Hochschule der Medien Stuttgart

Fakultät Electronic Media

Audiovisuelle Medien (AM7)

Bachelorarbeit zum Thema:

Konzeption und Entwicklung einer SQL-basierten Sound-Library mit integrierter GUI

Zur Erlangung des akademischen Grades Bachelor of Engineering

Vorgelegt von:

Sven Michael Anderle

E-Mail: sa083@hdm-stuttgart.de

Matrikelnummer: 39968

Studiengang: Audiovisuelle Medien (AM7)

Vorgelegt am: 26.02.2026

Erstprüfer: Prof. Oliver Curdt

Zweitprüfer: Prof. Dipl.-Ing. Uwe Schulz

Ehrenwörtliche Erklärung

Hiermit versichere ich, Sven Michael Anderle, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel „Konzeption und Entwicklung einer SQL-basierten Sound-Library mit integrierter GUI“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quellen kenntlich gemacht. Ebenso sind alle Stellen, die mit Hilfe eines KI-basierten Schreibwerkzeugs erstellt oder überarbeitet wurden, kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO, § 23 Abs. 2 Master-SPO (Vollzeit)) einer unrichtigen oder unvollständigen ehrenwörtlichen Erklärung zur Kenntnis genommen.

Filderstadt, 26.02.2026

Hinweise zur Arbeit

In der vorliegenden Arbeit wird darauf verzichtet, bei Personenbezeichnungen sowohl die weibliche als auch die männliche und diverse Form zu nennen. Alle Bezeichnungen gelten gleichermaßen für alle Geschlechter und dienen ausschließlich der besseren Lesbarkeit.

Für die technische Umsetzung der Implementierung wurde das KI-basierte Werkzeug ChatGPT (OpenAI) als Coding-Assistenz eingesetzt. Dies umfasst die Generierung von Code-Fragmenten sowie das Debugging auf Basis eigens formulierter Anforderungen. Sämtliche konzeptionelle Entscheidungen - darunter die Wahl des Datenbankmodells, die Architektur des Gesamtsystems, das Import- und Scan-Konzept sowie die Gestaltung der GUI - wurden vollständig eigenständig erarbeitet und der KI lediglich als Umsetzungsvorgabe übergeben. Die inhaltliche Auseinandersetzung mit den technischen Grundlagen, die wissenschaftliche Argumentation, die Konzeption, die Auswertung sowie die schriftliche Ausarbeitung sind ohne den Einsatz KI-basierter Schreibwerkzeuge entstanden und stellen ausschließlich eigene Leistung dar.
Die englischsprachige Kurzfassung wurde auf Basis der deutschen Kurzfassung mithilfe des maschinellen Übersetzungswerkzeugs DeepL erstellt und anschließend inhaltlich geprüft.

Der entwickelte Prototyp sowie der vollständige Projektordner mit allen Quelldateien sind dieser Arbeit als CD beigelegt und stehen zusätzlich unter folgenden Links zum Download bereit:

https://cloud.mi.hdm-stuttgart.de/index.php/s/eETJon6Qc88kQGe

https://drive.google.com/drive/folders/1lQU8o1_soy2Z2tMkmkOfYupeFiwJkBhm?usp=drive_link

Danksagung

Mein besonderer Dank gilt Prof. Oliver Curdt, der mich nicht nur als Betreuer dieser Arbeit begleitet hat, sondern mir auch über mein gesamtes Studium hinweg wertvolles Wissen in Tontechnik und Sounddesign vermittelt hat.

Ebenso möchte ich Prof. Uwe Schulz danken, der mich in den ersten Semestern meines Studiums in der Informatik unterrichtet hat und dabei Interesse und Wissen vermittelte, das mir auch in dieser Arbeit zugutekam - und trotz seines verdienten Ruhestands die Zweitprüfung übernommen hat.

Abschließend danke ich meinen Freunden und meiner Familie für ihre Unterstützung während meines Studiums und der arbeitsreichen Wochen dieser Arbeit.

Kurzfassung

Die vorliegende Arbeit beschreibt die Konzeption und Implementierung einer lokalen, SQL-basierten Sound-Library mit grafischer Benutzeroberfläche. Ausgangspunkt ist die in der Ton-Postproduktion häufig anzutreffende Problematik unstrukturierter Audiodatei-Verwaltung , die das Auffinden und Wiederverwenden von Audiomaterial erheblich erschwert. Das entwickelte System kombiniert eine SQLite-Datenbank zur zentralen Metadatenverwaltung mit einem automatisierten Scan-Prozess und einer Tkinter-basierten GUI. Audiodateien verbleiben dabei im lokalen Dateisystem, während Metadaten wie Tags, Kategorien, Beschreibungen und Bewertungen datenbankgestützt verwaltet und durchsuchbar gemacht werden. Ein SHA-256-basierter Hashwert-Abgleich ermöglicht zusätzlich die zuverlässige Erkennung verschobener und umbenannter Dateien. Der entwickelte Prototyp wurde anhand eines selbst aufgenommenen Datensatzes von 77 WAV-Audiodateien aus sechs Kategorien evaluiert und in einem informellen Funktionstest mit zehn externen Testern auf macOS- und Windows-Betriebssystemen getestet.

Abstract

This thesis describes the design and implementation of a local, SQL-based sound library with a graphical user interface. The starting point is the problem of unstructured audio file management, which is frequently encountered in audio post-production and makes it considerably more difficult to find and reuse audio material. The developed system combines an SQLite database for central metadata management with an automated scanning process and a Tkinter-based GUI. Audio files remain in the local file system, while metadata such as tags, categories, descriptions, and ratings are managed in a database and made searchable. An SHA-256-based hash value comparison also enables reliable detection of moved and renamed files. The developed prototype was evaluated using a self-recorded dataset of 77 WAV audio files from six categories and tested in an informal functional test with ten external testers on macOS and Windows operating systems.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1 Motivation

Bei der Arbeit mit selbst aufgenommenem Audiomaterial kommt es häufig zu einem Problem: Die Dateien liegen irgendwo auf der Festplatte, verteilt über mehrere Ordner, benannt nach Aufnahmedatum oder einem schnell vergebenen Arbeitstitel, der langfristig kaum noch nachvollziehbar ist. Das Vorhören einer konkreten Aufnahme erfordert manuelles Durchsuchen von Verzeichnissen, und die Frage, ob ein bestimmter Sound schon aufgenommen wurde oder wo er liegt, lässt sich oft nur durch zeitaufwendiges Suchen beantworten.

