Melody recognition
via microphone

Inhalt

1.Motivation

Es gibt bereits zahlreiche Anwendungen, die es ermöglichen per MIDI-Keyboard Melodien einzuspielen. Jedoch haben diese Anwendungen den Nachteil, dass man dafür ein Keyboard besitzen muss. Aus dieser Grundlage heraus entstand die Idee Melodien per Mikrofon aufzunehmen, zu erkennen und darzustellen. Dadurch ist es für alle Musiker möglich ihre Ideen und Gedanken schnell musikalisch darzustellen, anstatt sie mühsam per Hand auszuschreiben. Egal ob Gesang, Streich- oder Blasinstrument. Da das Programm die Notation der Melodie übernimmt, wird der Musiker nicht mehr in seinem kreativen Schaffen ausgebremst. Er kann seinen Instinkten folgend improvisieren und die Melodie erschaffen und muss nicht parallel mitschreiben oder im Nachhinein aufgenommene Melodien mühevoll nach Gehör abbilden. Das spart eine Menge Arbeit.

Zudem ermöglicht die automatische Erkennung der Melodie auch Stellen in bereits bekannten Stücken wiederzuerkennen und eröffnet so auch ganz neue Forschungsmöglichkeiten, wie die automatische Positionierung in Noten oder Applikationen, die selbstständig umblättern für Solisten.

2. Projektdefinition

2.1. Projektziele / Nicht-Ziele

Ziel des Projekts ist es eine einstimmige Melodie über das Mikrofon erkennen zu können und grafisch im westlichen Notensystem darzustellen. Aufgrund der Komplexität mehrstimmiger Eingaben und möglicher Interferenz zwischen den Frequenzen wird die Mehrstimmigkeit zunächst nicht weiter berücksichtigt. Die Taktart wird vom Nutzer vorgegeben, muss also nicht von dem Programm erkannt werden. Der Fokus liegt darin die musikalische Stimmung an das Instrument des Nutzers anzupassen, Transponierung zu ermöglichen, sowie adaptiv leichte Tempoveränderungen des Nutzers zu erkennen und in der Notation wieder zu korrigieren. Oberstes Ziel ist eine funktionierende und simple Oberfläche, um den dahinterliegenden Mechanismus nutzbar zu machen. Sie soll vor allem dazu dienen einfach und schnell Noten „niederzuschreiben“. User Experience ist zunächst vernachlässigbar und wird nur betrachtet, wenn der zeitliche Rahmen des Projekts dies zulässt. Orientalische Notensysteme (Vierteltöne, etc.) werden nicht berücksichtigt. Es ist ebenfalls nicht möglich, sämtliche musikalischen Notationsstile komplett abzudecken oder Stücke ohne feste Taktart/mit wechselnder Taktart darzustellen. Die Anwendung ist dazu gedacht, Melodien zu notieren, die dann mit weiteren Programmen (z.B. MuseScore) um Lautstärkenangaben, etc. erweitert werden können. Aufgrund der weiteren Bearbeitbarkeit des MusicXML-Formats in anderen Programmen, erlaubt das Projekt eine hohe Fehlerschwelle bei der Rhythmuserkennung. Das Hauptaugenmerk liegt darauf, die Tonhöhe richtig zu erkennen, da dies ist, was den meisten Musikern am schwersten fällt.

2.1.1. Muss-Ziele

Die Anwendung muss auf jeden Fall einzelne Melodien erkennen und richtig darstellen können. Dabei sollte sie in der Lage sein über intonatorische[1] und rhythmische Ungenauigkeiten des Nutzers hinwegzusehen. Eine Export-Funktion der MusicXML-Datei zur weiteren Bearbeitung in externen Programmen ist ebenfalls essentiell.

2.1.2. Kann-Ziele

Mögliche Ziele sind priorisiert die Transponierung der Stücke, sowie eine selbstständige Tonarterkennung des Programms. Ist dies gewährleistet, soll die Notation erweitert werden, um sich intelligent verschiedenen Musikstilen anzupassen. (z.B. Erkennen musikalischer Muster wie ein Swing Rhythmus, Triller, usw.). Zudem bestünde die Möglichkeit, ein UI zu designen, das es ermöglicht, nacheinander mehrere Stimmen einzuspielen und diese in einer MusicXML-Datei darzustellen.

[1] Eine korrekte Intonation bezeichnet das (nahezu) exakte Treffen der musikalisch korrekten Frequenz.

