Konzeption und Realisierung eines mit Web Audio entwickelten, browserbasierten Audio Editors

Bachelor-Thesis

zur Erlangung des akademischen Grades B.Sc.

Jakob Sudau

2028702

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

Erstprüfer: Prof. Andreas Plaß

Zweitprüfer: Prof. Thomas Görne

Hamburg, 28. 8. 2015

"I happen to think that computers are the most important thing to happen to musicians since the invention of cat-gut which was a long time ago."

Robert Moog, 1990

Zusammenfassung

In der heutigen Welt der Computer spielt das Internet und die Mobilität eine immer größere Rolle - Computer werden mobiler, Smartphones, Tablets oder kompakte Notebooks ersetzen mehr und mehr die klassischen Workstation Computer - im privatem sowie geschäftlichen Bereich. Das Internet wird durch neue Technologien immer leistungsfähiger und durch den Ausbau der Dateninfrastruktur sowie dem kontinuierlich weiter verbreiteten mobilen Internet immer allgegenwärtiger.

Die Webentwicklung hat in den vergangen Jahren große Fortschritte gemacht: von meist statischen, informativen Websites hin zu interaktiven Single-Page Web Applikationen wie Google Docs, sozialen Netzwerken und Chat-Clients wie Facebook und Tools wie Online-Kalender, Karten, Mail-Clients oder Browsergames.

Die in den letzten Jahren entwickelte Web Audio API ermöglicht es, Klänge und Töne im Browser zu generieren, zu bearbeiten und mit umfangreichen Funktionen diese in Websites und Web Applikationen zu integrieren. Hierdurch öffnen sich viele neue Möglichkeiten Audio im Browser leistungsfähig und umfangreich umzusetzen.

Diese Bachelorarbeit verbindet die aufgeführten, aktuellen Themen und beschreibt die Entwicklung eines browserbasieten Audio Editors von der Konzeption über die Entwicklung bis zur Fertigstellung. Es werden mit Hilfe von aktuellen Technologien wie der Web Audio API, HTML Canvas Elementen oder responsive Webdesign eine moderne, client-basierte Web Applikation erstellt und im Detail beschrieben. Es wird auf Audiovisualisierungsfunktionen, Scheduling und Timing, Track- und Waveformmanagement sowie Designaspekte eingegangen sowie ein Ausblick auf zukünftige Entwicklungen gegeben.

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung

Die Landschaft der Musikproduktion hat sich in den letzten 20 Jahren stark gewandelt. Mit dem Einzug der Computer in den Alltag der Endverbraucher haben sich neue Entwicklungen der digitalen Musik- und Audioverarbeitung bis in die Wohn- und Schlafzimmer verbreitet. Was zuvor nur den großen Tonstudios mit professioneller Ausrüstung möglich war, wurde dem Markt mehr und mehr eröffnet: das Aufnehmen, schneiden und bearbeiten von Tonspuren, Musikinstrumenten und Audiosignalen. Mehr und mehr leistungsfähige digitale Instrumente machen den Computer heute zu dem Zentrum jeder Musikproduktion. Immer mehr Musiker, Journalisten oder Produzenten greifen auf Tonaufnahmesoftware als Arbeitsmittel ihrer Wahl zurück.

Eine noch beachtenswertere Entwicklung hat das Internet in den letzen Jahren und Jahrzehnten vollzogen: die Verbreitung und Zugänglichkeit in alle Regionen der Welt schreitet stetig voran und mehr und mehr Menschen nutzen das Internet täglich, um umfassende Aufgaben zu bewältigen.

In den vergangenen Jahren gab es einen großen Sprung bei der Leistungsfähigkeit des Internets: vorangetrieben von den Browserherstellern wird der technologische Fortschritt immer größer und neu etablierte APIs wie die WebSocket, WebGL oder Web Audio API ermöglichen es, von grafischem 3D Rendering über real-time-collaboration bis zur Audioverarbeitung viele Bereiche, die bis jetzt betriebssystemnativen Programmen vorbehalten waren, auch Web basiert im Browser auszuführen Wyse & Subramanian (2014).

1.2 Idee

Die Idee dieser Bachelorabeit vereint zwei neue technologische Entwicklungen: zum einen die stetig erweiterte Funktionalität des Internets und der Browser, zum anderen die Mobilität der Musikproduktion. Es wird ein Audio Editor mit den Grundansätzen einer DAW erarbeitet und programmiert. Bei dem Programm soll dabei grundlegend über ein eingebautes oder angeschlossenes Mikrofon aufgenommen und in einer Editor Übersicht bearbeitet werden können. Das Programm soll durch ein einfaches und modernes Design einen klaren und leichten Einstieg bieten. Die in den letzten Jahren eingeführte und stetig weiterentwickelte Web Audio API (MDN (2015)) ermöglicht hierfür den tiefen Zugriff auf Audiodaten und Aufnahmemöglichkeiten. Heute ist die Web Audio API Bestandteil aller modernen Browser und kann trotz der jungen Entwicklung als stabil und einsetzbar betrachtet werden.

1.3 Struktur

Nach dem ersten Einführungsteil dieser Bachelorarbeit wird in dem **2. Kapitel** die Anforderungsanalyse aufgeführt und damit die Motivation, Rahmenbedingungen, Zielgruppen und Plattform dieser Thesis erläutert.

Der **Abschnitt I** beschreibt die dieser Bachelorarbeit zugrunde liegende Theorie und beleuchtet die Hintergründe der Audioproduktion, der technischen Entwicklung der Web Audio API und weiteren Komponenten wie dem HTML Canvas Element.

Anschließend wird im **Abschnitt II** die Konzeption des entwickelten Programms dargelegt; von den Funktionen über das Softwaredesign bis zu dem Design der Benutzeroberfläche und des Logos.

Als Hauptteil dieser Bachelorarbeit behandelt **Abschnitt III** die Realisierung und Programmierung des Audio Editors. Hierbei werden alle Teilaspekte des Programms erläutert und so die Umsetzung dargelegt. In den Kapiteln dieses Abschnitts werden die Funktionen der Audioverarbeitung, Audiovisualisierung, Zeitumsetzung sowie Hauptfunktionen erklärt. Im letzen **Abschnitt IV** dieser Thesis wird das Fazit der Arbeit gezogen. Hierbei wird das entwickelte Programm ausgewertet und eine mögliche Erweiterbarkeit vorgestellt. Ebenfalls wird ein Ausblick auf die Zukunft der Audioverarbeitung im Zusammenhang mit den Themen der Bachelorarbeit gegeben.

2 Anforderungsanalyse

2.1 Motivation

Die Öffnung des Audio- und Musikproduktionsmarktes verbunden mit der technologischen Entwicklung des Internets hat in den letzten Jahren viele Trends, Ansätze und Firmen hervorgebracht. So werden heute auf Plattformen wie Soundcloud oder bandcamp Songs und Musik direkt vom Künstler vertrieben, Streaming-Portale wie Spotify oder Apple Music haben den Musikmarkt revolutioniert und durch neue Entwicklungen sowie Design- und Usabilityreformen bei Musikproduktionsprogrammen wie Apple Logic Pro X oder GarageBand ist die Möglichkeit, Audio und Musik aufzunehmen, leichter denn je.

Immer mehr Anwendungen der kreativen Branche werden in das Web ausgelagert und die Möglichkeiten, cloud-basierte Dienste anzubieten und somit Vorteile wie standardisierte Versionskontrollen, zentrale Verwaltung der Software oder kollaboratives Arbeiten zu nutzen, eröffnen neue Arbeitsweisen und eine neue Produktivität.

2.2 Zielgruppe

Das in dieser Thesis erarbeitete Programm soll sich an Nutzer wenden, die einen schnellen und unkomplizierten Einstieg und stetigen mobilen Zugriff unabhängig des Endgerätes benötigen. Das Programm bietet hierfür grundlegende Funktionalität, um Audiodaten aufzunehmen und zu bearbeiten. Mögliche Anwendergruppen und Profile sind hierfür anfängliche bis fortgeschrittene Musikproduzenten, die nach einer einfachen und unkomplizierten Möglichkeit des Aufnehmens suchen, Songwriter, die schnell erste Ideen festhalten oder vorproduzieren wollen, oder Journalisten, die nach einer klaren und simplen Möglichkeit suchen, mobil und unabhängig der vorhandenen Geräte Mitschnitte oder Aufnahmen zu machen.

2.3 Funktionen & Umfang

Nach einer anfänglichen Recherche- und Testphase, in der verschiedene, bereits etablierte Programme wie Apple Logic Pro X, Apple Garageband, Steinberg Cubase und Avid Pro Tools (siehe 2.1) verglichen und getestet wurden, wurde eine umfangreiche Liste an Funktionen zusammengestellt, welche für den Nutzer und den Workflow des Programms essentiell sind.

Die vereinfachte Hauptfunktionalität einer jeden dieser DAWs[1] teilt sich dabei in drei Bereiche auf: den Hauptbedienelementen (siehe 6.2 A) am oberen Bildschirmrand, den Spurbedienelementen (siehe 6.2 B) am linken Bildschirmrand und dem Arrangierfenster (siehe 6.2 C) auf der rechten Seite des Bildschirms. In der Vergleichsgrafik A.4 sind die Bereiche in den verschiedenen DAWs farblich gekennzeichnet und mit (A) für die Hauptbedienelemente, (B) für die Spurbedienelemente und (C) für das Arrangierfenster beschriftet. Diese Funktionalitätsaufteilung lässt sich in den meisten Audiobearbeitungsprogrammen finden und wird auch in dem in dieser Bachechelorarbeit entwickelten Programm umgesetzt, um dem Nutzer eine bestmögliche, umfangreiche und vertraute Lösung zu bieten.

Die aufgelisteten DAWs verfügen alle über eine weitaus größere Funktionalität wie einem Mixer, Bus- und Routingmöglichkeiten, Automation, oder Plugin- und MIDI-Support. Um diese Bachelorarbeit in einem überschaubaren Rahmen zu halten und die Funktionalität klar und simpel zu halten, wird nur die elementare Funktionalität einer DAW implementiert und erläutert.

2.4 Arrangierfenster

Dieses Fenster ist die zentrale Arbeitsfläche des Programms. Hier kann der Nutzer die aufgenommenen Töne, welche als Waveform[2] visualisiert werden, schneiden oder innerhalb einer und zwischen mehreren Spuren verschieben. Das Arrangierfenster soll sich in der Breite über die zeitliche Länge des Projektes definieren und über eine Zeitleiste verfügen, welche am oberen Rand des Fensters lokalisiert ist, den zeitlichen Verlauf des Projektes darstellt und die jeweils aktuelle Abspielposition anzeigt.

[1] Digital Audio Workstation
[2] Zeitlicher Verlauf einer Schwingung

Waveforms

Die Waveforms sind die visuelle Darstellung der aufgenommenen Töne. Diese sollen frei im Arrangierfenster verschiebbar sein und somit dem Nutzer eine freie zeitliche Zusammensetzung der aufgenommenen Waveforms ermöglichen. Der Nutzer soll eine Waveform an beliebiger Stelle schneiden können.

Zeitleiste

Die Zeitleiste erstreckt sich über die gesamte Länge des Arrangierfensters und soll den zeitlichen Verlauf in Takten, welche nummeriert sind, anzeigen. Die Skalierung soll sich entsprechend der von dem Nutzer in den Hauptbedienelementen gesetzten bpm[3] ergeben und bei hoher Zoomstufe auch halbe Takte und Vierteltakte anzeigen.

Abspielposition

Die aktuelle Abspielposition soll über eine dünne vertikale Linie dargestellt werden, die sich über das gesamte Arrangierfenster streckt. Diese Linie soll beim Abspielen oder bei der Aufnahme stets die jeweils aktuelle Abspielposition anzeigen und sich somit im Arrangierfenster von links nach rechts bewegen. Ebenfalls soll die Abspielposition durch den Nutzer frei verändert werden können, indem dieser auf einen beliebigen Zeitpunkt in der Zeitleiste klickt.

Gitternetzlinien

Zur präzisieren Positionierung der Waveforms durch den Nutzer soll ein Gitter vorhanden sein, welches sich, je nach Zoomstufe, variabel auf Takte, Halbtakte oder Vierteltakte einstellt. Die Gitternetzlinien sollen hierbei nur vertikal verlaufen und die Waveforms sollen sich, ab einem bestimmten, geringen Abstand zu einer Gitternetzlinie, an diesen orientieren. Die Gitternetzlinien sollen für den Nutzer dezent visuell dargestellt sein.

[3] beats per minute

2.5 Hauptbedienelemente

Die Hauptbedienelemente dienen dem Nutzer der allgemeinen Steuerung und Benutzung des Programms und sollen folgende Funktionen beinhalten.

Projekt Name

Der Nutzer soll den Namen des aktuell bearbeiteten Projektes festgelegen können, um Übersichtlichkeit, eine klare Zuordnung sowie Bezug zu dem Projekt zu schaffen. Der Name des Projektes soll anschließend auch dem Download dienen und die vom Nutzer runtergeladene Datei korrekt beschriften.

Play

Mit dieser Funktion soll der Nutzer die Wiedergabe des Projektes starten können.

Pause

Die Funktion Pause ermöglicht das Pausieren der aktuellen Wiedergabe, während der Aufnahme wird diese gestoppt.

Stop

Die Funktion Stop hält die aktuelle Wiedergabe des Projektes an. Im Gegensatz zur Pause Funktion wird hierbei jedoch die aktuelle Abspielposition auf den Anfang des Projektes zurückgesetzt. Während der Aufnahme wird durch die Funktion Stop diese gestoppt, die aktuelle Abspielposition jedoch nicht auf Anfang zurückgesetzt, da sich das Stoppen in diesem Moment auf die Aufnahme und nicht auf das Projekt bezieht.

Record

Durch die Funktion Record kann der Nutzer eine Aufnahme starten. Hierbei wird in der aktuell ausgewählten Spur eine neue Waveform erzeugt und die Mikrofoneingabe in dieser gezeichnet.

Spur hinzufügen

Der Nutzer kann eine neue Spur erzeugen und dem Projekt hinzufügen.

Zeitanzeige

Über die Zeitanzeige soll die aktuelle Zeit und der aktuelle Takt angezeigt werden und somit die Orientierung im Projekt gegeben werden.

Metronom

Der Nutzer soll zur Wiedergabe und Aufnahme ein Metronom ein- und ausschalten können. Dieses kann in der Geschwindigkeit variabel eingestellt werden und orientiert sich an den Takten der Zeitleiste und der aktuellen Abspielposition.

Master Lautstärke

Über diese Funktion soll der Nutzer die Hauptlautstärke des Projektes einstellen und diese ebenfalls stummschalten können.

Vollbildmodus

Der Nutzer soll die Möglichkeit haben, das Programm im Vollbildmodus auszuführen und kann somit eine höhere Produktivität durch mehr Arbeitsfläche erreichen.

Download

Mit der Download Funktion soll der Nutzer sein aktuelles Projekt als Datei im WAVE[4]-Format[5] herunterladen können.

Help

Der Nutzer soll Zugang zu einem Hilfebildschirm haben, welcher die Funktionalität der einzelnen Elemente erklärt und weitere Informationen angibt.

Zoom

Das Programm soll über eine Zoom-Funktionalität verfügen, über welche der Nutzer die Ansicht im Arrangierfenster vergrößern und verkleinern kann.

[4] https://msdn.microsoft.com/en-us/library/windows/hardware/dn653308(v=vs.85).aspx (geprüft 9.8.2015)

[5] Dateiformat zum Speichern von digitalen Audiodaten

2.6 Spurbedienelemente

Die Spurbedienelemente dienen dem Nutzer zur Steuerung und Bedienung der einzelnen Spuren.

Spurname

Für die Organisation und Übersichtlichkeit des Projekts soll der Nutzer die Möglichkeit haben, jeder Spur einen Namen zu geben.

Spur entfernen

Der Nutzer soll jede Spur löschen können, bis nur noch eine Spur im Projekt vorhanden ist.

Spurlautstärke

Die Lautstärke der Spur kann vom Nutzer gesteuert und stumm geschaltet werden.

Panning

Jede Spur soll die Möglichkeit des Stereopannings[6] besitzen. Hierbei kann der Nutzer die Ausgabe der Spur auf den linken und rechten Stereokanal einstellen.

