenorganisation** ist eine andere, da die Antworten vom Nutzer selbst eingegeben werden. Hierbei ist die Anzahl der unterschiedlichen Antworten nicht begrenzt, somit erscheint eine Darstellung als Diagramm wenig sinnvoll. Der Dozent soll die Möglichkeit haben die Antworten einfach per Drag-and-Drop zu gruppieren, um so beispielsweise ein Brainstorming durchzuführen. Bei der Gruppenorganisation soll der Dozent zusätzlich den Namen der Gruppe an den Antwortenden zurückgeben.

3.1.3 Anwendungsfall Hörer

Während der Vorlesung verteilt der Dozent eine bestimmte URL an seine Hörer. Wenn diese die URL in ihrem Webbrowser aufrufen betreten sie den virtuellen Klassenraum des Dozenten und können sofort die Fragen des Dozenten sehen und beantworten. Der Hörer sieht eine Liste an Fragen des Dozenten (falls dieser welche vorbereitet hat) und kann selbst bestimmen welche Frage er zuerst beantwortet. Wurde jedoch eine Frage einmal beantwortet, so kann der Hörer diese nicht wieder ändern. Auch wenn der Hörer den Browser schließt und später wieder öffnet soll er nicht erneut abstimmen können.

3.1.4 Funktionale Anforderungen

Allgemein

- Es muss zwei verschiedene Ansichten geben: Eine für den Fragesteller („Dozent“) und eine für den der die Frage beantworten soll („Hörer“).
- Es muss verschiedene „Räume“ geben in denen Fragen gestellt werden können, diese Räume sollen völlig unabhängig voneinander sein. So sollen mehrere Vorlesungen gleichzeitig die Anwendung nutzen können.

- Folgende verschiedene Fragetypen müssen unterstützt werden:
  - Multiple Choice
  - Wahr/Falsch
  - Eigene Antwort eingeben
  - Gruppenorganisation
- Die Gesamtanzahl der Nutzer soll für jeden einsehbar sein.
- Die Anwendung muss ohne vorherige Anmeldung nutzbar sein.
- Der Zustand des Systems (Fragen, Antworten) darf nicht verloren gehen wenn ein Benutzer die Seite verlässt und sich erneut verbindet.

Dozent

- Nur der Vortragende darf Fragen hinzufügen oder löschen.
- Der Dozent soll die abgegebenen Antworten der Hörer in Gruppen einteilen können.
- Der Vortragende muss die Möglichkeit haben, Fragen bereits vor der Vorlesung vorzubereiten.
- Nur der Vortragende sieht die Anzahl der Nutzer, die eine Frage beantwortet haben.
- Lediglich der Dozent hat eine Übersicht über die abgegebenen Antworten.

Hörer

- Nur der Student darf Fragen beantworten
- Ein Student darf jede Frage nur einmal beantworten.

3.1.5 Nichtfunktionale Anforderungen

- Die Anonymität aller Anwender muss gewahrt werden.
- Die Anwendung soll auf der Mehrheit der gängigen Geräten lauffähig sein und außerdem ohne Vorkenntnisse bedienbar sein.
- Die Anwendung soll auf Geräten mit unterschiedlichsten Bildschirmdimensionen optisch ansprechend dargestellt werden.
- Die Anwendung soll erweiterbar sein.

- Die Übertragung der Daten erfolgt korrekt und ohne große Verzögerung.
- Die Anwendung soll robust sein, sie soll nicht abstürzen falls der Benutzer falsche oder unvorhergesehene Eingaben tätigt

3.2 Vergleich mit bereits bestehenden Systemen

Im nächsten Schritt soll nun beispielhaft vorgestellt werden welche ARS Anwendungen bereits bestehen und welche Möglichkeiten diese mit sich bringen.

3.2.1 eduVote

EduVote[1] ist ein komerziell vertriebenes Audience Response System. Es wurde von der Firma „SimpleSoft“entwickelt, hinter der sich zwei Absolventen der TU Braunschweig verbergen. Die Software wird an einigen deutschsprachigen Hochschulen eingesetzt[2], eine Einzellizenz kostet pro Jahr 299 € bzw. 2800 €[3] für eine gesamte Institution. Das Produkt bildet ein klassisches ARS – ähnlich wie bei „Wer wird Millionär?“der Publikumsjoker – ab. Der Dozent muss allerdings vor der Verwendung eine Lizenz kaufen und sich mit seiner E-Mail und dem Lizenzschlüssel registrieren. Ist dies getan kann er damit beginnen die Fragen für seinen Vortrag vorzubereiten. Er hat nur die Möglichkeit Multiple-Choice Fragen zu erstellen, lediglich die Anzahl der Antwortmöglichkeiten ist variabel, nicht jedoch der Antworttext. Auch die konkrete Fragestellung ist in der Anwendung nicht ersichtlich. Der Dozent startet die Umfrage manuell während des Vortrags und beendet sie auch wieder manuell, daraufhin werden die Ergebnisse als Balkendiagramm dargestellt. Die Studenten können an der Umfrage teilnehmen indem Sie die E-Mail Adresse des Dozenten eingeben. Jeder Student darf nur einmal an jeder Umfrage teilnehmen.

Für Studenten gibt es eduVote als Mobile-App für Android, iOS, Windows Phone 8 und BlackBerry, als Desktop-App für Windows sowie Mac OS und schließlich als Browseranwendung für alle Geräte. Dozenten haben die Wahl zwischen einer Desktop-App für Windows oder Mac OS, oder aber einer Browseranwendung. EduVote bietet ein klassisches ARS mit modernen Technologien, es bietet Apps für alle gängigen Platformen an. Allerdings sind sowohl die Fragetypen als auch die Antwortmöglichkeiten stark eingeschränkt.

[1] http://www.eduvote.de/, zuletzt abgerufen am 15.03.2016
[2] Siehe Referenzen auf der Website http://www.eduvote.de/referenzen.html, zuletzt abgerufen am 15.03.2016
[3] Siehe Lizenzmodelle http://www.eduvote.de/lizenz.html, zuletzt abgerufen am 02.04.2016

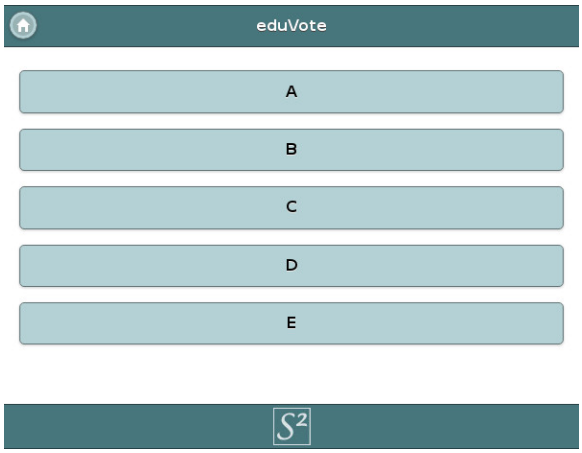

Abbildung 3.1: eduVote: Ansicht für einen Studenten – lediglich die Anzahl der Antwortmöglichkeiten unterscheidet sich

3.2.2 ARSnova

ARSnova ist eine Open Source Projekt der Technische Hochschule Mittelhessen (THM) und bietet nicht nur Lösungen für Hochschulen, sondern auch für Schulen und Seminare. In drei Jahren Forschungs- und Entwicklungsarbeit haben sich einige Features und Use-cases angesammelt:

- möglicher Login über Google+, Facebook, ARSnova oder den Uni-Server der THM
- Import und Export von Sessions
- Verschiedenste Fragetypen: Multiple Choice, Ja/Nein, Freitext, Benotung, Lernkarte, Bildmarkierung
- Vorschaufunktion für Fragen
- Enthaltungsmöglichkeit
- Möglichkeit um Lösungshinweise zu hinterlegen
- Möglichkeit, Fragen aus dem Publikum entgegen zu nehmen

ARSnova wurde als webbasierter Dienst entwickelt und kann auf allen Geräten über einen Browser aufgerufen werden. Sowohl die Bedienung als auch die Gestaltung der Web-App ist für Mobilgeräte mit kleinen Bildschirmen optimiert. Allerdings kann der von Geräten mit größeren Bildschirmen zusätzlich verfügbare Platz nicht genutzt werden, da lediglich ein Drittel des Bildschirms von der Anwendung genutzt wird. ARSnova steht als Open Source Projekt auch anderen Institutionen zur

Verfügung und ist unter der GPLv3[4] Lizenz freigegeben. Falls eine Institution sich nicht selbst um die Installation, Integration und Wartung kümmern möchte bietet die TransMIT GmbH eine Ferninstallation und Einbindung in die bestehende Hochschulinfrastruktur für 3900 €[5] zzgl. MwSt. an. Für die Bedienung der Anwendung gibt es eine umfangreiche Dokumentation, Handbücher und Lernvideos. Weiterhin bietet ARSnova die Desktop-App „ARSnova Overlay“an, die für für Windows und Mac OS verfügbar ist. „ARSnova Overlay“ bietet einen Anzeigenmonitor der sich auf dem Bildschirm des Dozenten über die Präsentationsfolien legt. Diese App stellt das Live Feedback der Studenten in einem Balken, Icon oder als Smiley dar. Der Dozent kann so sehen, ob die Studierenden seinen Ausführungen folgen können. Eine weitere Anwendung für Dozenten ist der „ARSnova Presenter“, ein weiteres webbasiertes Frontend für ARSnova, welches den Fokus auf die Vorbereitung und Präsentation von ARSnova Sessions legt. Mit dieser Anwendung kann ein Dozent die Statistiken seiner Sessions einsehen, er findet dort aufbereitete Graphen und Diagramme die zu Präsentationszwecken dienen. ARSnova Presenter bietet für Dozenten den gleichen Funktionsumfang wie das Mobile-Frontend, wobei es für die Anwendung auf großen Bildschirmen optimiert ist.

Zusammengefasst bietet ARSnova ein modernes ARS und erweitert das Angebot durch zusätzliche Funktionen und Anwendungen, die den Alltag einer Hochschule bereichern sollen. ARSnova legt den Fokus auf den Dialog zwischen Lehrenden und Studierenden, nicht nur an Hochschulen – sondern auch für kleine Seminare oder Schulen.

[4] GNU General Public License, eine Open Source Software-Lizenz. Ein Überblick ist hier zu finden: http://www.gnu.org/licenses/quick-guide-gplv3.en.html, zuletzt abgerufen am 02.04.2016

[5] Quelle: https://arsnova.thm.de/blog/technischer-support/, zuletzt abgerufen am 02.04.2016

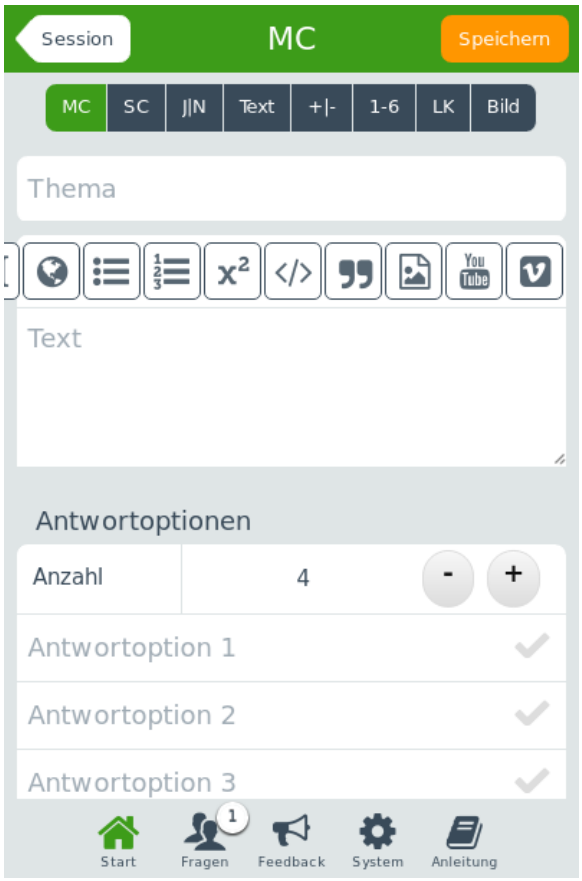

Abbildung 3.2: Erstellen einer Frage in ARSnova – Ansicht für einen Dozenten

3.2.3 Socrative

Socrative ist ein anmeldepflichtiger Webdienst zur Beurteilung von Schülern und Studenten. Die Zielgruppe ist laut eigenen Angaben der Primäre und Sekundäre Bildungsbereich, also in etwa die ersten zwölf Schuljahre. Mit diesem Dienst können die Antworten von Studenten in Echtzeit zusammengebracht, analysiert und dargestellt werden. Lehrer haben die Möglichkeit Fragen oder Quiz anzulegen, zusätzlich können Sie mit dem Aufgabentyp „Space Race“ Studenten spielerisch dazu auffordern Probleme zu lösen. Socrative besticht durch sein elegantes Design und einen sehr einfachen Einstieg in die Software: Der Benutzer wird mit einem Demo-Video und einer Einladung zu einem kostenlosen Account begrüßt und es finden sich viele Trainingsangebote und ein eigenständiges Hilfe-Center auf der Website. Socrative bietet eigenständige Apps für Studenten und Lehrer, jeweils für iOS, Android, Chrome OS und Windows. In der Webanwendung können verschiedene Berichte exportiert werden: Für die gesamte Klasse, für jeden Studenten einzeln oder nach Fragen sortiert.

