Softwarebasierte Realisierung eines Funkmeldeempfängers / Fünftonfolgendecoders mittels Javascript

Eine Hausarbeit von:

David Breidert Talstraße 33

07545 Gera [david.breidert@me.com](mailto:david.breidert@me.com)

Eingereicht bei:

Thüringer Landesfeuerwehr- und Katastrophenschutzschule

Silbitzer Weg 6

07586 Bad Köstritz

# Inhaltsverzeichnis

[1Einleitung 3](#_TOC_250015)

* 1. [Die analoge Alarmierung 3](#_TOC_250014)
     1. Die Fünftonfolge 4
     2. Vor- und Nachteile der analogen Alarmierung 5

[2Das Projekt: ZVEI-Decoder 7](#_TOC_250013)

* 1. [Ziele 7](#_TOC_250012)
     1. Funktionen 7
  2. [Auswahl der Programmiersprache 7](#_TOC_250011)

[3Analyse des Audiosignals 9](#_TOC_250010)

* 1. [Grundlagen 9](#_TOC_250009)
  2. [Abtasten des analogen Signals 9](#_TOC_250008)
     1. Ansatz 1: Autokorrelation 10
     2. Ansatz 2: Frequenzspektrumananalyse 11

[4Umsetzung in Javascript 12](#_TOC_250007)

* 1. [Die Benutzeroberfläche 12](#_TOC_250006)
  2. [Funktion mounted 13](#_TOC_250005)
  3. Funktion updateData 14
  4. [audio.js 16](#_TOC_250004)
     1. Funktion getUserAudio 16
     2. Funktion createContext 16
     3. Funktion getCurrentFrequencyFft 17
  5. [zvei.js 18](#_TOC_250003)
     1. Funktion getTonNummer 18
     2. Funktion getValidatedTonfolge 18
  6. [Die Einsatzmitteldatenbank 20](#_TOC_250002)

[5Anhang 21](#_TOC_250001)

* 1. [Bedienungsanleitung 21](#_TOC_250000)
     1. Links 21
     2. Benötigte Hardware 21
     3. Installation 21
     4. Nötige Windows-Einstellungen 21
     5. Nach dem Programmstart 22
     6. Dashboard 24
     7. Einsatzmitteldatenbank 25

# Einleitung

3

In Thüringen gibt es 1.609 freiwillige Feuerwehren mit insgesamt ca. 34.000 akti- ven Mitgliedern in den Einsatzabteilungen ([1]).Während in den größeren Städten Berufsfeuerwehren 24 Stunden und 7 Tage die Woche auf der Feuerwache in Ein- satzbereitschaft sind, sind die freiwilligen Kräfte vor allem im ländlichen Bereich essentiell, um den Brandschutz und die dringliche technische Hilfe sicherzustellen.

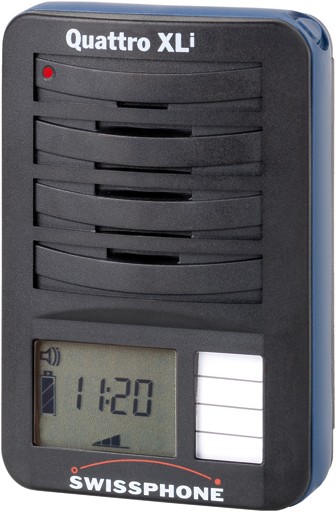
Diese Hausarbeit beschäftigt sich primär mit der Alarmierung dieser freiwilligen Kräfte.

## Die analoge Alarmierung

Die Alarmierung von Feuerwehren und Rettungsdiensten in Thüringen erfolgt flä- chendeckend noch über den 4m-Analogfunk, also im gleichen Netz in dem auch der Sprechfunkverkehr stattfindet. Alle Teilnehmer dieses Netzes können sich ge- genseitig hören und sich untereinander verständigen.

Da auf dem selben Kanal hunderte verschiedene Kräfte alarmiert werden, ist es aufgrund der niedrigen Einsatzfrequenz einzelner Feuerwehren / Rettungsdienste

– im Vergleich mit der Gesamteinsatzfrequenz – unrealistisch, dass jeder Feuer-



*Abb. 1: typischer Funkmeldeempfänger*

*(Quelle:* [*https://www.gfd-katalog.com/ludwig\_feuerschutz/quattro-xlixlsi/quattro-xli/25597*](https://www.gfd-katalog.com/ludwig_feuerschutz/quattro-xlixlsi/quattro-xli/25597)*)*

wehrmann bzw. Rettungsdienstmitarbeiter stets ein Funkgerät mit sich trägt und darauf wartet angesprochen zu werden. Deshalb wurde ein alternatives System entwickelt, mit dem die Einsatzkräfte auf eine Alarmierung aufmerksam gemacht werden:

Einsatzkräfte größerer freiwilliger Feuerwehren sowie von Berufsfeuerwehren und Rettungsdiensten tragen für die Alarmierung meist einen sogenannten Funkmel- deempfänger (FME) mit sich. Die Funkmeldeempfänger werden im Einsatzfall von der alarmierenden Stelle – meist ist dies die zuständige zentrale Leitstelle – aus- gelöst. Hierzu wird eine, auf dem genutzten Funkkanal einzigartige, Tonfolge in den Funk eingespielt, die auf den entsprechenden Funkmeldeempfängern pro- grammiert ist. Bei Erkennen einer programmierten Tonfolge gibt der Melder ein Alarmsignal und schaltet im Anschluss den Funk laut, woraufhin der Disponent eine Alarmdurchsage mit den Einsatzdaten durchsagen kann.

Da die Anschaffung und Programmierung von Funkmeldeempfängern ein großer Kostenfaktor sein kann, wählen kleinere Feuerwehren oft eine andere Variante, um die Alarmierung der eigenen Kräfte sicherzustellen: die Sirenenalarmierung.

Bei der Sirenenalarmierung wird von der alarmierenden Stelle statt den Funkmel- deempfängern eine Sirene ausgelöst, die im betroffenen Ort zentral platziert ist, um möglichst viele Einsatzkräfte zu erreichen.

* + 1. Die Fünftonfolge

Egal ob Sirene oder Funkmeldeempfänger, die Auslösung des Alarms basiert auf einem deutschlandweit einheitlichen System.

Im Alarmfall spielt die alarmierende Stelle in den Funkkanal, auf den die Funkmel- deempfänger und/oder Sirenen programmiert sind, sogenannte Fünftonfolgen ein. Fünftonfolgen sind – wie der Name schon sagt – eine Abfolge von fünf einzelnen Tönen, die von dem jeweiligen Empfangsgerät erkannt werden.

Der Aufbau dieser Tonfolge ist deutschlandweit einheitlich und in einer techni- schen Richtlinie[1](#_bookmark0) festgelegt worden. Unter ist in dieser Richtlinie zu finden welche Frequenzen (siehe Abb. 2) sowie Dauer die einzelnen Töne der Tonfolge haben sollen, wie oft diese wiederholt werden soll, und welche zeitlichen Abstände zwi-

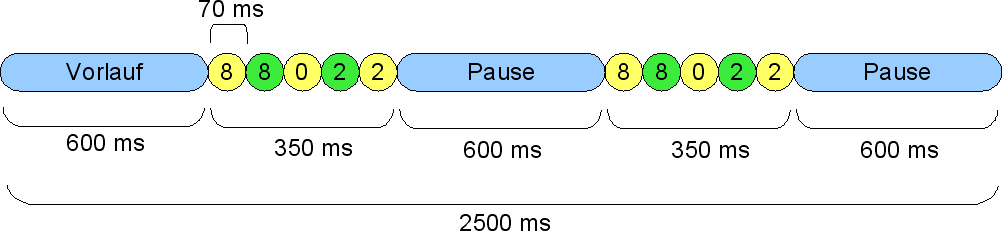
1 https://[www.lfs-bw.de/Fachthemen/Digitalfunk-Funk/Documents/Pruefstelle/TRBOS-Funkalar-](http://www.lfs-bw.de/Fachthemen/Digitalfunk-Funk/Documents/Pruefstelle/TRBOS-Funkalar-) mierung.pdf

schen mehreren hintereinander eingespielten Tonfolgen eingehalten werden sol- len (siehe Abb. 3). Es wird auf zehn verschiedene Frequenzen (Ziffer 0-9) und eine Ausweichfrequenz für sich wiederholende Töne (Ziffer R) zurückgegriffen.