2.2. Projektumfang

Im Rahmen des Projekts wurde der zeitliche Anteil von Dokumentation, Initiierung & Deployment, Bugfixes, Testing und Entwicklung ermittelt. Die zusätzlichen Anforderungen der Uni (z.B. Betreuen des Media-Night-Stands) wurden nicht berücksichtigt, da sie keine unmittelbare Arbeit am Projekt darstellen. Von den ermittelten Aufwänden wurden ca. 57% für die eigentliche Entwicklung verwendet. Nachfolgende Diagramme zeigen die Anteile von Dokumentation, Initiierung & Deployment, Bugfixes, Testing und Entwicklung, sowie eine Unterteilung der Anteile innerhalb der Entwicklung in Prozent:

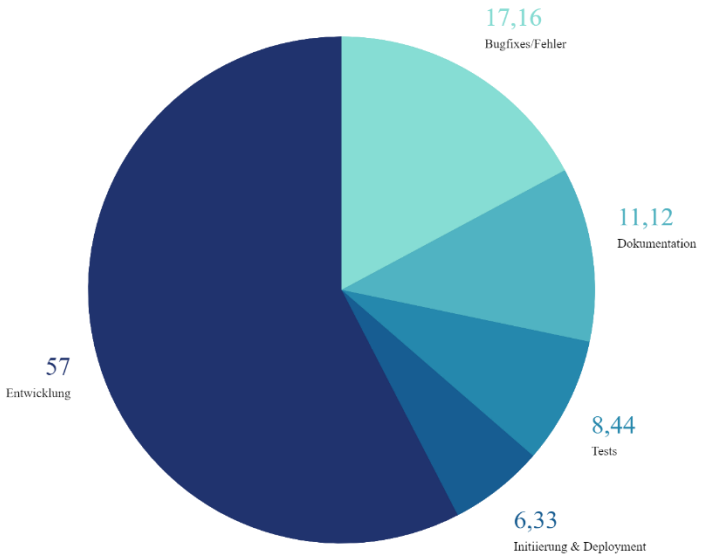

Abbildung 1: Anteile der Bereiche an Gesamtaufwand

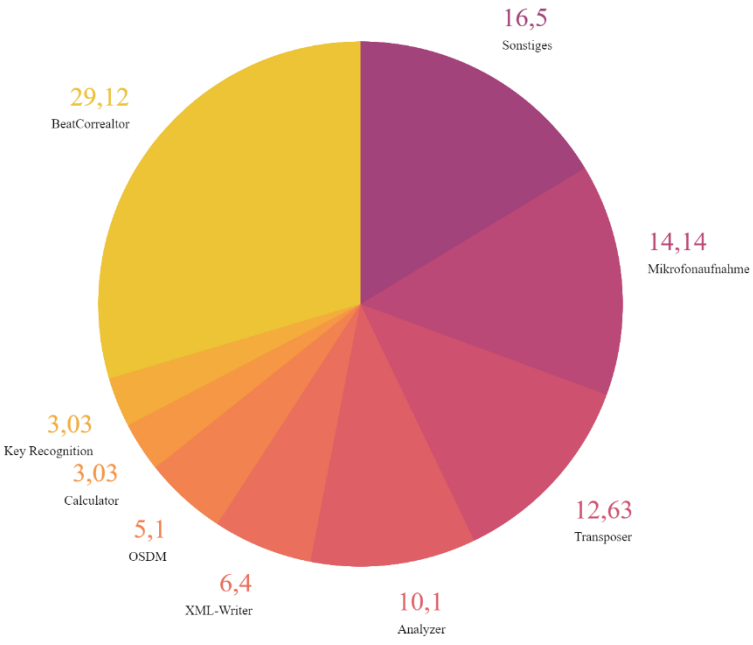

Abbildung 2: Anteile der Bereiche innerhalb der Entwicklung

2.3. Technologien

Aufgrund der schlechten Lesbarkeit von JavaScript und der Komplexität der Berechnungen wurde beschlossen, soweit möglich, TypeScript zu verwenden.
Für die Manipulation des DOMs wird React verwendet.
Als Testing-Framework wird jest verwendet und zum Ausführen des Programms NodeJS.

2.3.1. Software

Zur Erkennung der Frequenz wurde das wad-Framework von rserota[2] verwendet, da dieses bereits einen Tuner enthält, der eine sehr leichte Anbindung an das Mikrofon ermöglicht und direkt die ermittelte Frequenz zurückgibt.
Zur grafischen Darstellung der MusicXML im Browser wurde OpenSheetMusicDisplay[3] verwendet.

2.3.2. Hardware

Es wird keine spezielle Hardware benötigt. Es wird empfohlen ein Headset zu verwenden, da ansonsten bei Windows während der Ausgabe des Metronoms das Mikrofon sehr stark runtergeregelt wird, damit der Computer sich selbst nicht hört.

3. Konzeption

3.1. Architektur-Entwurf

Das Projekt folgt einer angepassten MVC-Architektur. Es ist aufgeteilt in Models, Algorithmen und Views. Die Architektur sollte möglichst modular gestaltet werden.