Diese Workflow-Problematik war der konkrete Ausgangspunkt dieser Arbeit. In der Ton-Postproduktion, unabhängig ob für Film, Videospiele oder andere audiovisuelle Medien, ist ein strukturiertes und schnell zugängliches Audiomaterial ein wesentlicher Faktor für effizientes Arbeiten. Professionelle Sound-Libraries lösen dieses Problem, sind aber entweder auf kuratierten Fremdcontent ausgerichtet oder erfordern die Abhängigkeit von kommerziellen Tools mit festem Funktionsumfang, Abonnementmodellen und externen Update-Zyklen. Eine einfache, lokal kontrollierbare Lösung, die den eigenen Workflow abbildet ohne sich an fremde Systeme anpassen zu müssen, fehlte.

Daraus entstand die Idee, eine eigene Sound-Library zu entwickeln. Sie sollte ein schlankes Werkzeug darstellen, das selbst aufgenommene Audiodateien automatisch erfasst, strukturiert und durchsuchbar macht, dabei ohne Cloud-Zwang, ohne Accounts und ohne Abhängigkeit von kommerzieller Software auskommen. Die Verbindung aus eigenem Bedarf, Interesse an der Kombination von Audio- und Softwareentwicklung und dem Ziel, einen praxisnahen Prototyp zu bauen, bildete die Grundlage für diese Arbeit.

1.2 Zielsetzung und Abgrenzung

Ziel der Arbeit ist die Konzeption und Implementierung eines funktionalen Prototyps einer lokalen, SQL-basierten Sound-Library mit grafischer Benutzeroberfläche. Der Prototyp soll zeigen, dass sich die Kernfunktionen eines Workflows mit einer Sound-Library, also automatisches Erfassen von Audiodateien, strukturierte Metadatenverwaltung, Suche, Filterung und Vorhören, mit überschaubaren Mitteln und ohne externe Abhängigkeiten realisieren lassen.

Ein zentraler Aspekt der Zielsetzung ist dabei die Eigenständigkeit des Workflows. Das Dateisystem bleibt die primäre Ablage der Audiodateien, die Sound-Library ergänzt diese um eine Verwaltungs-

und Suchschicht. Neu abgelegte Dateien sollen automatisch erfasst werden, manuell gepflegte Metadaten sollen bei wiederholten Scans erhalten bleiben. Das Ergebnis ist eine Anwendung, die den eigenen Workflow unterstützt, ohne ihn zu erzwingen.

Gleichzeitig sind die Grenzen dieser Arbeit klar definiert. Das Ergebnis ist ein Prototyp, kein marktreifes Produkt. Der Fokus liegt auf dem Funktionsnachweis des Konzepts, nicht auf einem vollständigen Funktionsumfang, was konkret bedeutet, dass auf Cloud-Integration, DAW-Anbindung und Mehrbenutzer-Betrieb verzichtet wird. Die Anwendung ist für einen lokalen Single-User-Betrieb konzipiert. Auch der im Rahmen der Arbeit aufgebaute Audio-Datensatz ist kein kommerziell nutzbarer Inhalt, sondern ein praxisnahes Evaluierungsmaterial zur Erprobung der Software-Funktionen. Eine wissenschaftlich strukturierte Usability-Studie ist ebenfalls nicht Bestandteil der Arbeit, die Evaluation beschränkt sich auf einen informellen Funktionstest unter realen Nutzungsbedingungen.

1.3 Vorgehensweise und Aufbau

Die Arbeit folgt einer Struktur, die von den theoretischen Grundlagen über die Konzeption bis zur Evaluation und Diskussion führt. Jeder Abschnitt baut dabei auf dem vorherigen auf und bildet die Grundlage für den nächsten.

Kapitel 2 legt die theoretischen Grundlagen dar, die für die Arbeit relevant sind. Dazu gehören Field-Recording und Audiobearbeitung als praktische Basis für die Erstellung des Evaluierungsdatensatzes. Audio-Metadaten als konzeptionelle Grundlage für die Datenbankstruktur sowie relationale Datenbanken und GUI-Entwicklung in Python als technische Fundamente der Implementierung. Die Theorie wurde dabei gezielt auf die für diese Arbeit relevanten Aspekte beschränkt.

Kapitel 3 überführt die theoretischen Grundlagen in konkrete Anforderungen und Designentscheidungen. Es beschreibt die Zielsetzung und Abgrenzung der Sound-Library, die Anforderungen, das Datenmodell, das Import- und Scan-Konzept sowie das GUI-Konzept. Dieses Kapitel bildet die Planungsgrundlage für die Implementierung in Kapitel 4.

Kapitel 4 beschreibt die praktische Umsetzung. Es umfasst die Aufnahme und Bearbeitung des Audio-Datensatzes, die Implementierung der SQLite-Datenbank inklusive Schema und Zugriffsschicht, die Entwicklung der Tkinter-GUI sowie die Bereitstellung des Prototyps als plattformübergreifende Anwendung. Abweichungen vom Konzept, etwa der Wechsel von PostgreSQL zu SQLite oder die

Entscheidung nur WAV-Formate zu berücksichtigen, werden in den jeweiligen Abschnitten begründet.

Kapitel 5 evaluiert den Prototyp aus zwei Perspektiven, einem qualitativen Vergleich mit bestehenden kommerziellen Sound-Libraries sowie einem informellen Funktionstest durch externe Tester auf macOS- und Windows-Systemen. Die Ergebnisse werden abschließend den in Kapitel 3 formulierten Anforderungen gegenübergestellt.

In Kapitel 6 wird eine kritische Reflexion durchgeführt. Es wird auf konzeptionelle Stärken und Grenzen des Prototyps eingegangen sowie auf die Grenzen der Evaluation und das Optimierungspotenzial im Entstehungsprozess, insbesondere im Hinblick auf die zeitliche Schwerpunktsetzung zwischen Audio- und Softwareteil.

Kapitel 7 schließt die Arbeit mit einem Fazit ab, das die wesentlichen Ergebnisse zusammenfasst und einem Ausblick, der konkrete nächste Schritte für eine Weiterentwicklung des Prototyps skizziert.

2. Theoretische Grundlagen

Im Folgenden werden die theoretischen Grundlagen dargelegt, die für die Konzeption und Implementierung der Sound-Library relevant sind, wobei sich auf die für diese Arbeit relevanten Aspekte beschränkt wird.
Dabei werden Field Recording und Audiobearbeitung als praktische Grundlage für die Aufnahme- und Bearbeitungsprozesse behandelt, zudem die Loudness-Normalisierung als abschließender Schritt der Audioproduktion.
Des Weiteren wird auf die Audio-Metadaten als konzeptionelle Grundlage für die Datenbankstruktur und auf relationale Datenbanken als technisches Fundament der Sound-Library eingegangen.
Abschließend wird die GUI-Entwicklung in Python als Grundlage für die Benutzeroberfläche thematisiert.