Solo

Jede Spur soll eine Solo Funktion beinhalten. Sobald der Nutzer eine oder mehrere Spuren des Projekts auf Solo stellt, werden nur noch diese Spuren abgespielt. Das Stummschalten einer Spur beeinflusst die Solo Funktion nicht, die Solo Wiedergabe des Projektes steht in der Hierarchie über dem Stummschalten der Spuren.

Mikrofonwiedergabe

Der Nutzer soll das Mikrofon auf der aktuellen Tonausgabe aktivieren können. Damit kann der Nutzer mit Kopfhörern die Mikrofoneingabe und das Metronom hören, ohne dass es auf das Mikrofon überspricht. Sind jedoch das Mikrofon und die Tonausgabe zu nah bei einander, kann es zu Übersteuerungen kommen.

[6] http://www.dawsons.co.uk/blog/what-is-panning (geprüft 18.8.2015)

Eingabedarstellung

Die aktuelle Eingabe einer jeden Spur soll visuell dargestellt werden. Dies kann entweder die Mikrofoneingabe sein, oder die abzuspielenden Waveforms während der Wiedergabe. Der Dezibelpegel der Eingabe soll sowohl grafisch, als auch numerisch dargestellt werden um den Nutzer bessere Analyse- und Kontrollmöglichkeiten zu ermöglichen.

2.7 Plattform

Als Plattform wird bei diesem Programm der Webbrowser gewählt. Über 40 Prozent der Weltbevölkerung[7] hat Zugriff auf einen Internetzugang und moderne Browser wie Chrome und Firefox haben einen Marktanteil von über 54 Prozent[8]. Durch HTML5 und CSS3 neu eingeführte Standards erlauben eine leistungsfähige und umfangreiche Programmierung und Gestaltung WC3 (2014). Der Hauptteil des Programms wird in Javascript geschrieben und mit der seit 2012 eingeführten Web Audio API realisiert, weitere Mittel sind die navigator.getUserMedia() API und das HTML Canvas Element.

Das Programm soll leichtgewichtig sein und auf ein Back-End oder Server verzichten. Die Anwendung wird client-basiert konzeptioniert und umgesetzt, wodurch eine für dieses Programm und den Anwendungsbereich der Nutzer wichtige Offline-Funktionalität erreicht wird und die Web App überall eingesetzt werden kann, unabhängig der Verbindung und Verfügbarkeit des Internets.

Das Programm soll kompatibel mit den Browsern Google Chrome, Mozilla Firefox und Opera unter Windows, Mac und auf Android Smartphones und Tablets sein.

2.8 Markt & Konkurrenz

Die auf dem Musikproduktionsmarkt am meisten genutzten Programme sind Avid Pro Tools, Steinberg Cubase, Ableton Live und Apple Logic Pro X. Neben diesen marktführenden DAWs existieren noch weitere vollwertige Programme mit großem

[7] http://www.internetlivestats.com/internet-users/ (geprüft 6.8.2015)

[8] http://gs.statcounter.com//#all-browser-ww-monthly-201508-201508-bar (geprüft 6.8.2015)

Funktionsumfang wie FL Studio oder Propellerhead Reason. Erste Anwendungen wie Ohm Studio oder Steinbergs VST Connect Pro[9] versuchen bereits, einen kollaborativen oder cloud-basierten Ansatz zu integrieren. All diese Programme sind jedoch native Standalone-Programme von meist weit über 3 GB Größe. Webbasierte Dienste wie Audiotool oder AudioSauna haben einen ähnlichen Funktionsumfang, benötigen jedoch keine weitere Installation und sind über den Browser abrufbar. Diese Web Apps basieren jedoch auf dem sicherheitsanfälligen Flash Plugin, wessen Unterstützung durch Webbrowser stark rückgängig ist.

Im Jahr 2014 veröffentliche das schwedische Start-Up SoundTrap (ehm. PlayWerk) das Web basierte Programm SoundTrap. Hierbei handelt es sich um eine Web Applikation, welche zum ersten Mal moderne Standards wie HTML5 und die Web Audio API benutzt, um auf einer professionellen Ebene ein Musikbearbeitungsprogramm zu veröffentlichen, welches zudem über kollaborative Features verfügt. Alle genannten Programme sind in der Tabelle 2.1 aufgelistet.

Anforderungsanalyse: Markt & Konkurrenz (geprüft am 6.8.2015)

Avid Pro Tools	https://www.avid.com/DE/products/pro-tools-software
Steinberg Cubase	http://www.steinberg.net/fi/products/cubase/start.html
Ableton Live	https://www.ableton.com/de/live/
Apple Logic Pro X	https://www.apple.com/de/logic-pro/
FL Studio	https://www.image-line.com/flstudio/
Propellerhead Reason	https://www.propellerheads.se/reason
Ohm Studio	https://www.ohmstudio.com/
Audiotool	http://www.audiotool.com/
Audiosauna	http://www.audiosauna.com/
Soundtrap	https://www.soundtrap.com/

Tabelle 2.1: Anforderungsanalyse: Markt & Konkurrenz

[9] http://www.steinberg.net/se/products/vst/vst_connect/vst_connect_pro.html (geprüft 7.8.2015)

Teil I

Theorie

3 Audio Produktion

Von den Anfängen der Musikproduktion bis heute haben sich viele der Techniken grundlegend geändert. Das Grundprinzip jedoch, ein zeitbasiertes Medium wie Ton wiederholt hörbar zu machen, ist im Kern stets das Gleiche geblieben.

In diesem Kapitel wird ein kleiner Einblick in die Geschichte und die Praktiken der Musikproduktion gegeben, sowie auf die aktuelle Entwicklung der Web Audio API und dem für das Programm wichtige HTML Canvas Element eingegangen. Der Umfang dieser Bachelorarbeit ermöglicht keine tiefergehende, analytische Auseinandersetzung mit diesen Themenbereichen, daher werden diese nur als kleine Ausschnitte, zugeschnitten auf den Inhalt der Bachelorarbeit, wiedergegeben. Die Praktiken werden hierbei durch persönliche Erfahrung meinerseits als Musiker in verschiedenen Musikproduktionen wiedergegeben und beinhalten nur die Teilaspekte einer kompletten Musikproduktion, welche relevant für das geplante Programm sind.

3.1 Geschichte

Die Aufnahmetechnik hat ihre Anfänge um die Wende des 20. Jahrhunderts. Die ersten Versuche, Ton auf einem Medium aufzunehmen, festzuhalten und wieder abspielbar zu machen fanden in dieser Zeit statt. Thomas Edison gelang hierbei 1877 der erste Durchbruch. Bei diesen ersten, rein akustischen Aufnahmen wurde eine Membran durch den durch einen Trichter gesprochenen Ton in Schwingung versetzt. Auf der anderen Seite der Membran befand sich eine stumpfe Nadel, die durch die Bewegung der Membran in Schwingung versetzt wurde und diese Informationen auf eine dünne Zinnfolie schrieb. Die Zinnfolie war auf einem Zylinder angebracht und wurde mittels einer handbetrieben Kurbel gedreht. Anschließend konnte der geschriebene Ton über den umgekehrten Mechanismus abgespielt werden (siehe Abb. 3.1) (Gronow & Saunio 1999: 1).

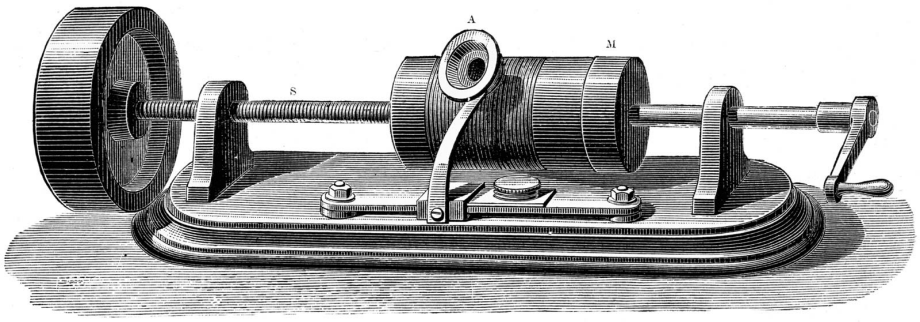

Abbildung 3.1: Erstes Modell eines Edison Phonographs, ca. 1877

Die sogenannten Phonographen wurden bis in die 1920er weiter ausgefeilt und erreichten ihren Höhepunkt in der Entwicklung des Grammophons. Ohne jegliche elektronische Technik wie Mikrofone konnte bei diesem Vorgang jedoch nur eine beschränkte Qualität erreicht werden. Mit der Erfindung und Verbreitung des Mikrofons und elektrischen Verstärkern in den 1920ern und Folgejahren wurde der Aufnahmeprozess von einem rein elektronischen abgewechselt und die Musikindustrie erreichte in den USA und Europa ein neues Hoch. Audio Engeneering Society (2005)

Beschränkt durch die technischen Möglichkeiten, nur eine Spur und ein Mikrofon aufzunehmen, wurden große Tonstudios errichtet, um, von kleineren Jazz Ensembles bis zu großen Symphonie Orchestern, Musik aufzunehmen. In dieser Zeit spezialisierte sich die Tontechnik auf verschiedene Mikrofontypen wie Kondensatormikrofone, dynamische Bändchenmikrofone oder Kohlemikrofone und deren Zusammenspiel in großen Räumen, um eine bestmögliche Aufnahmequalität zu erreichen. Neue Entwicklungen von Klangbearbeitungsgeräten wie Kompressoren und Equalizern ermöglichten einen weiteren Sprung in der Aufnahmequalität und neue Tonbearbeitungsmöglichkeiten (Gronow & Saunio 1999: 36 ff.).

In den 1950ern wurde erstmals eine neue Technik der Tonaufnahme eingeführt, die sich bis in die späten 1970er als Standard etablierte: die Aufnahme auf magnetischen Tonbändern mit Hilfe von Tonbandgeräten. Die bereits zu Beginn des 20. Jahrhunderts von Oberlin Smith erfundene und durch die deutsche Firma AEG in den 1930ern weiterentwickelte Technologie ermöglichte eine weit höhere Qualität sowie die Möglichkeit, mehrere Spuren auf einem Band aufzunehmen, wodurch erste Möglichkeiten der Stereoaufnahme entstanden und der Beginn der neuzeitigen Mu-

sikproduktion eingeläutet wurde: die Mehrspuraufnahme (Gronow & Saunio 1999: 96 ff.).

Durch Mehrspurrekorder mit 4 bis zu 24 Spuren konnten nun Instrumente separat aufgenommen werden und großräumige Tonstudios wichen mehr und mehr kleineren Studios mit separaten und isolierten Gesangs- und Instrumentenkabinen. Ebenfalls wurde durch die magnetischen Tonbänder die Möglichkeit eingeführt, bereits aufgenommene Bänder zu überschreiben und somit Fehler, die während der Aufnahme oder des Spielens entstanden sind, zu berichtigen. Auch war das physikalische Schneiden einzelner Teile möglich um somit verschiedene Takes zusammenzuführen. Durch diesen technisches Fortschritt wurde auch die Overdubbing-Technik, welche in den Jahren zuvor nur durch einen aufwendigen Prozess, in dem die Spuren direkt auf eine Vinylplatte gepresst wurden, umgesetzt war, stark vereinfacht und findet bis heute eine weite Verbreitung. Diese Technik beinhaltet das Übereinanderlegen von Spuren und Takes, wodurch neue kreative Möglichkeiten erschaffen werden, wie einen Chor mit nur einer Person aufzunehmen oder eine Song in voller Bandbesetzung mit nur einem Multiinstrumentalist zu realisieren Audio Engeneering Society (2005).

In den späten 1970ern wurde schließlich die digitale Aufnahmetechnik auf dem kommerziellen Markt eingeführt. Vorangetrieben von Firmen wie Denon werden bei dieser Technik Audio Signale in digitale Zahlenströme konvertiert und im PCM Format auf einem Speichermedium gespielt. Hierfür genutzte Formate wie das Digital Audio Tape wurden in den 1990ern in der Aufnahmetechnik von Computern abgelöst. Größere Festplattenkapazitäten sowie schneller CPUs ermöglichen eine immer leistungsfähigere Aufnahmeumgebung mit fast unendlich vielen Spuren, digitalen Effekten und Bearbeitungsmöglichkeiten.

3.2 Praktiken

Ton und Musik sind ein zeitbasiertes Medium. Bei der heutigen üblichen Praxis werden hierbei digital an einem PC in mehreren Spuren mehrere Mikrofone entweder gleichzeitig oder nach und nach übereinander aufgenommen. Dies kann entweder als kompletter Take oder in einzelnen Abschnitten passieren, die anschließend zusammengefügt werden. Einzelne aufgenommene Abschnitte können verschoben, geschnit-

ten und kopiert oder gelooped[1] werden. Jede aufgenommene Spur kann anschließend mit Effekten versehen, in Lautstärke und anderen Parametern bearbeitet und in allen Aspekten zeitbasiert automatisiert werden. Weitere Möglichkeiten über diesen grundlegenden Prozess hinaus beinhalten u.a. ein virtuelles oder analoges Mischpult, an dem alle verfügbaren Spuren in Kanalzügen bearbeitet und in Parametern, wie der Lautstärke, angepasst werden können.

[1] in einer Schleife beliebig oft wiederholt

4 Programmierung

4.1 Web Audio API

Die Audioentwicklung im Web hat in den letzen Jahren große Fortschritte gemacht. Nachdem in den anfänglichen Jahren von HTML schnell Bedarf an multimedialen Inhalten auf Websites entstand, wurden, nach den ersten Versuchen von Microsoft mit dem bgsound Element[1], als browserübergreifende Lösungen externe Plugins angeboten, welche Sounds und Videos abspielen konnten. Das am weitesten verbreitete Plugin heißt Flash und wird noch heute auf vielen Websites, trotz vieler Sicherheitslücken und kostenpflichtiger Lizensierung, verwendet. 2009 wurde von Firefox erstmals das `audio` Element[2] eingeführt, welches eine native Unterstützung zum Abspielen von Sounds ermöglichte. 2014 wurde dieses Element in die, in dem Jahr veröffentlichte, HTML5 Spezifikation aufgenommen und wird inzwischen von allen Browsern unterstützt WC3 (2014).

Dieses Element gab Entwicklern erste Möglichkeiten, Audiodateien in Websites zu integrieren. Da sich dieses Element jedoch nur auf das Laden und Abspielen von Audiodateien beschränkt, entstand bald der Bedarf, diese Audiodateien auch zu bearbeiten, visualisieren oder zu erzeugen. Hierfür entwickelte Chris Rodgers 2012 die Web Audio API.

Die Web Audio API ermöglicht tiefgreifenden Zugriff auf Audiodateien und deren Bearbeitung sowie die synthetische Generierung von Sounds. Die API verfügt ebenfalls über eine umfangreiche Audio Analyse, Timing Funktionalität sowie Mixing, Processing und Filtering Möglichkeiten und erschafft somit die Möglichkeit, komplexe Audio Anwendungen im Web zu implementieren Adenot & Wilson (2015).

Heute ist die Web Audio API in den meisten Browsern wie Chrome, Firefox und

[1] `https://msdn.microsoft.com/library/ms535198(v=vs.85).aspx` (geprüft 6.8.2015)
[2] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio` (geprüft 6.8.2015)

Opera vollständig und in Safari und Internet Explorer teilweise unterstützt.

4.1.1 Audio Nodes & Graphs

Die Web Audio API ist als Node-basierte API realisiert. Es wird ein AudioContext Objekt erstellt und an diesem alle weiteren Nodes wie Oszillatoren, Soundbuffer, Analyser, Filter oder Effekte zusammengeschlossen. Durch ein Verbinden der Nodes mit der `AudioContext.destination` wird anschließend die Soundausgabe gesteuert. Hierdurch kann komplexe Soundanalyse oder Soundbearbeitung implementiert werden Smus (2013).