Die Views enthalten nur die unmittelbare Logik für das UI.
Die Views sind in einer Index.tsx eingebunden und ihre Anzeige (und Ausgabe von Ton) wird von dieser kontrolliert. Dadurch hat nur die Index.tsx Abhängigkeiten auf die Views und es sind keine Abhängigkeiten zwischen den Views nötig:

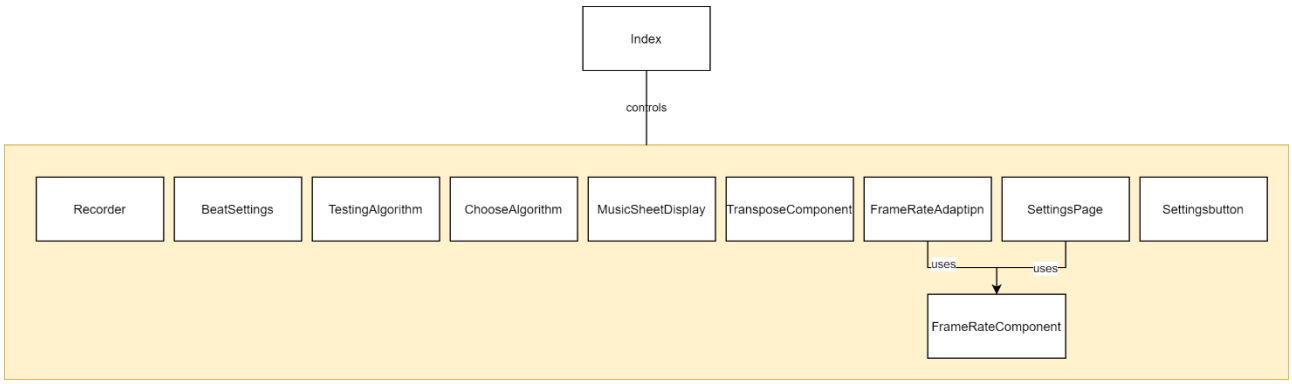

Abbildung 3: Architektur der Views

Die Backend-Logik folgt der Idee einer Pipeline. Diese enthält keine Abhängigkeiten zu den Views. Die Algorithmen lassen sich aufteilen in die Dateien Analyzer, Calculator, BeatCorrelator, KeyRecognizer, Transponer und XML-Writer, die sich wie in einer Pipeline „aneinanderstecken“ lassen. Damit die Algorithmen voneinander unabhängig sind, gibt es auch dort eine Steuerklasse, die sogenannte Pipeline. Diese ist aufgeteilt in zwei Abschnitte. Zunächst durchläuft der Input die drei verschiedenen Algorithmen im Analyzer und danach mit dem jeweiligen Ergebnis den Calculator. Sobald das geschehen ist, endet der Aufruf der Backend-Logik und dem Nutzer werden die drei Ergebnisse auditiv ausgegeben. Nachdem

[2] https://github.com/rserota/wad#pitch-detection
[3] https://opensheetmusicdisplay.org/

dieser sich für einen Algorithmus für die Frequenzzusammenfassung entschieden hat, wird die Pipeline fortgesetzt.