2.1 Field Recording

Field Recording bezeichnet die Aufnahme von Tönen und Geräuschen außerhalb kontrollierter Studioumgebungen. Die Aufnahmen werden dabei direkt in realen Umgebungen wie Straßen, Natur oder Innenräumen durchgeführt, wodurch auf akustisch optimierte Bedingungen verzichtet wird. Dies bringt eigene Herausforderungen mit sich, bietet aber dafür authentische und natürliche Klangbilder (Ableton, 2016).

Das Field Recording ist eine etablierte Methode in der Ton-Postproduktion für Film und einer Vielzahl anderer audiovisueller Medien (Marks, 2010). Somit bilden Field Recordings eine zentrale Grundlage für Sound-Libraries, die als organisierte Sammlungen von Audiomaterial im Sounddesign eingesetzt werden. Die technischen Eigenschaften der Aufnahmen, wie Abtastrate, Bittiefe und Dateiformat bestimmen dabei maßgeblich die Qualität und spätere Nutzbarkeit des Materials. Die folgenden Unterkapitel erläutern die relevanten technischen Grundlagen der digitalen Audioaufnahme.

2.2 Digitale Audiowandlung und Abtasttheorem

Schall ist physikalisch betrachtet eine kontinuierliche Druckwelle, die durch ein Medium wie Luft übertragen wird (OpenStax, 2016, S.790, 793). Um dieses analoge Signal digital verarbeiten und speichern zu können, muss es zunächst in eine diskrete, digitale Darstellung umgewandelt werden (Kosonocky & Xiao, 1999, S.1). Dieser Prozess wird als Analog-Digital-Wandlung bezeichnet und bildet die technische Grundlage jeder digitalen Audioaufnahme.

Bei Tonaufnahmen ist die Abtastrate ein maßgeblicher Faktor. Diese gibt in Hertz an, wie oft der analoge Schallpegel pro Sekunde digital erfasst wird. Das Nyquist-Shannon-Abtasttheorem dient dabei als theoretische Grundlage. Es besagt, dass für die verlustfreie digitale Rekonstruktion eines Signals die Abtastrate mindestens doppelt so hoch sein muss wie die höchste im Signal enthaltene Frequenz (Shannon, 1949, S.448). Dies ergibt sich aus der Tatsache, dass ein sinusförmiges Signal mindestens zwei Abtastpunkte pro Schwingung benötigt, um eindeutig rekonstruierbar zu sein, einen für den positiven und einen für den negativen Anteil. Wird das Signal nicht mit ausreichender Frequenz abgetastet, entsteht das sogenannte Aliasing, bei dem hohe Frequenzen fälschlicherweise als niedrige Frequenzen interpretiert werden (Nordström, 2017, S.7).

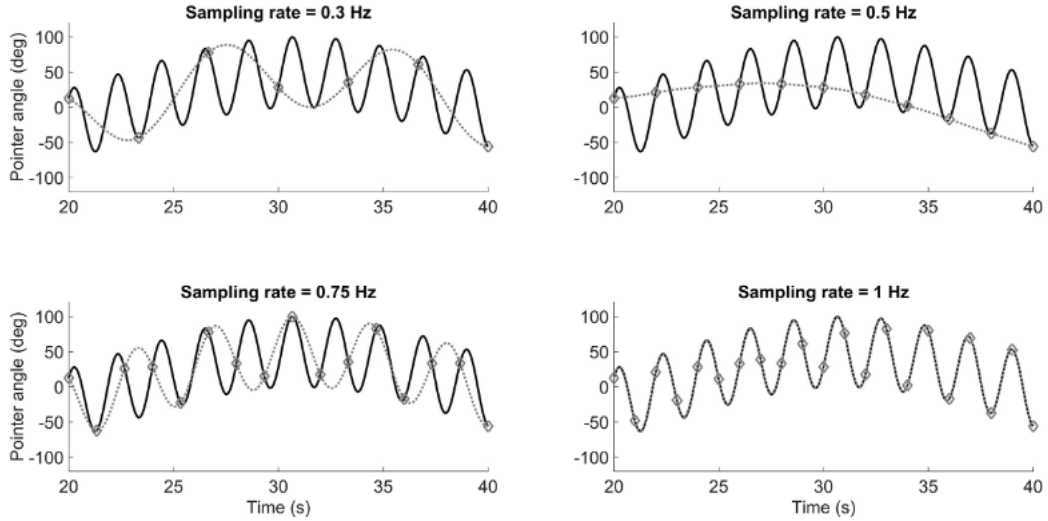

Abbildung 1: Illustration des Nyquist-Shannon-Abtasttheorems bei unterschiedlichen Abtastraten (Eisma et al., 2023, Abschnitt „Nyquist-Shannon Sampling Theorem“)

Der menschliche Hörbereich wird üblicherweise zwischen 20 Hz und 20 kHz angegeben (Moller & Pedersen, 2004, S.37), woraus sich nach dem Theorem eine Mindest-Abtastrate von 40 kHz ergibt. 48 kHz hat sich dabei als Standard für Film, Videospiele und viele andere audiovisuelle Medien etabliert und bietet einen Sicherheitspuffer oberhalb der theoretischen Mindestrate.

Eine Abtastrate von 96 kHz erfasst entsprechend Frequenzanteile bis 48 kHz und bietet dabei deutlich mehr Raum für extreme Pitch-Shifting oder Time-Stretching Bearbeitungen ohne hörbare Verluste. Bei Pitch-Shifting wird die Tonhöhe eines Signals verändert, was einer Skalierung des Frequenzspektrums entspricht (Laroche & Dolson, 2000, S.1). Wird ein Signal um eine Oktave nach oben verschoben, verdoppeln sich alle enthaltenen Frequenzen. Frequenzanteile, die beispielsweise bei 20 kHz lagen, würden dadurch auf 40 kHz verschoben und damit außerhalb des bei 48 kHz möglichen Bereichs abgeschnitten werden. Für typische Anwendungsfälle ohne extreme Bearbeitungen bedeuten höhere Abtastraten allerdings einen proportional größeren Speicherbedarf (Griscom, 2006, S.15) ohne wahrnehmbaren Mehrwert für diesen Anwendungsfall, da die zusätzlich erfassten Frequenzanteile oberhalb des Hörbereichs liegen.