Hinter Socrative steckt die Firma Mastery Connect, die sich auf formale Beurteilung von Studenten und die Entwicklung von Lernanwendungen im Bildungsbereich

spezialisiert hat. Mastery Connect hat laut CrunchBase[6] seit 2009 rund 30 Millionen US-Dollar von Investoren aufgebracht, zuletzt 5 Millionen US-Dollar im September 2015 von Mark Zuckerberg und Priscilla Chan.

Für einen kostenlosen Dienst macht Socrative einen wirklich sehr guten Eindruck, gerade wegen des ansprechenden Designs und der hohen Einsteiger- und Benutzerfreundlichkeit. Die Anmeldepflicht gilt nur für Dozenten und macht daher auch Sinn, da diese hier die vergangenen Aktivitäten einsehen und auswerten können.

Abbildung 3.3: Aufgabentyp Space Race in der Anwendung Socrative

[6]Quelle: https://www.crunchbase.com/organization/masteryconnect, zuletzt abgerufen am 18.03.2016

4 Technische Grundlagen

In diesem Kapitel werden die technischen Grundlagen für die Entwicklung einer Webanwendung erläutert. Es wird hierbei ein besonderes Augenmerk auf die Echtzeitfähigkeit des HTTP Protokolls gerichtet. Es gilt darauf hinzuweisen, dass die zu behandelnden technischen Grundlagen einer stetigen Weiterentwicklung unterliegen, daher sind Abweichungen zum wirklichen Entwicklungsstand nicht auszuschließen.

4.1 HTTP

Das Hypertext Transport Protocol (HTTP) ist ein sehr weit verbreitetes Protokoll der Anwendungsschicht und dient der Datenübertragung zwischen einem Client (z.B. Webbrowser, Smartphone App) und einem Server. HTTP baut direkt auf dem in der Transportschicht liegenden Transmission Control Protocol (TCP) auf.

Die Kommunikation läuft nach dem Anfrage - Antwort Prinzip ab: Der Client (Anwender bzw. Webbrowser) schickt eine HTTP-Nachricht an den Server und dieser Antwortet wiederum mit einer HTTP-Nachricht. Letztere kann nun ein HTML Dokument, eine Video- oder Audiodatei oder ein einfaches Textdokument sein. Jede HTTP Nachricht besteht aus einem Nachrichtenkopf („Header“) und einem Nachrichtenrumpf („Body“). Der Header enthält Informationen über die Beschaffenheit des Bodies, beispielsweise über die verwendete Kodierung oder Kompression, die Länge der Nachricht oder ähnliches. Der Body enthält hingegen die eigentliche Nachricht.

Das HTTP Protokoll ist zustandslos. Das bedeutet, dass jede einzelne Anfrage unabhängig von anderen Anfragen vom Server verarbeitet wird, dabei ist die Reihenfolge in der die Anfragen geschickt werden irrelevant, desweiteren gehen Informationen aus früheren Anfragen verloren.
Damit der Client nicht die Zustandsinformationen mit jeder Nachricht mitsenden muss, wie zum Beispiel Login-Informationen des Benutzers oder Produkte im Warenkorb, kann ein Sitzungsbezeichner (engl. session id) mit jeder Nachricht mitgeführt werden. Das ist effizienter, wenngleich nicht sicherer, da die Informationen in Klartext übertragen werden. Ergänzend zu HTTP wurde HTTPS zur verschlüsselten Datenübertragung standardisiert, welches zusätzlich das Verschlüsselungsprotokoll TLS einsetzt.

Früher wurde für jede Anfrage eine neue TCP-Verbindung aufgebaut und nach der vollständigen Übertragung der Antwort wurde diese TCP-Verbindung wieder

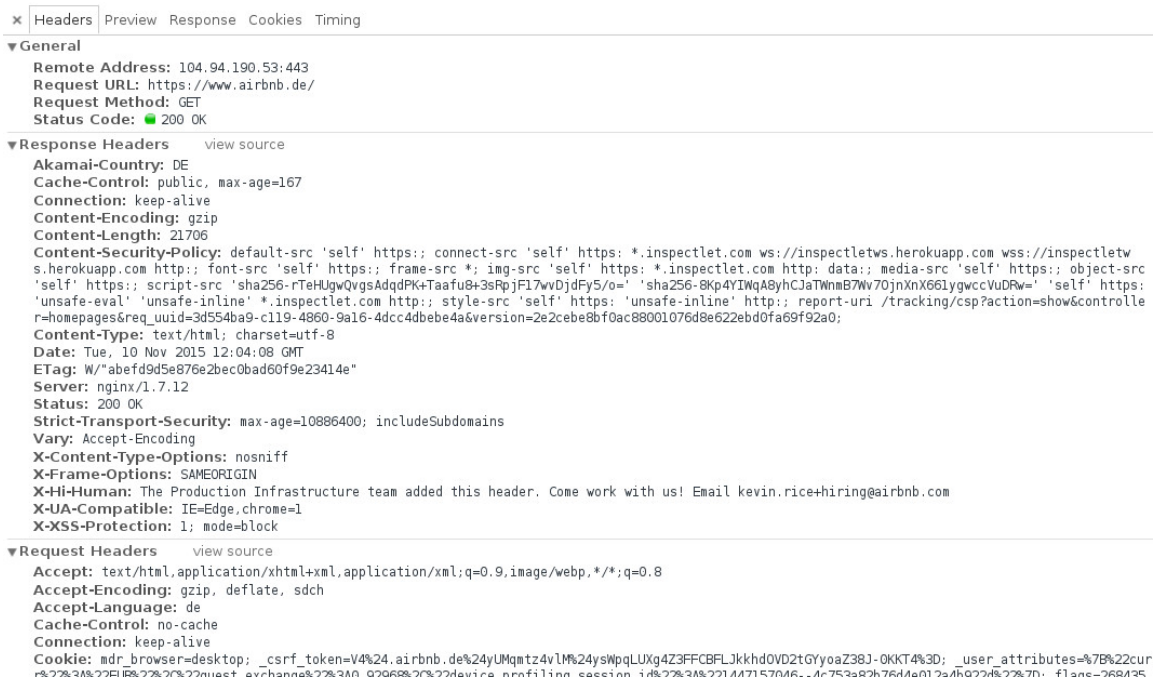

Abbildung 4.1: HTTP Anfrage- und Antwortheader

abgebaut[1]. Folglich musste für jedes Bild, das in einem HTML-Dokument enthalten war eine neue TCP-Verbindung aufgebaut und wieder geschlossen werden. Mit dem 1999 eingeführten Standard HTTP/1.1 (RFC 2616 1999) kann nun der Client mit dem zusätzlichen Headerfeld „Keepalive“ den Server bitten die TCP-Verbindung zwischen den Anfragen aufrecht zu erhalten, womit sich der Mehraufwand für den Verbindungsauf- und abbau erübrigt. Weiterhin können Clients mit Version 1.1 durch Pipelining mehrere Nachrichten über eine einzelne TCP-Verbindung nach dem First-In First-Out (FIFO) Verfahren versenden und empfangen. Aufgrund des FIFO Verfahrens kann es allerdings zum sogenannten Head-of-Line-Blocking (HOL-Blocking) kommen, dabei staut sich die Abarbeitung der darauffolgenden Anfragen auf, während die erste Anfrage noch bearbeitet wird.
Während der Entwicklungsphase der Anwendung wurde eine neue Spezifikation zu HTTP Version 2 (HTTP/2) von der Internet Engineering Task Force (IETF) verabschiedet[2]. Diese erweitert die Protokollsemantik von HTTP 1.1 und ist Abwärtskompatibel. HTTP/2 ermöglicht das Multiplexing von mehreren Nachrichten über eine TCP-Verbindung. Dadurch wird das HOL-Blocking eliminiert, da die Antworten nicht nach dem FIFO-Verfahren zurückgesandt werden müssen. Weiterhin wird dem Server die Möglichkeit gegeben auf eine Anfrage hin präventiv mehrere Ant-

[1]Das galt für HTTP/1.0, siehe RFC 1945, https://tools.ietf.org/html/rfc1945, Abschnitt 1.3 „Overall Operation“, zuletzt Abgerufen am 10.03.2016
[2]Mai 2015, RFC7540 https://tools.ietf.org/html/rfc7540, zuletzt Abgerufen am 12.03.2016

worten zu schicken („*server push*“). Dies ist nützlich, wenn der Server weiß welche Anfragen der Client als nächstes schicken wird. So kann der Server mit dem initial angefragten HTML-Dokument gleich alle darin enthaltenen Bilder und weitere benötigte Ressourcen, wie CSS-Stylesheets und Javascript Dateien, mitsenden. Die *server push* Technologie ersetzt damit das Einbetten von Ressourcen in ein HTML-Dokument („ressource inlining“). Dies war vor HTTP/2 weit verbreitet um den Performanzengpass, der durch eine große Anzahl an HTTP Anfragen entsteht, zu vermeiden.*Server Push* ermöglicht es jedoch **nicht** Nachrichten unabhängig von einer Anfrage zu schicken!

4.2 Echtzeit mit HTTP

Wir nutzen heute Webanwendungen um in Echtzeit mit unseren Freunden zu chatten und wenn wir auf der Facebooktapete nach unten scrollen erwarten wir, dass Inhalte nachgeladen werden. Aber welche Technologie liegt hier Zugrunde, die diese bidirektionale Kommunikation ermöglicht?

HTTP wurde ursprünglich entworfen um Nachrichten zwischen zwei Parteien auszutauschen: der Client stellt eine Anfrage und der Server antwortet, somit muss für die Aktualisierung der Information die Ressource erneut angefordert werden. Aufgrund dieser Architektur lassen sich keine Nachrichten vom Server zum Client schicken ohne, dass der Client vorher eine Anfrage geschickt hat. Zudem werden bei einer erneuten Anfrage redundante Informationen in Form von HTTP-Headern mitgeschickt, wodurch sich längere Wartezeiten ergeben, vor allem wenn sich die Antwort des Servers verzögert oder gar ausbleibt.

Es scheint also keine angemessene oder effiziente Möglichkeit zu sein Echtzeitinformationen, wie Verkehrslage oder Börsenwerte, mittels HTTP zu synchronisieren. Um dieses Problem zu umschiffen bedienten sich Entwickler einst Übergangslösungen, die in den nächsten Unterkapiteln erläutert werden.

4.2.1 AJAX

Aus dieser Problemstellung heraus hat sich AJAX („Asynchronous Javascript and XML“) entwickelt. Dies ermöglicht ein partielles, asynchrones Nachladen von Teilinformationen. Genauer gesagt beschreibt AJAX ein Denkmuster und ein Konglomerat aus Technologien, die die Architektur einer Webanwendung beeinflussen.
Die Technologien, die AJAX zugrunde liegen existierten in ihrer ersten Form bereits 1998. Der Begriff AJAX wurde 2005 von Jesse James Garrett durch seinem Aufsatz „Ajax: A New Approach to Web Applications“[3] maßgeblich geprägt. Heute wird der Begriff AJAX Synonym für XMLHttpRequest verwendet. Garrett beschreibt AJAX

[3] https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php, zuletzt Abgerufen am 12.03.2016

wie folgt:

> „Every user action that normally would generate an HTTP request takes the form of a JavaScript call to the Ajax engine instead. Any response to a user action that doesn't require a trip back to the server - such as simple data validation [...] - the engine handles on its own. If the engine needs something from the server in order to respond [...] the engine makes those requests asynchronously, usually using XML [...] without stalling a user's interaction with the application."