Diese Frequenzen richten sich in Deutschland nach dem ZVEI-Standard.



*Abb. 2: Tonfrequenzen, aus denen die Fünftonfolgen aufgebaut werden (Quelle: https://de.wikipedia.org/wiki/5-Ton-Folge)*



*Abb. 3: zeitlicher Ablauf einer Funkalarmierung Quelle: https://de.wikipedia.org/wiki/5-Ton-Folge*

So würde sich die Tonfolge 88910 zum Beispiel folgendermaßen aufbauen:

2000Hz – 2600Hz (da die Ziffer „8“ hier auf eine weitere Ziffer „8“ folgt wird die Re- servefrequenz codiert) – 2200Hz – 1060Hz – 2400Hz.

Dass diese Parameter einheitlich festgelegt sind ist wichtig, damit die empfangen- den Geräte die Tonfolgen zuverlässig erkennen können.

* + 1. Vor- und Nachteile der analogen Alarmierung

Die analoge Alarmierung bietet einige Vorteile, aber auch Nachteile gegenüber di- gitalen Alternativen, wie etwa eine auf POGSAC oder TETRA basierende Varian- te.

Einer der größten Nachteile der analogen Alarmierung – und auch der Grund war- um deutschlandweit der Trend immer mehr in Richtung digitale Alarmierung geht – ist die fehlende Verschlüsselung. Theoretisch kann jeder mit einem Funkgerät und dem nötigen technischen Fachwissen den Funk und somit auch die Alarmierung abgreifen.

Weitere Nachteile sind die Überlastung des Funkkanals durch lange Sprachdurch- sagen (Jede Alarmierung belgegt den Funkkanal für ca. 20-30 Sekunden), sowie

der fehlende Feldstärkealarm bei analogen Funkmeldeempfängern, wodurch beim Aufhalten in einem Funkloch eine Alarmierung verpasst werden kann.

Größter Vorteil gegenüber den digitalen Alternativen stellt die Alarmierungsge- schwindigkeit dar. Vom Auslösen des Alarms in der Leitstelle bis zum Ertönen des Funkmeldeempfängers vergeht meistens nicht mehr als eine Sekunde.

# Das Projekt: ZVEI-Decoder

In diesem Kapitel soll mithilfe von Quellcodeauszügen die Arbeitsabläufe des Pro- gramms erklärt werden, die zur Erkennung und Ausgabe der Fünftonfolgen führen

## Ziele

Ziel des Programms, dass in dieser Projektarbeit entwickelt wird ist die software- basierte Decodierung von Fünftonfolgen. Später soll auf diese Funktionalität mit zusätzlichen Modulen / Funktionalitäten aufgebaut werden, um Einsatzkräfte in ih- rer Arbeit zu unterstützen. Diese Funktionen können von einer SMS-Alarmierung, bis zu einer Audio-Aufzeichnung der Sprachdurchsage reichen.

Um diese Zusatzfunktionen im Falle einer Alarmierung entwickeln zu können ist es zwingend notwendig den Alarmfall zügig und zuverlässig erkennen zu können. Da Feuerwehren – wie in der Einleitung beschrieben – flächendeckend über Fünfton- folgen alarmiert werden, ist die Decodierung dieser zwingend notwendig um spä- ter darauf aufbauen zu können.

* + 1. Funktionen

Folgende Funktionen sollen im Rahmen der Entwicklung realisiert werden:

* + - * zuverlässige Decodierung der Fünftonfolgen eines Funkkanals über einen Funkscanner.
      * Visuelle Darstellung der registrierten Fünftonfolgen und des Zeitpunkts der Registrierung.
      * Verknüpfung von Fünftonfolgen mit einer Datenbank von Einsatzmitteln, die im Programm bearbeitet werden kann.

## Auswahl der Programmiersprache

Vor Beginn des Projekts galt es die Entscheidung zu Treffen, welche Program- miersprache verwendet wird. Aufgrund meiner persönlichen Erfahrung in der Web-

entwicklung entschied ich mich für einen unorthodoxen Weg und wählte Java- Script als alleinige Programmiersprache für ZVEI-Decoder.

JavaScript ist ursprünglich als Programmiersprache von Webbrowsern entstanden um einfache Skripte für Webseiten zu entwickeln. Mit der Zeit wurde das Potential von JavaScript immer größer. Heutzutage ist es möglich JavaScript Apps zu ent- wickeln und diese mittels Zusatzpaketen als eigenständige Programme zu packen.

ZVEI-Decoder nutzt folgende Technologien:

* ElectronJS ([https://www.electronjs.org](https://www.electronjs.org/)), für das Packen der Web-App als ei- genständige Anwendung
* VueJS ([https://vuejs.org](https://vuejs.org/)), als JavaScript-Framework
* VuetifyJS ([https://vuetifyjs.com](https://vuetifyjs.com/)), als User-Interface Bibliothek

Einen kompletten Webbrowser als App zu verpacken hat natürlich gegenüber an- deren Alternativen sowohl Nach- als auch Vorteile:

Durch die zwischengeschaltete JavaScript-Engine ist die Ausführgeschwindigkeit vieler Prozesse langsamer als bei einer hardwarenäheren Programmiersprache wie z.B. C++.

ZVEI-Decoder muss, um seine Funktion zu erfüllen eine Echtzeitanalyse eines Au- diostreams vornehmen, was bei mangelnder Leistung der Engine oder des Rech- ners zu Problemen führen kann.

Aus diesem Grund wurden vor dem endgültigen realisieren des Projekts einige Tests meinerseits durchgeführt, um die Leistungsfähigkeit dieser Umgebung zu testen. Das Ergebnis dieser Tests lies vermuten, dass die Leistung der JavaScript- Engine auf haushaltsüblichen PCs völlig ausreichend ist um eine solche Echtzeit- analyse durchzuführen.

Somit konnte die vermutete Einschränkung durch die Programmiersprache nicht bestätigt werden und das Vorhaben ungehindert weiter durchgeführt werden.

Ein großer Vorteil der gewählten Programmierumgebung ist hingegen die Portier- barkeit. Das fertige Programm kann in kürzester Zeit von einer Windowsanwen- dung auch als Macanwendung, oder sogar als Webanwendung umgeschrieben werden. Somit sind Nutzer nicht an ein Betriebssystem gebunden.