Ein weiterer relevanter Parameter ist die Bittiefe. Diese bestimmt die Anzahl der möglichen Amplitudenstufen bei der Digitalisierung und damit den Dynamikumfang der Aufnahme. Bei n Bit stehen 2^n mögliche Amplitudenstufen zur Verfügung, bei 16 Bit also 65.536, bei 24 Bit bereits über 16 Millionen Stufen (Neuhaus et al., 2021, S.3). Der Abstand zwischen zwei benachbarten Amplitudenstufen wird als Quantisierungsrauschen bezeichnet. Je mehr Stufen vorhanden sind, desto kleiner ist dieser Abstand und desto geringer das Rauschen. Der theoretische Dynamikumfang berechnet sich nach der Formel $SNR = 6{,}02 \cdot n + 1{,}76\ dB$, wobei n die Bittiefe darstellt (Kester, 2009, S.3). Bei 24 Bit ergibt sich damit ein theoretischer Dynamikumfang von etwa 146 dB. 16 Bit wie bei Audio-CDs bieten einen Dynamikumfang von etwa 98 dB, was für Aufnahmen mit wechselnden Lautstärken deutlich weniger *Headroom* lässt. 24 Bit ermöglicht damit eine sichere Pegelwahl auch bei unvorhersehbaren Lautstärkespitzen ohne Clipping zu riskieren.
32 Bit Float speichert Amplitudenwerte als Gleitkommazahl statt als Ganzzahl, wodurch auch Werte außerhalb des normalen Pegelbereichs gespeichert werden können, ohne dass Clipping entsteht, solange das Format beibehalten wird. Das Format ist jedoch hardwareabhängig (Grundström, 2013, S.4-5, 8). Voraussetzung für die Nutzung dieser Eigenschaft ist, dass keine analoge Übersteuerung stattgefunden hat. Nachträgliche Pegelanpassungen sind damit ohne Qualitätsverlust möglich, was vor allem bei komplexen Produktionsworkflows von Vorteil ist.

Beim Aufnahmeformat ergeben sich verschiedene Spezifikationen. PCM-WAV (Pulse Code Modulation) ist aufgrund seiner breiten Kompatibilität mit DAWs, Game-Engines und Audio-Tools der de-facto Standard in der professionellen Ton-Postproduktion. PCM-WAV speichert jeden Abtastwert durch Abtastung, Quantisierung und Kodierung als unkomprimierten digitalen Zahlenwert (Tomić & Drljača, 2023, S42-43). Dabei wird das Signal unkomprimiert gespeichert (Edge, 2016, S.156), wodurch jeder Abtastpunkt mit seiner vollen Bittiefe erhalten bleibt.

Verlustbehaftete komprimierte Formate wie MP3 oder AAC nutzen psychoakustische Modelle, um die Dateigröße zu reduzieren (Brandenburg & Popp, 2000, S.2, 7). Dabei werden Frequenzanteile basierend auf psychoakustischen Maskierungseffekten codiert, bei denen leisere Töne durch lautere Töne maskiert werden können (vgl. Brandenburg & Popp, 2000, S.7). Bei weiterer Bearbeitung verstärken sich diese Verluste und können als hörbare Artefakte auftreten (Maher, 2003, S.256). Verlustfreie komprimierte Formate wie FLAC reduzieren die Dateigröße ohne Qualitätsverlust durch mathematische Kompressionsverfahren (van Beurden & Weaver, 2024, S.1, 5), haben jedoch eine geringere Kompatibilität in professionellen Workflows und bieten für den vorliegenden Anwendungsfall keinen relevanten Vorteil gegenüber WAV.

2.3 Grundlagen der Audiobearbeitung

Nach der Aufnahme stellt die Audiobearbeitung den nächsten Schritt zum finalen Audiomaterial dar. Gerade bei Sound-Libraries ist das Ziel dabei die technische Bereinigung ohne Veränderung des natürlichen Klangcharakters, um Anwendern später selbst klangfärbende Maßnahmen offen zu halten.

Filter gelten dabei als grundlegende Werkzeuge zur frequenzabhängigen Bearbeitung von Audiosignalen. Hierbei werden bestimmte Frequenzbereiche durchgelassen, andere gedämpft oder blockiert. Analog aufgebaute Filter basieren dabei auf RC-Gliedern, also Schaltungen aus Widerstand und Kondensator, deren Grenzfrequenz durch $f_g = \frac{1}{2\pi \cdot R \cdot C}$ bestimmt wird. Digitale Filter simulieren dieses Verhalten mathematisch. Hochpassfilter lassen Frequenzen oberhalb einer definierten Grenzfrequenz passieren und dämpfen tiefere Frequenzanteile, Tiefpassfilter funktionieren analog in die entgegengesetzte Richtung. Neben der Grenzfrequenz ist die Flankensteilheit ein weiterer relevanter Parameter, der beschreibt, wie stark Frequenzen jenseits der Grenzfrequenz gedämpft werden, angegeben in dB pro Oktave, bzw. Dekade (Matzik & Anwar, S.543-550).
Filter höherer Ordnung werden dabei durch mehrere hintereinandergeschaltete Filterstufen realisiert, was eine steilere Flanke ermöglicht, jedoch gleichzeitig zu einer stärkeren

frequenzabhängigen Phasenverschiebung führt, da jede Stufe eine eigene Phasenverschiebung einbringt (Walter, 2014, S.3).

Neben Filtern spielen bei Field Recordings auch Noise Gates eine wichtige Rolle. Dabei handelt es sich um dynamische Prozessoren, die ein Audiosignal unterhalb eines definierten Schwellenwertes dämpfen oder stummschalten, um unerwünschte leise Signalanteile wie Hintergrundrauschen zwischen Klangereignissen auszublenden. Der Threshold dient dabei als zentraler Parameter und definiert den Pegel, ab dem das Gate öffnet. Die Attack-Zeit bestimmt, wie schnell das Gate öffnet, wenn der Threshold überschritten wird. Ist sie zu lang gewählt, öffnet sich das Gate zu langsam, wodurch Transienten abgeschnitten werden und hörbare Verzerrungen entstehen können. Die Hold-Zeit definiert, wie lange das Gate nach Unterschreiten des Thresholds noch geöffnet bleibt. Ist diese zu kurz, kann das Gate zu schnell umschalten und Modulations-Artefakte erzeugen. Die Release-Zeit bestimmt, wie schnell das Gate anschließend schließt, wobei zu kurze Release-Zeiten das Ausklingen eines Signals unnatürlich abschneiden können. Die Kombination dieser Parameter bestimmt maßgeblich das natürliche Klingen des Gates (Terell et al., 2010, S.1-4).

Auch eine spektrale Rauschreduktion kann sich bei der Bearbeitung von Field-Recordings als hilfreich erweisen. Dabei wird das Rauschspektrum während signalfreier Perioden geschätzt und anschließend von den aufgenommenen Signalen subtrahiert. Die Methode unterdrückt gezielt Frequenzen mit niedrigem Signal-Rausch-Verhältnis. Eine zu starke Anwendung kann allerdings zu hörbaren Artefakten führen (Gustafsson et al., 1999).

Neben der technischen Bereinigung ist die Anpassung der wahrgenommenen Lautheit ein wichtiger Schritt zur konsistenten Nutzbarkeit von Audiomaterial in einer Sound-Library.