Es werden also Teile der Anwendungsslogik vom Server in den Client ausgelagert bzw. dupliziert um die Benutzererfahrung erheblich zu verbessern, wie beispielsweise das Validieren von Benutzerdaten in einem Formular, bevor sie zum Server versendet werden. Der Versand selbst kann, bzw. laut der obigen Beschreibung MUSS, ebenfalls asynchron erfolgen.

Es gibt mehrere Möglichkeiten um die Echtzeitanforderung mithilfe von asynchronen HTTP-Anfragen abzubilden.

Polling

Wenn ein Client in regelmäßigen Intervallen Nachrichten an den Server schickt, spricht man von polling (engl. etwa „automatischer Abruf von Nachrichten"). Diese Technik kann entweder als harter Seitenreload implementiert werden oder in Form einer asynchronen Anfrage (AJAX) ohne, dass das Dokument erneut vollständige herunterladen werden muss. Bedingt durch die regelmäßige Zeitspanne zwischen zwei Anfragen kann es sein, dass entweder die Information stark verspätet eintrifft, oder dass serverseitig eine unnötige Last erzeugt wird. Das macht polling sehr ineffizient. Ist jedoch das Intervall, in dem Informationen aktualisiert werden, regelmäßig, kann hiermit eine geschickte und effiziente Lösung implementiert werden.

Long Polling

Von Long Polling spricht man, wenn der Server die Anfrage des Clients nicht sofort beantwortet, sondern erst dann wenn eine neue Information verfügbar ist. Daraufhin sendet der Client eine erneute Anfrage an den Server und das Spiel beginnt von vorn.

4.3 WebSocket

Das WebSocket Protokoll ermöglicht eine vollduplex Kommunikation zwischen zwei Teilnehmern. „WebSocket" bezeichnet eine Client API für den Browser[4] und ein Netz-

[4] standardisiert vom W3C https://html.spec.whatwg.org/multipage/comms.html#network, zuletzt abgerufen am 14.03.2016

werkprotokoll[5]. Das WebSocket Protokoll ist ein von HTTP unabhängiges, auf TCP basierendes Protokoll, dessen Einsatzgebiet hauptsächlich in der bi-direktionalen Kommunikation zwischen einem Browser und einem Server liegt. Das Protokoll besteht im Grunde aus zwei Teilen: dem Handshake, der genutzt wird um die Parameter der Verbindung zu bestimmen und dem Datentransfer, der eine effiziente, nachrichtenbasierte Kommunikation für Text und binäre Daten ermöglicht. Das Websocket Protokoll kann auch ausserhalb des Browsers genutzt werden, es gibt Implementierungen für viele Sprachen und Plattformen.

4.3.1 WebSocket Handshake

Das Websocket Protokoll bietet eine Vielzahl an Features, doch bevor überhaupt eine Nachricht ausgetauscht werden kann, müssen die Verbindungsparameter bestimmt werden. Das ist die Aufgabe des WebSocket Handshake, er hat bewusst eine Ähnlichkeit mit einer HTTP Nachricht. Dadurch kann das Protokoll die bestehende HTTP Infrastruktur in Form von Servern, Proxies, Firewalls und ebenfalls die „UPGRADE"-Mechanik[6] von HTTP 1.1 nutzen. Das Protokoll verwendet ebenfalls die gleichen TCP Ports wie HTTP: 80 für reguläre Verbindungen und 443 für WebSocket Verbindungen die über TLS getunnelt werden.

Der WebSocket Handshake ist nichts anderes als eine HTTP Upgrade Anfrage mit eigenen WebSocket Headern, die die Parameter der WebSocket Verbindung aushandeln:

- **Sec-WebSocket-Version**: Wird vom Client gesendet, er definiert die Version des WebSocket Protokolls. (z.B. „13" für RFC6455)
- **Sec-WebSocket-Key**: Wird vom Client gesendet, er zeigt dem Server dass es sich um einen WebSocket Handshake handelt. Dies soll sicherstellen, dass der Server keine WebSocket Verbindungen zu HTTP-Clients aufbaut.
- S**ec-WebSocket-Accept**: Wird vom Server gesendet, er bestätigt hiermit den WebSocket Handshake.
- **Sec-WebSocket-Protocol**: Wird zwischen beiden Parteien ausgetauscht um sich über ein Subprotokoll auf Nachrichtenebene einig zu werden. *(optional)*
- **Sec-WebSocket-Extensions**: Wird zwischen beiden Parteien ausgetauscht um sich über zu verwendende Erweiterungen einig zu werden. *(optional)*

Der vollständige WebSocket Handshake des Clients:

[5] standardisiert durch die IETF https://tools.ietf.org/html/rfc6455, zuletzt abgerufen am 14.03.2016

[6] https://en.wikipedia.org/wiki/HTTP/1.1_Upgrade_header

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: d2hpdGVkZXNrIG5vbmNl==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 8
```

Listing 4.1: WebSocket Handshake Request

Der Server antwortet im Erfolgsfall mit dem „Switching Protocols“ Status und schließt damit den Handshake ab.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: a34DKS2G7H6KdfgGzzhZxOo=
Sec-WebSocket-Protocol: chat
```

Listing 4.2: WebSocket Handshake Response

4.3.2 Websocket Datentransfer

Nach dem erfolgreichen Handshake können beide Endpunkte unabhängig voneinander Daten senden. Es werden sogenannte „Data Frames“ übertragen, jedoch definiert das WebSocket Protokoll nicht den Inhalt der Nachrichten. Ein einzelnes bit definiert lediglich die Codierung der Nachricht: Entweder ein UTF-8 String oder eine binäre Nachricht. Dadurch wird sichergestellt dass die Nachrichten der beiden Parteien effizient decodiert werden können.

Im Unterschied zu HTTP enthalten die WebSocket Data Frames keine Metadaten, deshalb ist es unter Umständen notwendig, dass die beiden Kommunikationspartner ihr eigenes Subprotokoll implementieren oder bekannte Protokolle nutzen[7].

4.4 Node.js

Node.js ist eine plattformübergreifende Javascript Laufzeitumgebung für den Betrieb von Netzwerkanwendungen und wurde 2009 von Ryan Dahl erschaffen. Heute ist es ein Gemeinschaftsprojekt mit einer konstant wachsenden Community. Node.js (kurz: Node) nutzt unter der Haube Googles Javascript Laufzeit „V8“. Für Entwickler bedeutet das, dass sie die selbe Technologie für Clientseitige Anwendungen benutzen können wie für Serverseitige. Im Moment ist Javascript die einzige Sprache mit dieser Art der Wiederverwendbarkeit und schließt die Lücke zwischen Browser, Server und sogar Mobilen Anwendungen. Die ereignisorientierte Architektur von Node ermöglicht

[7] WebSocket Subprotocol Name Registry https://www.iana.org/assignments/websocket/websocket.xml

asynchrone I/O Operationen. Aufgrund dieser Ausgestaltung lassen sich Netzwerkanwendungen hinsichtlich ihres Datendurchsatzes und ihrer Skalierbarkeit optimieren. Node.js wurde entwickelt, weil die Abbildung von nebenläufigen Prozessen in serverseitigen Sprachen schwer fällt und oft zu schlechter Performanz führt. Node arbeitet in einem einzelnen Thread, so können mit Hilfe von asynchronen I/O Operationen mehrere Zehntausend Verbindungen gleichzeitige bedient werden, ohne den Mehraufwand von Kontextwechseln, welche üblicherweise bei der Verwendung von mehreren Threads notwendig sind. Allerdings hat die Verwendung eines einzelnen Threads auch Nachteile: Dieser darf nämlich niemals blockiert werden, sonst werden damit auch alle Netzwerkverbindungen und sonstigen Berechnungen blockiert. Asynchrone I/O Operationen werden in Node mit Callbacks, also Rückruf-Funktionen abgebildet. Diese werden genau dann ausgeführt, wenn eine Operation abgeschlossen ist. Wenn man nun mehrere Operationen sequenziell ausführen möchte, muss man jene Callbacks verschachteln, was zu unleserlichem Code führen kann.

5 Konzeption & Design

Dieses Kapitel geht auf die Planung der Softwarearchitektur und das Design der Anwendung basierend auf den in den letzten Kapiteln erarbeiteten Anforderungen ein.

5.1 Softwarearchitektur

Die globale Systemarchitektur lässt sich als Client-Server Architektur, wie in Abbildung 5.1 modelliert, beschreiben. Dieses Entwurfsmuster ist das Standardkonzept für Dienste, die im World Wide Web genutzt werden. In diesem Fall wird die Architektur benötigt um die Kommunikation verschiedener Geräte zu koordinieren. Der Client ist eine Anwendung, die sich einerseits um die Präsentation der Daten und andererseits um den Datenaustausch mit dem Server kümmert, indem beide über Internetprotokolle miteinander kommunizieren. Der Server wiederum stellt Schnittstellen für den Datenaustausch bereit und implementiert die notwendige Geschäftslogik.

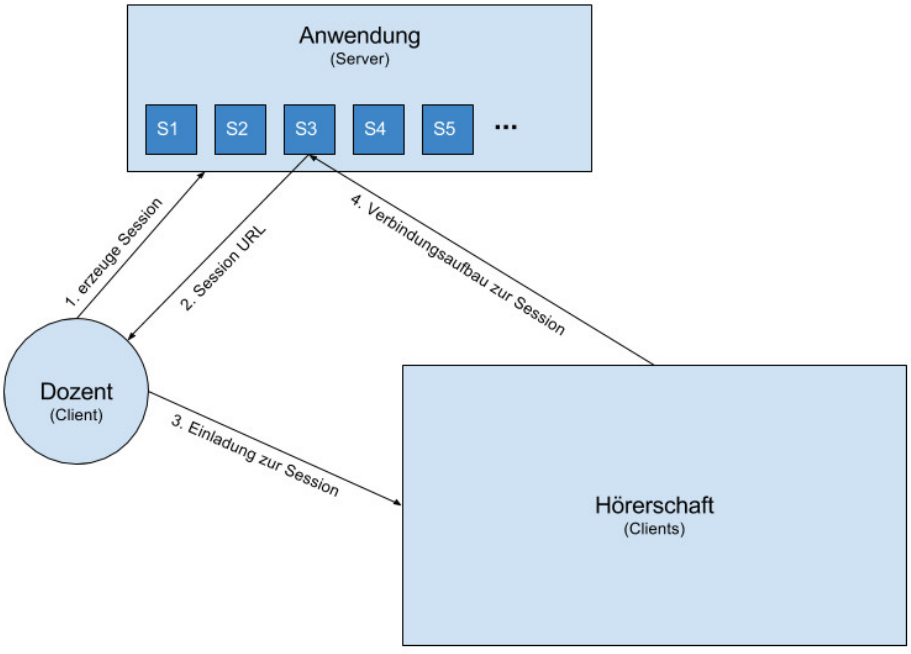

Abbildung 5.1: Erzeugung und Teilnahme an einer Sitzung

5.1.1 Client

Die Clientseite dieses Systems soll durch einen Webbrowser abgebildet werden. Dadurch wird sichergestellt, dass die Anwendung auf einer Vielzahl von Endgeräten ohne aufwendige Installation lauffähig sein wird. Denkbar wäre die Entwicklung von nativen Applikationen, jeweils eine für jede zu unterstützende Plattform. Durch native Applilationen hat man einerseits Zugriff auf bestimmten Hardwarekomponenten, wie zum Beispiel Kamera, Gyroskop und GPS, andererseits können bestimmte plattformspezifische Dienste (z.B. Apple Push Notification Service, Google Cloud Messaging, Google Identity Plattform) nur über native APIs angesprochen werden. Ganzheitlich betrachtet bieten native Applikationen eine weitaus bessere Performance als Webapplikationen. Jedoch muss für jede native Applikation eine neue Anwendung geschrieben werden, da jede Plattform eigene Anforderungen an Sprache und Entwicklungswerkzeuge mit sich bringt. Aufgrund dieses Mehraufwandes lohnt sich hier die Entwicklung einer nativen Applikation nicht. Alternativ könnte auch eine sogenannte Hybridapplikation erstellt werden, welche sich dadurch auszeichnet, dass ein von nativem Code initiierter Browser eine Webanwendung ausführt. Je nach Implementierung liegen hier die Vor- und Nachteile eher bei deren einer Webanwendung, jedoch könnten zusätzlich Hardwarekomponenten und plattformspezifische Dienste wie bei einer nativen Applikation angesprochen werden.