In dem folgenden, einfachen Beispiel wird ein Ton über einen Oszillator mittels der Web Audio API ausgegeben. Als erstes wird ein `AudioContext` erzeugt und als Variable gespeichert. Anschließend wird ein Oszillator und eine Gainnode zum Steuern der Lautstärke erzeugt. Der Oszillator wird mit einer Frequenz und die Gainnode mit einem Lautstärkewert von 0 bis 1 initialisiert. Der Oszillator wird nun an die Gainnode angeschlossen, die Gainnode mit der Soundausgabe über `AudioContext.destination` verbunden und anschließend wird der Oszillator mit dem Funktionsaufruf `.start()` gestartet (siehe Listing 4.1).

Listing 4.1: Web Audio Beispiel

```
1    var audioCtx = new AudioContext();
2    var oscillatorNode = audioCtx.createOscillator();
3    var gainNode = audioCtx.createGain();
4    gainNode.gain.value = 0.8;
5
6    oscillatorNode.connect(gainNode);
7    gainNode.connect(audioCtx.destination);
8    oscillatorNode.start();
```

4.2 getUserMedia() API

Eine weitere, für dieses Projekt wichtige API ist die navigator.getUserMedia() API. Der Bedarf auf native Funktionen des Anwendungsgerätes wie GPS Daten, Orientierung oder GPU zuzugreifen, ist mit dem stetigen Funktionsumfang von Web Apps

immer mehr gestiegen. So wurde 2011 von dem W3 Consortium[3] ein Arbeitsgruppe zur Standardisierung der verschiedenen Media Capture APIs, welche die eben genannten Zugriffe steuern, gegründet. Die Device APIs Working Group spezifizierte die navigator.getUserMedia() API, um einen einheitlichen Zugriff auf Webcam und Mikrofon zu etablieren. Nach ersten Versuchen mit HTML Media Capture[4] und dem `<device>`-Element, letzteres welches in den Browsern weitestgehend nicht implementiert blieb, wurde mit der WebRTC[5] der Grundsatz der darauf folgenden getUserMedia API gelegt.

Die navigator.getUserMedia() API kann nativ im Browser ohne Plugins auf Webcam und Mikrofon zugreifen, und seit 2014 ist der Mikrofonzugriff im Zusammenhang mit der Web Audio API benutzbar Burnett & Bergkvist & Jennings (2015). Hierbei wird ein `MediaStreamAudioSourceNode`[6] Objekt erstellt, welches den in der `navigator.getUserMedia()` Funktion überreichten `stream` weiterverarbeiten kann (siehe Listing 4.2). In dem Beispielcode 4.2 wird diese Node über einen Filter mit der Tonausgabe verbunden.

Listing 4.2: navigator.getUserMedia() API Beispiel

```
1  var audioCtx = new AudioContext();
2  navigator.getUserMedia({audio: true}, function(stream){
3      var microphone = audioCtx.createMediaStreamSource(
           stream);
4      var filter = audioCtx.createBiquadFilter();
5
6      microphone.connect(filter);
7      filter.connect(audioCtx.destination);
8  }, function(){alert("error");});
```

[3] Internationales Gremium zur Standardisierung der Techniken im World Wide Web
[4] http://www.w3.org/TR/html-media-capture/ (geprüft: 3.8.2015)
[5] Web Real Time Communications, http://w3c.github.io/webrtc-pc/ (geprüft 3.8.2015)
[6] https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamAudioSourceNode (geprüft 3.8.2015)

4.3 HTML Canvas

Als weiteres wichtiges Element zur Audiovisualisierung wird das HTML Canvas eingesetzt. Mit diesem Element können dynamische, skript-basierte Renderings erstellt werden. Das HTML Canvas wurde 2004 von Apple zunächst als interne Lösung entwickelt und später von der WHATWG[7] als Teil der HTML5 Spezifikation aufgenommen.

Über JavaScript können hierbei in dem `canvas`-Element komplexe geometrische Formen durch Mittel wie Bézierkurven, Kreisbögen, Transformationen, oder Farbverläufe erstellt werden Cabanier & Mann & Munro & Wiltzius & Hickson (2015).

In dem folgenden, einfachen Beispiel 4.3 wird ein rotes Rechteck in ein Canvas gezeichnet. Mit der Funktion `.getContext("2d")` wird ein zweidimensionaler Kontext erzeugt, auf dem gezeichnet werden kann. Mit der `.fillStyle` Eigenschaft kann die Zeichenfarbe gesetzt werden, die `.fillRect(x, y, width, height)` Funktion zeichnet anschließend das Rechteck, wobei sich die vier Parameter aus der x und y Position der oberen linken Ecke des Rechtecks sowie der Breite und Höhe des Rechtecks zusammensetzen.

Listing 4.3: HTML Canvas Beispiel

```
1  <html>
2    <canvas id="myCanvas" width="100" height="50"></canvas>
3    <script>
4      var c = document.getElementById("myCanvas");
5      var ctx = c.getContext("2d");
6      ctx.fillStyle = "red";
7      ctx.fillRect(0,0,10,25);
8    </script>
9  </html>
```

4.4 JavaScript Objektorientierung

JavaScript ist eine dynamische Programmiersprache und keine klassische Objektorientierte Sprache wie C, C++ oder Java. JavaScript ist eine Skriptsprache, die die

[7] Web Hypertext Application Technology Working Group

meisten Ansätze der Objektorientierung wie Polymorphie oder Datenkapselung auf ähnliche Weise wie die klassischen Objekt-orientierten Sprachen implementiert, jedoch den wichtigen Aspekt der Vererbung auf eine für die klassischen Sprachen untypische Art der Prototypbasierten Programmierung[8] umsetzt. In Javascript existieren keine primitiven Klassen, stattdessen werden diese mit Funktionen oder Prototypen umgesetzt Kropff (2014).

In dem Beispielcode 4.4 wird eine einfache Klasse mit Klassenvariablen und Klassenfunktionen mittels einer Konstruktor/Prototyp Kombination in Javascript erzeugt. Zunächst wird die Klasse `Test` mittels der Konstruktorfunktion `function Test(someNumer)` mit dem Parameter `someNumber` erstellt und die Klassenvariable `aNumber` initialisiert. Dies ist der gängigste Weg, eine Klasse in JavaScript zu definieren. Im nächsten Schritt wird der `.prototype` Eigenschaft der Klasse eine Klassenfunktionen hinzugefügt, wodurch die Funktion für alle Instanzen der Klasse verfügbar wird. Hiernach wird in der Variable `aTest` ein neues Objekt der Klasse `Test` erstellet und die Klassenfunktion `addNumber` aufgerufen.

Listing 4.4: Objektorientierung in Javascript Beispiel

```
1  function Test(someNumber){
2      this.aNumber = someNumber;
3  }
4
5  Test.prototype = {
6    constructor: Test,
7
8    addNumber:function(numberToAdd){
9      return this.aNumber += numberToAdd;
10   },
11 }
12
13 var aTest = new Test(5);
14 aTest.addNumber(7); //returns 12
```

In dem Programm AEditor sind meisten Klassen nach diesem Muster programmiert.

[8] klassenlose Programmierung

Teil II

Konzeption

5 Softwaredesign

Bei der Planung der Software wurden Paradigmen erstellt, um eine saubere, übersichtliche und leistungseffiziente Implementierung zu gewährleisten. Hierbei wurde auf folgende Aspekte geachtet:

- Es sollen keine weiteren Assets und Ressourcen außer HTML, Javascript und CSS verwendet werden. Alles Notwendige soll aus dem Code generiert werden um somit ein leistungsfähiges Programm zu entwickeln, ohne Ladezeiten durch Zugriffe auf Bilder oder Ähnliches. Auch auf jegliches Back-End wie serverseitiger Code soll verzichtet werden.
- Es soll, soweit in Javascript realisierbar, objektorientiert programmiert werden.
- Alle weiteren Programmierprinzipien[1] wie Redundanzvermeidung, Abstraktion, Einfachheit oder Single Responsibility sollen weitestgehend befolgt werden.

5.1 Entwicklungsstrategie

Als Entwicklungsstrategie wurde das Bottom-Up-Prinzip[2] gewählt. Dieser Ansatz zur Programmentwicklung beinhaltet die Idee, essentielle Programmteile zu definieren und direkt zu implementieren. Darauf aufbauend werden die nächsten Programmteile realisiert und somit das Programm Stück für Stück entwickelt. Der Bottom-Up Ansatz steht im Gegensatz zu dem Top-Down-Prinzip, bei welchem zunächst alle Teile, Unterteile, Unter-Unterteile usw. des Programms spezifiziert werden um somit einen Überblick zu bekommen und Implementierungen zunächst zu vernachlässigen.

Der gewählte Ansatz bringt die Vorteile, schnellstmöglich mit der Implementierung zu beginnen und jede implementierte Funktionalität bereits im frühen Stadium

[1] http://www.artima.com/weblogs/viewpost.jsp?thread=331531 (geprüft 6.8.2015)
[2] http://www.paulgraham.com/progbot.html (geprüft 19.8.2015)

zu testen. Zudem kann bei diesem Ansatz Code Redundanz vermieden werden, da von Beginn an Code geschrieben wird und Stück für Stück weitere Implementierungen vorgenommen werden Pizka & Bauer (2005). Das Bottom-Up-Prinzip bietet sich vorallem bei kleinen Teams oder, wie in dieser Bachelorarbeit, Einzelpersonen an, die ein komplettes Programm entwickeln. Das Top-Down-Prinzip bietet sich für große Entwicklerteams an, da in diesem Ansatz nach der ersten Spezifikation parallel an Unterteilen des Programms gearbeitet werden kann und durch einheitlich definierte Tests Kontinuität gegeben ist.

5.2 Entwurfsmuster

Als Entwurfsmuster sollen bei dem Programm eine Verbindung aus dem Singleton-Pattern[3] und Prototype-Pattern eingesetzt werden. Durch das Singleton-Pattern kann von einer Klasse nur genau ein Objekt existieren. Dieses Pattern ist hilfreich für Klassen, die Aktionen über das gesamte Programm koordinieren und von denen es nicht vorhergesehen ist, mehrere Instanzen zu besitzen. (Gamma & Helm & Johnson & Vlissides 1995: 144) Der nachfolgende Code[4] 5.1 zeigt die Implementierung des Singleton-Patterns in JavaScript.

Listing 5.1: Singleton Pattern in Javascript

```
1   function MySingletonClass() {
2     if (arguments.callee._singletonInstance)
3       return arguments.callee._singletonInstance;
4     arguments.callee._singletonInstance = this;
5     this.Foo = function() {
6       //...
7     }
8   }
9   var a = new MySingletonClass()
10  var b = MySingletonClass()
11  Print( a === b ); //prints: true
```

[3] https://msdn.microsoft.com/en-us/library/Ee817670(pandp.10).aspx (geprüft 19.8.2015)
[4] https://code.google.com/p/jslibs/wiki/JavascriptTips#Singleton_pattern (geprüft 19.8.2015)

Dieses Pattern soll für die im Abschnitt 5.3 aufgeführte Klasse `App` verwendet werden. Für die restlichen aufgeführten Klassen `Track` und `Waveform` des Programms Soll das Prototyp-Entwurfsmuster[5] verwendet werden. Bei diesem Entwurfsmuster wird eine prototypische Instanz eines Objektes erstellt. Neue Objekte werden erzeugt, in dem das prototypische Objekt kopiert wird. Hierbei wird im klassischen Sinne keine Klasse definiert und der Code kann ressourcensparend implementiert werden, da keine Instanz einer Klasse erzeugt wird(Gamma & Helm & Johnson & Vlissides 1995: 133). Dieser Ansatz ist in JavaScript im Abschnitt 4.4 **JavaScript Objektorientierung** beschrieben und ist gleichzeitig der verbreitetste Weg, in Javascript Klassen und Objekte zu implementieren.

5.3 Klassen-Objekt-Struktur

Das Programm soll sich in fünf Klassen aufteilen: `App`, `Track`, `Waveform`, `metronome` und `recorder`. Die Klasse `Waveform` soll dabei alle Funktionen zur Waveformerstellung und -darstellung beinhalten. Die Klasse `Track` soll alle Funktionen zur Spurverwaltung, Spurerstellung und Spursteuerung behinhalten und die Klasse `App` alle weiteren Funktionen zur Verwaltung und Erstellung des Projekts, sowie Audioerstellungs- und Audiobearbeitungsfunktionen. Die Klasse `metronome` verfügt über den Code des Metronoms und die Klasse `recorder` beinhaltet das Plugin `recorder.js` zur Audioaufnahme (siehe 11.1.1). Weitere Klassen für Buttons oder Slider sind nicht notwendig, da diese bereits durch die HTML Spezifikation definiert sind.

[5] geile Footnote zum Design Pattern Buch

6 Design

Das Design des Programms ist einer der wichtigsten Bestandteile. Über die Usability und den Look & Feel entscheiden die Designvorgaben und -aspekte und sind somit ausschlaggebend für das Alleinstellungsmerkmal und die Konkurrenzfähigkeit des Produktes.

Firmen wie Google schreiben bei der Entwicklung von Apps für ihre Plattformen spezifische Design Guidelines[1] vor um die Integrität der Designsprache zu währen.

6.1 Guidelines

Die gewählten Design Guidelines beinhalten folgende Aspekte und wurden bei der Entwicklung des Programms in späteren Iterationen implementiert (siehe Abb. A.1).

Farbcode

Die Farben des Programms werden in Weißtönen mit geringen farblichen Akzenten gehalten. Dadurch wird eine visuelle Klarheit erstellt, welche die einfache und intuitive Bedienung unterstützt. Die einzelnen Spuren und die dazugehörigen Waveforms werden in unterschiedlichen, gedeckten Farben akzentuiert, wodurch Übersichtlichkeit geschaffen wird.

Transparenz

Das Design der einzelnen Elemente und Hintergründe soll in verschiedenen Transparenz-Stufen gehalten werden, wodurch Leichtigkeit kreiert und die Übersichtlichkeit unterstützt wird.

[1] https://www.google.com/design/spec/material-design/ (geprüft 5.8.2015)

Schlichtheit

Alle Formen, Symbole und Anordnungen werden in ihrer Erscheinung so schlicht wie möglich gehalten. Es sollen keine Verzierungen oder andere Elemente, die von der eigentlichen Funktion oder der minimalen Gestaltung ablenken, eingesetzt werden.

Schriftart, Buttons & Slider

Als Schriftart wird Source Sans Pro[2] gewählt. Die von Adobe unter freier Lizenz veröffentlichte Schriftart ist serifenlos und wird in dem Schriftschnitt Extra-Light verwendet.

Die Buttons werden in runder Form erstellt in mit vereinfachten, ikonographisch Abbildungen der Funktion versehen. Gedrückte Buttons werden durch einen blauen Hintergrund hervorgehoben. Den Slidern wird durch ein einheitliches Design browserübergreifende Integrität gegeben und soll sich dabei an den runden Buttons orientieren.

6.2 User Interface

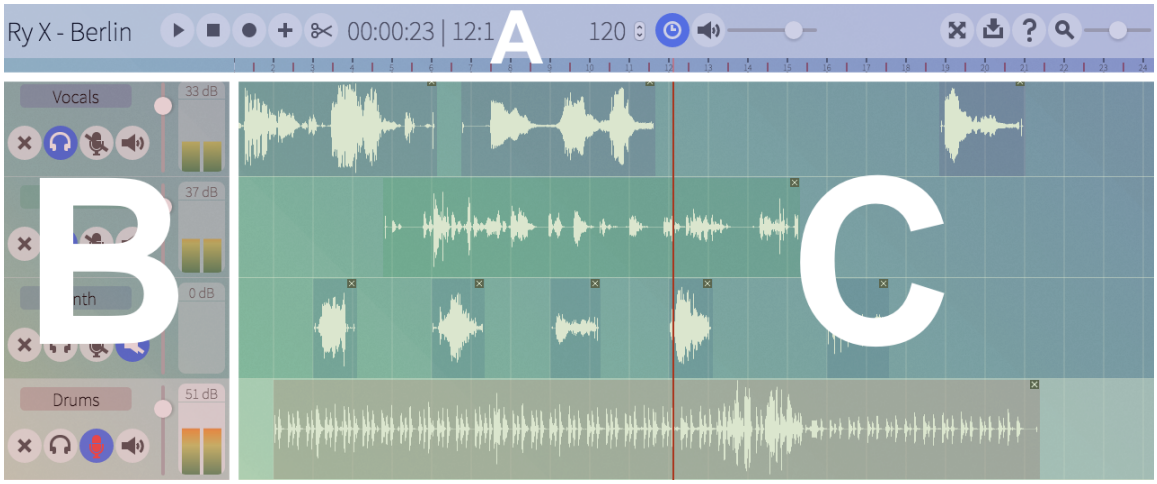

Abbildung 6.1: Screenshot des Programmes

Das Design des Programms soll sich an der Erfolgsmethode der meistgenutzten

[2] https://www.google.com/fonts/specimen/Source+Sans+Pro (geprüft 5.8.2015)

DAWs orientieren. In der westlichen Welt ist der Lese- und Schreibefluss von Links nach Rechts der Brauch, ebenso in der Audiovisualisierung.