# Analyse des Audiosignals

## Grundlagen

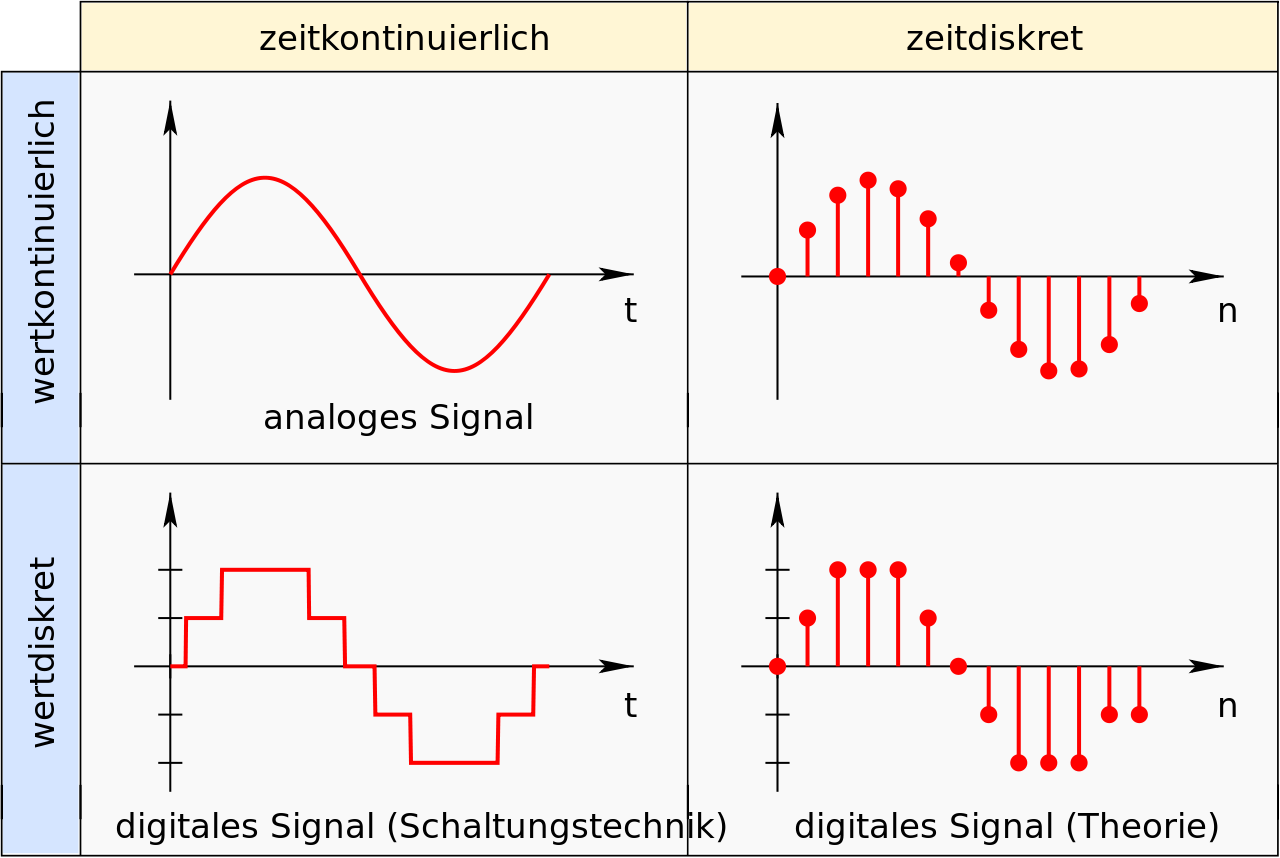
In den Funk eingesprochene Sprache oder eingespielte Töne sind analoge Signa- le, welche mittels Frequenzmodulation auf eine Trägerfrequenz moduliert werden, damit diese übertragen werden können. Der Empfänger des Signals demoduliert im Anschluss die sogenannte Niederfrequenz von der Trägerfrequenz, um diese dann z.B. über einen Lautsprecher auszugeben.

Um das analoge Funksignal mittels Software auswerten zu können, muss dieses in ein digitales Signal umgewandelt werden (Analog / Digital – Umsetzung).

Dabei wird ein zeit- und wertkontinuierliches Signal in ein zeit- und wertdiskretes Signal umgewandelt ([2])

## Abtasten des analogen Signals

Um digitale Werte aus dem zeit- und wertkontinuierlichen, analogen Signal zu ent- nehmen, muss dieses abgetastet werden.



*Abb. 4: Gegenüberstellung von analogen und digitalen Signalen (Quelle:*

*https://upload.wikimedia.org/wikipedia/commons/f/f2/%C3%9Cbersicht\_kontinuierliche\_und\_dis- krete\_Signale.svg)*

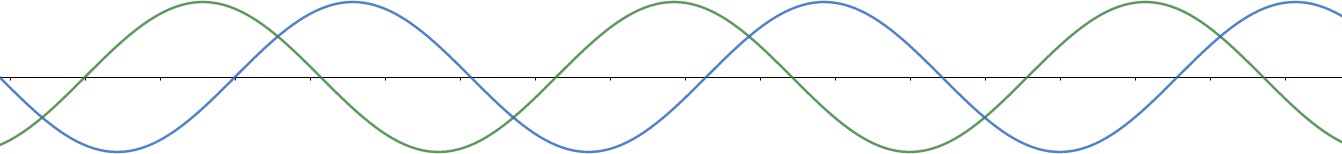
Das Abtasten des Signals übernimmt die Soundkarte des Rechners, ZVEI-Deco- der greift diese Werte mittels der nativen Web-Audio-API([3]) von JavaScript ab.

Diese Werte müssen nun weiter analysiert werden, um die spezifischen Frequen- zen einer Ruftonfolge zu erkennen. Dafür gibt es zwei wesentliche Ansätze.

* + 1. Ansatz 1: Autokorrelation

Autokorrelation ist der Vergleich eines Signals mit sich selbst zu einem früheren Zeitpunkt([4]).

Mit diesem Verfahren ist es möglich die Frequenz eines periodischen Signals (z.B. eines Analogfunksignals) zu einem gewissen Zeitpunkt zu ermitteln. Dies ge- schieht – vereinfacht gesagt – indem man das Signal solange mit sich selbst zu immer später zurückliegenderen Zeitpunkten vergleicht, bis es wieder annähernd gleich ist.



*Abb. 5: vereinfachte graphische Darstellung einer Autokorrelation. Der blaue Graph wird so lange nach hinten verschoben, bis er über dem grünen liegt.*

Hat man den Punkt erreicht, an dem das Signal dem Referenzsignal gleicht, kann man daraus auf die Periodendauer schließen, ergibt sich aus der Differenz des Zeitpunkts des Referenzsignals und des übereinstimmenden Signals. Die Perio- dendauer ist dann, im Falle eines Audiosignals, gleich der Frequenz in Hertz.

Dieser Ansatz eignet sich sehr gut für sehr saubere Signale, weniger jedoch für häufig störbelastete Signale wie den Analogfunk, da überlagernde Störfrequenzen den Algorithmus schnell an seine Grenzen bringen und so Frequenzen verfälscht werden. Für die gewünschte Anwendung ist dieser Ansatz dementsprechend nicht geeignet.

* + 1. Ansatz 2: Frequenzspektrumananalyse

Anders als bei der Autokorrelation wird bei der Frequenzspektrumanalyse nicht nur eine Frequenz ermittelt, sondern das Audiosignal wird in seine Einzelfrequen- zen zerlegt. Störbehaftete Signale (wie etwa durch Rauschen) stellen hier nicht so ein großes Problem wie bei der Autokorrelation dar, da das gesuchte Signal vom Störsignal getrennt werden kann.