Das Ablaufschema vom Erzeugen einer Sitzung zwischen dem Dozenten, Server und der Hörerschaft wird ebenfalls in Abbildung 5.1 (Seite 27) dargestellt. Die Kommunikation zwischen Client und Server während einer Sitzung soll ausschließlich über WebSockets abgebildet werden. Websocketverbindungen eignen sich besonders gut, dadurch können Daten in Echtzeit ausgetauscht werden – ohne dass die Seite neu geladen werden muss. Aufgrund dieser Eigenschaft empfiehlt sich die Entwicklung einer Single-page-Webanwendung. Das Echtzeitkriterum setzt voraus, dass die eintreffenden Daten vom Client verarbeitet werden und dort in angemessener Weise dargestellt werden.

In den Abbildungen 5.2 und 5.3 (Seite 29 / 30) werden die Anwendungsszenarien aus Sicht eines Dozenten, bzw. Studenten beschrieben. Es gilt zu beachten, dass die meisten Aktivitäten von der Session Seite ausgehen. Diese Seite wird eine Single-page-Webanwendung darstellen. Die Willkommen-Seite wird hingegen ein einfaches HTML-Dokument sein.

5.1.2 Server

Die Serverseite des Systems kann auf einem dedicated Server[1] im Hochschulnetzwerk installiert werden. Jedoch empfiehlt es sich die Serveranwendung im World Wide Web nach dem Software as a Service (SaaS) Modell bereitzustellen. Auf diese Weise wird sichergestellt, dass die Benutzer auch ohne das Hochschulnetzwerk den Dienst

[1] Ein eigens für die Anwendung eingerichteter Server, der nicht nebenbei für andere Aufgaben verwendet wird.

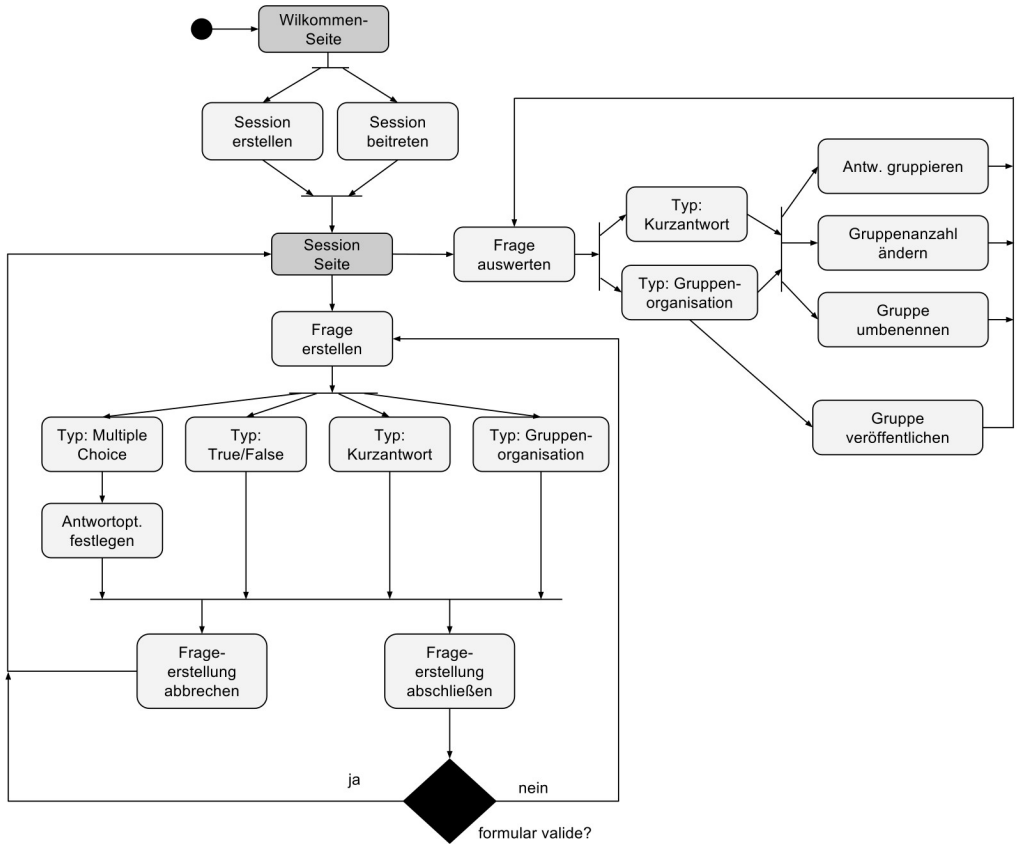

Abbildung 5.2: Aktivitätsdiagramm für die Sitzung eines Dozenten

nutzen können. Außerdem sinken so auch die technischen Hürden bezüglich der Installation Installation und Wartung des Systems. Hinsichtlich des Servers stehen einige Technologien zur Wahl. Wünschenswert ist hierfür eine Plattform welche über eine ereignisgesteuerten Architektur verfügt, da diese einerseits wesentlich performanter mit vielen gleichzeitigen Verbindungen umgehen können und andererseits diese Architektur zu der ebenfalls ereignisorientierten WebSockets API passt. Für die Serverseite der Applikation eignet sich Node.js besonders gut, da es diesen Anforderungen entspricht und zudem das Backend, wie auch das Frontend, in Javascript programmiert werden kann. Weiterhin sollen die Schnittstellen so gestaltet sein, dass sie unabhängig von einem Browser funktionieren sollten. So verbaut man sich nicht die Option später doch einen nativen Client oder Add-Ons, wie sie beispielsweise bei ARSnova angeboten werden, umzusetzen.

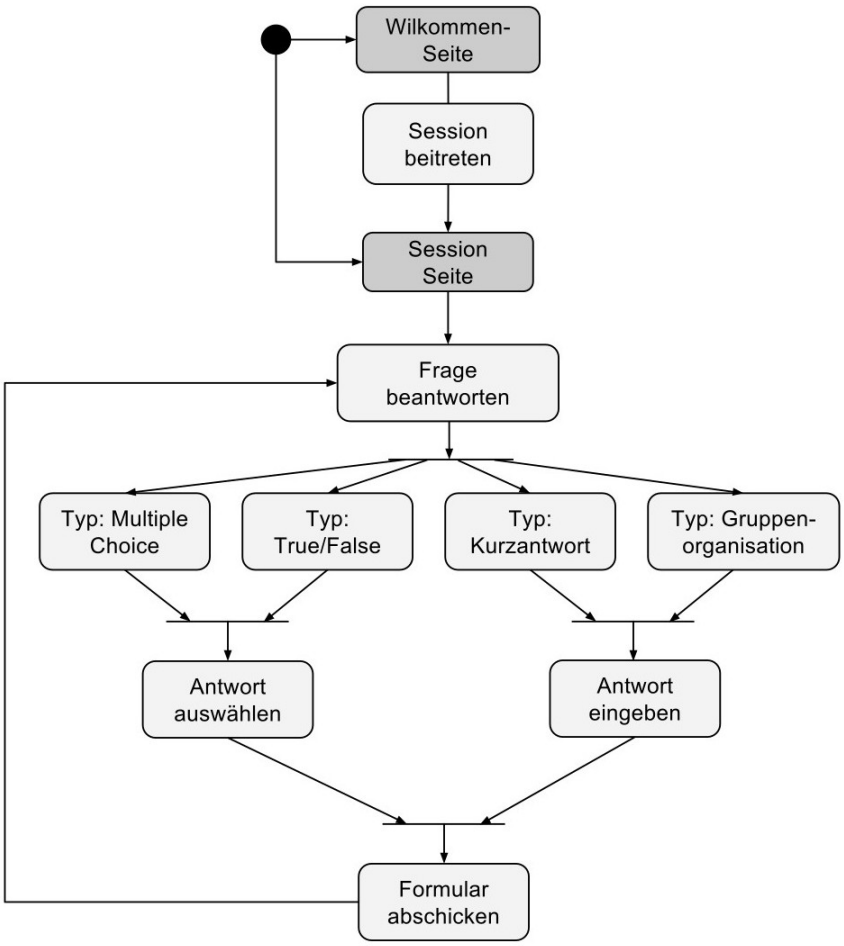

Abbildung 5.3: Aktivitätsdiagramm für die Sitzung eines Studenten

5.2 Design

Bei der Planung des Designs müssen nicht nur die Anforderungen aus Kapitel 3 umgesetzt werden, sondern es muss auch ein besonderes Augenmerk sowohl auf die Darstellung auf Geräten mit verschiedenen Bildschirmgrößen als auch auf die Gewohnheiten der Benutzer gerichtet werden. Das Design soll an die von Google entwickelte Designsprache „Material Design“[2] angelehnt werden. Hintergrund dafür ist, dass viele Anwender mindestens eines der Google Produkte besitzen, nutzen oder jedenfalls kennen. Sei es ein Smartphone mit dem Android Betriebssystem oder eines der Softwareprodukte wie Gmail, Inbox, Maps, Chrome, Drive oder Docs. Folglich kennt der Benutzer bereits das User-Interface dieser Anwendung und kann zumindest erahnen wie er Fragen erstellt oder auf welche Buttons er klicken muss um ein

[2] https://www.google.com/design/spec/material-design/introduction.html, zuletzt abgerufen am 09.03.2016

Formular abzuschicken. Ein weiterer Vorteil liegt darin, dass es bereits Bibliotheken oder Pakete gibt, die diese Design Sprache implementieren. Damit spart man sich bei der Entwicklung wertvolle Zeit.

5.2.1 Die Willkommen-Seite

Die Willkommens Seite (Abbildung 5.4) ist der Einstieg in die Webanwendung, hier können Nutzer eine neue Sitzung erstellen oder an einer bereits bestehenden Sitzung teilnehmen. Diese Seite ist in drei Teile aufgeteilt:

- Einen Kopfbereich für den Titel und eine kurze Beschreibung der Anwendung
- Einen Fußbereich als visuellen Abschluss der Seite, welcher einen Link zum Feedback-Formular enthält
- Und dazwischen einen Aktionsbereich in dem der Benutzer eine Sitzung erstellen bzw. an einer bestehenden teilnehmen kann

Mit einem Klick auf die Buttons „Session erstellen“ oder „Session beitreten“ wird der Benutzer auf die Session-Seite weitergeleitet.

5.2.2 Die Session-Seite

Diese Seite ist das eigentliche Herzstück der Applikation, hier können Fragen erstellt, beantwortet und ausgewertet werden. Das Design besteht aus einem schmalen Navigationsmenü am oberen Rand des Bildschirms, hier wird die Session-ID angezeigt und hinter dem drop-down-Menü verbirgt sich ein Link zum Feedback-Formular und die Anzahl der aktiven Teilnehmer in diesen Raum. Darunter befindet sich eine Liste von Fragen. Jede Frage wird als eine Art Karteikarte dargestellt. Je nachdem ob der Benutzer ein Dozent oder Student ist, wird der Inhalt anders dargestellt.

Die Studenten-Ansicht

Die Studentenansicht der Karteikarte ist sehr simpel gehalten: Zuerst die Fragestellung, darauf die Antwortmöglichkeiten und zum Schluss ein „Abschicken“ Button.

Die Dozenten-Ansicht

Die Ansicht für den Dozenten (Abbildung 5.6, Seite 33) unterscheidet sich auf den ersten Blick von der des Studenten. Zum einen ist die Navigationsleiste am oberen Teil das Bildschirms in einer anderen Farbe gekennzeichnet, so kann der Anwender auf den ersten Blick erkennen, ob er in der Rolle des Dozenten oder Studenten ist. Zum Anderen befindet sich in der rechten, unteren Ecke ein Aktionsbutton, der mit einem Plus-Symbol gekennzeichnet ist. Mit diesem Button kann der Benutzer sofort erkennen, dass er hier etwas hinzufügen kann. Die Dozentenansicht besteht ebenfalls

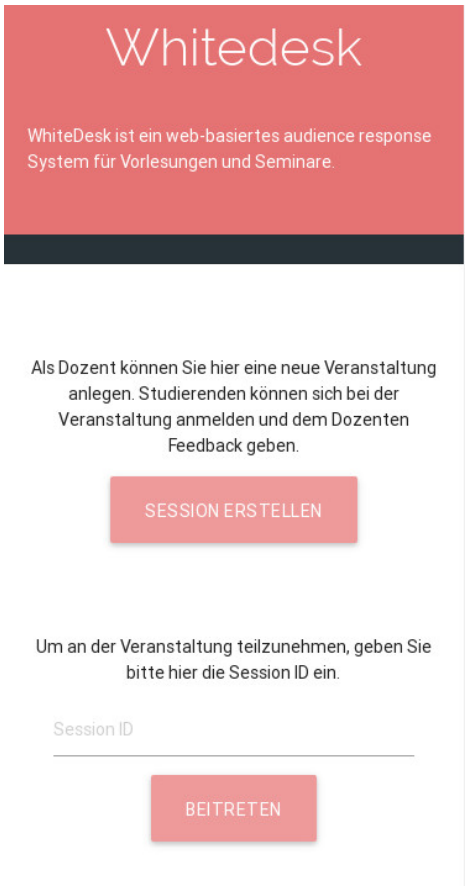

Abbildung 5.4: Willkommen-Seite auf einem Smartphone

aus Karteikarten, notwendigerweise unterscheiden sich jedoch die Inhalte von denen der Studentenansicht. Der Inhalt jeder Karteikarte hat dennoch immer das selbe Muster für den Dozenten: In der linken Hälfte befindet sich eine Auflistung über die bereits abgegebenen Antworten und in der rechten Hälfte eine grafische Auswertung der Antworten.