Das Hauptfenster ist bei diesem Programm in drei Regionen aufgeteilt: (6.2 A) die Hauptbedienelemente, (6.2 B) die Spurbedienelemente und (6.2 C) das Arrangierfenster (siehe Abb. 6.1).

(A) Hauptbedienelemente

Bei den Hauptbedienelemente handelt es sich um die in Abschnitt 2.5 beschriebenen Funktionen. Diese sind als Buttons, Textfelder und Slider umgesetzt. Die Leiste der Hauptbedienelemente soll jederzeit am oberen Rand des Fensters fixiert sein und sich in der Breite dem Browserfenster dynamisch anpassen.

(B) Spurbedienelemente

Die Spurbedienelemente setzen sich aus den in Abschnitt 2.6 beschriebenen Funktionen zusammen. Diese sind ebenfalls als Buttons und Silder realisiert. Das Volume Display ist als Pegelanzeige realisiert. Hierbei wird der aktuelle Dezibel Pegel des Mikrofoninputs als senkrechte Säule dargestellt und wessen Höhe und Färbung sich an der Lautstärke orientiert[3].

(C) Arrangierfenster

Das Arrangierfenster orientiert sich an dem zuvor beschriebenen, westlichen Schreibefluss von links nach rechts. In diesem Teil des Anwendungsfensters werden die einzelnen Spuren als Zeilen dargestellt. Eine durchgehende, rote Linie zeigt den aktuellen Zeitpunkt in dem Projekt an und am oberen Rand des Arrangierfensters befindet sich eine Zeitleiste mit Taktangaben zur Orientierung. Wenn die Tonaufnahme gestartet wird, wird die Mikrofoneingabe als Waveform von links nach rechts laufend gezeichnet. Alle so aufgenommenen Waveforms können in der jeweiligen Spur und zu anderen Spuren hin verschoben, geschnitten und gelöscht werden.

[3]je höher der dB Pegel, desto höher die Säule; Färbung von Grün (leise) zu Rot (laut)

6.3 Name

Der Name des Projektes lautet "Æditor". Diese Namenswahl ist eine Komposition aus den zwei Worten Audio und Editor, verbunden mit dem skandinavischen Æ, welches als Ligatur[4] aus A und E für die beiden Bestandteile des Programms steht. Hiermit wurde ein simpler und verständlicher Name gewählt, der das Programm in seiner Kernfunktionalität wiederspiegelt, durch seine Schlichtheit leicht zu merken ist und dadurch eingängig und griffig ist.

6.4 Logo

Abbildung 6.2: Entwicklungsschritte des Logos

Die Entwicklung des Logos wurde von Beginn an durch den Namen des Projektes definiert. So sollte das Æ im Titel als Grundlage für das Logo dienen und zusammen mit einer einfachen und klaren graphischen Darstellung für einen hohen Wiedererkennungswert sorgen. Das Logo wurde nach dem ersten Entwurf, welcher im Design an die ersten Entwürfe des Programms (siehe Abb. A.1) angelehnt war, an die Design Guidelines und Schriftart angepasst (siehe Abb. A.2 und A.3). Im letzten Schritt wurde das Logo an die neu etablierte, runde Buttonform angelehnt um somit ein in sich stimmiges und logisches Gesamtbild abzugeben (siehe Abb. 6.2).

[4] Buchstabenverbund

Teil III

Realisierung

7 Entwicklungsumgebung

Das Projekt wurde auf einem Apple MacBook Pro 13,3 Zoll aus dem Jahre 2011 unter dem Betriebssystem OS X Yosemite 10.10.4 programmiert. Die primäre Bildschirmauflösung bei der Entwicklung betrug 1280x800, zum Testen wurde eine zweiter Bildschirm mit der Auflösung 1680x1050 verwendet und für das Testen auf mobilen Geräten wurde ein Google Nexus 7 Tablet mit der Android Version 5.1.1 verwendet.

Zum Programmieren wurde der quelloffene Editor Brackets[1] von Adobe verwendet. Dieser Editor wurde 2014 veröffentlicht, ist weitestgehend in HTML, CSS und Javascript entwickelt und als Desktop Applikation, welche mit dem Chromium Embedded Framework[2] umgesetzt wurde, erhältlich. Brackets ist speziell auf die Web Entwicklung ausgelegt, indem Features wie eine Live Preview des Codes über einen internen Server im Google Chrome Browser mit Echtzeitänderungen oder Quick Edit Funktionen um verbundene HTML, CSS oder Javascript zusammenhängend zu editieren, zur Verfügung gestellt werden. Außerdem bietet Brackets eine umfangreiche, von der Community vorangetriebene Erweiterungsbibliothek, Integration des JavaScript Debuggers Theseus[3], eine Quick Docs Funktion, um zugehörige Inline-Kommentare anzuzeigen, sowie JSLint[4] und LESS[5] Support.

Als Zielplattformen wurden die Browser Chrome, Opera und Firefox gewählt. Die genannten Browser unterstützen die aktuelle Version der Web Audio API und navigator.getUserMedia() API, welche in dem Programm eingesetzt werden. Das Programm soll hierdurch sowohl auf PC, Mac, als auch auf Tablets und Smartphones mit neuen Versionen des Android-Betriebsystems funktionsfähig sein.

[1] http://brackets.io/ (geprüft 6.8.2015)
[2] https://code.google.com/p/chromiumembedded/ (geprüft 6.8.2015)
[3] https://github.com/adobe-research/theseus (geprüft 6.8.2015)
[4] http://jslint.com/ (geprüft 6.8.2015)
[5] http://lesscss.org/ (geprüft 6.8.2015)

8 Struktur

Das Programm AEditor besteht aus drei Bestandteilen: dem HTML, CSS und Javascript. Das HTML und CSS sind bei diesem Projekt für das User Interface verantwortlich. Die Struktur der Elemente des Interfaces wird durch das HTML definiert, das CSS ist für die Gestaltung und Positionierung verantwortlich und im Javascript wird die eigentliche Funktionalität implementiert.

8.1 Erweiterungen

Für die drag & drop Funktionalität dieses Programms wird die jQuery Erweiterung jQuery UI[1] verwendet. Es wird der Quellcode des von Chris Wilson entwickeltem Web Audio Metronom[2] in stark abgewandelter und angepasster Form, sowie das von Matt Diamond entwickelte `recorder.js`[3] Plugin zur Aufnahme und zum Export von Web Audio Nodes in das Projekt integriert.

8.2 Audio Node Graph

Jede mit der Web Audio API geschriebene Website verfügt über einen Audio Node Graph. Dieser Graph beschreibt die Verknüpfung der in JavaScript erstellen Web Audio Nodes.

Die Abbildung 8.1 zeigt den Audio Node Graph des entwickelten Programms AEditor. Auf der linken Seite befindet sich die `MediaStreamAudioSouce` Node, welche den Ton, der über das Mikrofon eingegeben wird, darstellt. Die `MediaStreamAudioSouce`

[1] https://jqueryui.com/ (geprüft 7.8.2015)
[2] https://github.com/cwilso/metronome (geprüft 7.8.2015)
[3] https://github.com/mattdiamond/Recorderjs (geprüft 7.8.2015)

Node wird anschließend mit einer `ScriptProcessor` Node, welche für die Tonaufnahme des `recorder.js` Plugins zuständig ist, und einer `ChannelSplitter` Node, welche das Audio Singal der `MediaStreamAudioSouce` Node in zwei Stereokanäle trennt, verbunden. An die zwei Stereosignale der `ChannelSplitter` Node werden zwei `Analyser` Nodes angeschlossen, welche für die grafische Darstellung der Pegelanzeige von der jeweils ausgewählten, aktiven Spur verwendet werden. Während die `ScriptProcessor` Node des `recorder.js` Plugins mit der Tonausgabe über die `AudioDestination` verbunden wird, wird eine der beiden `Analyser` Nodes mit einer weiteren `ScriptProcessor` Node verbunden, welche Funktionen während der Toneingabe aufruft und die anschließend ebenfalls mit der `AudioDestination` verbunden wird. Im unteren Teil der Abbildung 8.1 werden die Verknüpfungen der einzelnen Spuren mit der Masterlautstärke gezeigt. Die linke `Gain` Node steht für die Lautstärke einer Spur. Diese ist zum einen über einen `Panner` mit der Master `Gain` Node der Hauptlautstärke und zum anderen mit einer `ChannelSplitter` Node verbunden. Diese bietet, wie bereits vorher beschrieben, zwei daran angeschlossenen `Analyser` Nodes die Möglichkeit, das linke und rechte Stereosignal für die Darstellung der Pegeleinstellung der jeweiligen Spur zu analysieren. Die `ChannelSplitter` und `Analyser` Nodes einer jeden Spur dienen dabei der grafischen Darstellung der Wiedergabe von bereits aufgenommenen Tönen, während der `ChannelSplitter` und die `Analyser` Nodes, die mit der `MediaStreamAudioSouce` Node verbunden sind, zur graphischen Darstellung des der aktuellen Mikrofoneingabe dienen. Die `Gain` Node der Masterlautstärke wird über eine weitere `Gain` Node, mit welcher der Nutzer die Wiedergabelautstärke während des Downloads steuern kann, und eine weitere `ScriptProcessor` Node, welche für das `recorder` Objekt des Downloads zuständig ist, mit der Tonausgabe `AudioDestination` verbunden. All dies beschreibt den Audio Node Graph, welcher bei der Initialisierung des Programms erstellt wird.

Im Laufe der Benutzung des Programms wächst dieser Audio Node Graph um viele weitere Nodes und wird dynamisch erweitert und reduziert. Die Elemente `AudioBuffer`, in denen die aufgenommenen Audiodaten gespeichert sind und `AudioBufferSource` Nodes, welche zur Wiedergabe der `AudioBuffer` dienen, werden dynamisch erzeugt und entfernt.

In der Abbildung 8.2 ist ein Audio Node Graph des Programms abgebildet, welches über drei Spuren und auf jeder Spur einer aufgenommenen Audiodatei verfügt. Auf

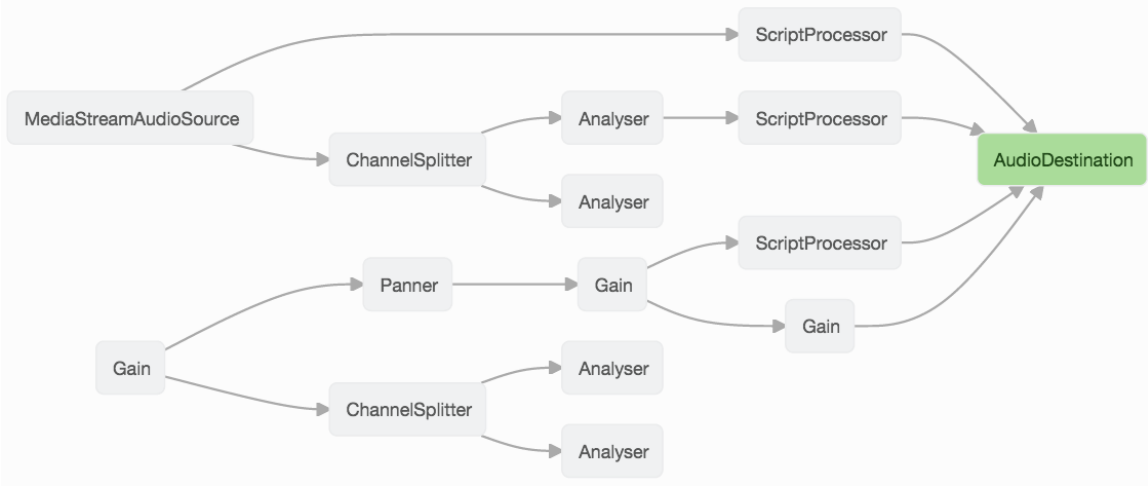

Abbildung 8.1: Initial Audio Node Graph des Programms AEditor

der linken Seite befinden sich die `AudioBuffer`, in welchen die durch das `recorder.js` Plugin aufgenommenen Audiodaten als Array, bestehend aus zwei Float32Arrays für die zwei Stereokanäle, gespeichert sind. Die `AudioBuffer` sind nicht direkt mit einem anderen Knoten verbunden, da diese als Basis für die `AudioBufferSource` Nodes dienen. Die `AudioBufferSource` Nodes dienen dem Abspielen einer, in einem `AudioBuffer` geladenen, Audiodatei und werden an die `Gain` Nodes der jeweiligen Spuren angeschlossen.

In der Web Audio API sinde viele Nodes wie die `AudioBufferSourceNode` als Einweg-Instanzen implementiert. Man kann die Funktion `.start()` nur einmal aufrufen. In diesem Projekt werden daher alle `AudioBufferSourceNodes` in dem Moment erzeugt, in dem diese abgespielt werden, und anschließend, nach dem Ende der Audiodatei, mit der Funktion `AudioBufferSourceNode.disconnect()` freigegeben und aus dem Audio Node Graph entfernt. Der interne JavasScript Garbage Collector[4] verwaltet die nicht verwendeten `AudioBufferSourceNodes` und entfernt diese.

[4]automatische Speicherverwaltung, minimiert den Speicherbedarf eines Computerprogramms

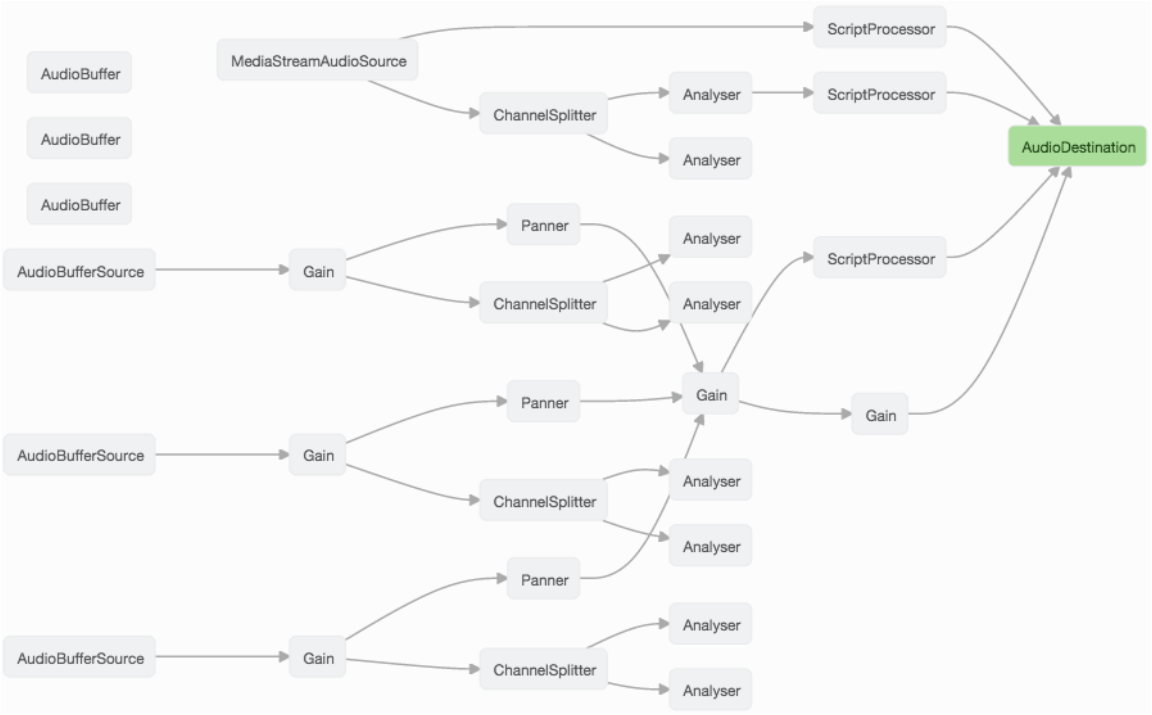

Abbildung 8.2: Audio Node Graph des Programms AEditor nach Tonaufnahme

9 HTML

Das HTML wird in dem Programm AEditor komplett dynamisch erzeugt, sodass die `index.html` Datei des Programms sich auf 20 Zeilen Code beschränkt und lediglich die Javascript Dateien lädt. In der Klasse `App` werden mit der Funktion `initialize()` die Hauptelemente des HTML geladen und in der Klasse `track` die HTML Elemente für eine neu erstellte Spur.