Die Berechnung eines Frequenzspektrums ist sehr rechenintensiv. Aufgrund mo- derner Computer und optimierten Algorithmen ist es heutzutage jedoch möglich eine Echtzeitberechnung des Frequenzspektrums durchzuführen.

Der am besten geeignete Algorithmus für eine Echtzeitberechnung des Frequenz- spektrums eines Audiosignal ist die Fast-Fourier-Transformation (FFT)([5]), die Er- klärung dieses Algorithmus würde den Umfang dieser Projektarbeit jedoch über- steigen.

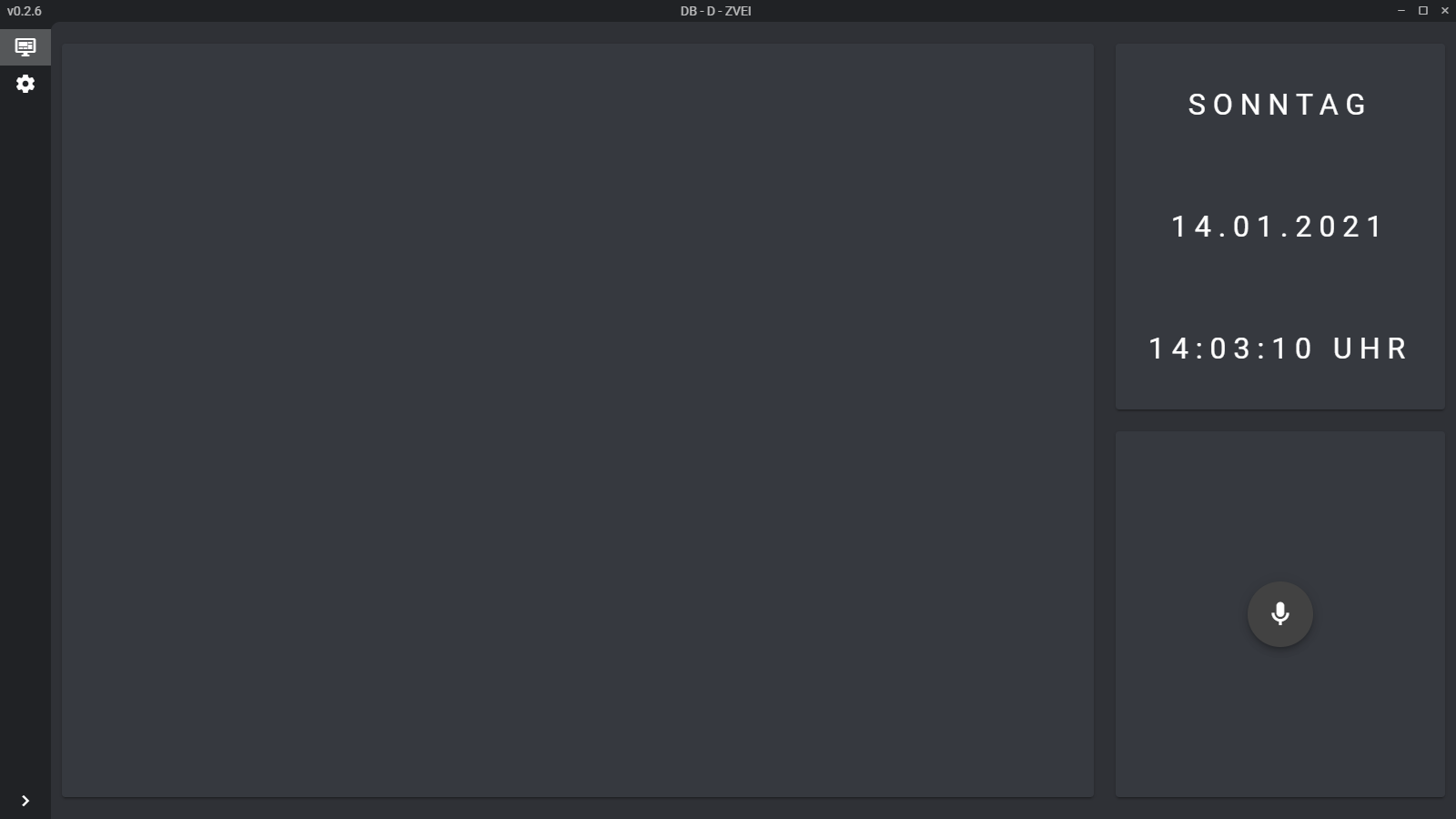
Durch den gewaltigen Vorteil der niedrigeren Störanfälligkeit ist dieser Ansatz der geeignetste für ZVEI-Decoder.

# Umsetzung in Javascript

Da die Programmierumgebung und die Methode zur Auswertung des Audiosignals nun geklärt sind, geht es in diesem Kapitel um die praktische Umsetzung des Pro- jekts.

## Die Benutzeroberfläche

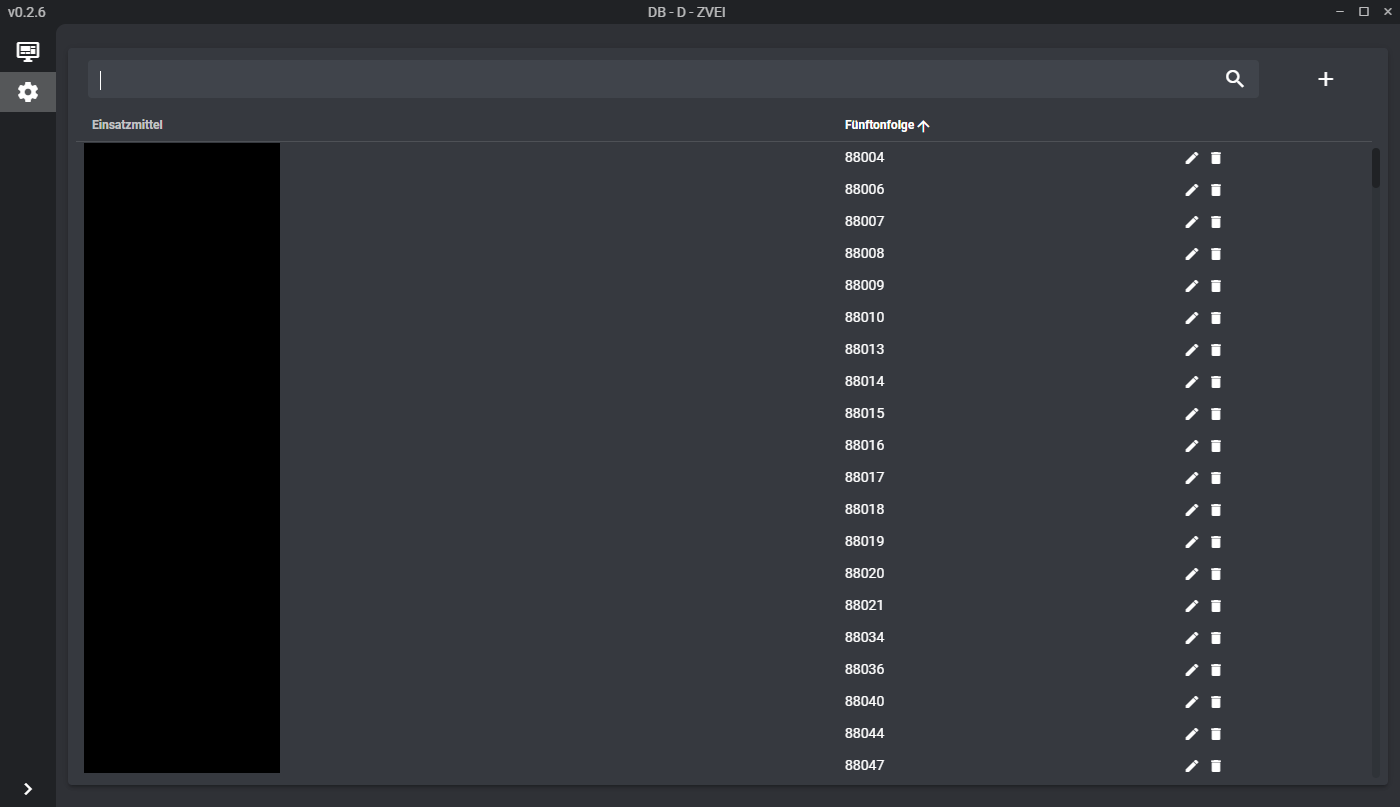
Wie in Kapitel 2.3 bereits erwähnt wird die Benutzeroberfläche mithilfe der Java- script Frameworks VueJS und VuetifyJs realisiert.