2.4 Loudness-Normalisierung

Bei der Normalisierung ist die Unterscheidung zwischen Pegel und wahrgenommener Lautheit von hoher Relevanz. Der Schalldruckpegel beschreibt dabei die physikalische Schalldruckgröße eines Signals in dB, die wahrgenommene Lautheit ist dagegen eine subjektive, psychoakustische Größe, die vom hör- und Wahrnehmungssystem des Menschen abhängt (Scharine et al., 2012, S-2-3). Das menschliche Gehör nimmt verschiedene Frequenzen bei gleichem Pegel unterschiedlich laut wahr, was in den Kurven gleicher Lautstärke nach Fletcher und Munson beschrieben wird (Yost, 2015, S.50). Diese zeigen, dass das Gehör im mittleren Frequenzbereich empfindlicher ist als bei tiefen oder hohen Frequenzen. Entsprechend eignet sich eine bloße Peak-Normalisierung nicht zur Angleichung

der Lautheit. Diese berücksichtigt nur den höchsten Momentanwert, nicht die wahrgenommene Lautheit. Zwei Signale mit gleichen Peak-Werten können also sehr unterschiedlich laut klingen.

Dafür wurden LUFS (Loudness Units relative to Full Scale) als Messgröße etabliert, welche auf dem ITU-R BS.1770-Verfahren basiert (International Telecommunication Union [ITU], 2017, S.1). Dieses Verfahren berücksichtigt die frequenzabhängige Empfindlichkeit des Gehörs durch eine K-Gewichtung des Signals (ITU, 2017, S.3). Dabei wird das Signal durch eine definierte Filterkurve vorverarbeitet, die einerseits hohe Frequenzen leicht anhebt, um die verstärkte Wahrnehmung am Kopf des Hörers zu berücksichtigen und andererseits sehr tiefe Frequenzen ausblendet, da diese vom menschlichen Gehör kaum als Lautheit wahrgenommen werden (ITU, 2017, S.3).
Integrated LUFS dienen als zentraler Messwert und stellen die gemittelte Lautheit über die gesamte Dateilänge dar (European Broadcasting Union [EBU], 2023b, S.4-5). Auch Loudness-Gating ist ein Teil des Standards. Dabei werden sehr leise Passagen oder Stille bei der Messung ausgeblendet, um das Ergebnis nicht zu verfälschen (ITU, 2017, S.2; EBU, 2023b, S.5). Ergänzend kommen noch Momentary und Short-term LUFS als Messwerte hinzu um kurzfristige Lautheitsverläufe akkurat darstellen zu können (EBU, 2023b, S.5).
Weitere Werte sind True- und Sample Peaks. Sample Peaks beschreiben den höchsten gemessenen Abtastwert eines digitalen Signals. Bei der Digital-Analog-Wandlung können zwischen zwei Abtastpunkten jedoch höhere Pegel entstehen als die Abtastwerte selbst anzeigen, diese werden als Intersample-Peaks bezeichnet (ITU, 2017, S.1).
True Peaks berücksichtigen diese Intersample-Peaks durch Überabtastung des Signals vor der Messung und sind damit der relevantere Grenzwert für die Pegelkontrolle (Lund, 2007, S.7). Dabei hat sich -1 dBTP als typischer Grenzwert in der Postproduktion etabliert, um Übersteuerungen bei der Wiedergabe zu vermeiden (EBU, 2023a, S.4).

Neben dem tatsächlichen Inhalt von Audiodateien müssen auch ihre technischen Inhalte sowie ihre organisatorischen Eigenschaften strukturiert und verwaltet werden. Dies führt zur Notwendigkeit eines Metadatenkonzepts als Grundlage für die Sound-Library.

2.5 Audio-Metadaten

Metadaten sind strukturierte Informationen, die ein Objekt beschreiben, ohne dessen eigentlichen Inhalt zu verändern (NISO (2004), zitiert nach Day, 2005, S.1). Sie ermöglichen eine strukturierte Suche, Filterung und Organisation großer Datenbestände. Die Relevanz nimmt dabei mit der Menge an Dateien zu, bei wenigen Dateien reicht ein Dateisystem, bei großen Beständen wird eine

strukturierte Metadatenverwaltung notwendig. Das Grundprinzip ist dabei eine klare Trennung zwischen dem eigentlichen Inhalt und den beschreibenden Informationen.

Grundsätzlich kann in verschiedene Arten von Metadaten unterschieden werden. Deskriptive Metadaten sind beschreibende Informationen, um Inhalt und Kontext einzuordnen. Administrative Metadaten enthalten Verwaltungsinformationen wie Erstellungsdatum und Änderungsdatum sowie technische Metadaten, welche Abtastrate, Bittiefe, Kanalzahl und Dauer beschreiben. Auch rechtliche Informationen wie Lizenzangaben und Urheberrechte fallen in diese Kategorie. (International Association of Sound and Audiovisual Archives [IASA], 2009, Abschn. 3.5.1).

Audiodateien können Metadaten grundsätzlich direkt in ihre Dateistruktur einbetten. WAV-Dateien nutzen dabei das RIFF-Format (Resource Interchange File Format) als Container, die RIFF-Struktur besteht aus sogenannten Chunks. Dabei enthält der data-Chunk die eigentlichen Audiodateien, während der format-Chunk technische Parameter speichert (Chalmers, 1997, S.4), wie beispielsweise die Abtastrate oder Bittiefe. Zudem gibt es einen INFO-Chunk, der als optionaler Bestandteil für eingebettete Textmetadaten wie Titel oder Kommentar dienen kann (Federal Agencies Audio-Visual Working Group, 2012, S.1, 12). BWF (Broadcast Wave Format) stellt eine Erweiterung des WAV-Formats dar mit zusätzlichen bext-Chunks für professionelle Produktionsmetadaten wie Erstellungsdatum und Ursprungsinformationen (Chalmers, 1997, S.3-4).

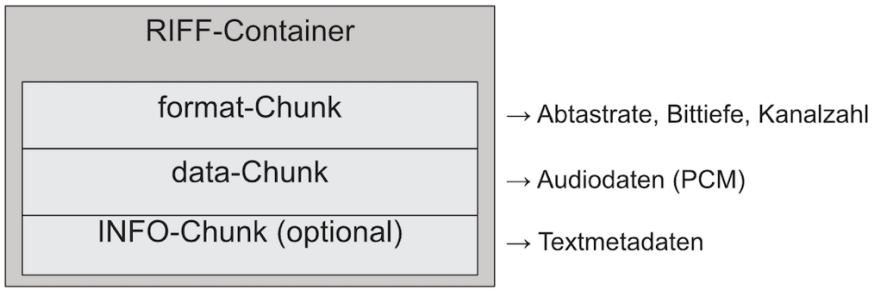

Abbildung 2: Vereinfachte Chunk-Struktur einer WAV-Datei im RIFF-Format (eigene Darstellung)

Daneben existiert eine Vielzahl anderer Metadatenformate. Dazu gehören die weitverbreiteten ID3-Tags, welche ursprünglich für MP3-Dateien entwickelt wurden, inzwischen aber auch für viele andere Formate genutzt werden können (Library of Congress, 2021; The Broadcast Bridge, 2025). XMP (Extensible Metadata Platform) von Adobe ist ein XML-basiertes Format, das in verschiedene Dateiformate eingebettet werden kann (Adobe, 2020, S.5, 7). JSON (JavaScript Object Notation) ist ein leichtgewichtiges, textbasiertes Datenaustauschformat (Bray, 2017, S.1), welches durch seine minimale und portabel Struktur für die Speicherung strukturierter Daten geeignet ist (Bray, 2017, S.3). Metadaten können auch als externe Dateien (sogenannte Sidecar-Dateien) gespeichert werden.