Eine drei dimensionale Visualisierung der HTML Struktur und einzelnen Elemente wurde zur Anschauung mit dem 3D Inspektor[1] von Mozilla Firefox erstellt und in der Abbildung A.5 angehängt.

In Abbildung 9.1 wird die HTML Struktur als hierarchischer Ordnerbaum dargestellt. In dem HTML `body` wird der Hauptbehälter `main_container` erstellt, in welchem sich alle weiteren Elemente befinden, mit Ausnahme des `help_controls_container`, in welchem die Elemente für den Hilfebildschrim organisiert sind, und `logo_container1`, in welchem sich das Logo und der zugehörige Infotext befinden. Diese beiden Behälter werden ebenfalls dem HTML `body` angehängt, sind jedoch aus Überschaubarkeitsgründen nicht mit auf der Abbildung 9.1 verzeichnet. In diesem Abschnitt wird auf den Inhalt des erstellten `main_container` eingegangen, mit welchem alle Hauptfunktionalität des Programms verknüpft ist. Weitere Informationen über die Behälter `help_controls_container` und `logo_container1` können dem Quellcode und der Dokumentation des Programms entnommen werden.

1. main_container

Dem `main_container`, welcher dem `body` des HTML dynamisch bei der Initialisierung des Programms angehängt wird, werden alle weiteren Elemente hinzugefügt. Der `main_container` setzt sich aus den zwei weiteren Behältern

[1] https://developer.mozilla.org/de/docs/Tools/3D_untersuchung (geprüft 10.8.2015)

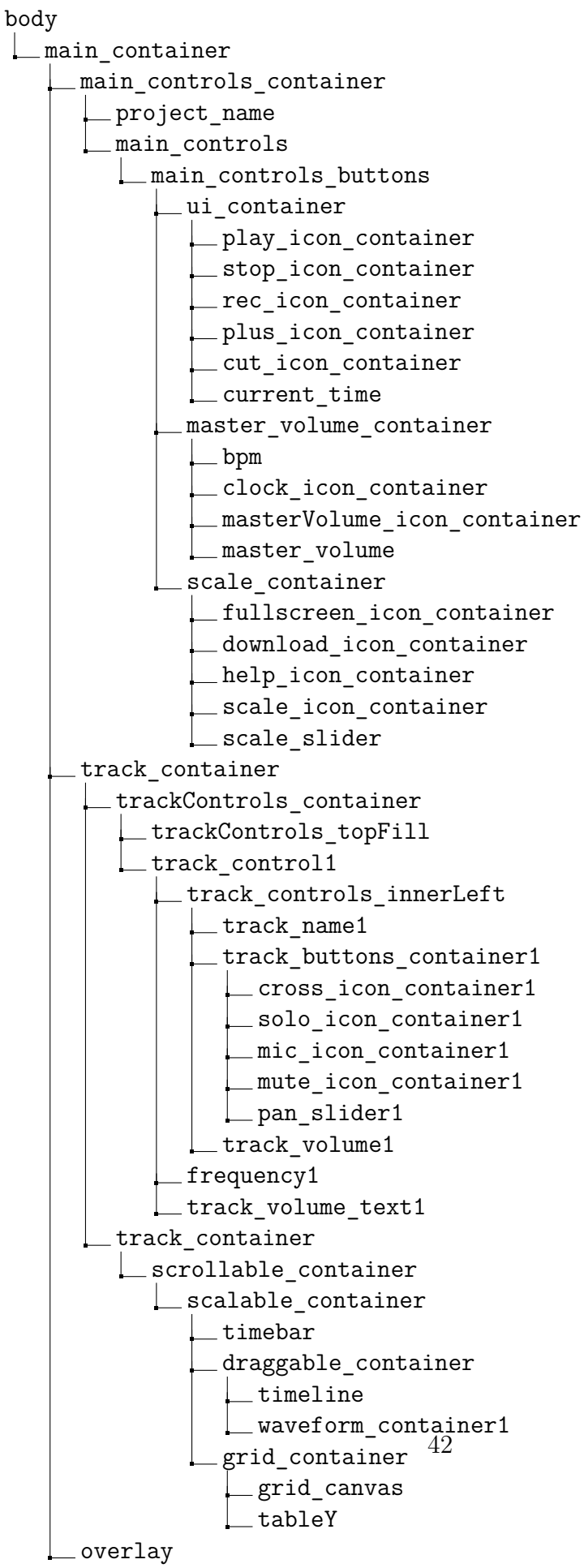

Abbildung 9.1: HTML Struktur

main_controls_container und track_container, sowie dem für den anfänglichen und bei dem Hilfebildschirm eingesetzten Unschärfeeffekt verantwortlichen overlay Element, zusammen.

2. main_controls_container

Der main_controls_container beinhaltet die Hauptbedienelemente (siehe 2.5) project_name für die Eingabe des Projektnamens und den main_controls Behälter, in wessen Unterbehälter main_controls_buttons sich in den drei Behältern ui_container, master_volume_container und scale_container alle Bedienelemente wie Buttons, Silder oder Eingabefelder der Hauptbedienelemente befinden. Die Struktur ist dabei durch die graphische Gestaltung und Layoutansätze, wie vertikales Scrollen oder Mindestbreiten, mitgeprägt.

3. track_container

In dem track_container befinden sich alle Elemente des Arrangierfensters (siehe 2.4 C) innerhalb des Behälters scalabe_container und alle Elemente der Spurbedienelemente (siehe 2.6 B) in dem Behälter trackControls_container.

4. trackControls_container

Der trackControls_container setzt sich dabei aus dem graphischen Füllelement trackControls_topFill und dem track_control1 Behälter zusammen, der alle Kontrollelemente jeweils einer Spur beinhaltet und von welchem für jede vorhandene Spur eine Instanz existiert (track_control1, track_control2, track_control3, etc.). Innerhalb eines jeden Behälters wie track_control1 befinden sich die Elemente track_controls_innerLeft, welchem wiederum alle Buttons, Silder und Eingabefelder der Spurbedienelemente innerhalb des track_buttons_container1 angehängt sind, sowie der frequency1 und track_volume _text1 aus welchen beiden sich das graphische Volume Display mit numerischer Anzeige zusammensetzt.

5. scrollable_container

Für das Arrangierfenster und dessen Inhalte verantwortlich ist der scrollable _container. Innerhalb des scrollable_container befinden sich in dem scalable

`_container` alle weiteren Elemente des Arrangierfensters. Diese Aufteilung sorgt für eine korrekte Scroll- und Zoom-Funktionalität. In dem `scalable_container` wiederum befindet sich die in 2.4 beschriebene Zeitleiste `timebar`, der `draggalbe_container`, sowie der `grid_container`, worin sich das canvas `grid_canvas` zur graphischen und die Tabelle `tableY` zur funktionalen Implementierung des in 2.4 beschriebenem Gitterliniennetzes, befindet.

In dem `draggalbe_container` befindet sich letztendlich die in 2.4 beschriebene Zeitachse und der für jede Spur jeweils erzeugte `waveform_container1`, `waveform _container2`, `waveform_container3` usw., worin die Waveforms der jeweiligen Spur gespeichert werden.

10 CSS

Das CSS des Programms AEditor ist ein wichtiger Bestandteil, denn in diesem wird nicht nur das Layout und die entsprechenden Parameter wie Scrollbarkeit, teilweise Skalierung oder das responsive Design des Programms implementiert, sondern auch die graphische Gestaltung aller Elemente und Symbole, da in diesem Projekt auf Bildquellen verzichtet wurde um das Programm möglichst effizient und performant zu halten.

10.1 Buttons

Die Buttons, welche aus dem JavaScript der `App` und `Track` Klassen generiert werden, bestehen aus unterschiedlich vielen HTML Divs[1]. Die Struktur der Buttons besteht dabei jeweils aus einem `icon_container` Div, welches die einheitliche Größe, Form und Farbe des Buttons bestimmt (siehe Abb. 10.1), sowie dem jeweils darin enthaltenem Symbol, bestehend aus einem bis zwei Divs, welchen über die CSS Pseudoelemente[2] `::before` und `::after` weiterer Inhalt und Styling gegeben wird.

In dem Beispielcode 10.1 wird anhand des Zoom Buttons (siehe Abb. 10.1) der Vorgang des Buttonstylings erklärt. Über den, in diesem Beispiel nicht enthaltenem, `icon _container` wird die Hintergrundfarbe, Größe und Positionierung gesetzt und über das Attribut `border-radius` des `icon_container` die runde Form definiert. Das Symbol des Buttons wird über das Div `scale` entworfen, indem das CSS Pseudoelement `::after` verwendet wird. Das `scale` Div selbst wird als runder Teil der zu entwerfenden Lupe ähnlich dem `icon_container` über das `border-radius` Attribut als Kreis dargestellt und über den Außenabstand `margin` positioniert. In dem Pseudoelement `scale::after` wird der Stab der Lupe erstellt, indem der Inhalt `content`

[1] http://wiki.selfhtml.org/wiki/HTML/Textstrukturierung/div (geprüft 10.8.2015)
[2] http://www.w3schools.com/css/css_pseudo_elements.asp (geprüft 10.8.2015)

mit dem Parameter ' ' auf leer gesetzt wird und somit nur die eigentliche Form des Pseudoelements relevant wird, welche anschließend in Höhe und Breite definiert, positioniert und über `transform: rotate` zu dem gewünschten Winkel rotiert wird.

Abbildung 10.1: Design des Zoom Buttons

Listing 10.1: Beispielcode zum Styling des Zoom Buttons

```
1       #scale {
2           height: 8px;
3           width: 8px;
4           background: transparent;
5           border-radius: 50px;
6           border: 4px solid #555;
7           margin: 7px 0 0 7px;
8       }
9
10      #scale::after {
11          content: '';
12          width: 4px;
13          height: 12px;
14          background: #555;
15          position: absolute;
16          top: 17px;
17          left: 21px;
18          transform: rotate(-45deg);
19
20      }
```

10.2 Responsive Webdesign

Unter dem Begriff Responsive Webdesign sammeln sich viele verschiedene Ansätze des Webdesigns. Das Grundprinzip dieser Techniken ist jedoch die variable Anpassung des Layouts und Designs von Websites an das Endgerät des Benutzers. In der heutigen Welt ist die Vielfalt an Bildschirmformaten und Auflösungen durch verschiedene Endgeräte wie Smartphones, Tablets, Laptops oder Desktop PCs sehr hoch und unterschiedlich, womit sich die Fragen der Kompatibilität des Designs in einer Zeit, in der immer mehr Websites leistungsfähiger und umfangreicher werden, stellt.

Der Name responsive Webdesign stammt aus dem Jahr 2010 und wurde von Ethan Marcotte erstmals erwähnt Marcotte (2010). Responsive Webdesign umfasst heute verschiedene Layouttechniken wie fixed, fluid oder elastic Layout[3] und wird vornehmlich über die in CSS3 etablierten Media Queries[4], mit welchen Regeln für beliebig wählbare Displaygrößen erstellt werden können, prozentualen Positionierungs- und Größenangaben oder Frameworks wie Bootstrap[5] realisiert.

Das in dieser Bachelorarbeit erarbeitete Programm AEditor ist für Endgeräte mit einer minimalen Auflösung 400 x 150 Pixel benutzbar und somit auch auf Smartphones und Tablets einsetzbar. Da die in 4.2 beschriebene getUserMedia API jedoch zur Zeit nur auf neueren Androidgeräten verfügbar ist, wurde die Konzeption des Programms hauptsächlich auf Laptop und Desktop PCs sowie Tablets und Smartphones mit den entsprechenden Androidversionen ausgelegt.

Dafür wurde bei der Umsetzung des CSS auf eine Hybridlösung aus dem fluid und fixed Layout gesetzt: einige Elemente wie die Höhe der Leiste der Hauptbedienelemente, Buttons, Slider oder Eingabefelder und die Spurbedienelemente (siehe Abb. 6.1) haben eine in Pixel festgesetzte Größe entsprechend dem fixed Layoutentwurf, alle weiteren Elemente wie die Breite der Leiste der Hauptbedienelemente, Breite und Höhe des Arrangierfensters oder des in Abschnitt 9 beschriebenem `main_container` passen sich jedoch durch prozentuale Angaben der Dimensionen des Browserfensters dynamisch an.

[3] http://www.smashingmagazine.com/2009/06/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/ (geprüft 11.8.2015)
[4] http://wiki.selfhtml.org/wiki/CSS/Media_Queries (geprüft 11.8.2015)
[5] http://getbootstrap.com/ (geprüft 11.8.2015)

11 JavaScript

In diesem Teil der Realisierung wird auf die Umsetzung der Funktionalität in JavaScript eingegangen. Es werden alle Hauptfunktionen thematisiert und in den Grundzügen erklärt. Während auf einige der Funktionen detailliert mit Codebeispielen eingegangen wird, kann die restliche entsprechende Umsetzung dem Quellcode des Programms entnommen werden.

11.1 Audioverarbeitung

In diesem Abschnitt wird die Audioverarbeitung des Programms AEditor mit Hilfe des `recorder.js` Plugins beschrieben.

11.1.1 recorder.js

Das Plugin `recorder.js` (siehe 8.1) ermöglicht die Aufnahme von Audiodateien über die getUserMedia API. Aufgebaut ist dieses Plugin nach der Web Worker[1] API, welche seit der Spezifikation von HTML5 Multithreading[2] in Javascript erlaubt.

Bei diesem Plugin wird ein `new Recorder(source)` Objekt erstellt mit der `MediaStreamSourceNode` als Parameter `source` für die Mikrofoneingabe initialisiert. Mit den Funktionen `rec.record()` und `rec.stop()` wird anschließend die Aufnahme gestartet und gestoppt. Dabei wird in dem `recorder.js` Plugin ein Array mit zwei Float32Arrays[3] als Inhalt erstellt und in diese kontinuierlich die Audiodaten der Mikrofoneingabe mittels der in dem Plugin erzeugten `ScriptProcessorNode` geschrieben.

[1] http://www.html5rocks.com/de/tutorials/workers/basics/ (geprüft 11.8.2015)
[2] https://de.wikipedia.org/wiki/Softwareseitiges_Multithreading (geprüft 11.8.2015)
[3] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Float32Array (geprüft 11.8.2015)

In dem Beispielcode wir die `onaudioprocess`[4] Event Handler der `ScriptProcessorNode` aufgerufen, welcher kontinuierlich bei Mikrofoneingabe aufgerufen wird. Wird nun in der Funktion `record()` des `recorder.js` Plugins die Variable `recording` auf `true` gesetzt, so wird das Array `buffer` erstellt und daran die Audiodaten der verfügbaren Kanäle über die Array Funktion `push()` angefügt. Das Schreiben der Audiodaten wird in dem in `recorderWorker.js` implementierten Web Worker ausgeführt, indem über die Web Worker Funktion `postMessage` der auszuführende Befehl `'record'` und das `buffer` Array übergeben wird. Um das Schreiben der Audiodaten zu stoppen, wird die Funktion `stop()` des `recorder.js` Plugins aufgerufen, welche lediglich die `recording` Variable auf `false` setzt.