Neben einer „Dashboard“-Ansicht, die neben der Liste mit den erkannten Tonfol- gen noch eine digitale Uhr und einen Aktivitätsmonitor für den Audiostream bein- haltet, gibt es noch eine zweite Ansicht zur Bearbeitung der Datenbank.

*Abb. 6: Dashboard-Ansicht von ZVEI-Decoder*

Umrandet werden diese beiden Ansichten von einer Kopf- und einer Seitenleiste, in der Kopfleiste finden sich Informationen zu Name und Version des Programms, sowie die Fenstersteuerung.

In der Seitenleiste lässt sich zwischen den beiden Hauptansichten umschalten.



*Abb. 7: Datenbankansicht von ZVEI-Decoder mit geschwärzten Einsatzmitteln*

## Funktion mounted

Die Funktion mounted ist der Ein- stiegspunkt des Programms. Sie hat folgende Aufgaben:

* Einrichtung benötigter Kompo- nenten wie dem Audiostream des Funks (mehr dazu in Kapi- tel 4.4)
* Einrichtung von Event-Listen- ern für die Datenbankmanipu- lation
* Einrichtung der Programmschleife. Um das Audiosignal kontinuierlich zu analysieren muss die entsprechende Funktion (updateData) regelmäßig (im Falle von ZVEI-Decoder 100 mal pro Sekunde) aufgerufen werden.

## Funktion update-

Data

updateData() ist die Hauptfunkti- on des Programms. In ihr werden alle Daten der Helferfunktionen gesammelt, weitergeleitet und eventuelle Ergebnisse abgespei- chert.

Der grobe Ablauf der Funktion ist dabei wie folgt:

1. Überprüfung der momen- tanen Frequenz des Audi- ostreams und ob diese eine festgelegte Mindest- lautstärke überschreitet, ist dies nicht der Fall endet die Ausführung und wird 10ms später erneut ge- startet.
2. Hat das Programm eine Frequenz erkannt wird die- se einem Ton nach Abb. 2 (Seite 4) zugeordnet, sollte die Frequenz mit keinem Ton übereinstimmen endet die Ausführung hier
3. Der erfasste Ton wird zwi- schengespeichert und ein Timer gestartet. 600 Milli- sekunden werden weiter

*Abb. 8: Funktion updateData in der Datei App.vue*

Töne gesammelt und im Anschluss an eine Helfermethode gesendet. Da die Funktion updateData alle 10ms einen Durchlauf macht, wird jeder Ton einer Tonfolge (Dauer 70ms) mehrfach erkannt werden, deshalb müssen die gesammelten Töne validiert und auf fünf Töne zusammengefasst wer- den. Dieses Konzept dient dazu eventuelle Fehlerkennung aussortieren zu können.

1. Wurde eine Tonfolge erfolgreich validiert, wird diese abgespeichert, damit sie vom User-Interface gerendert werden kann.

## audio.js

audio.js ist die erste von zwei JavaScript-Dateie, die für die Auswertung der Fünf- tonfolgen verantwortlich ist. Sie exportiert verschiedene Funktionen, die von der Hauptfunktion updateData aufgerufen werden

* + 1. Funktion getUserAudio

In Zeile 1-3 findet man die Funktion getUserAudio().

Diese Funktion macht sich das native MediaDevices-Interface von JavaScript zu- nutze, um zugriff auf den Audioeingang des Benutzers zu bekommen.

Im Anschluss gibt sie den erhaltenen Audiostream an die Aufrufende Funktion zu- rück, damit diese Ihn in einer Varia-

ble abspeichern kann.

* + 1. Funktion createCon- text

Die Funktion createContext() nutzt wiederum den Stream, den sie als Parameter übermittelt bekommt, und erschafft um ihn herum einen sogenannten AudioContext. Audio- Context ist eine Schnittstelle der zu- vor in Kapitel 3.2 erwähnten Web- Audio-API. Ein erstellter AudioCon- text bietet Möglichkeiten zur Mani- pulation und Analyse des eingege- benen Audiosignals. Da das Audio- signal nicht manipuliert, sondern analysiert werden soll, wird hier noch eine AnalyserNode erstellt, die

*Abb. 9: audio.js*

die Analysefunktionen der Web-Audio-API bereitstellt

* + 1. Funktion getCurrentFrequencyFft

Die Funktion getCurrentFrequencyFft führt die Fast-Fourier-Transformation durch, hierfür bekommt sie den zuvor erstellten AudioContext und die dazugehörige Ana- lyserNode als Parameter übermittelt.

Nachdem die Fast-Fourier-Transformation das Signal in seine Einzelfrequenzen aufgesplittet hat, wird die Frequenz mit dem höchsten Amplitudenwert (Lautstärke) an die Funktion updateData zurückgegeben.

## zvei.js

audio.js liefert uns die Frequenz des analysierten Audiosegments. Die erste Funktion in zvei.js heißt getTonNummer und arbeitet nun mit dieser Frequenz, um sie einer Tonziffer entsprechend Abb. 2 (Seite 4) zuzuweisen.

* + 1. Funktion getTonNummer

Hierzu werden zuertst alle Ziffern (0-9 und R) in einem Array deklariert. Für eventuelle Funktions- erweiterungen wurde auch die Frequenz für den Sirenenauslöseton (S) mit eingefügt.

Im Anschluss wird die übergebene Frequenz mit einer gewissen Toleranz mit den Frequenzen der einzelnen Ziffern verglichen. Fällt die überlieferte Frequenz auf eine Ziffer, wird diese zurückgege- ben, ansonsten wird der Wert -1 zurückgegeben.

* + 1. Funktion getValidatedTonfolge Die Aufgabe von getValidatedTonfolge ist die Reduzierung und Validierung einer Tonsamm- lung (Siehe Kapitel 4.3).

Eine Tonsammlung (siehe Abb. 11) beinhaltet alle Töne, die updateData innerhalb von 600ms nach Erkennung des ersten Tones erkannt hat. Durch die Ausführung von updateData alle 10 Millisekunden wird jeder Ton öfters erkannt, die- se gilt es nun zu zusammenzufassen.

Im absoluten Idealfall würde jeder Ton 7 mal er- kannt werden (70ms Tondauer / 10ms Ausfüh- rung), dieser Idealfall tritt jedoch seltenst ein.