Durch diese Trennung wird die Datei selbst nicht verändert, allerdings besteht das Risiko, dass Metadaten und Datei voneinander getrennt werden (Adobe, 2020, S.7).

Eine externe Verwaltung von Metadaten über Datenbanken stellt eine Alternative zu eingebetteten Metadaten dar. So können diese unabhängig von der Datei angepasst werden, während die Datei selbst unverändert bleibt. Zudem ermöglichen solche Systeme effektive Verwaltungsmöglichkeiten für große Bestände. Allerdings werden Metadaten und Dateien voneinander getrennt, wodurch bei der Weitergabe von Dateien die Verbindung zu den Metadaten verloren gehen kann (Smith et al., 2014, S. 52-53).
Kryptografische Hashfunktionen wie SHA-256 berechnen aus dem Dateiinhalt einen eindeutigen Prüfwert. SHA-256 erzeugt dabei einen 256-Bit-Hashwert. Gleiche Dateiinhalte erzeugen immer denselben Hash, während Änderungen am Inhalt mit sehr hoher Wahrscheinlichkeit einen anderen Hash erzeugen (Dang, 2012, S.2, 6; Eastlake & Hansen, 2006, S.3-4). Durch die Kollisionsresistenz ist es praktisch unmöglich mit zwei verschiedenen Dateien einen identischen Hash zu erzeugen (Dang, 2012, S.6; Eastlake & Hansen, 2006, S.3).
Ein solcher Hash-Wert ermöglicht eine eindeutige Identifikation und Integritätsprüfung unabhängig vom Speicherort (Farrell et al., 2013, S.1, 3).

Um eine strukturierte externe Metadatenverwaltung zu realisieren, ist ein geeignetes Datenhaltungskonzept erforderlich. Dafür bieten sich relationale Datenbanken als natürliche Wahl für die Verwaltung strukturierter, miteinander verknüpfter Metadaten an.

2.6 Datenbanken und SQL

Eine Datenbank ist ein strukturiertes System zur Speicherung, Verwaltung und Abfrage von Daten. Ein Datenbankmanagementsystem (DBMS) dient als Software zur Verwaltung und zum Zugriff auf die Datenbank. Im Gegensatz zu einer einfachen Dateispeicherung bieten DBMS die Möglichkeit strukturierter Abfragen, konsistenter Datenhaltung und der Vermeidung von Redundanzen (Watt & Eng, 2014, Kap. 2-3).

Relationale Datenbanken beruhen auf dem relationalen Modell. Daten werden in Tabellen (Relationen) organisiert, die bestehen aus Zeilen (Datensätzen) und Spalten (Attributen). Primärschlüssel werden als eindeutige Identifikation eines Datensatzes in einer Tabelle verwendet, Fremdschlüssel dienen als Referenz auf einen Primärschlüssel einer anderen Tabelle, wodurch Beziehungen zwischen Datensätzen verschiedener Tabellen ermöglicht werden. Durch die

sogenannte Referenzielle Integrität wird sichergestellt, dass keine ungültigen Referenzen entstehen (Watt & Eng, 2014, Kap. 7, 9).

Um Redundanzen und Anomalien in der Datenhaltung zu vermeiden, können Datenbanken normalisiert werden. Dabei wird zwischen erster, zweiter und dritter Normalform unterschieden (1NF, 2NF, 3NF). Diese stellen aufeinander aufbauende Stufen der Normalisierung dar (Watt & Eng, 2014, Kap. 12).

Beziehungen zwischen Tabellen werden in relationalen Datenbanken in unterschiedlichen Beziehungstypen dargestellt. Bei einer 1:1-Beziehung entspricht einem Datensatz in Tabelle A genau ein Datensatz in Tabelle B. In einer 1:n-Beziehung können einem Datensatz in Tabelle A mehrere Datensätze in Tabelle B zugeordnet sein. Bei einer m:n-Beziehung (viele-zu-viele) können mehrere Datensätze der Tabelle A mehreren Datensätzen der Tabelle B zugeordnet sein. Um in einer höher normalisierten Datenbank eine m:n-Beziehung darstellen zu können, wird sie technisch über eine Zuordnungstabelle umgesetzt (Watt & Eng, 2014, Kap. 8).

Relationale Datenbanken werden über die standardisierte Sprache SQL (Structured Query Language) implementiert und verwaltet. SQL wird dabei in verschiedene Bereiche unterteilt: DDL (Data Definition Language) stellt dabei Befehle zur Definition der Datenbankstruktur, wie `CREATE`, `DROP` und `ALTER`. DML (Data Manipulation Language) umfasst Befehle zur Manipulation von Daten, beispielsweise `INSERT`, `UPDATE` oder `DELETE`. .Um mehrere Tabellen in einer Abfrage zu verknüpfen, werden JOIN-Operationen genutzt. Je nach Anforderungen können dabei beispielsweise `FULL-JOINs`, `INNER-JOINs` oder `OUTER-JOINs` genutzt werden (Watt & Eng, 2014, Kap. 15-16).

Indizes stellen in DBMS zusätzliche Datenstrukturen zur Beschleunigung von Abfragen dar. Ähnlich wie bei einem Buchindex ermöglicht ein Datenbankindex den direkten Zugriff auf relevante Datensätze, ohne die gesamte Tabelle durchsuchen zu müssen (SQLite Documentation, 2025a).

Im Verlauf dieser Arbeit wird hauptsächlich auf SQLite eingegangen. Dabei handelt es sich um eine serverlose, dateibasierte Implementierung eines relationalen Datenbanksystems. Hier wird die gesamte Datenbank in einer einzigen Datei gespeichert, so dass keine separate Serverinstallation notwendig ist und eine direkte Einbindung in die Anwendung ermöglicht wird. Allerdings gibt es einige Einschränkungen im Gegensatz zu vollwertigen DBMS wie PostgreSQL oder MySQL, so ist kein

Mehrbenutzer-Betrieb und nur eine begrenzte Skalierbarkeit bei sehr großen Datenmengen möglich (SQLite Documentation, 2024; SQLite Documentation, 2025b).