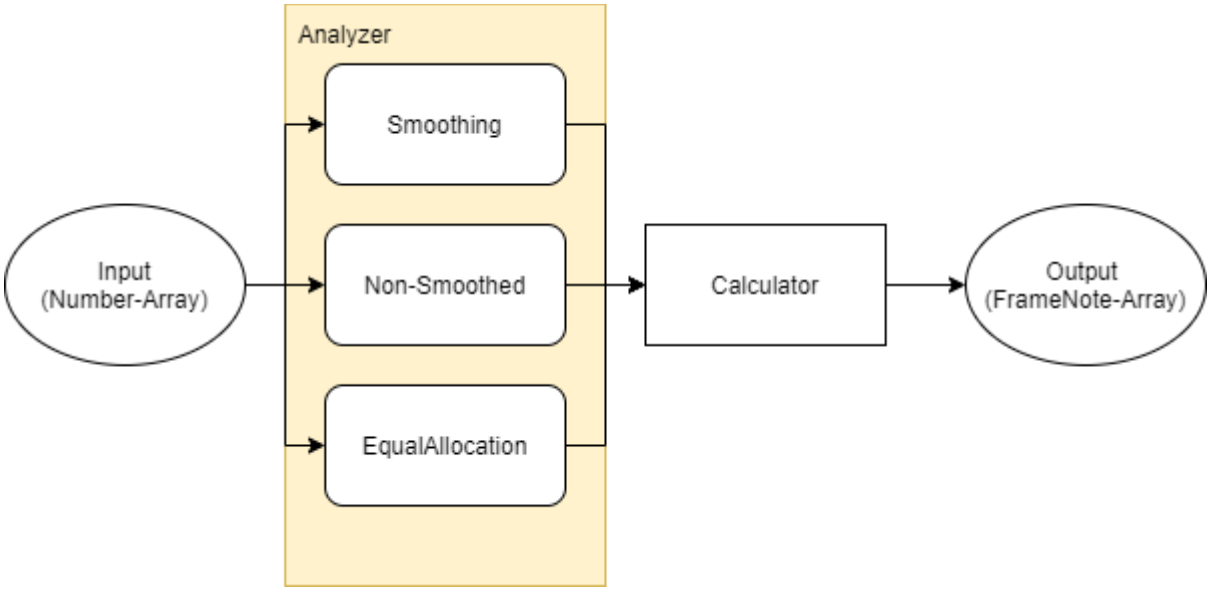

Abbildung 4: Teil 1 der Pipeline

Der zweite Teil der Pipeline besteht aus KeyRecognizer oder Transponer (je nachdem, ob transponiert wird - und somit die Tonart bereits bekannt ist - oder nicht), BeatCorrelator und XML-Writer. Dieser zweite Teil wird nach Auswahl des verwendeten Analyzer-Algorithmus bei jeder Änderung in den Einstellungen auf der Endseite neu aufgerufen. Hierbei bleibt der Datentyp der Daten bei dem KeyRecognizer und dem Transponer gleich, es wird lediglich auf dem FrameNote-Array gearbeitet (s. Abb. 5) (Kreuze zu Bs konvertiert, bei Tonarten mit Bs als Vorzeichen) und zusätzlich Output generiert, den später der XML-Writer benötigt.

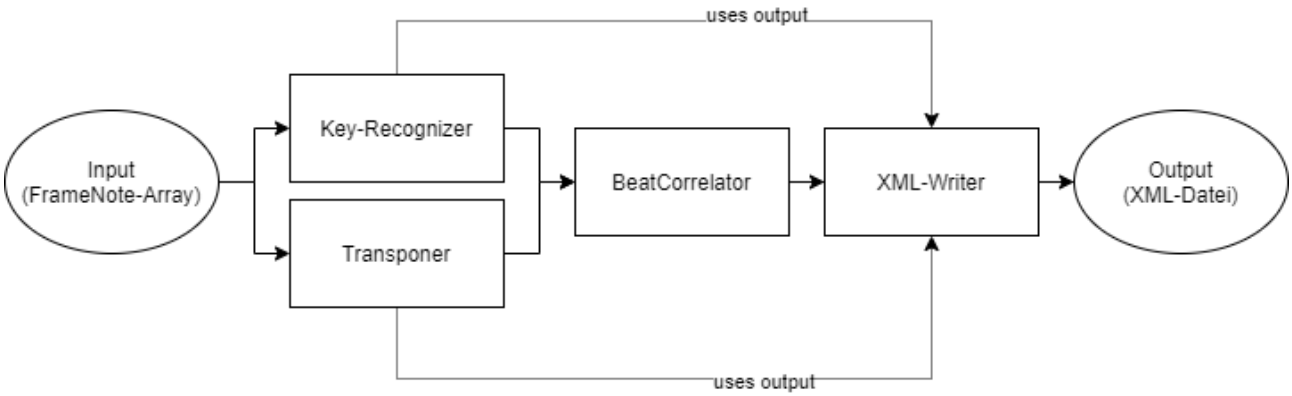

Abbildung 5: Teil 2 der Pipeline

Die Models enthalten Klassen, die meist gar keine, selten wenige Methoden enthalten. So enthält Beats.tsx z.B. Methoden, die allerdings rein informativer Natur über die Taktart sind und keine ausführende Logik enthalten. Diese Models dienen als Datenstrukturen oder enthalten globale Konstanten.

Außerdem gibt es noch Helper-Klassen, die aus mehreren Stellen im Code verwendet werden.