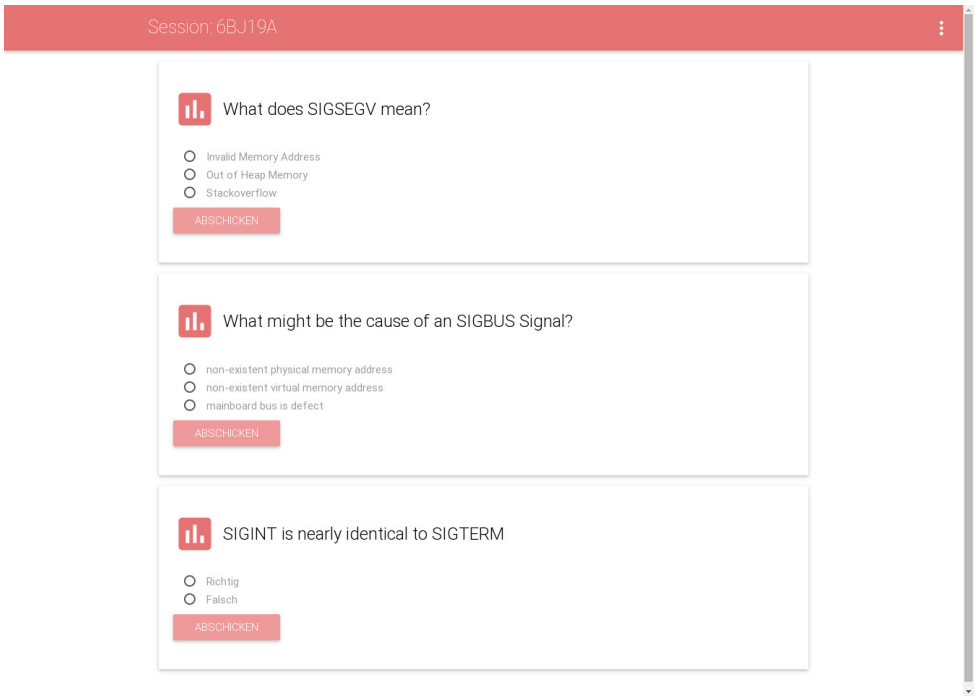

Abbildung 5.5: Hörer-Ansicht auf einem Desktop-Computer

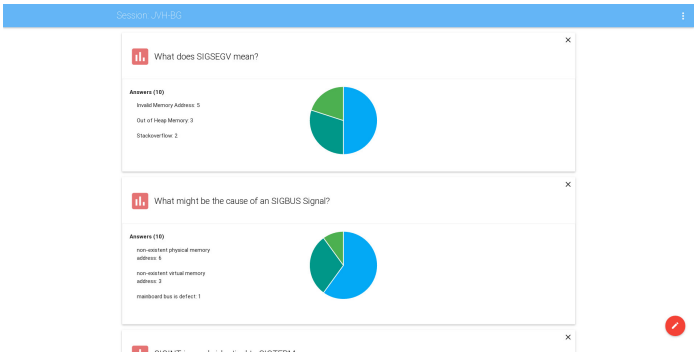

Abbildung 5.6: Dozenten-Ansicht auf einem Desktop-Computer

6 Umsetzung & Implementierung

Mit der im vorigen Kapitel beschriebenen Softwarearchitektur im Gedächtnis ist nun eine geradlinige Implementierung des Audience-Response-Systems möglich, die dieses Kapitel beschreibt. Zunächst werden die wesentlichen Elemente der Serveranwendung und deren Schnittstellen erklärt, anschließend folgt eine Beschreibung des Clientsystems. Abschließend wird auf bekannte Einschränkungen und Fehler hingewiesen. Das Backend, wie auch das Frontend, wurde in Javascript geschrieben. Zusätzlich wurden diverse Pakete aus den Paketquellen von npm genutzt um den Entwicklungsaufwand möglichst gering zu halten. Die wichtigsten Pakete werden in der Beschreibung jedes Teilsystems kurz aufgelistet und es wird erklärt, warum sie benutzt wurden.

6.1 Backend

Die Serveranwendung wurde in Javascript mithilfe der Platform Node.js implementiert. Die wichtigsten Pakete die hierfür verwendet wurden sind:

Express vereinfacht die Erstellung eines Webservers. Es kümmert sich in dieser Anwendung um das Management der Middlewares, die Routingfunktionalität und das Ausliefern von statischen Dateien.

socket.io kümmert sich um den Websocket Handshake und stellt eine API zur Kommunikation mit dem Client bereit.

cookie-parser ist ein Paket zum Lesen und Schreiben von signierten Cookies, und wird zur Identifikation von Clients verwendet.

handlebars ist ein Paket um „Handlebars“ Templates zu rendern. Hiermit lassen sich komfortabel HTML Schnipsel kombinieren und dynamisch gestalten.

rethinkdb ist ein Javascript Treiber für die RethinkDB Datenbank. Dieses Paket dient der Kommunikation zwischen der Serveranwendung und der Datenbank.

Der Quellcode der Anwendung ist auf der beiliegenden DVD, beziehungsweise auf GitHub[1] zu finden. Der Einstiegspunkt für die Serveranwendung liegt in der Datei *app.js*. Dort werden sowohl alle benötigten Fremdpakete als auch selbst geschriebenen Module mit der *require()* Funktion geladen. Daraufhin wird Express initialisiert und

[1] https://github.com/moolen/thesis, zuletzt abgerufen am 04.04.2016

sogenannte *Middlewares* mit der Funktion *app.use()* in dem Framework registriert. Eine Übersicht über den Ablauf der Middlewarefunktionen befindet sich in Abbildung 6.1. Middlewares sind Funktionen die im Anfrage-Antwort Zyklus das Anfrage- und Antwort Objekt manipulieren und die darauffolgende Middlewarefunktion aufrufen. Die letzte Middleware stellt den Router dar, hier werden sogenannte Route-Handler registriert. Dies sind Funktionen, die eine HTTP-Anfragemethode und URL, z.B. **`GET http://example.com/foo`** auf eine Funktion abbilden. Abschließend wird mit dem Aufruf der *listen()* Funktion dem HTTP Server mitgeteilt, dass er auf eingehende TCP Verbindungen unter einem fest definiertem Port lauschen soll.

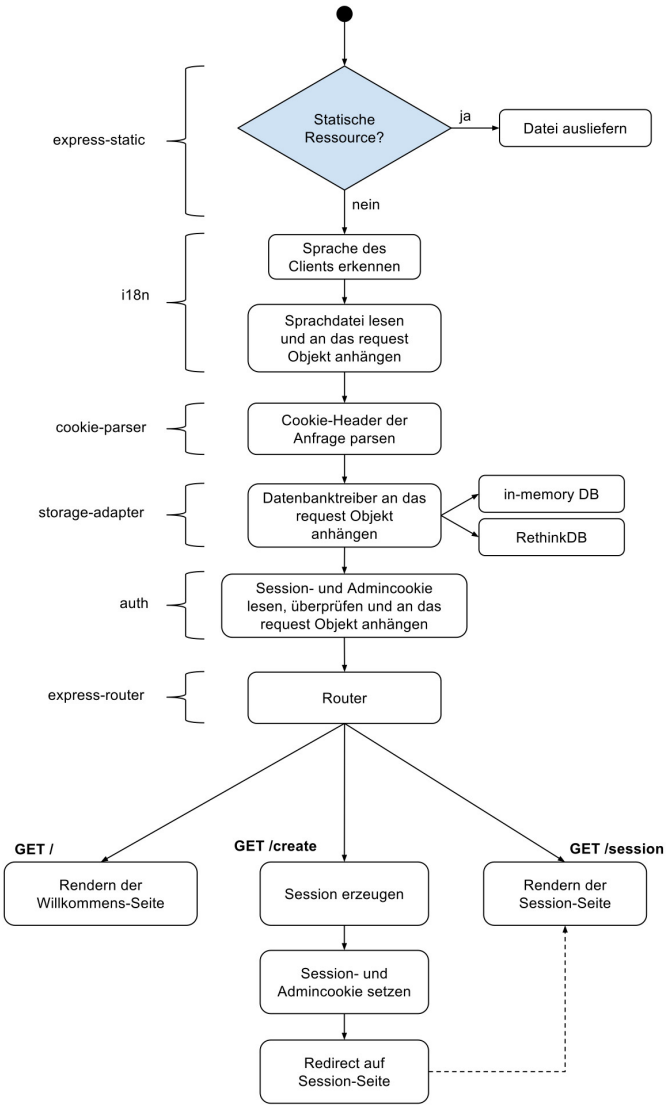

Abbildung 6.1: Ablaufdiagramm der Middlewares in der Anwendung

Die Middlewarefunktionen der Applikation haben immer die gleiche Funktionssignatur: void fn(request, response, next). Sie können beliebigen Code ausführen, Änderungen am Anfrage- oder Antwortobjekt vornehmen oder den Anfrage-Antwort Zyklus beenden. Mit der ersten Middleware *express.static('public')* wird der Server so konfiguriert, dass er die angeforderte Datei aus dem Ordner *public* als Antwort zurückgibt, sofern diese existiert. Daraufhin wird mit der *i18n.middleware* das Anfrageobjekt um sprachspezifische Attribute erweitert. Dabei wird die Sprache des Benutzers vom *accept-language* Header ausgelesen, die jeder Browser mit seiner Anfrage sendet. Das hat den Hintergrund, dass die Anwendung mit Hinblick auf eine eventuelle Übersetzung bzw. Internationalisierung einfach und schnell erweitert werden kann. Die *cookieParser(config.cookieSecret)* Middleware analysiert den *cookie* Header der Anfrage und erweitert das Anfrageobjekt. Alle von der Anwendung zurückgegebenen Cookies sind mit einem *Message Authentication Code*[2] signiert. Dadurch kann sichergestellt werden, dass der Wert eines Cookie nicht von einem Benutzer beliebig manipuliert werden kann. Die von der Anwendung gesetzten Cookies enthalten die Information welchem Benutzer welche Rolle zugeordnet ist — Student oder Dozent. Wenn die Cookies nicht signiert wären, könnte ein Benutzer den Wert des Cookies einfach ändern und sich als ein anderer Benutzer ausgeben.

Als Datenbanktechnologie fiel die Wahl auf RethinkDB, eine dokumentenorientierte Datenbank. Diese schickt in Echtzeit Nachrichten an ihre Clients, sobald Änderungen an bestimmten Datensätzen vorgenommen wurden. Die Verwendung einer externen Datenbank ist in dieser Anwendung optional, die Anbindung an die Datenbank wurde mit dem Strategie-Entwurfsmuster modelliert und ist deshalb austausch- und erweiterbar. Es wurden 2 konkrete Strategien implementiert: Eine flüchtige in-memory Datenbank im Heap[3] des Node.js Prozsseses und eine persistierende, welche den Treiber für die RethinkDB Datenbank nutzt. Die flüchtige Variante wurde zuerst implementiert um den Entwicklungsprozess zu beschleunigen. Jedoch hat diese Variante schwerwiegende Nachteile: Zum Einen kann diese Implementierung nicht in einer nebenläufigen Umgebung (Multithreading) genutzt werden, da sonst pro Thread eine Datenbank existieren würde und der Benutzer Glück haben müsste, dass seine Anfrage an den richtigen Thread geschickt wird und zum anderen würde die Datenbank mit dem Ende des Node.js Prozesses gelöscht werden. Dieser Datenbanktreiber ist nur für Evaluationszwecke gedacht. Darauf aufbauen wurde ein weiteres Modul für eine Produktionsumgebung programmiert, sodass sich auch die Datenbank unabhängig vom Applikationsserver skalieren lässt und die Daten tatsächlich persistent gespeichert werden.