Für den Zugriff auf die Datenbank und die Interaktion mit dem Nutzer wird eine grafische Benutzeroberfläche notwendig, womit die GUI-Entwicklung in Python den nächsten theoretischen Baustein darstellt.

2.7 GUI-Entwicklung

Eine grafische Benutzeroberfläche oder GUI (Graphical User Interface) ermöglicht die Interaktion mit einer Anwendung über visuelle Elemente wie Fenster, Buttons, Eingabefelder und Listen. Dadurch kommt es zu einer Abstraktion der zugrundeliegenden Programmlogik für den Nutzer. Im Gegensatz zu kommandozeilenbasierten Schnittstellen oder CLI (Command Line Interface) ermöglicht eine GUI eine intuitive Bedienung ohne Kenntnisse der Programmiersprache oder Befehlen.

GUIs basieren auf dem Prinzip der ereignisgesteuerten Programmierung. Das Programm wartet auf Ereignisse (Events) wie Mausklicks, Tastatureingaben oder Fenstergrößenänderungen. Diese Ereignisse lösen dabei sogenannte Callbacks oder Handler-Funktionen aus, also Funktionen die gezielt für die Reaktion auf ein bestimmtes Ereignis definiert werden und die entsprechende Programmlogik ausführen. Der Event-Loop (Ereignisschleife) stellt ein zentrales Konzept dar. Dabei handelt es sich um eine kontinuierlich laufende Schleife die Ereignisse entgegennimmt und an die entsprechenden Handler weitergibt. Zeitintensive Operationen im Main-Thread blockieren den Event-Loop und führen zu einem eingefrorenen UI, weshalb solche Operationen in separate Threads ausgelagert werden (Python Software Foundation, 2026a).

Python bietet verschiedene Frameworks zur komfortablen GUI-Entwicklung.
Tkinter ist Teil der Python-Standardbibliothek, wodurch es plattformübergreifend verfügbar ist (Python Software Foundation, 2026a).
PyQt stellt eine leistungsfähigere Alternative dar, mit einer umfangreichen Widget-Bibliothek von über 1000 Klassen. Allerdings erfordert PyQT externe Abhängigkeiten, da Qt separat bezogen werden muss. Außerdem unterliegt es einer dualen Lizenzierung und einer kommerziellen Lizenz (Riverbank Computing, 2026).
Eine weitere Alternative ist wxPython. Diese nutzt native Betriebssystem-Widgets für ein plattformspezifisches Erscheinungsbild (wxWidgets, 2026).

Webbasierte Ansätze wie Electron ermöglichen die Entwicklung von Desktop-Anwendungen mit JavaScript, HTML und CSS, was jedoch die Komplexität der Anwendung erhöhen kann (Electron Documentation, n.d.).

Für die Implementierung der Sound-Library wurde Tkinter gewählt, da es als Teil der Python-Standardbibliothek frei verfügbar ist. Tkinter basiert auf dem Tk-Framework, das ursprünglich für die Skriptsprache Tcl entwickelt wurde und heute plattformübergreifend für Windows, macOS und Linux verfügbar ist. Es ermöglicht einen Widget-basierten Aufbau, bei dem UI-Elemente wie Buttons, Labels, listboxes und Eingabefelder als Objekte instanziiert und im Fenster platzier werden können. Mit den Geometry Mangern pack, grid und place können diese Widgets platziert werden (Python Software Foundation, 2026a).
Ttk ist eine moderne Erweiterung von Tkinter, die in Tk 8.5 eingeführt wurde. Sie bietet themes und ein deutlich besseres Erscheinungsbild auf verschiedenen Plattformen als die klassischen Tk-Widgets (Python Software Foundation, 2026b). Da Tkinter single-threaded arbeitet, sollten zeitintensive Operationen wie Datenbankabfragen in separate Threads ausgelagert werden, um ein Blockieren der Ereignisverarbeitung zu vermeiden (Python Software Foundation, 2026a).

Die vorgestellten theoretischen Grundlagen bilden die Basis für die Konzeption und Implementierung der Sound-Library. Im folgenden Kapitel werden diese Grundlagen in konkrete Anforderungen und Designentscheidungen überführt.

3. Konzeption der Sound-Library

Dieses Kapitel beschreibt die vorgenommene Planung und Konzeption der Sound-Library, die als Basis für die spätere Implementierung (siehe Kapitel 4) dient.

Wie in Kapitel 1 angesprochen, liegen Audiodateien oft verteilt in verschiedenen Ordnerstrukturen, wodurch die Organisation und die Suche nach bestimmten Dateien oftmals sehr aufwendig wird. Auch die Metadatenpflege der Dateien ist oftmals umständlich. Der Leitgedanke dieser Sound-Library ist es, eine Tool-unabhängige Lösung bereitzustellen, die einen eigenen, individuellen Workflow ermöglicht, ohne sich an den Funktionsumfang oder den Update-Zyklen bestehender Tools und Anwendungen anpassen zu müssen.
Das Dateisystem fungiert für dieses Konzept als zentrale Quelle der Audiodateien, während eine zusätzliche Schicht die Suche, Organisation und Metadatenpflege vereinfachen soll. Um die entsprechenden Dateien zuverlässig und unaufwändig in das System zu integrieren, wird ein

Mechanismus benötigt, der die Dateien automatisiert verwaltet. Die gesamte Konzeption soll so angelegt sein, dass zusätzliche Anforderungen im weiteren Verlauf ergänzt werden können.
Im Folgenden werden Grundanforderungen, zentrale Konzeptionsentscheidungen, mögliche Workflows sowie Bedienkonzepte beschrieben, welche als Grundlage für die Implementierung im Kapitel 4 dienen.

3.1 Zielsetzung

Der Fokus der Sound-Library liegt auf einem Tool-unabhängigen Workflow.
Dabei erfolgt die Ablage der Audiodateien im lokalen Dateisystem, die Sound-Library an sich ergänzt diese Ablage um eine Verwaltungs- und Such-Schicht. Neu exportierte oder abgelegte Dateien sollen ohne manuelles Importieren in die Library übernommen werden können, wo eine effiziente Suche, Filterung und Sortierung ermöglicht wird, um Sounds schneller auffindbar zu machen. Zudem sollen Sounds über Kategorien und frei verfügbare Tags (Schlagwörter) detailliert strukturiert werden können.
Eine weitere zentrale Anforderung ist die einfache Metadatenpflege. Nicht-technische Metadaten sollen zentral in der Sound-Library angelegt oder bearbeitet werden können, ohne dabei die Dateinamen oder die Ordnerstruktur als alleinige Informationsquellen zu verwenden.
Des Weiteren soll ein direktes Vorhören innerhalb der Sound-Library möglich sein, so dass Sounds direkt abgespielt werden können, um einen schnellen und unkomplizierten Entscheidungsprozess zu unterstützen.
Eine Qualifizierung durch eine einfache Bewertung, beispielsweise von 1 bis 5 soll die Auswahl geeigneter Sounds unterstützen. Abschließend soll das System so konzipiert werden, dass neue Metadatenfelder, Filterkriterien oder Funktionen ergänzt werden können.