3.2. Use-Case Diagramm

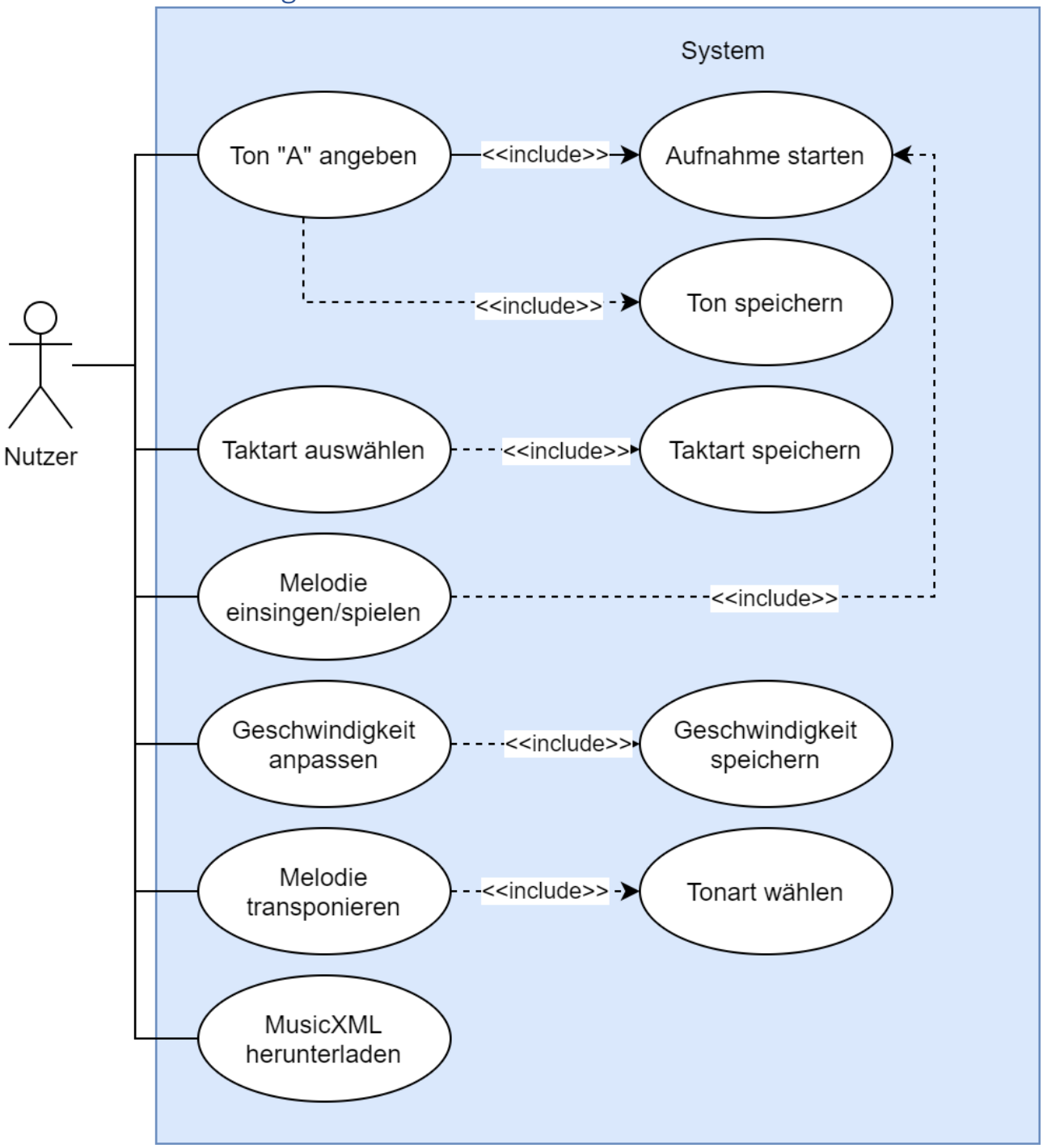

Abbildung 6: Das Use-Case-Diagramm des Programms

3.3. Programmablaufplan

Der Programmablaufplan ist aufgrund der Größe in die Einstellung des Systems (Referenzton erstellen und Taktart wählen) und das Einsingen/Spielen und Verarbeiten der Melodie aufgeteilt. Die Konfigurationsmöglichkeiten sind nicht mit im Programmablaufplan aufgeführt, da sie nicht Teil des typischen Programmablaufs sind, sondern diesen zu beliebiger Zeit unterbrechen. Bei der Verarbeitung der Melodie ist die Pipeline sehr gut zu erkennen.