*Abb. 10: zvei.js*



*Abb. 11: Beispiel einer Tonsammlung, die an getValidatedTonfolge übergeben wird*

Um jeden Ton genau sieben mal zu erkennen müsste die erste Ausführung von updateData genau auf den Anfang des ersten Tons fallen, ist dies nicht der Fall, wird updateData im Verlauf der Tonfolge auch genau zwischen zwei aufeinander- folgenden Tönen ausgeführt, was einer Erkennung im Wege stehen kann. Hier- durch, und durch andere Einflüsse wie z.B. Störungen am Funk, kann es passie- ren, dass Töne öfter oder seltener als sieben mal erkannt werden.

Deshalb muss ein Bereich festgelegt werden, wie oft ein Ton in der Tonsammlung vorhanden sein darf, um von getValidatedTonfolge als Teil einer regelrechten Ton- folge erkannt zu werden. Dies geschiet durch die Parameter minTonCount und maxTonCount. Nach ausführlichen Tests unter unterschiedlichen Bedingungen wurden minTonCount auf 3 und maxTonCount auf 11 festgelegt. Mit diesen Wer- ten wurde die bisher beste Balance aus zuverlässiger Erkennung und Reduzie- rung von Fehlerkennungen beobachtet. In späteren Versionen des Programms wird der Endnutzer diese Werte selbst an seine Gegebenheiten anpassen können.

Ist ein Ton in der Tonsammlung also mindestens so oft wie minTonCount und höchstens so oft wie maxTonCount vorhanden, gilt er als erkannt. Das Ergebnis hat dann im Idealfall genau fünf erkannte Töne, in diesem Fall gilt die gesammte Tonfolge als validiert und wird an updateData zurückgegeben.

*Abb. 12: Die Tonsammlung aus Abb. 11 nach Durchlaufen von getValidatedTonfolge*

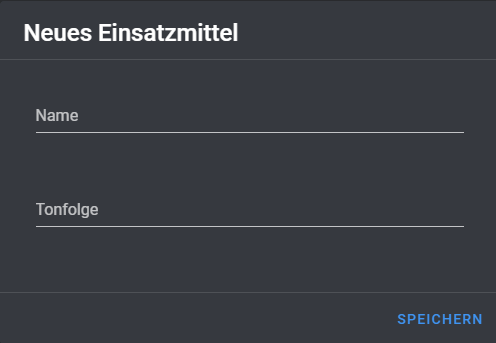
Sind hingegen zu wenige Töne erkannt worden, gilt die Tonfolge als nicht validiert und wird verworfen, bei zu vielen erkannten Tönen wird die Tonfolge weiter redu- ziert, indem die Töne, die am nähesten an der Grenze des Bereiches zwischen minTonCount und maxTonCount gelöscht werden.

## Die Einsatzmitteldatenbank

Damit erkannte Tonfolgen nicht ihrer rohen Form ausgegeben werden, sondern stattdessen die zugehörigen Einsatzmittel, kann der Nutzer den Tonfolgen einen Namen zuordnen (z.B. Kleine Schleife, große Schleife, o.Ä.).

Diese Beziehung wird dann in ei- ner lokalen Datenbank gespei- chert und kann jederzeit geän- dert oder gelöscht werden.

*Abb. 13: Datenbanklogik im Hintergrundprozess*



*Abb. 14: Dialog zur Erstellung eines neuen Einsatzmittels*

# Anhang

## Bedienungsanleitung

Die folgende Anleitung bezieht sich auf ZVEI-Decoder Version 0.2.6. Aufgrund fortlaufender Entwicklung kann sich das Programm mit zukünftigen Updates ver- ändern.

* + 1. Links

**Quellcode:** *https://github.com/david-breidert/zvei-decoder*

**Download Version 0.2.6:** [*https://github.com/david-breidert/zvei-*](https://github.com/david-breidert/zvei-decoder/releases/tag/v0.2.6)[*decoder/releases/tag/v0.2.6*](https://github.com/david-breidert/zvei-decoder/releases/tag/v0.2.6)

**Download aktuellste Version:** [*https://github.com/david-breidert/zvei-decoder/re-*](https://github.com/david-breidert/zvei-decoder/releases)[*leases*](https://github.com/david-breidert/zvei-decoder/releases)

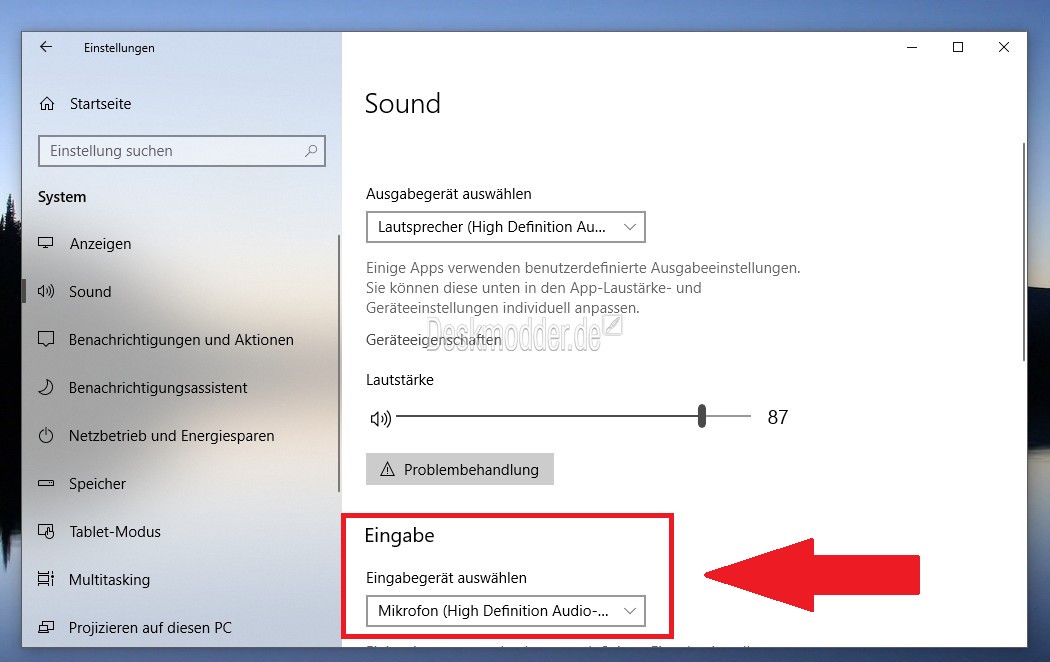
* + 1. Benötigte Hardware
       - Ein PC mit Microsoft Windows
       - ein Gerät zum Funkempfang mit Audioausgang (z.B. Ein Funkscanner) an- geschlossen an den PC ODER ein Mikrofon sowie ein Fünftongeber oder eine aufgenommene Fünftonfolge
    2. Installation

Nach Download und Ausführung des Installers (zvei-decoder-Setup-0.2.6.exe) wird das Programm installiert und nach der Installation automatisch geöffnet.