[2] Ein MAC dient dazu die Integrität einer Nachricht zu überprüfen. Hierfür wird eine Prüfsumme aus der Nachricht und einem Geheimen Schlüssel erzeugt.

[3] Der Heap ist ein Speicherbereich, der zur Laufzeit dynamisch zugewiesen wird.

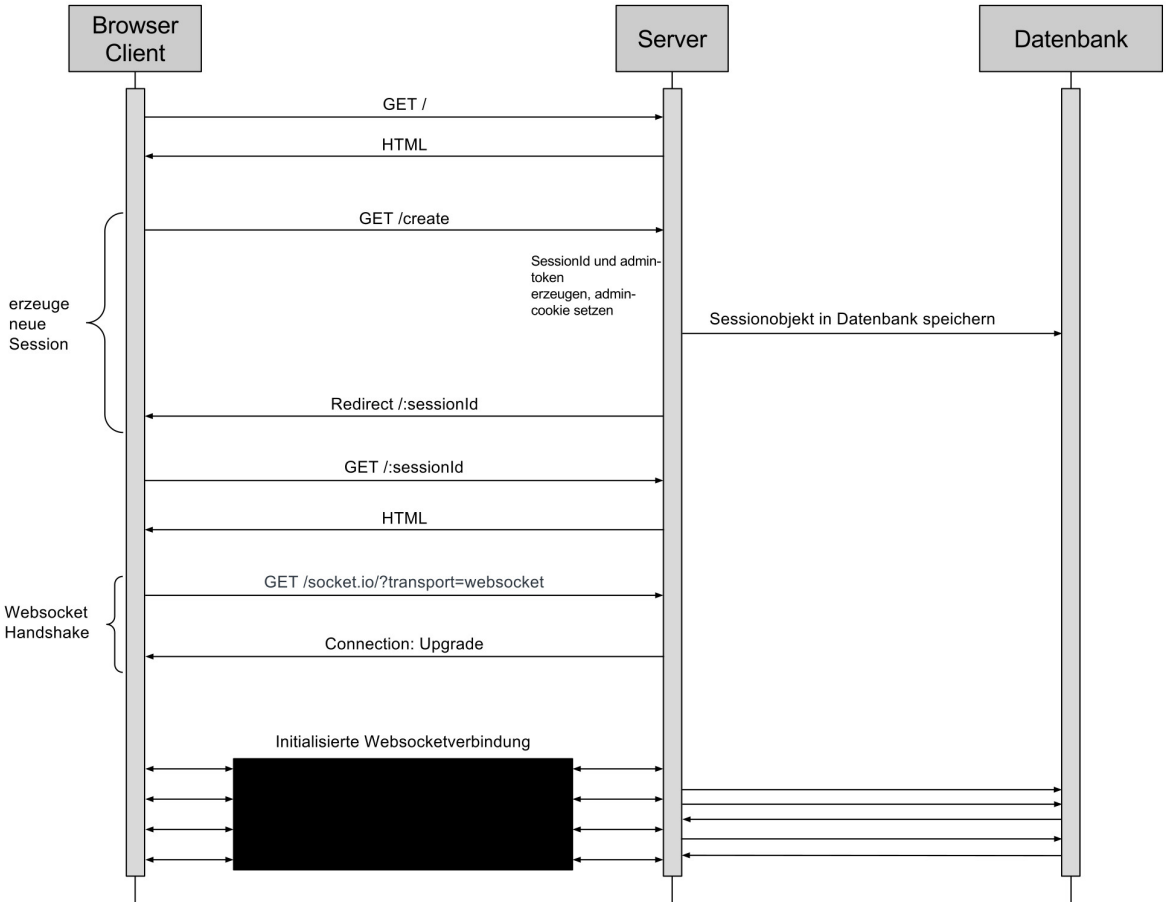

Abbildung 6.2: Sequenzdiagramm der Initialisierung einer Session

```
1
2  interface DBTreiber {
3    void              construct(Object opts);
4    Promise{Session}  get(string sessionId);
5    Promise{mixed}    getData(string sessionId, string key);
6    Promise{bool}     setData(string sessionId, string key, mixed value);
7    Promise{Session}  createSession(string sessionId, string adminToken);
8    Promise{bool}     removeSession(string sessionId);
9    Promise{bool}     isAdmin(string sessionId, string adminToken);
10   Promise{bool}     hasAnswered(string sId, string qId, string uToken);
11   Promise{Answer}   addAnswer(string s, string q, string u, Answer a);
12   Promise{Question} addQuestion(string sessionId, Question question);
13   Promise{Question} updateQuestion(string sessionId, Question question);
14   Promise{bool}     removeQuestion(string sessionId, Question question);
15 };
```

Listing 6.1: Interface des Datenbanktreibers in Pseudocode

Das *storage-adapter* Modul lädt den gewünschten Datenbanktreiber und stellt ihn den darauffolgenden Middlewares zur Verfügung. Jeder Datenbanktreiber implementiert jeweils eine bestimmte API, welche von anderen Teilen des Programms aufgerufen wird, um Informationen aus der Datenbank abzufragen oder hinzuzufügen. Javascript unterstützt das Interface-Konzept nicht, deshalb ist diese Herangehensweise auf jeden Fall akzeptabel, jedoch kann dieses Entwurfsmuster nur in statisch getypten Sprachen voll ausgeschöpft werden. Die API des Datenbanktreibers verlässt sich hauptsächlich auf das *promise* Konstrukt, auch bekannt als *future* oder *deferred*. Hierbei handelt es sich um Objekte, die als Platzhalter für einen konkreten Wert genutzt werden, weil dieser im Moment noch unbekannt ist. Dies ist typischerweise bei asynchronen Funktionsaufrufen der Fall: Während beispielsweise eine Datei vom darunterliegenden Betriebsystem gelesen und decodiert wird, kann der aufrufende Prozess andere Aufgaben wahrnehmen statt auf deren Verarbeitung zu warten. Diese Art der Programmierung ermöglicht ein hohes Maß an nebenläufiger Parallelisierung von Prozessen.

```
1
2  interface Session {
3    string id,
4    string adminToken,
5    int    userCount,
6    []Question questions
7  };
8
9  interface Question {
10   string    id,
11   string    question,
12   string    type,
13   date      createdAt,
14   []string  acceptedOptions,
15   []Answer  answers,
16   []string  answered
17 };
18
19 interface Answer {
20   string id,
21   string value
22 };
```

Listing 6.2: Interface der Datenbankdokumente

Die darauffolgende Middleware „auth“ ist ausschließlich zur Authentifizerung des Benutzers zuständig. Sie liest die Werte der beiden Cookies *session-token* und *admin-token* aus und überprüft mit Hilfe des Datenbanktreibers, ob sie mit den in der Datenbank hinterlegten Werten übereinstimmen. Das Ergebnis der Abfrage wird an das Anfrageobjekt angeheftet und so allen darauffolgenden Middlewares zur Verfügung gestellt.

Zu guter Letzt werden die Route-Handler als Middleware registriert. Zunächst gibt

es Routen die mit gewöhnlichem HTML antworten, dazu gehören die Willkommens-Seite `GET /` und die Sitzungs-Seite `GET /:sessionId`. Die Letzte Route `GET /create` erzeugt eine neue Sitzung und initialisiert die WebSocket Schnittstellen, mehr dazu in Unterkapitel 6.2. Die HTTP Upgrade Anfrage für die Initialisierung der WebSocket-Verbindung wird von dem socket.io Modul realisiert.

Alle Konfigurationseinstellungen sind in einer eigenen Datei *config.js* ausgelagert. So kann man bequem den port oder die Datenbankparameter ändern.

Mit der Initialisierung der WebSocket-Verbindung hat die Serveranwendung alle Schritte getan, um das Frontend vollständig zu initialisieren. Jegliche Kommunikation mit dem Server wird von nun an über Websockets realisiert. Das gilt auch für das Erzeugen von HTML, dies wird ebenfalls Client-Seitig berechnet.

6.2 Schnittstellen

Der Datenaustausch zwischen dem Front- und Backend erfolgt ausschließlich über WebSockets. Um die Schnittstelle zu nutzen muss sowohl der Client als auch der Server auf bestimmte Lifecycle Events zurückgreifen. Zunächst muss beim Verbindungsaufbau ein entsprechender „Namespace“ angegeben werden. Das ist eine Funktion von socket.io, mit der Teilnehmer in verschiedene Gruppen eingeteilt werden können. Nachrichten, die an einen bestimmten Namespace addressiert sind, werden vom Server auch nur an diejenigen Teilnehmer übertragen, die auch eine Verbindung zu dem jeweiligen Namespace haben. Diese Namespaces werden genutzt um die Sessions voneinander abzugrenzen. Dies ist notwendig, da sonst ein Broadcast-ähnliches System erstellt würde, welches eine unnötige Netzwerklast erzeugt.

Nachdem sich der Client mit dem richtigen Namespace verbunden hat, registriert der Server Rückruffunktionen für alle nötigen Lifecycle events. Mit dem Aufbau einer neuen Verbindung (event: *connect*) als auch mit dem Abbau (event: *disconnect*) teilt der Server den anderen Teilnehmern mit, dass sich die Teilnehmerzahl verändert hat (event: *usercount:change*). Daraufhin wartet der Server auf das *client:ready* event, in dem soll der Client das *admin-token* – das token zur Authentifizierung des Dozenten – an den Server schicken, worauf hin er dieses verifiziert und selbst das *server:ready* „feuert“. In diesem schickt er dem Client die restlichen Daten, die zur Darstellung im Frontend benötigt werden, also die bisher erstellten Fragen und die Anzahl der momentan verbundenen Benutzer in diesem Namespace. Von nun an lauscht der Server auf Events wie zum Beispiel *answer:submit* oder *questions:add*, die, wieder der Name schon sagt, zur Änderung der Dokumente in der Datenbank gedacht sind. In Abbildung 6.3 sind die wichtigsten Events sowie Teile der Programmlogik dargestellt.

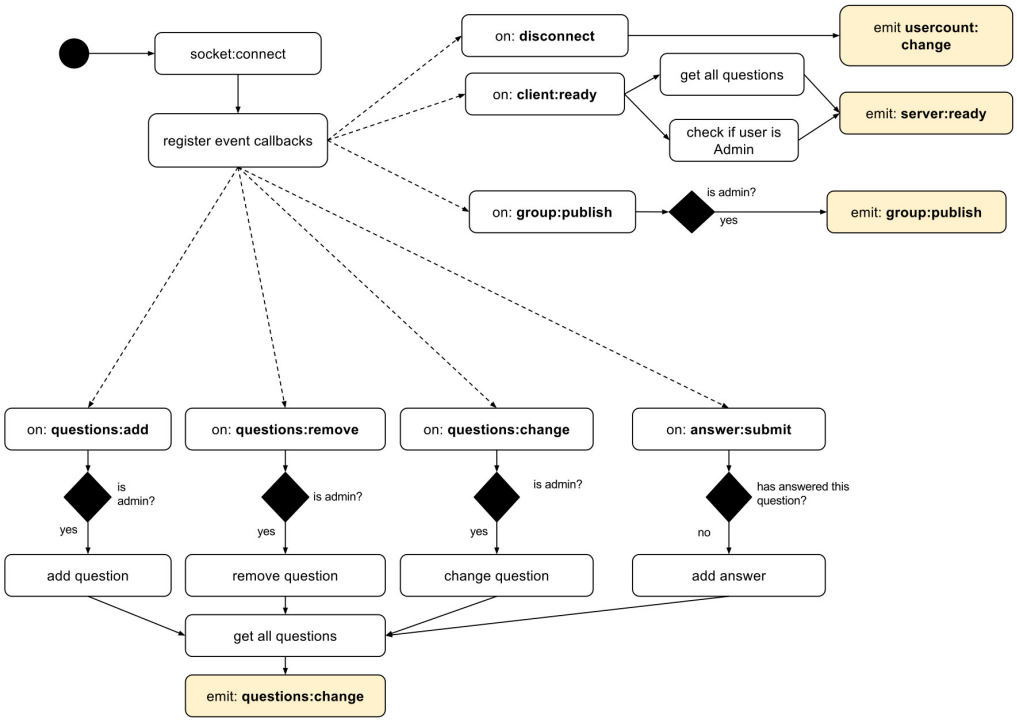

Abbildung 6.3: Programmablaufplan der WebSocket Schnittstellen

6.3 Frontend

Das Frontend stellt eine Single-Page-Application („SPA“) dar, das ist eine Webanwendung, die aus einem einzelnen HTML Dokument besteht. Um aber dennoch dynamische Inhalte darzustellen werden zunächst über Schnittstellen (AJAX, WebSocket) Daten ausgetauscht und mithilfe der Document Object Model API (kurz: „DOM“) die bestehende HTML Seite manipuliert. Im Gegensatz zu einer SPA stehen herkömmliche Webanwendungen, die aus mehreren, verlinkten HTML-Dokumenten bestehen. Durch die Verwendung einer SPA kann die Serverlast reduziert werden, da einerseits weniger HTTP Anfragen durchgeführt werden und andererseits nur die „nackten“Daten übertragen werden – also nicht, wie sonst bei klassischen Webanwendungen üblich, ein HTML-Gerüst, das um die Daten herum gelegt wird. Allerdings muss dafür die Geschäftslogik – zumindest teilweise – sowohl Client- als auch Serverseitig abgebildet werden, was als zusätzlicher Aufwand zu verbuchen ist. Bei einer SPA spricht man deswegen auch von einem „Fat Client“ System. Für die Implementierung einer SPA gibt es immer mehr Pakete und Frameworks, die die Erstellung und Verwaltung des Codes erleichtern sollen, da an vielen Stellen Funktionen benötigt werden um rasch eine Anwendung aufzubauen. Die Browseranwendung umfasst momentan zirka 11.500 Zeilen Code, welche sich auf viele Dateien verteilen. Das sind in Summe ungefähr 1.7 Megabyte (382 Kilobyte mit gzip), die initial vom Browser