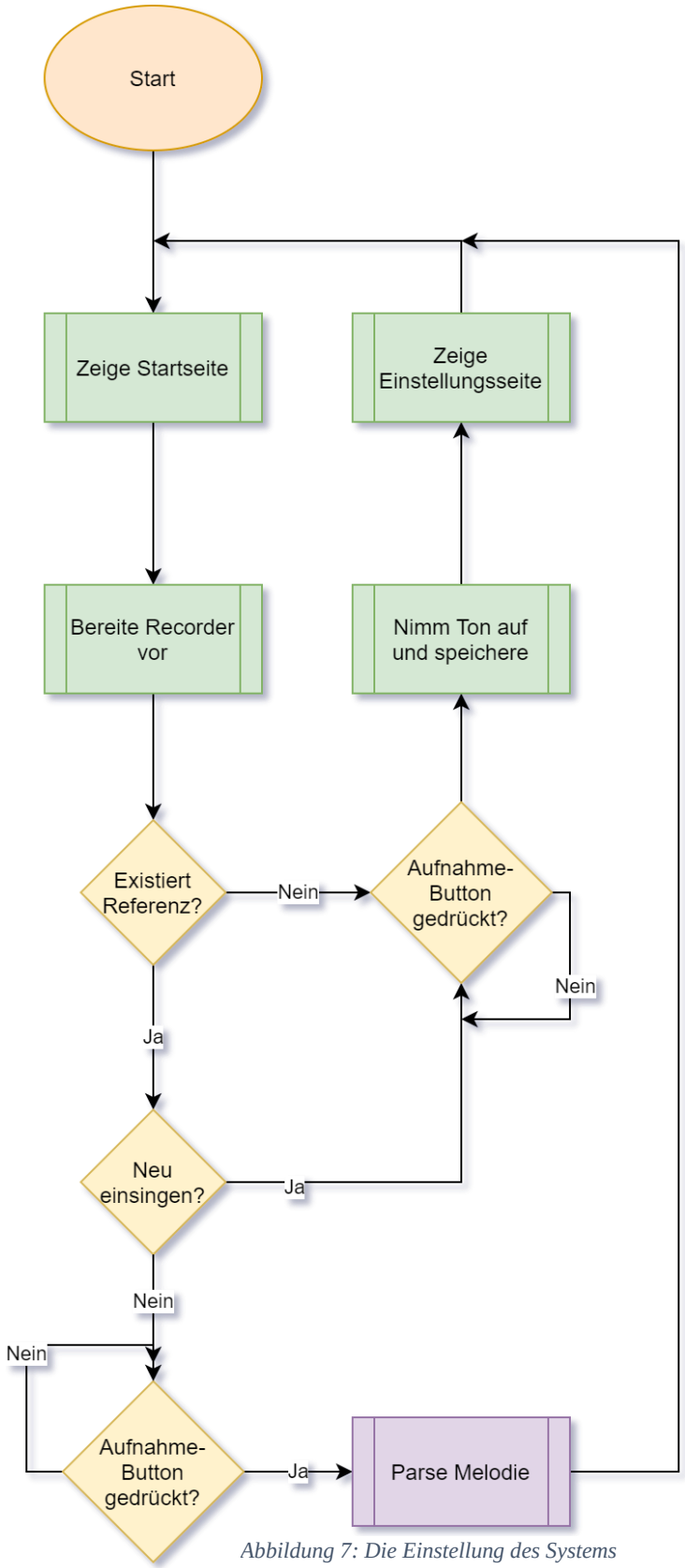

Abbildung 7: Die Einstellung des Systems

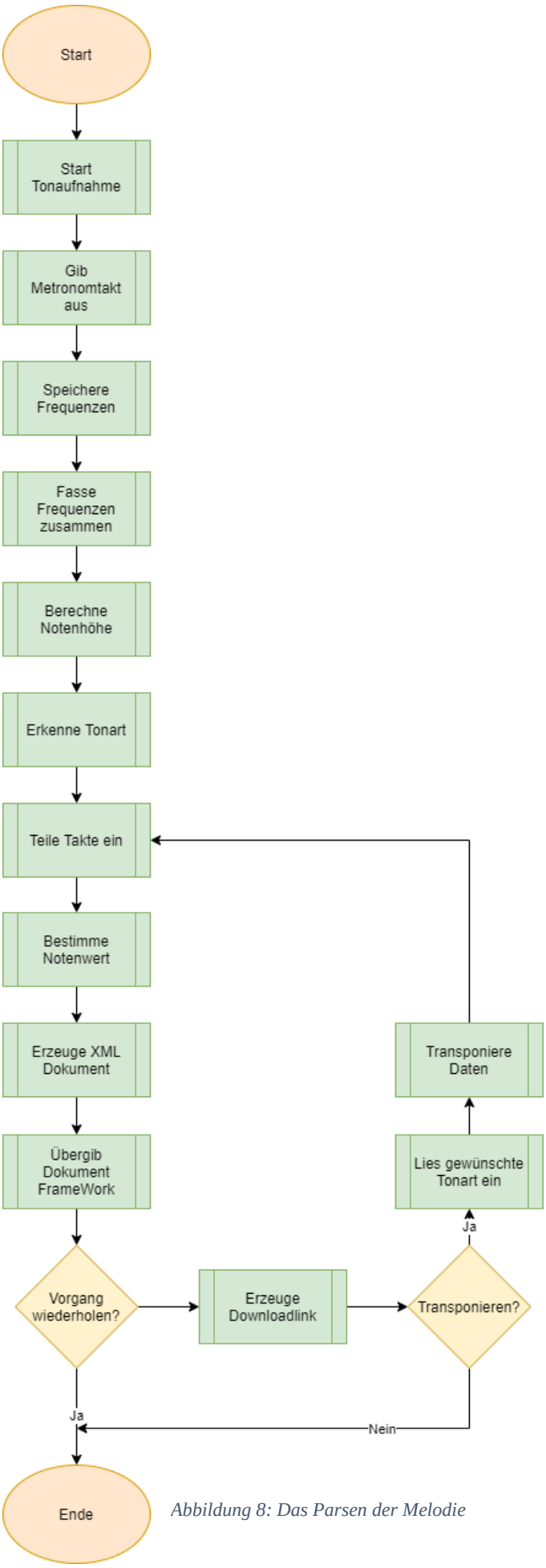

Abbildung 8: Das Parsen der Melodie

3.4. Risikoanalyse

Die Risikoanalyse ist in einem separaten Dokument angehängt und wurde im Rahmen der IT-Projektmanagement-Vorlesung angefertigt.