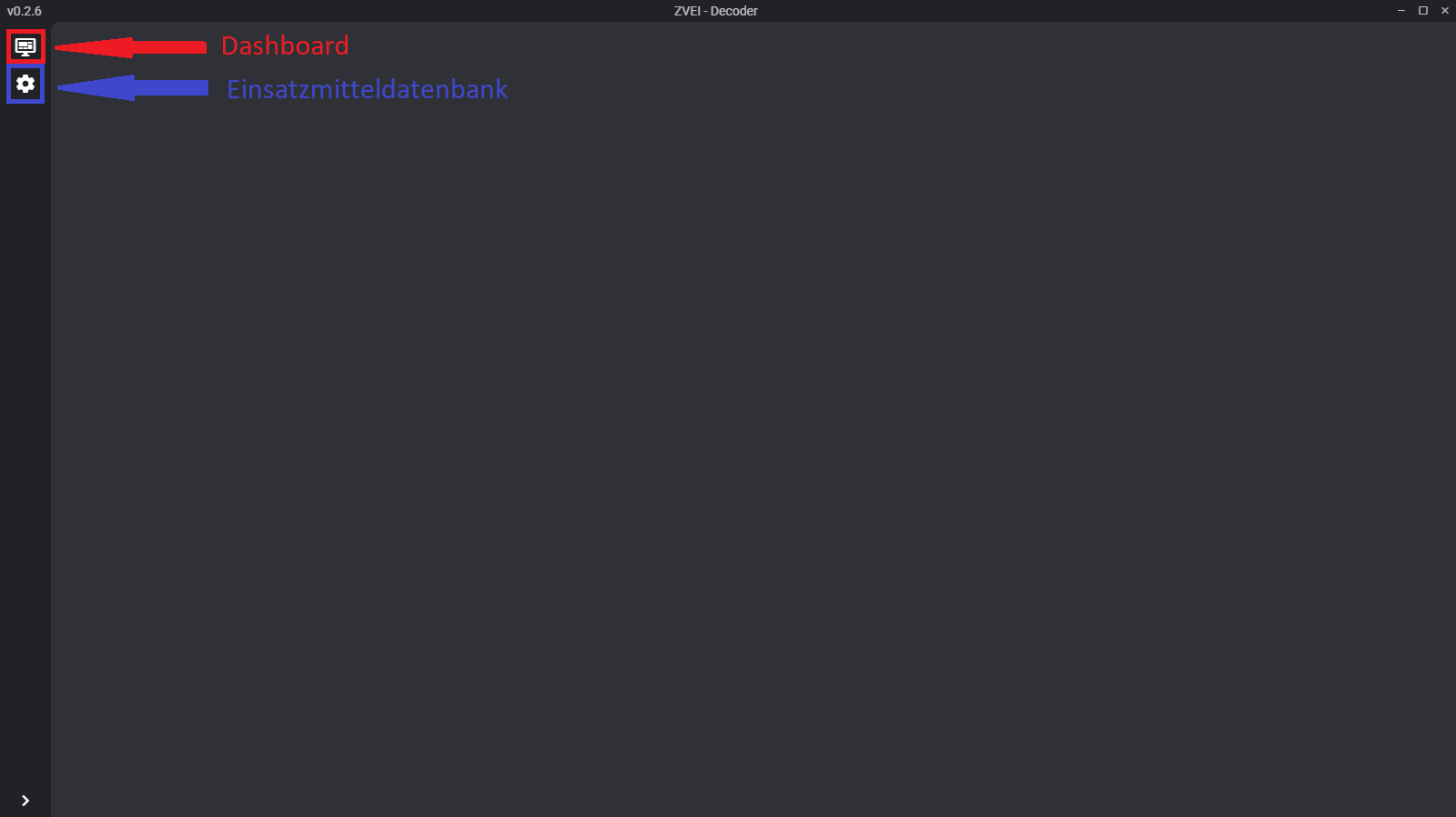
Sollte sich das Programm nicht automatisch öffnen, kann es über die erstellte Ver- knüpfung auf dem Desktop geöffnet werden.

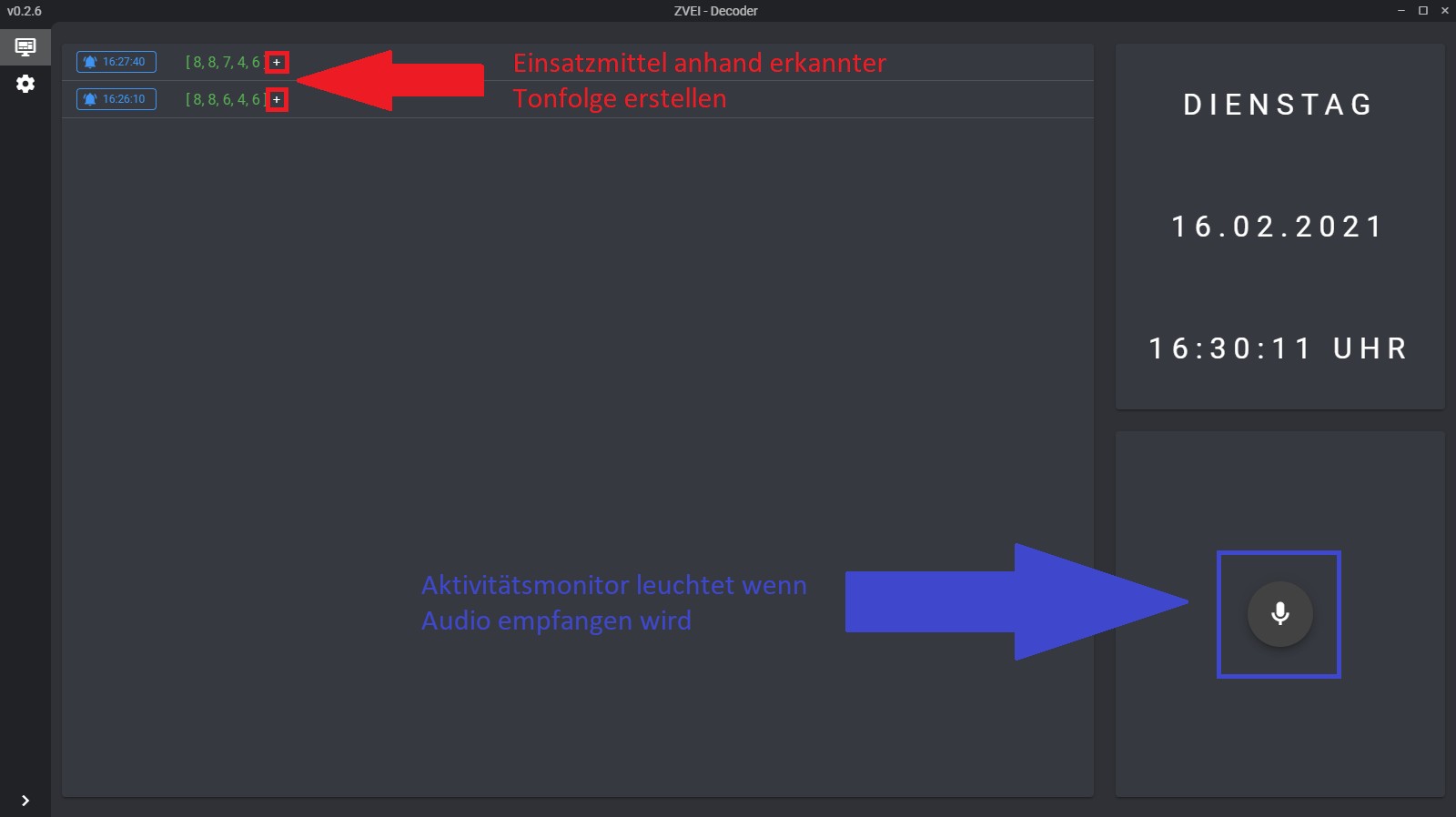
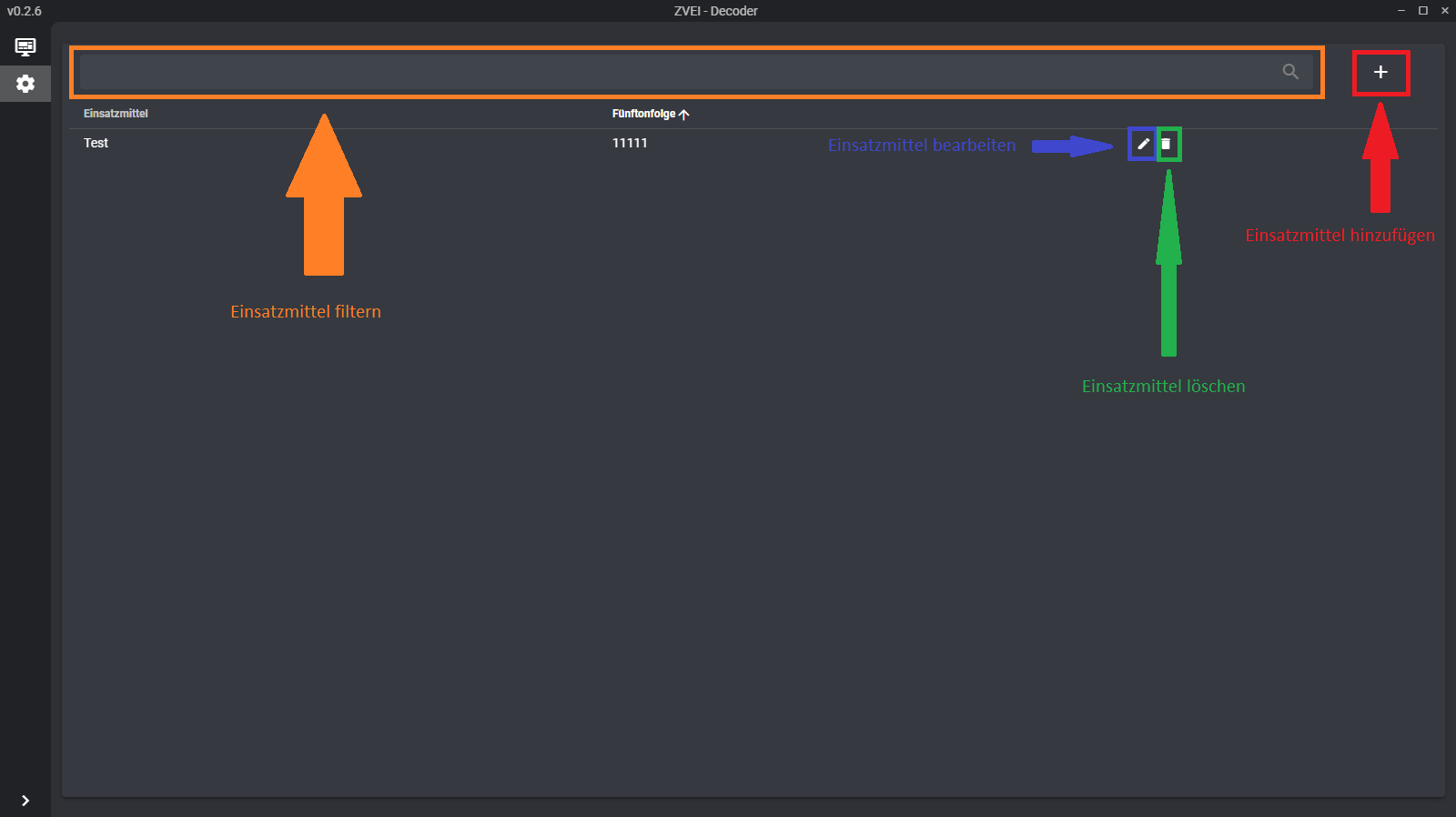
* + 1. Nötige Windows-Einstellungen