Über die Funktion `clear()` des Plugins werden die Inhalte der Buffer und Arrays gelöscht und auf Null gesetzt, wodurch das `recorder` Objekt bereit ist für die nächste Aufnahme. Bevor dies ausgeführt wird, wird in dem Programm AEditor die Funktion `createWaveformBuffer` aufgerufen, welche wiederum über die Funktion `getBuffers` des `recorder.js` Plugins auf die gespeicherten Audiodaten zugreift und diese in einem `AudioBuffer` der Web Audio API speichert, wodurch diese für den weiteren Zugriff im Programm zur Verfügung stehen.

Listing 11.1: Ausschnitt der Aufnahmefunktionalität des recorder.js Plugins

```
1  this.node.onaudioprocess = function(e){
2    if (!recording) return;
3    var buffer = [];
4    for (var channel = 0; channel < numChannels; channel++){
5      buffer.push(e.inputBuffer.getChannelData(channel));
6    }
7    worker.postMessage({
8      command: 'record',
9      buffer: buffer
10   });
11 }
```

[4] https://developer.mozilla.org/en-US/docs/Web/API/ScriptProcessorNode/onaudioprocess (geprüft 11.8.2015)

11.2 Audiofunktionen

Die Audiofunktionen des Programms AEditor werden in der Klasse `App` umgesetzt. Als Ausgangssituation dient dabei die Funktion `setupAudioNodes`, welche die in 8.2 beschriebenen Nodes erstellt und verknüpft. Da diese Funktion das Grundgerüst des Programmes setzt und durch den `navigator.getUserMedia()` Aufruf erst bei Aktivierung der Mikrofoneingabe durch den Nutzer aufgerufen wird, wird hier auch der Rest des Programms durch den Funktionsaufruf `initialize()` initialisiert.

In den nachfolgenden Abschnitten wird auf einige Funktionen und Funktionalitäten des Programms eingegangen und diese beschrieben, die vollständige Anzahl der Funktionen können im Quellcode eingesehen werden.

11.2.1 Play/Pause, Stop & Record

Die Implementierung der Aufnahme- und Wiedergabefunktionalität des Programms teilt sich in viele Unterfunktionen auf und greift auf viele Klassenvariablen der Klasse `App` zu. Einer der Hauptteile der Wiedergabe und Aufnahme ist die Funktion `whileInput` der Klasse `App`, welche die in Codebeispiel 11.2 aufgeführte Funktion `onaudioprocess` der `javaScriptNode`, wie im Abschnitt 11.1.1 beschrieben, aufruft. Diese Funktion wird konstant aufgerufen, solange die Mikrofoneingabe aktiviert ist, setzt die aktuelle Zeit des `audioContext` in der Variablen `currentTime` der Klasse `App`, ruft die Funktion `drawFrequencies()` auf, welche die Visualisierung der Spureingabe umsetzt und führt eine Reihe von Funktionen aus, wenn das Programm auf Wiedergabe oder Aufnahme gestellt ist. Diese Funktionen bestehen u.a. aus der Funktion zur Bewegung der visuellen Abspielposition `moveTimeLine()`, der Funktion `displayTime()` zur Darstellung der Zeit und Takte, oder den Funktionen `startWaveforms()` und `drawWaveform()`, welche, je nachdem ob das Programm über die beiden Variablen `playing` und `recording` auf Wiedergabe oder Aufnahme gesetzt ist, das Abspielen oder die Aufnahme steuern.

Die Funktionen `playRecording()`, `stopRecording` und `startRecording()` werden durch die Benutzereingabe über die jeweiligen Buttons Start, Stop und Rec aufgerufen und steuern über die Variablen `playing` und `recording` der Klasse `App` die Aufnahme und Wiedergabe. Zusätzlich werden noch weitere Aktionen wie der Aufruf der Hilfsfunktion `stopWhileRecoding()`, das Setzen der Abspielposition über die

Klassenvariable `currentTimebarLocation`, das Abspielen des Metronoms oder das Setzen der der aktuellen Projektzeit vollzogen.

Listing 11.2: Audiofunktionen der JavaScriptNode

```
1  this.javascriptNode.onaudioprocess = function() {
2    app.currentTime = app.audioCtx.currentTime;
3    app.trackObjects[0].drawFrequencies(app);
4    if (app.playing == true || app.recording == true){
5      app.extendProject();
6      app.focusTimeline();
7      app.moveTimeLine(app.timebarColumn + app.
           currentTimebarLocation);
8      app.timebarColumn++;
9      app.displayTime();
10     if (app.playing == true){app.startWaveforms();}
11     if (app.recording == true){app.drawWaveform();}
12   }
13 }
```

11.2.2 Cut

Um die Bedienung des Programms AEditor so klar und einfach wie möglich zu halten wurde auf ein selbsterstelltes Rechtsklickmenü verzichtet. Stattdessen wurde die Schneidefunktionalität der Waveforms über einen Button in den Hauptbedienelementen implementiert. Das Schneiden wird durch das Drücken dieses Buttons aktiviert, wodurch sich der Mauszeiger zu einem Kreuz ändert. Nun kann jede Waveform an beliebiger Stelle durch einen Klick geschnitten werden. Nach dem Klick auf ein beliebiges Element deaktivieren sich der Button und die Schneidefunktionalität wieder und der Mauszeiger ändert sich zurück.

Realisiert wurde dies durch zwei Eventlistener[5], welche die zwei Funktionen `cutWaveformActive()` und `cutWaveform()` der Klasse `App` aufrufen. Der Eventlistener des Schneidebuttons in den Hauptbedienelementen löst die Funktion `cutWaveformActive()` aus, welche die boolesche Klassenvariable `cut` der Klasse `App`

[5]Ein Aktion wird bei einem bestimmten Ereignis ausgelöst

auf `true` setzt, den Cursor über den Befehl `document.body.style.cursor = "crosshair";` als Kreuz definiert und den Butten in einen gedrückten Zustand visualisiert. Nun ist das Schneiden aktiviert und über den zweiten Eventlistener (siehe Codebeispiel 11.3), welcher über `document` auf das gesamte Programm gelegt wird und durch den Parameter `"click"` bei einem Mausklick ausgelöst wird, wird die Schneidefunktion `cutWaveform` aufgerufen, falls durch den Schneidebutton über die zuvor beschriebene `cutWaveformActive()` Funktion das Schneiden aktiviert wurde, indem die Variable `cut` auf `true` gesetzt wurde.

Bei der eigentlichen Schneidefunktion `cutWaveform()` wird über den Eventlistener 11.3 das geklickte Element, dessen Identifizierung und die Mausposition übergeben, um anschließend mit diesen Parametern die Schneideposition auf dem geklickten Waveformcanvas zu ermitteln, zwei neue Waveformobjekte und -canvas zu erstellen, diese dem Spurobjekt anzufügen und die ursprüngliche Waveform zu löschen. Die linke der beiden neu erstellen Waveforms bekommen als Audiodatei den `audioBuffer` der Ursprungswaveform, da diese den gleichen Anfang haben und die Länge über die Breite des Waveformcanvas bestimmt wird. Für die rechte der beiden neuen Waveforms wird ein neuer `audioBuffer` erstellt, welcher sich aus dem ursprünglichen `audioBuffer`, bei dem der erste Teil bis zur Schneideposition abgeschnitten wurde, zusammensetzt. Nach diesen Aktionen wird der Mauszeiger, Schneidebutton und die `cut` Variable wieder zurückgesetzt und somit das Schneiden deaktiviert.

Listing 11.3: Eventlistener der Schneidefunktion

```
1       document.addEventListener("click", function(e) {e.
            preventDefault(); if(app.cut == true){app.
            cutWaveform(e.target, e.target.id, e.clientX, app)
            ;}}, false);
```

11.2.3 Mute

Die Stummschaltefunktionalität des Programms ist in zwei Funktionen aufgeteilt, die `masterMute()` Funktion der Klasse `App` und die `mute()` Funktionalität der Klasse `Track`. Die Klasse `App` und jedes Objekt der Klasse `Track` verfügt jeweils über eine boolesche Klassenvariable `mute` bzw. `muted`, welche in der Funktion invertiert wird und aufgrund dessen die jeweilige `gainNode` entweder auf Null oder auf den Wert

des zugehörigen Sliders gesetzt wird. Der Solostatus der stummzuschaltenden Spur wird mit der Objektvariablen `solo` des jeweiligen `track` Objektes abgeglichen und nur wenn dieser auf `false` ist, wird die Spur stummgeschaltet (siehe Abschnitt 2.6). Ebenfalls wird in dieser Methode die graphische Hervorhebung des jeweiligen Buttons gesetzt.

11.2.4 Solo

Um diese Funktionalität vollständig umzusetzen, hat die Klasse `App` die boolesche Instanzvariable `anySolo` und `Track` die boolesche Instanzvariable `solo`. Bei der Solo Funktion der Spuren wird die Funktion `soloTrack` der Klasse `track` aufgerufen. Diese Funktion setzt die `solo` Variable der ausgewählten Klasse auf `true`, die Variable `anySolo` auf false und läuft anschließend in einer `for`-Schleife über die Spurenobjekte, die in dem Array `trackObjects` der Klasse `App` gespeichert sind. Jede Spur, dessen `solo` Variable auf `false` gesetzt ist und die somit nicht auf Solo gestellt ist, wird stummgeschaltet, indem der Wert der `gainNode` der Spur auf Null gesetzt wird. Die `gainNode` jeder weiteren Spur, dessen `solo` Variable auf `true` gesetzt ist und sich somit im Solomodus befindet, wird auf den Wert des Lautstärkesliders der Spur gesetzt und gleichzeitig wird die `anySolo` Variable der Klasse `App` auf `true` gesetzt, um zu signalisieren, dass sich mindestens eine Spur im Solomodus befindet. Für den Fall, dass sich keine Spur im Solomodus befindet und die Variable `anySolo` nach der eben beschriebenen `for`- Schleife auf `false` gesetzt ist, wird eine weitere `for`-Schleife durchlaufen, in der alle `gainNodes` der jeweiligen Spuren auf den Wert des zugehörigen Laustärkesliders der Spur oder, falls die Spur stummgeschaltet ist, auf Null gesetzt werden.

11.2.5 Lautstärke & Panning

Die Lautstärken einer jeden Spur und des Gesamtprojektes werden über `gainNodes`[6] gesteuert, das Panning einer jeden Spur über `pannerNodes`[7]. Diese Nodes werden nach der Erstellung in der `initialize()` Funktion der Klasse `App` bzw. der `init()` Funktion der Klasse `Track` mit dem Audio Node Graph des Projektes verbunden.

[6] https://developer.mozilla.org/de/docs/Web/API/GainNode (geprüft 18.8.2015)
[7] https://developer.mozilla.org/de/docs/Web/API/PannerNode (geprüft 18.8.2015)

Die `mainGainNode` der Gesamtlautstärke wird in dem Audio Node Graph vor die `gainNode` der Downloadlautstärke, welche mit der Tonausgabe verbunden ist, gesetzt. Die jeweilige `trackGainNode` der Spurlautstärke wird mit der `pannerNode` `trackPanner` der jeweligen Spur verbunden und an die `mainGainNode` gebunden (siehe Abb. 8.1).

Listing 11.4: Gain und Panning einer Spur

```
1       var trackGainNode = app.audioCtx.createGain();
2       trackGainNode.gain.value = 0.8;
3       var trackPanner = app.audioCtx.createPanner();
4       app.audioCtx.listener.setPosition(0, 0, 0);
5       trackPanner.setPosition(0, 0, 1);
6       trackPanner.panningModel = 'equalpower';
7
8       trackPanner.connect(app.mainGainNode);
9       trackGainNode.connect(trackPanner);
```

In dem Beispielcode 11.4 aus der `init()` Funktion der Klasse `Track` wird die `gainNode` und `pannerNode` einer Spur erstellt. Die `gainNode` wird mit dem `value` 0.8 in dem definierten Bereich von 0 bis 1 initialisiert, die `pannerNode` mit der der Funktion `setPosition` in den Nullzustand gesetzt und über das `panningModel` definiert. Anschließend wird der `trackPanner` mit der `mainGainNode` der Gesamtlautstärke über die Funktion `connect()` verbunden und die `trackGainNode` der Spurlautstärke dem `trackPanner` angefügt.

11.2.6 Mikrofonausgabe

Die Mikrofonausgabe für jede Spur ist standardmäßig deaktiviert, um Übersprechungen durch die Ausgabe auf die Lautsprecher zu vermeiden. Wird diese vom Nutzer aktiviert, um z.B. über Kopfhörer die Mikrofoneingabe zu hören, so wird die Funktion `setMicActive()` der Klasse `Track` aufgerufen. Diese Funktion setzt die boolesche Variable `micActive` der ausgewählten Spur und trennt oder verbindet die Mikrofoneingabe `sourceNode` mit der `mainGainNode` der Klasse `App`.

11.3 Audiovisualisierung

Die Audiovisualisierungsfunktionen sind für die Erstellung und Darstellung der Waveforms und der Spurpegel zuständig.

11.3.1 Frequency Canvas

Die Funktionen `drawFerquencies()` und `drawFrequency()` der Klasse `Track` sind für die Visualisierung der Lautstärke und Mikrofoneingabe der jeweiligen Spur, wie in 2.6 beschrieben, zuständig. Dabei wir die Funktion `drawFrequencies()` in der in 11.2 aufgeführten Funktion `whileInput()` der Klasse `App` aufgerufen, welche wiederum die Funktion `drawFrequency()` aufruft. Falls die Variable `play` der Klasse `App` auf `true` gesetzt ist und die Wiedergabe des Projektes somit aktiv ist, wird über eine `for`-Schleife für jede Spur das Frequencycanvas und die Dezibelanzeige mit dem Input der aufgenommenen Waveforms über die `analyserNodes` der Spur gezeichnet (siehe Audio Node Graph Abb. 8.2 und Abschnitt 8.2). Ist die Variable `play` auf `false` gesetzt und die Wiedergabe somit nicht aktiv (also entweder die Aufnahme oder der Ruhezustand aktiv), so wird die Funktion `drawFrequency()` mit den `analyserNodes` der Hauptklasse `App` aufgerufen, welche wiederum mit der Mikrofoneingabe verbunden sind und diese somit visualisiert wird.

In der Funktion `drawFrquency(id, node1, node2)` wird die eigentliche Visualisierung realisiert. Dabei wird in das Frequenzcanvas der mit der `id` übergebenen Spur über die beiden im vorhergehenden Absatz beschriebenen `analyserNodes`, welche ebenfalls übergeben werden, der Dezibel Pegel gemalt und als Text geschrieben. Es werden zwei `Uint8Arrays`[8] erstellt, deren Länge über den `frequencyBinCount`[9] der jeweiligen `analyserNode` definiert wird und welche den Inhalt der Buffer der beiden `analyerNodes` für den rechten und linken Stereokanal über die Methode `analyserNode.getByteFrequencyData()` übergeben bekommen. Nun werden mit der funktionsinternen Methode `getAverageVolume` die Durchschnittswerte der beiden Arrays berechnet, diese in das Canvas gemalt und in das entsprechende `div` als Text geschrieben Dirksen (2012).

[8] https://msdn.microsoft.com/de-de/library/br212477(v=vs.94).aspx (geprüft 15.8.2015)
[9] https://developer.mozilla.org/en-US/docs/Web/API/AnalyserNode/frequencyBinCount (geprüft 15.8.2015)

11.3.2 Waveform Canvas

Die Funktion zum Zeichnen eines `waveformcanvas` bei Aufnahme wird in der Klasse `Waveform` implementiert und in der bereits beschriebenen Funktion `whileInput()` aufgerufen, sobald die Variable `recording` auf `true` gesetzt wird. Bei jedem Aufruf der Funktion `drawWaveform()` wird dabei eine weitere Spalte bzw. Pixel des `waveformcanvas` gemalt.

Dabei wird über die `analyserNode` der Klasse `App`, welche mit der Mikrofoneingabe verbunden ist, das `amplitudeArray` als Buffer erstellt und mit Daten beschrieben, wie in Abschnitt 11.3.1 erläutert. Anschließend wird der minimale und maximale Wert des Arrays ermittelt, gespeichert und in das aktuelle `waveformcanvas` gemalt Jones (2013). Um das Canvas zu erweitern wird ein Hilfscanvas erstellt und der bereits gezeichnete Stand darin gespeichert. Nun wird das eigentliche `waveformcanvas` um eine Spalte bzw. Pixel erweitert, wodurch der Inhalt gelöscht wird, und der aktuelle Stand wieder durch das Hilfscanvas eingefügt.