4. Architektur

4.1. Projektstrukturplan

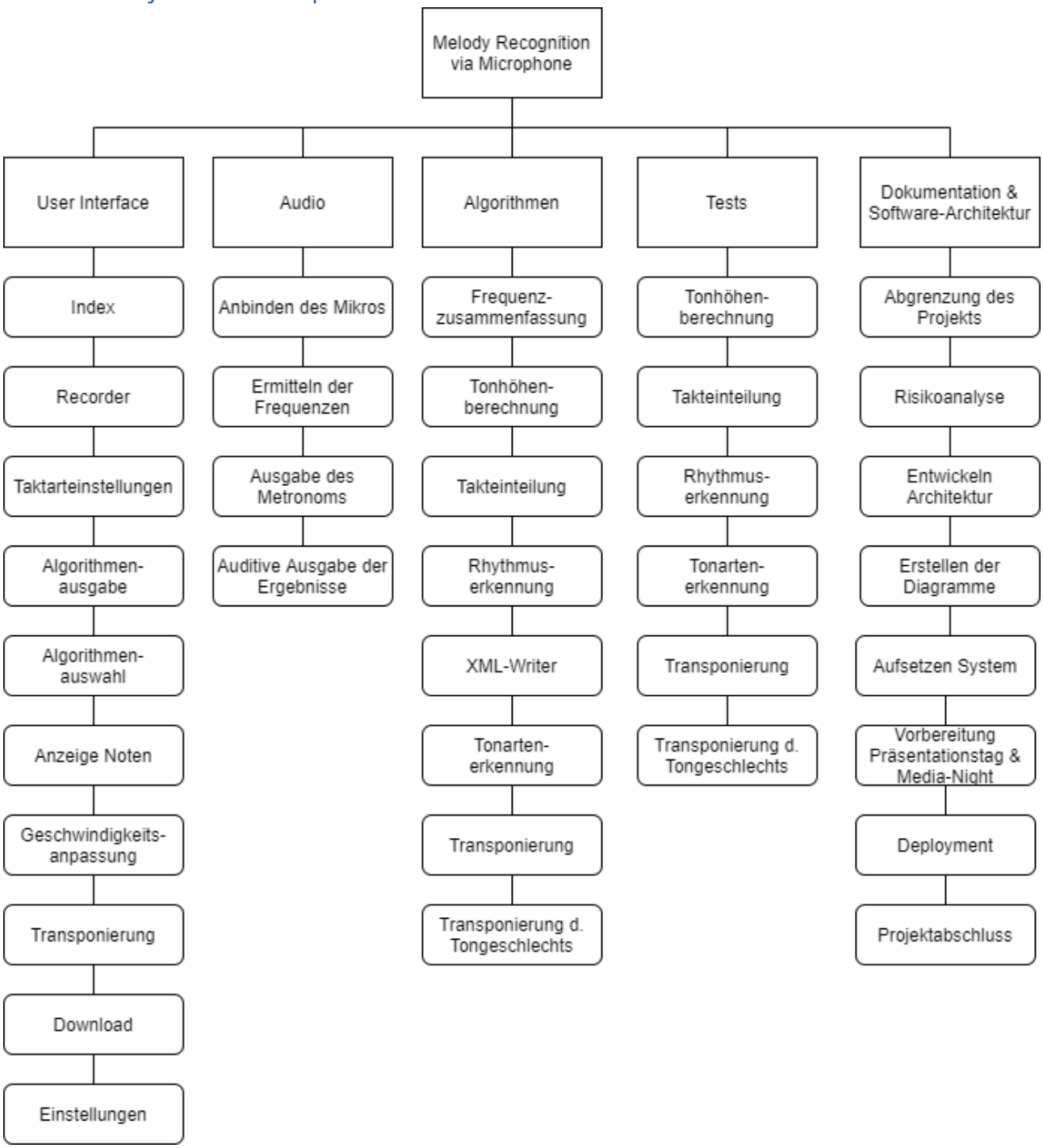

Abbildung 9: Der Projektstrukturplan

4.2. Grafisches User Interface

Abbildung 10: Nach Einsingen der Melodie

Abbildung 11: Einstellungen

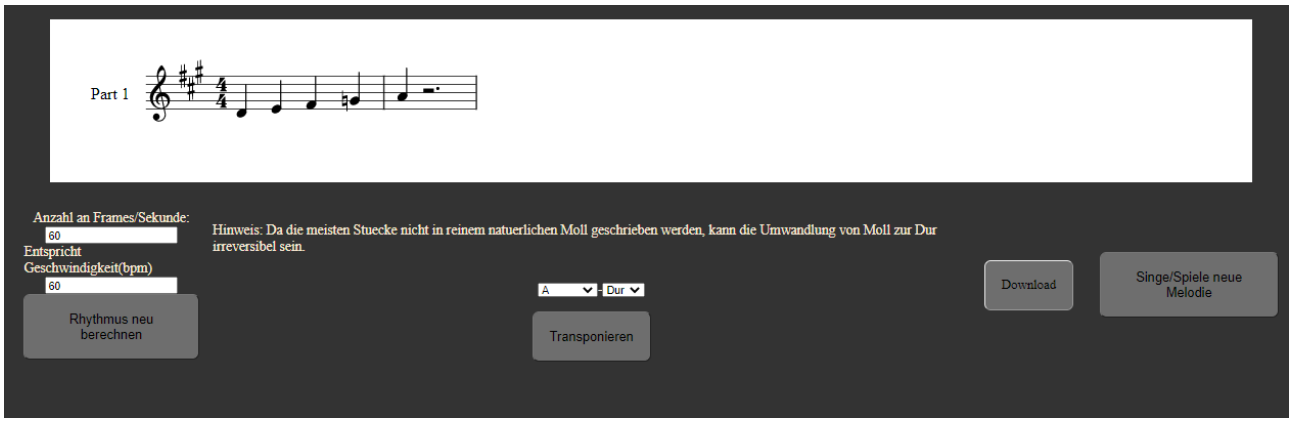

Abbildung 12: Ergebnisseite

5. Ergebnis

5.1. Projektablauf

Zu Beginn wurde ein Großteil der Projektdokumentation angefertigt (Umfang, Ziele und Nicht-Ziele) und sich eine grundlegende Architektur überlegt. Im Rahmen dieser Initiierungsphase wurde dann entschieden, Typescript zu verwenden, um die Qualität des Codes trotz der Größe des Projektes zu gewährleisten. Daraufhin wurde mit der Entwicklung begonnen. Schnell wurde jedoch klar, dass das DOM so viel manipuliert wird, dass jQuery keine sinnvolle Lösung dafür darstellt. Ich wurde auf React aufmerksam und das Projekt wurde damit neu aufgesetzt.

In der Entwicklungsphase wurde zunächst die Grundfunktionalität der Melodieerkennung bis zur Darstellung in MusicXML implementiert. Danach wurden Features hinzugefügt, wie das Anpassen der Geschwindigkeit, das Transponieren der Tonart und die Konfiguration verschiedener Settings.

Im Rahmen der Media-Night kam das Projekt dann auch in die Deployment-Phase. In dieser wurde es als Open-Source-Projekt auf Github veröffentlicht und mit Vercel deployt. Die Anwendung ist unter dem Link https://melody-recognition-via-microphone.vercel.app/ bereitgestellt.

5.2. Erfahrungen und Knowhow