heruntergeladen werden müssen. Aufgrund der Tatsache, dass eine SPA zunächst viel Code vom Server herunterladen muss, liegt der Performanzengpass an der Anzahl der HTTP-Anfragen, die der Browser an den Server schicken muss. Um diese zu minimieren müssen die Abhängigkeiten des Quellcodes aufgelöst werden und alle Dateien in einer einzelnen zusammengefasst werden. Dieser Prozess wird natürlich nicht manuell erledigt, dafür gibt es Programme, die den Build-Prozess von Softwareanwendungen automatisieren. Folgende Pakete wurden zur Erstellung der Single-Page-Application genutzt:

- **ampersand** ist eine Sammlung von lose gekoppelten Paketen, die die Erstellung einer SPA erleichtern. Funktionen für das Modifizieren von HTML, Registrieren von DOM-Events, sowie Synchronisieren des Zustandes mit dem Server wurden genutzt, um schneller eine SPA zu erstellen.
- Wie auch serverseitig wird die Templatesprache **handlebars** hier genutzt um dynamisch HTML zu erzeugen.
- **chart.js** erleichtert das Erstellen von Diagrammen. Diese werden in der Ansicht des Dozenten verwendet um Frageauswertungen darzustellen.
- **Materialize** bietet auf dem Material Design basierende Frontend Komponenten. Hiermit wurden große Teile des User-Interface gestaltet.
- **gulp** ist task-runner für das Frontend welches Quellcode kompiliert und mehrere statische Dateien zusammenfasst. Gulp benutzt **browserify**, einen Modulbundler für das Frontend mithilfe dessen Abhängigkeiten des Javascript Codes aufgelöst werden, sowie den CSS-Präprozessor **less**.
- **socket.io** wird ebenfalls in der Frontendanwendung genutzt. Hiermit wird die bi-direktionale Echtzeitkommunikation abwärtskompatibel auf der Clientseite abgebildet.

Zunächst besteht die Anwendung aus einem *Application* Objekt, welches die Frontend-Anwendung initialisiert und die nötigen Schnittstellen, die im vorigen Kapitel angesprochen wurden, anspricht. Dieses Objekt erweitert die vom *ampersand-app* bereitgestellte „Klasse“, die das Singleton Entwurfsmuster implementiert. Dieses Application Objekt (Abbildung 6.4, Seite 43) ist der Eigentümer über anwendungsweite Instanzen von Models, Views, Collections und des Routers. Außerdem hält dieses Objekt eine Referenz auf das von socket.io initialisierte *socket* Objekt. In der Anwendung gibt es immer genau eine Instanz des Fragenkatalogs („QuestionCollection“) sowie des Models, welche die Anzahl der User repräsentiert. Diese Instanzen werden dann je nach Bedarf an weitere Objekte übergeben (*Dependency Injection*).

In einer klassischen Webanwendung sind die Dokumente in verschiedenen Ordnern hinterlegt und können über sogenannte Uniform Resource Identifier (URI) angesporchen werden. Bei einer SPA allerdings wird nur ein Dokument ausgeliefert, damit geht diese Struktur und die damit verbundene semantische Logik verloren.

```
1 http://www.example.com/
2     the homepage of Example Inc.
3 http://www.example.com/people/alice
4     the homepage of Alice
5 http://www.example.com/people/bob
6     the homepage of Bob
```

Listing 6.3: Semantische Bedeutung von URIs[4]

Damit man in einer SPA dennoch semantisch wertvolle URIs abbilden kann, wurde die HTML5 Pushstate API eingeführt. Das Router Modul nutzt diese API um einerseits auf Änderungen in der URI zu reagieren, wie sie beispielsweise durch das Anklicken von Links entstehen, oder andererseits um die URI selbst programmatisch zu manipulieren. Letztere Möglichkeit wird vor allem von den View-Komponenten des Systems verwendet um einen neuen, reproduzierbaren Zustand des Systems durch eine neue URI zu kennzeichnen. Zu einem früheren Entwicklungsstand gab es Einzelansichten für Fragen, so konnten Fragen nach dem Schema */{session-id}/question/{question-id}* aufgerufen werden. Das bedeutet, dass ein Benutzer beim Aufruf dieser URI direkt auf der Detailansicht der Frage gelandet ist. Heutzutage ist allerdings nur noch die URI für die Erstellungsmaske einer Frage nach dem Schema */{session-id}/question/new* implementiert. Nichtsdestotrotz ist diese API sehr gewichtig. Man beachte, dass die SPA aus einem einzelnen HTML Dokument besteht und – sofern es korrekt implementiert ist – anhand der URI den Zustand des Systems erkennt und reproduziert. Nur dadurch sind heutzutage Deeplinks in Single-page-Webanwendungen möglich.

Models stellen die Grundlage der Geschäftslogik in der Anwendung dar. Sie kapseln die eigentlichen Informationen bzw. Daten der Anwendung ein und erlauben das Speichern dieser auf dem Server. Immer wieder benötigt man eine geordnete Menge von gleichartigen Models – hier ist das der Fragenkatalog. Dieses Problem wird durch eine sogenannte Collection gelöst. Diese ist nicht einfach nur ein Array von gleichartigen Model-Instanzen, sondern ebenfalls ein eigener Datentyp. Auf dieser Abstraktionsebene muss ebenfalls Geschäftslogik abgebildet werden – beispielsweise das Filtern, Sortieren oder Speichern von Daten. Umgekehrt können Models ebenfalls Collections als Objekt-Attribut besitzen, so können beliebige Beziehungen zwischen Models und Collections abgebildet werden. In dieser Anwendung erweitern die Models das *ampersand-state* Objekt, welches die eigentlichen Daten in einem internen Objekt einkapselt. Hierauf kann nicht direkt zugegriffen werden, es ist nur möglich über getter oder setter den Zustand zu verändern. Ausserdem implementiert dieses Objekt das Beobachter Entwurfsmuster (genauer gesagt Event Emitter), wodurch von außen Rückruffunktionen an dieses Objekt angebunden werden können, welche bei Zustandsübergängen aufgerufen werden.

Views stellen die Beziehung zwischen den Models, Collections und dem DOM her. Sie besitzen üblicherweise Models und/oder Collections und referenzieren HTML-

[4]Quelle: https://www.w3.org/TR/cooluris/#oldweb, zuletzt abgerufen am 17.03.2016

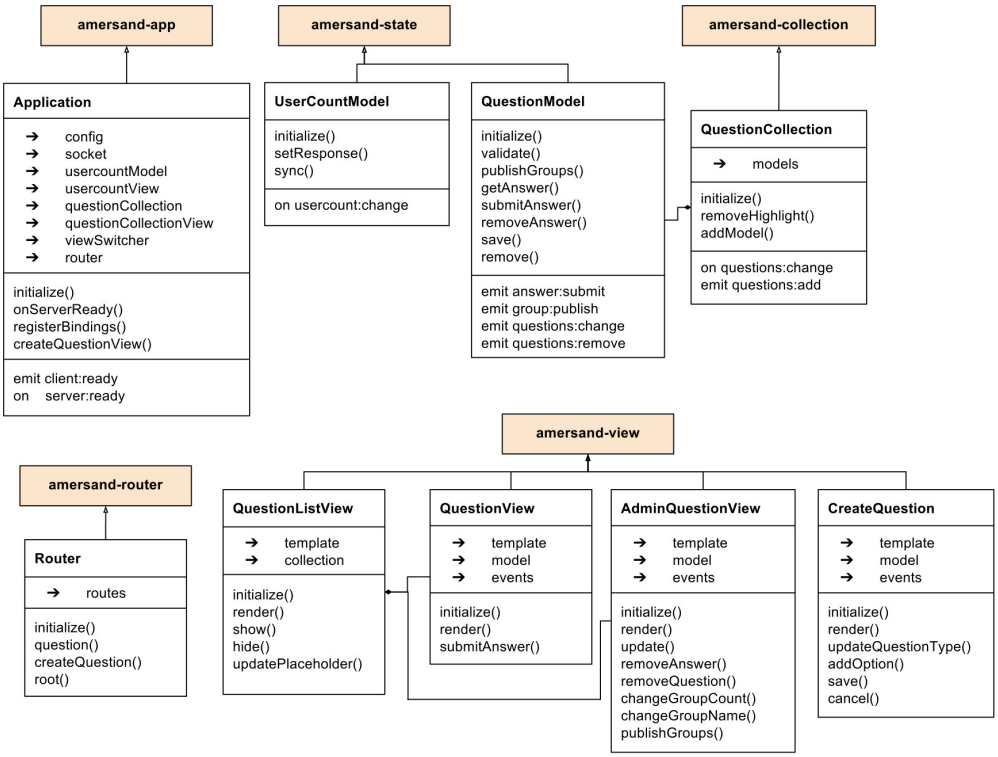

Abbildung 6.4: Klassendiagramm des Frontends

Templates, in der Regel enthalten Views auch einfache Geschäftslogik. Diese Views entsprechen also in etwa dem Controller in einer klassischen MVC-Architektur. Weiterhin implementieren diese Views das Beobachter Entwurfsmuster. Dabei lauscht das View auf Änderungen des Models oder der Collection und aktualisiert entsprechend das HTML-Dokument. Umgekehrt kann auch auf Änderungen des HTML reagiert werden und dann entsprechend das Model aktualisiert werden, dann spricht man von „two-way data binding“. Das *ampersand-view* Paket stellt eine reichhaltige API zur Verfügung um Views hierarchisch zu strukturieren. Einerseits gibt es die Möglichkeit untergeordnete Views einzubinden, die den Lebenszyklus des übergeordneten Views teilen, anderseits kann jedes Model aus einer Collection auf ein untergeordnetes View, auf die eben beschriebene Art und Weise, abgebildet werden. Damit wurde beispielsweise die Beziehung zwischen dem *QuestionListView* und dem QuestionView realisiert, wie in Abbildung 6.4 zu sehen ist.

6.4 Einschränkungen & Fehler

Durch die in Kapitel 3.1 definierten Anforderungen ergeben sich zwangsläufig konzeptionelle Einschränkungen für die Anwendung, weiterhin wurden bei Testläufen kleinere Fehler gefunden:

- Ein Teilnehmer kann beispielsweise eine Antwort auf eine Frage erneut abgeben, wenn er einen anderen Browser oder die „*Privates Browsen*“-Funktion des Browsers nutzt. Um dies zu unterbinden gibt es technisch keine andere Möglichkeit als eine Registrieren-Funktion zu implementieren. Allerdings widerspricht dies dem Ziel, die Zugangsschranken niedrig zu halten.

- Die Session-ID kann beim Erzeugen einer Session mit einer bestehenden kollidieren und so die bestehenden Fragen und Antworten löschen. Die Wahrscheinlichkeit für so eine Kollision ist allerdings sehr gering. Die Session-ID besteht aus 6 Zeichen, jedes davon kann aus 64 verschiedenen Symbolen bestehen, damit ergeben sich $64^6 \approx 6,8 * 10^{10}$ mögliche Session-IDs.