11.4 Zeitumsetzung

11.4.1 Timeline & Timebar

Die Zeitleiste und Zeitachse sind in dem Programm AEditor als zwei separate Elemente umgesetzt.

Die Zeitachse `timeline` wird als dünne rote Linie, die über dem Arrangierfenster liegt und die aktuelle Abspielposition anzeigt, realisiert. Diese wird über die Funktion `moveTimeLine(amount)` der Klasse `App` bewegt, indem das CSS Attribut `style.left` des Elementes über die Variable `timeLinePosition` der Klasse `App` gesetzt wird. Diese Funktion wird wiederum in den Funktionen `stopRecording()` bei der Zurücksetzung der Abspielposition, `clickTimebar()` beim Setzen der Abspielposition durch einen Klick auf die Zeitleiste, sowie der bereits beschriebenen Funktion `whileInput()` aufgerufen.

Die Zeitleiste `timebar`, welche sich am oberen Rand des Arrangierfensters befindet und das Zeit- und Taktraster anhand der aktuell ausgewählten Taktgeschwindigkeit darstellt, wird über die Funktion `drawTimebar()` der Klasse `App` gezeichnet. Dabei wird die Variable `tempo` der Klasse `App`, in welcher die aktuelle Taktgeschwindigkeit

gespeichert ist, übergeben und anhand dieser die `timebar` gezeichnet. In der Funktionsvariablen `pixelBeat` wird die Geschwindigkeit der `timeline` mit einberechnet und je nach Zoomstufe entweder nur ganze Takte oder zusätzlich halbe und Vierteltakte zur Orientierung gezeichnet. Die Funktion `drawTimebar()` wird in dem Programm zur Initialisierung, bei Änderung der Zoomstufe oder Taktgeschwindigkeit und bei der Erweiterung des Projektes, wenn bei der Wiedergabe oder Aufnahme die maximale Länge erreicht wird, aufgerufen.

11.4.2 Timedisplay

Die Zeitanzeige des Programms AEditor ist in zwei Teile aufgeteilt, zum einen in die Sekunden-, Minuten- und Stundenanzeige und zum anderen in die Taktanzeige. Die Anzeige wird dabei in Klassenvariablen wie `currentSeconds`, `currentBeats` oder `currentTimeOffset` der Klasse `App` gespeichert und über die zwei Funktionen `calculateTime()` und `displayTime()` der Klasse `App` implementiert. Dabei wird die Funktion `calculateTime()` während der Wiedergabe oder Aufnahme in der Funktion `whileInput()` aufgerufen, berechnet die aktuelle Zeit und Takte anhand des `audioContext.currentTime`[10]. Nach jeder Sekunde oder Vierteltakt wird die Funktion `displayTime()` aufgerufen, welche die Zeitanzeige darstellt und die Sekunden- und Minutenumbrüche berechnet.

11.4.3 Metronom

Das Metronom wurde auf Grundlage des Artikels *A Tale of Two Clocks - Scheduling Web Audio with Precision* von Chris Wilson und dem von ihm dazu bereit gestelltem Metronom[11] entwickelt Wilson (2013). Dabei wird das Metronom nicht über die verbreitete Javascript Clock mit der Methode `.setTimeout()` gesteuert, sondern mit der Web Audio internen Uhr über `audioContext.currentTime`. Dadurch werden wesentlich präzisiere Timing Möglichkeiten geboten, denn neben der Ungenauigkeit durch `Date.now()`, welches einen Wert in nur Millisekunden zurückgibt, sind die Funktionen `.setTimeout()` und `.setInterval()` in dem Javascript Hauptthread ausgeführt

[10] https://developer.mozilla.org/en-US/docs/Web/API/AudioContext/currentTime (geprüft 15.8.2015)

[11] https://github.com/cwilso/metronome (geprüft 16.8.2015)

und somit anfällig für andere Aufrufe durch Rendering, Layout oder Garbage Collection. Die Web Audio Clock wird in einem `double` Wert geschrieben, sodass eine präzise Zeit mit bis zu 14 Nachkommastellen möglich ist.

Das von Chris Wilson entwickelte Metronom basiert auf einem Scheduler, der mit der Web Audio Clock und einem Web Worker arbeitet, um präzises und stabiles Timing zu garantieren. In dem Programm AEditor wurde die Grundfunktionalität des Metronoms übernommen und stark abgeändert. So ist das Metronom nun objektorientiert als Klasse implementiert und arbeitet ohne globale Variablen oder ähnliche Zugriffe. Das Metronom Objekt `metronome` wird dabei in der Klasse `App` erstellt. Zudem wurde das graphische Interface entfernt und in das Projekt eingebettet. Die Funktionalität des Metronoms wurde angepasst und um eine Positionsberechnung erweitert, um eine korrekte Anpassung auf die Zeitleiste und Abspielposition im Projekt zu gewähleisten, sowie in anderen Aspekten reduziert, um eine klare, programmdienliche und leistungsfähige Programmierung zu gewähren. Der Javascriptcode des Metronoms wurde von 192 Zeilen um über 53% auf 90 Zeilen reduziert.

In der Klasse `App` wird das Metronomobjekt in der Funktion `initialize()` erstellt und initialisiert. Die Klassenfunktion `clickOnOff` gestaltet den Metronombutton und setzt die Klassenvariable `click` auf `true` oder `false`. Die Helferfunktion `playClick` wird anschließend in den wiedergabe- und aufnahmerelevanten Funktionen `startRecording()`, `stopRecording()`, `playRecording()` und `stopWhile Recording()` sowie der Positionierungsfunktion `clickTimebar()` aufgerufen und ruft selbst wiederum bei aktiviertem `click` die Funktion `playMetronome()` der Klasse `metronome` auf, welche die Wiedergabe des Metronoms stoppt oder startet.

11.5 Hauptfunktionen

11.5.1 Download

Um die Downloadfunktionalität zu implementieren wurde das `recorder.js` Plugin um die Funktion `download()` erweitert. Diese Funktion ruft zunächst die `exportWAV()` Funktion des `recorderWorker` auf, welche den Inhalt des aufrufenden `recorder`-Objekts im .wav Dateiformat speichert und ruft anschließend die `forceDownload()` Funktion der `recorder`-Klasse auf um die gespeicherte Datei runterzuladen.

Die eigentliche `download()` Funktion des Programms AEditor, die in der Klasse `App` definiert ist, prüft zunächst, ob bereits Waveforms aufgenommen wurden. Ist dies der Fall, so wird ein neues `recorder` Objekt erstellt, die Abspielposition auf Null gesetzt, die Wiedergabe des Projekts gestartet und mit der Aufnahme des `recorder`-Objektes über die Funktion `recorder.record()` begonnen. Die Funktion `checkDownload()` der Klasse `App` prüft während der Downloadwiedergabe, ob die Waveform an der letzten Position des Projektes abgespielt wurde. Ist dies eingetreten, so wird die Wiedergabe über die Klassenvariable `playing` gestoppt und die `download()` Funktion der Klasse `App` aufgerufen, welche nun die Abspielposition auf Null zurücksetzt, das erstellte `recorder`-Objekt über die Funktion `recorder.stop()` anhält, die Funktion `recorder.download()` aufruft um das aufgenommene Projekt runterzuladen und über `recorder.clear()` den Inhalt des `recorder`-Objekts löscht.

In dem Codebeispiel 11.5 ist die `download()` Funktion des `recorder.js` Plugins aufgeführt, welche, wie beschrieben, die Funktionen `Recoder.forceDownload()` und `exportWAV()` des `recorderWorker` aufruft.

Listing 11.5: Download Funktion des recorder.js Plugins

```
1     this.download = function(filename) {
2         worker.postMessage({
3           command: 'exportWAV',
4           type: 'audio/wav'
5         });
6
7         worker.onmessage = function(e){
8           Recorder.forceDownload(e.data, filename);
9         };
10    }
```

11.5.2 Add/Delete Track

Die Funktionalität des Erstellens und Löschens einer Spur definiert sich über das HTML des Programms und ist in den Klassen `App` und `Track` implementiert. Beim Erstellen einer Spur wird die Funktion `createTrack()` der Klasse `App` aufgerufen. Diese Funktion erstellt ein neues `track` Objekt, fügt dieses dem Array `trackObjects`

hinzu und ruft die Funktion `init` der `track` Klasse auf, in welcher alle Elemente der Spur erstellt und hinzugefügt werden. Das Löschen einer Spur geschieht über die Funktion `deleteTrack` der Klasse `track`. Hierbei werden alle HTML Elemente aus dem DOM[12] entfernt, alle Web Audio Nodes über die Funktion `disconnect()` von dem Audio Node Graph getrennt und das Spurobjekt aus dem `trackObjects` Array gelöscht.

11.5.3 Add/Delete Waveform

Das Hinzufügen einer Waveform geschieht, wie in den Abschnitten 11.3.2 und 11.2.1 beschrieben, bei der Aufnahme. Bei der Erstellung einer Waveform wird durch die Funktion `addWaveformDeleteButton()` der Klasse `waveform` auf dem erstellten Canvas in der rechten oberen Ecke ein kleiner Button zum Löschen gezeichnet und das Canvas mit einem Eventlistener versehen, der bei einem Mausklick die Funktion `deleteWaveform()` der Klasse `waveform` aufruft. Die Funktion prüft, ob die geklickte Position auf dem Waveform auf dem gezeichneten Button liegt und löscht im gegebenen Fall das Canvas-Element aus dem DOM sowie das Waveformobjekt aus dem `waveformArray` der zugehörigen Spur.

11.5.4 Draggable

Für die Möglichkeit, einzelne Waveforms innerhalb einer Spur und zu anderen Spuren zu verschieben, wurde das draggable Widget der jQuery UI Library[13] herangezogen. Dieses Widget bietet umfangreiche Möglichkeiten, eine Drag & Drop Funktionalität mit vielen Einstellungen zu dem entwickeltem Programm hinzuzufügen. Die Funktionalität wird in der Funktion `makeWaveformDraggable()` der Klasse `waveform` implementiert und beinhaltet Optionen wie `grid` zur Erstellung und Einhaltung eines Rasters, `snap` und `snapTolerance` zur Einstellung der Orientierungsobjekte, `stack` zur Stapelung der Elemente bei Überlappungen, oder `containment` zur Begrenzung der Fläche. Über die Optionen `start`, `drag` und `stop` werden die Aktionen für den jeweiligen Status der Aktion definiert. Bei den Optionen `start` und `drag` wird jeweils die Skalierung der Ansicht mit einberechnet, bei der Option `stop` wird geprüft, ob

[12] http://wiki.selfhtml.org/wiki/JavaScript/Objekte/DOM (geprüft 16.8.2015)
[13] http://api.jqueryui.com/draggable/ (geprüft 16.8.2015)

die verschobene Waveform auf eine andere Spur gezogen wurde. Ist dies der Fall, so wird die Funktion `appendWaveformTo()` aufgerufen, welche die bewegte Waveform löscht und eine neue Waveform auf der ausgewählten Spur erzeugt. Dies hat den Hintergrund, dass bei einem einfachen Verschieben und Anhängen an ein anderes DOM Element sich die Positionierung sonst fehlerhaft verschiebt.

11.5.5 Grid

Um die in Abschnitt 2.4 beschriebenen Gitternetzlinien zu implementieren, wurde eine Kombination aus zwei HTML Elementen gewählt: zum einen ein `canvas`, um die Gitternetzlinien visuell darzustellen, und zum anderen das Tabellenelement `table`[14], über wessen Spaltenelemente `td` die Gitternetzlinienfunktionalität umgesetzt wird. Dabei wird das `canvas` Element `grid_canvas` über die Funktion `drawGridLines()` entsprechend der Zoomstufe gezeichnet. Dies geschieht bei der Initialisierung des Programms und bei Änderung der Zoomstufe. Ist die Zoomstufe hoch genug, so werden neben den Taktstrichen auch Vierteltaktstriche gezeichnet zur besseren Orientierung.

Die eigentlichen Gitternetzlinien, an denen sich die Waveforms durch die `snap` Option des `draggable` Widgets orientieren sollen, werden über ein `table` Element und die zugehörigen Spaltenelemente `td` umgesetzt, da die eigentliche Gitternetzlinienfunktionalität nicht über das Canvas implementiert werden kann, denn die `snap` Option orientiert sich an den Rändern eines definierten Elementes. Deshalb wird in der Funktion `createGridLines()` der Klasse `App` ein Tabellenelement erstellt und diesem in einer `for`-Schleife Spaltenelemente angehängt mit einem der Zoomstufe entsprechenden Abstand. Durch die Option `snap: "td"` orientieren sich die Waveforms an den Rändern der Spaltenelemente `td` der Tabelle und können somit neben der freien Bewegung anhand eines variablen Rasters bewegt werden.

Die Funktion `createGridLines()` wird bei der Initialisierung des Programms in der Funktion `initialize()` aufgerufen sowie in der für den Zoom zusändigen Funktion `changeScale()`, wodurch eine dynamische Anpassung an die Zoomstufe realisiert wird.

[14] http://www.w3schools.com/html/html_tables.asp (geprüft 17.8.2015)

Teil IV

Fazit

12 Auswertung

Das in dieser Bachelorarbeit konzipierte und realisierte Programm AEditor zeigt die Möglichkeiten der Audioaufnahme und Bearbeitung im Web durch die Web Audio API. Durch diese API und weitere neue, in dieser Thesis vorgestellte APIs, ist es möglich, mit der heutigen Browsertechnologie leistungsfähige Programme zu erstellen. Der Bereich der Audioaufnahme und Bearbeitung kann durch diese Entwicklungen vollständig in das Web ausgelagert werden und würde somit viele neue Möglichkeiten, wie das kollaborative Arbeiten, mit sich bringen.

Die entwickelte Funktionalität des Programms zeigt die Grundzüge einer jeden DAW auf und wie diese im Browser umgesetzt werden können. Die in dieser Bachelorarbeit aufgezählten, professionellen DAWs beinhalten alle eine weitaus größere Funktionalität und greifen auf viele Jahre Entwicklungszeit und Forschungsarbeit zurück, um ein möglichst umfangreiches und funktionales Programm zu bieten. Bei dem entwickeltem Programm AEditor wurde auf Klarheit und Schlichtheit in dem Design und der Funktionalität gesetzt, um diese möglichst effizient umzusetzen. In der Zukunft könnte dieses Programm jedoch auf der bereits entwickelten Basis zu einer umfangreichen DAW weiterentwickelt werden und eine Alternative zu den Desktopanwendungen bieten.

12.1 Erweiterbarkeit

Das Programm AEditor bietet in seiner Funktionalität alle nötigen Grundzüge zur Aufnahme und Bearbeitung von aufgenommenen Audiodateien. Darüber hinaus können in der Zukunft noch weitere Funktionalitäten implementiert und erweitert werden, um das Programm weiter und näher an die Funktionalität einer vollwertigen DAW zu bringen. Einige dieser Erweiterungen werden hier aufgeführt.

Import/Export

Um die Speicherung eines Projekts und dadurch das Bearbeiten zu einem beliebigen Zeitpunkt zu ermöglichen, kann eine Import und Export Funktionalität implementiert werden. Dabei kann der Nutzer sein bestehendes Projekt speichern und über ein Menü gespeicherte Projekte laden. Dies könnte mit Hilfe von JSON Dateien realisiert werden.

Effekte

Web Audio bietet die Möglichkeit, umfangreiche Filter über verschiedene Nodes wie die `DynamicsCompressorNode`[1], die `delayNode`[2] oder die `convolverNode`[3] zu implementieren. So kann das Progamm AEditor mit Effekten wie Kompressoren, Delay oder Reverb auf einzelnen Spuren oder dem Gesamtprojekt erweitert werden.

Copy/Paste

Eine Kopier- und Einfügefunktionalität ist in allen Editor Programmen, DAWs wie auch Texteditoren, realisiert und dient einer schnelleren Arbeitsweise durch Vermeidung von sich wiederholenden Arbeitsschritten.