Im Zuge des Projektes habe ich TypeScript und React gelernt. Außerdem habe ich mich mit dem Testing-Framework jest auseinandergesetzt, mit dem (ausgenommen Analyzer) sämtliche Algorithmen getestet wurden.

Außerdem musste ich mich im Zuge des Projekts sehr intensiv mit der Mathematik hinter Musik auseinandersetzen. Dadurch habe ich viel Erfahrung über Quintenzirkel, rhythmische Einheiten und das westliche Notensystem gelernt.

Auch wenn ich im letztendlichen Projekt keine Fourier-Transformation verwendet habe und ein entsprechendes Framework für die Frequenzerkennung gefunden habe, habe ich im Rahmen meiner Recherche doch auch diesbezüglich und auch zum PCM-Format einiges gelernt.

Außerdem habe ich den Umgang mit Visual Studio 2019 und Visual Studio Code gelernt, sowie das Debuggen von Tests in Typescript mithilfe von VS Code.

Viele Algorithmen waren hoch komplex, sodass die Notwenigkeit von Tests, ganz besonders für Randfälle, mir schnell klar wurde. Gerade dort habe ich Ablaufdiagramme zur Orientierung bei der Implementierung schätzen gelernt und mehrfach verwendet. So habe ich im Zusammenhang mit der Entwicklung auch meine Fähigkeiten im Erstellen von Flussdiagrammen und dem Umgang mit draw.io verbessert.

Im Rahmen der Media-Night habe ich mich zudem mit dem Deployment von NodeJS Projekten auseinandergesetzt.

6. Anhang

6.1. Quellcode auf Github

https://github.com/Sahara150/melody-recognition-via-microphone