- Die Anzeige auf mobilen Endgeräten ist noch nicht optimal, da lange Texte durch Zeilenumbrüche für eine unschöne Darstellung sorgen.

- Wenn der Dozent einen Gruppennamen bearbeitet springt die Seite nach oben, dies wird durch das re-rendering des Fragenkatalogs verursacht, angemessen wäre das re-rendern einer einzelnen Frage.

7 Praxistest

Ein Nutzertest ist zunächst dafür da um die Anwendung mit mehreren Benutzern gleichzeitig zu belasten. Dadurch können Fehler beobachtet werden, die erst durch eine höhere Last entstehen, wie beispielsweise Verbindungsabbrüche. Weiterhin ist ein Schulterblick bei den Anwendern hinsichtlich der Benutzerführung sowie der Darstellung der Anwendung auf verschiedenen Endgeräten aufschlussreich. Schon während des Entwicklungsprozesses wurde die Anwendung immer wieder auf Funktion und Darstellung hin getestet. Nichtsdestotrotz sind Fehler bei einem Praxistest zu erwarten, da man als Entwickler dazu neigt betriebsblind gegenüber der Anwendung zu sein. Gerade unvoreingenommenen Nutzer besitzen eine andere Erwartung an die Anwendung, die dabei Hilft die Benutzererfahrung zu verbessern. Mit dem Feedback der Benutzer soll die Anwendung in mehreren Entwicklungszyklen weiter optimiert werden um somit eine nachhaltige Lösung zu schaffen.

Vor dem eigentlichen Testlauf wurde im Gespräch mit der Dozentin geklärt, welche Features dem ARS noch fehlen bevor sie es einsetzen kann. Zum einen wurde angeregt einen Upload von Bildern zu ermöglichen, damit Bilder in eine Fragestellung eingebettet werden können. Zum Anderen wurde der Wunsch geäussert einen Beschreibungstext zu einer Frage angeben zu können.

Der Praxistest wurde mit diesen neuen Features in einer Vorlesung der Dozentin mit ungefähr 35 Teilnehmern durchgeführt, dazu wurden zwei Sessions vorbereitet: Die Erste soll das Wissen der Studierenden mit einem fachlichen Quiz testen. In der zweiten Session werden die Studierenden gebeten Feedback zu der Art und Weise des Unterrichts der Dozentin abzugeben. Die Dozentin hat während der Vorlesung die URL zu dem ARS über die Lernplattform verteilt, woraufhin den Studierenden einige Minuten Zeit gegeben wurde um die Fragen zu beantworten. Daraufhin ist die Dozenten mit den Studierenden Frage für Frage das Quiz durchgegangen um so Missverständnisse auszuräumen und Detailfragen zu einzelnen Fragen zu klären. Während der Durchführung wurde beobachtet, dass es in einigen Browsern zu Fehldarstellungen gekommen ist. Diese wurden dokumentiert, damit sie später beseitigt werden können.

Der Dialog mit den Studierenden nach dem Praxistest hat ergeben, dass sie den Einsatz des ARS als durchweg positiv empfunden haben und sie sich gut vorstellen können dieses ARS auch in anderen Kursen zu nutzen. Besonders gelobt wurde das klare Design und die Möglichkeit die Webanwendung auf einem Mobiltelefon aufzurufen. Jedoch wurde bemängelt, dass teilweise das Abschicken des Antwortformulars nicht verzögerungsfrei von statten ging. Weiterhin hatten die Studierenden den Eindruck, dass sie sich aktiv mit jeder einzelnen Fragestellung auseinandersetzen muss-

ten, sowohl während der Beantwortung der Frage als auch in der Nachbesprechnung des Quiz.

Der Dialog hat ebenfalls Verbesserungsvorschläge zu Tage gebracht. Zum einen wurde eine „bearbeiten“-Funktion für Fragen angeregt, da sonst bei einem Rechtschreibfehler die Frage gelöscht und neu erstellt werden müsse. Zum anderen wurde seitens der Studierenden ein einzelner „absenden“-Button gewünscht, sodass nicht jede Frage einzeln beantwortet werden müsse. Diese Ideen sind besonders wertvoll für den weiteren Ausbau der Anwendung, da sie von den Benutzern selbst kommen. Nur mit ihrer Hilfe kann die Anwendung langfristig optimiert werden.

In einer abschließenden Betrachtung kann gesagt werden, dass die Anwendung dem Einsatz in einer Unterrichtssituation standhalten konnte. Sie wurde grundsätzlich positiv wahrgenommen und hat ihren Zweck im Unterricht erfüllen können.

8 Fazit

Primäres Ziel war die Konzeption und Entwicklung eines webbasierten ARS, welches Dozenten als Werkzeug zur Verfügung stehen soll um ihre Hörerschaft in den Unterricht miteinzubeziehen. Die recherchierten Vorteile und Voraussetzungen für Audience Response Systeme dienen als Grundlage für die Anforderungsanalyse. Hierbei sollten einerseits Vorzüge vollständig abgebildet und andererseits Schwierigkeiten, wie sie bei der Verwendung eines ARS entstehen können, möglichst ausgehebelt werden. Es stellte sich heraus, dass das zu entwickelnde ARS sowohl in der Bedienung als auch in der Einrichtung und Wartung benutzerfreundlich gestaltet sein muss, damit der Nutzen in der Lehre möglichst hoch ist.

Die Konzeption des Softwaresystems zeigt, wie dieses als Ganzes abstrakt modelliert wird und wie die Entscheidungungen für oder gegen gewisse Technologien getroffen werden und welche weitreichenden Auswirkungen diese Entscheidungen haben. Die Wahl fiel auf eine Webanwendung, da diese ohne vorherige Installation von jedem browserfähigen Endgerät genutzt werden kann. Der Datenaustausch mit dem Server erfolgt über fest definierte Schnittstellen auf Basis des WebSocket Protokolls, so kann die Anwendung zu einem späteren Zeitpunkt gegebenenfalls um weitere Clients erweitert werden. Weiterhin wird die Serveranwendung nach dem SaaS-Modell bereitgestellt, wodurch der Aufwand für die Einrichtung des Systems weiter sinkt. Das Design der Anwendung nutzt die Gewohnheiten und Erfahrungen der Benutzer, sodass diese sich schnell in der Anwendung zurechtfinden. Weiterhin adaptiert sich die optische Gestaltung der Anwendung an die verschiedenen Bildschirmgrößen, um so eine gute Darstellung auf allen Endgeräten zu ermöglichen.

Nachfolgend wird dieses System in zwei kleinere, in sich geschlossene Systeme aufgeteilt – Frontend und Backend – und die konkrete Umsetzung dieser beschrieben. Jedoch wird kein Quellcode gezeigt, sondern mithilfe von Sequenz- und Klassendiagrammen sowie Interfaces aufgezeigt wie diese Systeme auf eine abstrakte Art und Weise operieren. Denn die Programmiersprache, die Laufzeitumgebung und die verwendeten Bibliotheken und Frameworks sind auf lange Sicht irrelevant. Die Technologie, vor allem aber die Frameworks sind bei dieser rasanten Entwicklung des Internets schnell Schnee von gestern. Deshalb wurde in diesem Kapitel der Fokus einerseits auf die Wahl und die Begründung der verwendeten Entwurfsmuster, und andererseits auf die Beschreibung der Architektur und der Eigenschaften des Systems gelegt. Diese Arbeit zeigt keinen Königsweg auf wie man Software programmiert. Es zeigt einen möglichen Weg, denn viele Komponenten – sei es *RethinkDB*, *Node.js* oder die *ampersand* Pakete – sind einfach austauschbar.

Der anschließende Praxistest zeigt auf, dass die Anwendung trotz kleinerer Feh-

ler im Unterricht eingesetzt werden kann und wie wichtig das Feedback der Nutzer für eine Anwendung ist. Die Benutzer konnten die Anwendung ohne vorige Schulung bedienen und haben sie als Bereicherung für den Unterricht wahrgenommen. Die Anwendung kann die Kommunikation zwischen Vortragenden und Hörern erleichtern. Der Fokus der Anwendung liegt aber nicht nur auf der Vereinfachung der Kommunikation, sondern vor allem auch auf der unkomplizierten Nutzung des Systems. So kann der Vortragende ohne technische Hürden voll und ganz auf seine Hörer eingehen und mit ihnen zusammen die Veranstaltung gestalten.

Betrachtet man abschließend die Anwendung, wird auch dank dem Feedback der Benutzer, ersichtlich, dass noch viele Möglichkeiten für Erweiterungen bestehen, um so die Bedürfnisse der Benutzer noch besser abbilden zu können. In meinen Augen kann sie jedoch auch in der bestehenden Fassung eine Bereicherung im primären und sekundären Bildungsbereich sein. Daneben halte ich auch einen Einsatz auf Tagungen, Fachkongressen oder Seminaren durchaus für denkbar.

Letztlich liegt die Entscheidung beim Dozenten ob er die Anwendung als Unterstützung für seinen Unterricht nutzt oder nicht. Jedoch stellt sie eine minimalistische und einfache Alternative zu den bereits bestehenden Systemen dar, zu welchen sie sich in einigen Punkten bewusst unterscheidet. Sie ist ein Werkzeug, um den Unterricht zu gestalten und eine der Möglichkeiten, wie der Unterricht von den neuen Technologien die das Internet mit sich bringt, profitieren kann.

A Anhang

A.1 Inhalte der DVD

- **/latex**: In diesem Ordner befindet sich der LaTeX-Quelltext zur Erstellung dieses Dokuments, inklusive der Bilder die in dieser Arbeit verwendet wurden.
- **/whitedesk**: In diesem Ordner befindet sich der Quellcode der Anwendung und die Datei **README.md**, dort sind alle Hinweise zur Installation der Anwendung hinterlegt. Das Projekt ist auch auf *GitHub* unter `https://github.com/moolen/thesis` verfügbar.
- **/BA_MBeller.pdf**: Diese Bachelorarbeit als PDF Dokument.

Abbildungsverzeichnis

Literaturverzeichnis

Abrahamson, L., 2006, „A brief history of networked classrooms: Effects, cases, pedagogy, and implications“.

Banks, D. A., 2006, „Reflections on the use of ARS with small groups“

Becker & Gordon, 1973, „Voting machine“, https://www.google.com/patents/US3766541, letzter Zugriff: 11.03.2016

Bergtrom, G., 2006, „Clicker sets as learning objects. Interdisciplinary Journal of Knowledge and Learning Objects“, http://ijklo.org/Volume2/v2p105-110Bergtrom.pdf, letzter Zugriff: 07.03.2016

Bonwell, Charles C. & Eison, James A., 1991, „Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports“

Caldwell, J. E., 2007, „Clickers in the large classroom: Current research and best-practice tips“

Draper, S. W., & Brown, M. I., 2004, „Increasing interactivity in lectures using an electronic voting system.“*Journal of Computer Assisted Learning*

Dukette, D., Cornish, D., 2009, „The Essential 20: Twenty Components of an Excellent Health Care Team“

Medienpädagogischer Forschungsverbund Südwest, 2014, http://www.mpfs.de/fileadmin/JIM-pdf14/JIM-Studie_2014.pdf, letzter Zugriff: 06.03.2016

Judson, E. & Sawada, D., 2006, „Audience Response Systems: Insipid Contrivanoes or Inspiring Tools?“

Kendrick, Roger A., 2010, „USING AN AUDIENCE RESPONSE SYSTEM (ARS) A.K.A. „CLICKER“ TO DO ATTENTION RESEARCH“

Simpson, V., & Oliver, M., 2007, „Electronic voting systems for lectures then and now: A comparison of research and practice“, *Australasian Journal of Educational Technology*

Van Dijk, L. A., Van Den Berg, G. C., & Van Keulen, H. (2001). „Interactive lectures in engineering education“, *European Journal of Engineering Education*

Berners-Lee, T., Fielding, T., Frystyk, H., Hypertext Transfer Protocol – HTTP/1.0, 1996, https://tools.ietf.org/html/rfc1945, zuletzt Abgerufen am 10.03.2016

Fielding, R., UC Irvine, Gettys, J., Compaq/W3C, Mogul, J., Frystyk, H., W3C/MIT, Masinter, L., Xerox, Leach, P., Microsoft, Berners-Lee, T., W3C/MIT Hypertext Transfer Protocol – HTTP/1.1, 1999, https://www.ietf.org/rfc/rfc2616.txt, zuletzt Abgerufen am 12.03.2016