Undo

Eine Funktion, um einzelne Arbeitsschritte rückgängig zu machen und somit Fehler schnell zu beheben, würde dem Programm AEditor weiter den vollwertigen DAWs näher bringen. Eine solche Funktionalität kann über Plugins wie Undo.js[4] realisiert werden.

Kollaboration

Eines der großen Vorteile von Web Apps ist die native Verbundenheit mit dem Internet, wodurch sich Möglichkeiten wie Kollaboration mit mehreren Nutzern umsetzen

[1] https://developer.mozilla.org/en-US/docs/Web/API/DynamicsCompressorNode (geprüft 18.8.2015

[2] https://developer.mozilla.org/de/docs/Web/API/DelayNode (geprüft 18.8.2015

[3] https://developer.mozilla.org/en-US/docs/Web/API/ConvolverNode (geprüft 18.8.2015

[4] https://github.com/jzaefferer/undo (geprüft 18.8.2015)

lassen und ohne weitere Software realisiert werden können. Die Technik der Synchronisation ist bereits vorhanden und wird in Web Apps wie Google Docs[5] verwendet, um viele Nutzer an einem Projekt gleichzeitig arbeiten zu lassen.

Plugins & MIDI

Eine weitere Möglichkeit, um das in dieser Bachelorarbeit entwickelte Programm umfangreicher und konkurrenzfähiger zu machen, ist die Implementierung der MIDI[6] Schnittstelle sowie die Möglichkeit, externe Audiosoftware als Plugins zu laden. Über die Web MIDI API[7] ist es möglich, die Eingabe über externe Musikinstrumente im Browser zu realisieren. Durch die Realisierung einer Plugin Schnittstelle, mit welcher Instrumente und Effekte von Drittherstellern in das Programm geladen werden können, würde sich das Programm AEditor der weiteren Erweiterung öffnen und es wäre möglich, unbegrenzt viele Instrumente und Effekte zu laden und aufzunehmen.

[5] http://googledrive.blogspot.de/2010/09/whats-different-about-new-google-docs.html (geprüft 18.8.2015)
[6] http://www.midi.org/aboutmidi/index.php (geprüft 18.8.2015)
[7] http://www.w3.org/TR/webmidi/ (geprüft 18.8.2015)

13 Ausblick

Die Konzeption und Realisierung des Web Audio Editors AEditor in dieser Bachelorarbeit zeigt die heutigen Möglichkeiten der Audiobearbeitung im Web. Das entwickelte Programm zeigt bereits in den Grundzügen, dass durch die Web Audio API und weitere Technologien umfangreiche Programme im Web realisiert werden können. Durch neue Standards wie HTML5, Web Audio und weitere APIs werden diese neuen Technologien immer weiter verbreitet und Web Apps finden ein immer größeres Publikum. Programme im Web ermöglichen durch Kollaboration, einheitliche Updates und Betriebssystemunabhängigkeit neue Möglichkeiten und Arbeitswege für Nutzer und Entwickler.

Der Entwicklung der Leistungsfähigkeit heutiger Browser und stetiger neuer Technologien zufolge wird es in den nächsten Jahren einen weiteren Schub an Programmen im Web geben und es werden mehr und mehr Konkurrenzangebote, wie schon heute, zu klassischen Desktopanwendungen als webbasierte Lösungen erscheinen, angepasst auf die Mobilität und Vernetzung der Nutzer in der heutigen Welt.

A Material

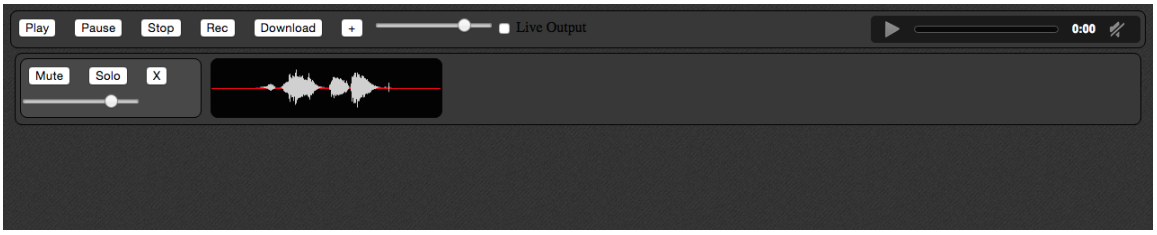

Abbildung A.1: 1. Entwicklungsschritt des Programms

Abbildung A.2: 2. Entwicklungsschritt des Programms

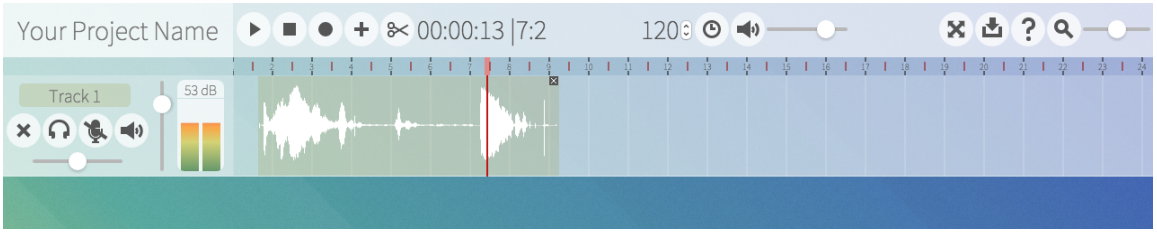

Abbildung A.3: 3. Entwicklungsschritt des Programms

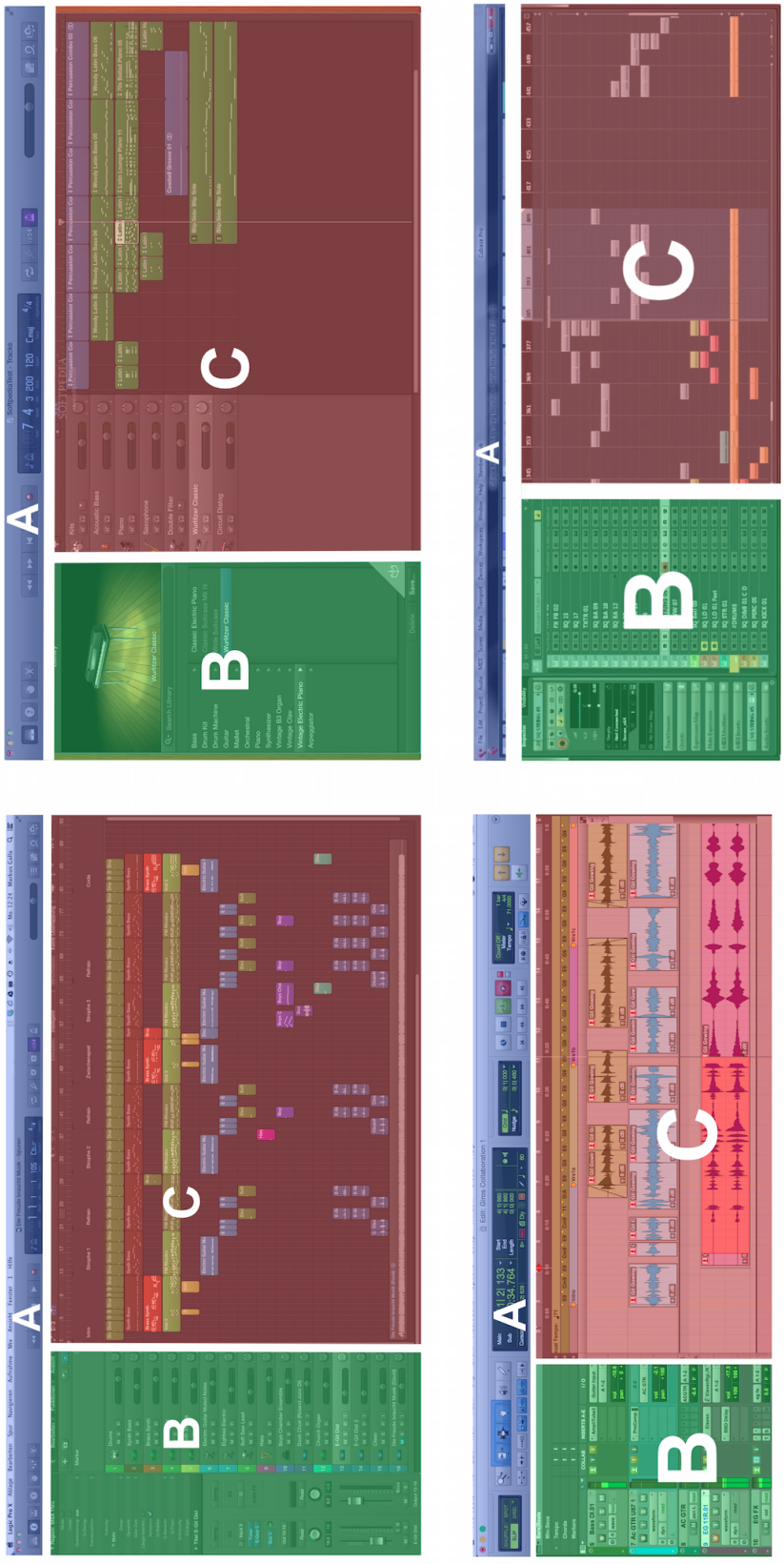

Abbildung A.4: Aufbauvergleich der DAWs Logic Pro X, Garageband, Pro Tools und Cubase (v.l.)

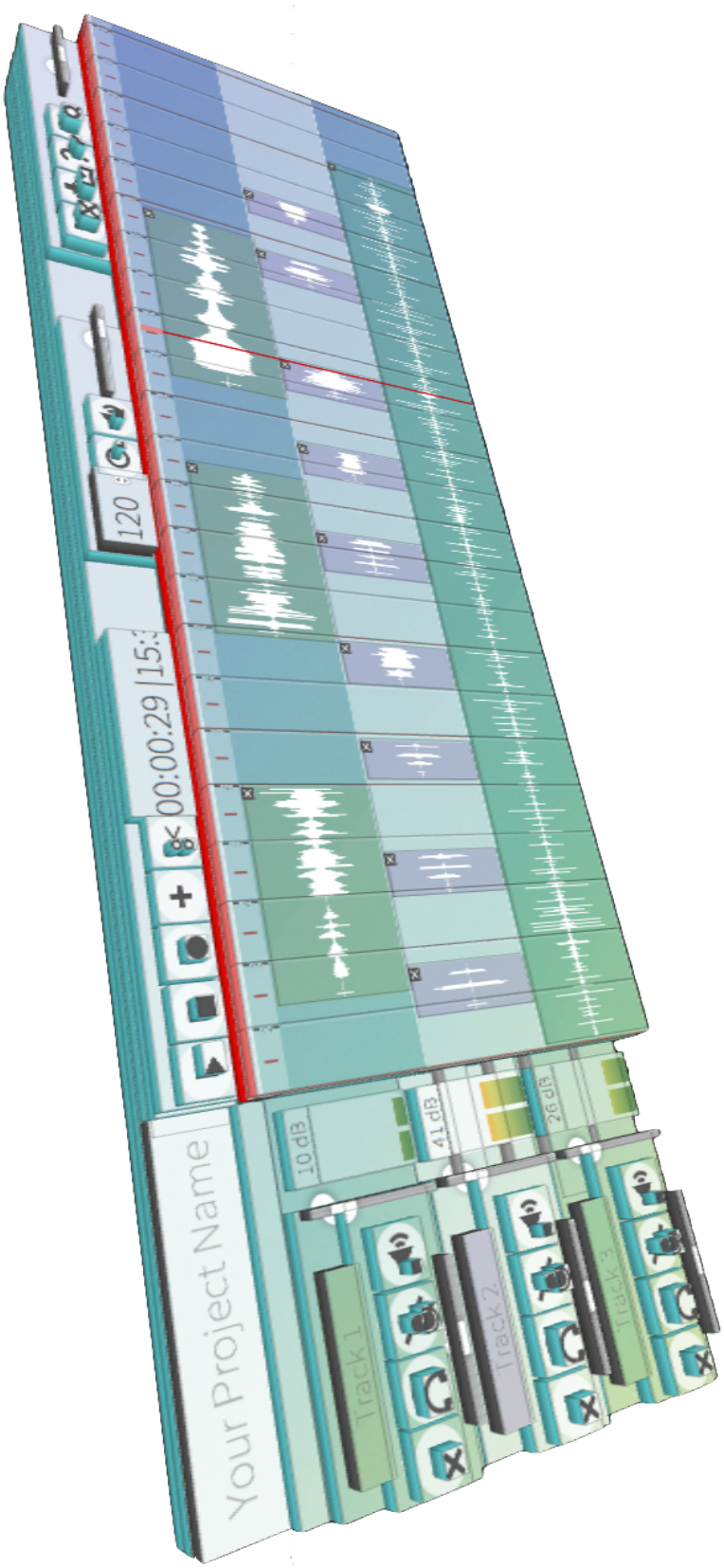

Abbildung A.5: 3D Visualisierung der HTML Divs

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

Adenot, Paul & Wilson, Chris: *Web Audio API*, http://webaudio.github.io/web-audio-api/, 2015, letzter Zugriff: 20. 8. 2015

Burnett, Daniel & Bergkvist, Adam & Jennings, Cullen: *Media Capture and Streams*, https://w3c.github.io/mediacapture-main/getusermedia.html, 2015, letzter Zugriff: 20. 8. 2015

Cabanier, Rik & Mann, Jatinder & Munro, Jay & Wiltzius, Tom & Hickson, Ian: *HTML Canvas 2D Context*, http://www.w3.org/TR/2dcontext/, 2015, letzter Zugriff: 20. 8. 2015

Dirksen, Jos: *Exploring the HTML5 Web Audio: visualizing sound*, http://www.smartjava.org/content/exploring-html5-web-audio-visualizing-sound, 2012, letzter Zugriff: 20. 8. 2015

Gamma, Erich & Helm, Richard & Johnson, Ralph & Vlissides, John: *Design Patterns - Elements of Reusable Object-Oriented Software*, Eddison-Wesley 1995, als eBook http://rasan.net/wp-content/uploads/DesignPatterns.pdf, letzter Zugriff 20.8.2015

Gronow, Pekka & Saunio, Ilpo: *An International History of the Recording Industry*, Cassell Academic 1999, als eBook https://books.google.de/books?isbn=030470590X, letzter Zugriff 20.8.2015

Jones, Robert: *Visualizing Audio #3 Time Domain Summary*, http://apprentice.craic.com/tutorials/32, 2013, letzter Zugriff: 20. 8. 2015

Kropff, Peter: *OOP mit JavaScript*, http://www.peterkropff.de/site/javascript/oop.htm, 2014, letzter Zugriff: 20. 8. 2015

Marcotte, Ethan: *Responsive Web Design*, http://alistapart.com/article/responsive-web-design, 2010, letzter Zugriff: 20. 8. 2015

Mozilla Developer Network: *Web Audio API*, https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API, 2015, letzter Zugriff: 20. 8. 2015

Pizka, Markus & Bauer, Andreas: *A Brief Top-Down and Bottom-Up Philosophy on Software Evolution*, Institut für Informatik, Technische Universität München 2005, als pdf http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.6071&rep=rep1&type=pdf, letzter Zugriff: 20. 8. 2015

Schoenherr, Steve: *Recording Technology History*, http://www.aes.org/aeshc/docs/recording.technology.history/notes.html, 2005, letzter Zugriff: 20. 8. 2015

Smus, Boris: *Web Audio API*, O'Reilly 2013, als eBook http://chimera.labs.oreilly.com/books/1234000001552/index.html, letzter Zugriff: 06. 8. 2015

WC3: *HTML5 Specification*, http://www.w3.org/TR/html5, 2014, letzter Zugriff: 20. 8. 2015

Wilson, Chris: *A Tale of Two Clocks - Scheduling Web Audio with Precision*, http://www.html5rocks.com/en/tutorials/audio/scheduling, 2013, letzter Zugriff: 20. 8. 2015

Wyse, Lonce & Subramanian, Srikumar: *The Viability of the Web Browser as a Computer Music Platform*, Computer Music Journal 37:4, pp. 10–23, Winter 2014 http://www.mitpressjournals.org/doi/pdf/10.1162/COMJ_a_00213, letzter Zugriff: 20. 8. 2015

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 Jakob Sudau