ZVEI-Decoder greift auf das Standard-Audioeingabegerät des Computers zu. Des- halb ist es wichtig, dass das Audiogerät, über das die Fünftonfolgen eingespielt werden in Windows als Standardgerät eingestellt ist.



* + 1. Nach dem Programmstart



* + 1. Dashboard
    2. Einsatzmitteldatenbank

[6][7][8][9][10]

# Literaturverzeichnis

1: Thüringer Ministerium für Inneres und Kommunales, Brand- und Katastrophen- schutzbericht 2018, 2018, https://innen.thueringen.de/fileadmin/Publikationen/An- lage\_1\_Jahresbericht\_2018.pdf

2: Hochschule Karlsruhe, Klassen von Signalen: Kontinuierliche und diskrete Si- gnale, , https://[www.eit.hs-karlsruhe.de/mesysto/teil-a-zeitkontinuierliche-signa-](http://www.eit.hs-karlsruhe.de/mesysto/teil-a-zeitkontinuierliche-signa-) le-und-systeme/zeitkontinuierliche-signale/klassen-von-signalen/kontinuierli- che-und-diskrete-signale.html

3: W3C Audio Working Group, Web Audio APIW3C Candidate Recommendation Snapshot, 2021, https://[www.w3.org/TR/webaudio/](http://www.w3.org/TR/webaudio/)

4: Wikipedia, Autokorrelation, 2020, https://de.wikipedia.org/wiki/Autokorrelation

5: Wikipedia, Schnelle Fourier-Transformation, 2020, https://de.wikipedia.org/wiki/Schnelle\_Fourier-Transformation

6: , Technische Richtlinie der Behörden und Organisationen mit Sicherheitsaufga- ben (BOS): Geräte für die Funkalarmierung , 2000

7: Elias Oenal, multimon-ng, 2020, https://github.com/EliasOenal/multimon-ng 8: VueJS Team, VueJS API, 2021, https://vuejs.org/v2/api/

9: ElectronJS Team, , 2021, https://[www.electronjs.org/docs/api](http://www.electronjs.org/docs/api)

10: VuetifyJS Team, VuetifyJS Guide, 2021, https://vuetifyjs.com/en/introduction/why-vuetify/

# Abbildungsverzeichnis

Abb. 1: typischer Funkmeldeempfänger

(Quelle: https://[www.gfd-katalog.com/ludwig\_feuerschutz/quattro-xlixlsi/quattro-](http://www.gfd-katalog.com/ludwig_feuerschutz/quattro-xlixlsi/quattro-) xli/25597) 2

Abb. 2: Tonfrequenzen, aus denen die Fünftonfolgen aufgebaut werden

(Quelle: https://de.wikipedia.org/wiki/5-Ton-Folge) 4

Abb. 3: zeitlicher Ablauf einer Funkalarmierung

Quelle: https://de.wikipedia.org/wiki/5-Ton-Folge 4

Abb. 4: Gegenüberstellung von analogen und digitalen Signalen (Quelle:

https://upload.wikimedia.org/wikipedia/commons/f/f2/%C3%9Cbersicht\_kontinuier- liche\_und\_diskrete\_Signale.svg) 9

Abb. 5: vereinfachte graphische Darstellung einer Autokorrelation. Der blaue Graph wird so lange nach hinten verschoben, bis er über dem grünen liegt 10

Abb. 6: Dashboard-Ansicht von ZVEI-Decoder 11

Abb. 7: Datenbankansicht von ZVEI-Decoder mit geschwärzten Einsatzmitteln 12

Abb. 8: Funktion updateData in der Datei App.vue 13

Abb. 9: audio.js 15

Abb. 10: zvei.js 17

Abb. 11: Beispiel einer Tonsammlung, die an getValidatedTonfolge übergeben wird 18

Abb. 12: Die Tonsammlung aus Abb. 11 nach Durchlaufen von getValidatedTon- folge 18

Abb. 13: Datenbanklogik im Hintergrundprozess 19

Abb. 14: Dialog zur Erstellung eines neuen Einsatzmittels 19