Andere denkbare Funktionen werden dagegen bewusst in dieser Arbeit nicht berücksichtigt. So wird auf eine vollständige Synchronisation mit dem Dateisystem verzichtet. Zwar werden gelöschte Dateien beim Scan automatisch bereinigt und verschobene Dateien über einen Hash-Abgleich erkannt, eine darüber hinausgehende automatische Synchronisation des Dateisystems ist jedoch nicht vorgesehen.
Auch auf einen Cloud- oder Multi-User-Betrieb wird verzichtet, der Prototyp wird für einen lokalen Single-User-Betrieb konzipiert.
Allgemein soll sich der Umfang der Anwendung auf Kernfunktionen und Workflows konzentrieren. Erweiterte Funktionen, wie sie teils in gängigen Sound-Libraries verbreitet sind, sollen noch kein Bestandteil des Prototyps sein. Dazu gehören beispielsweise DAW- oder Editor-Funktionen wie Schneiden, Effektbearbeitungen, Mono-Summierungen oder ähnliche signalverändernde Prozesse.

Zudem wird auf eine direkte Integration von DAWs, um beispielsweise Sounds direkt in eine Session zu transferieren, verzichtet. Es soll sich auf die Organisation, Suche und Vorhören fokussiert werden.

Anhand dieser Ziele und Abgrenzungen werden in den folgenden Abschnitten die konkreten Anforderungen abgeleitet.

3.2 Workflow-Anforderungen

In diesem Abschnitt werden konkrete Anforderungen für den geplanten Workflow mit der Sound-Library abgeleitet. Dabei soll die Dateisystem-basierte Ablage beibehalten werden, die Suche und die Pflege unterstützt werden. Die Anforderungen beschreiben, welche Funktionen der Prototyp zur Unterstützung dieses geplanten Workflows bieten muss.

Den grundlegenden Schritt des Workflows stellt eine simple Ablagestruktur dar. Hierfür muss ein Basisverzeichnis mit Unterordnern vorhanden sein, welche den genutzten Kategorien entsprechen. Nach der Ablage sollen Dateien automatisch erkannt, neue Dateien in die Datenbank und somit in die Sound-Library übernommen, geänderte Dateien aktualisiert werden. Der Scan nach neuen oder geänderten Dateien soll dabei automatisch stattfinden und keine Duplikate erzeugen.

Sind die Dateien in die Sound-Library aufgenommen, soll eine einfache Suche und Navigation ermöglicht werden. Über eine Textsuche sollen zentrale Felder erreicht werden, Kategorien, Kanalanzahl und Tags sollen als Filter fungieren. Des Weiteren soll eine schnelle Sortierung, beispielsweise alphabetisch, nach Bewertung oder der Aktualität ermöglicht werden.

Durch ein direktes Abhören in der Anwendung, sowie einen schnellen Wechsel zwischen den Treffern wird ein direkter Vergleich zwischen Sounds ermöglicht. Diese können durch eine Bewertung einfach priorisiert werden.

Als nächster Schritt ist die manuelle Metadatenpflege von hoher Relevanz. Nicht-technische Metadaten sind direkt in der Sound-Library editierbar. So können Tags hinzugefügt und entfernt werden, außerdem eine Kurzbeschreibung angelegt werden. Die Kategorie kann manuell gesetzt bzw. korrigiert werden, wobei es bei dem manuellen Korrigieren zu einem automatischen Schutz vor Überschreibungen kommt, so dass dieser Eintrag bei einem wiederholten Scan stabil bleibt und Updates ohne den Verlust von manuell bearbeiteten Metadaten durchgeführt werden können. Änderungen können durch die Sortier-Funktion der Sound-Library direkt nachvollzogen werden.

Auf Audio-Bearbeitung und DAW-Integration wird bewusst verzichtet, der Fokus liegt auf Organisation, Suche und Vorhören.

3.3 Konzeption des Audio-Datensatzes

Ursprünglich wurden für den Datensatz fünf Kategorien geplant: Atmosphären, Foleys, Haushaltsgeräusche, Autos und Schritte. Atmosphären werden in der Sound-Library statt „Atmos“ als „Ambience“ bezeichnet, um Verwechslungen mit dem Audio-Format Dolby Atmos zu vermeiden. Diese Kategorien decken einen großen Teil der in der Sound-Postproduktion regelmäßig benötigten Geräuschkategorien ab. Im Verlauf der Konzeption wurde Türen als sinnvolle Ergänzung identifiziert und als sechste Kategorie aufgenommen, da Türgeräusche einen eigenständigen und häufig genutzten Bereich darstellen.

Zweck des Audio-Datensatzes ist es, eine ausreichende Varianz abzubilden. Neben den unterschiedlichen Kategorien soll diese Varianz durch beispielsweise unterschiedliche Materialien und Umgebungen erreicht werden, wodurch die Filter- und Tagging-Funktionen praxisnah abgedeckt werden. Der Praxisbezug steht im Vordergrund und soll typische Arbeitssituationen in der Ton-Postproduktion abbilden.

Bezüglich des Dateiformats wurden konzeptionell keine Einschränkungen festgelegt. Die Entscheidung für ein konkretes Format wird im Rahmen der Implementierung getroffen und im Kapitel 4.9.1 begründet.

3.4 Datenmodell-Entscheidung

Der folgende Abschnitt beschreibt die Ableitung der Datenstruktur aus den Workflow-Anforderungen. Grundsätzlich bleiben bei dieser Konzeption Audiodateien im Dateisystem und das Datenmodell beschreibt, welche Informationen zusätzlich verwaltet werden müssen, um eine strukturierte Suche und Organisation zu ermöglichen. Dabei wird auf eine klare Trennung zwischen dem Audio-Inhalt der Datei und den Metadaten als beschreibende und organisatorische Informationen geachtet. Das Modell soll konsistent, übersichtlich und erweiterbar bleiben.

Grundsätzlich findet die Verwaltung auf Datei-Ebene statt. Das bedeutet, dass eine Datei einem Eintrag in der Datenbank und somit der Sound-Library entspricht. Das Dateisystem bleibt also die Quelle der Audiodateien.

Die Trennung zwischen technischen und organisatorischen Metadaten nimmt eine zentrale Rolle im Datenmodell ein. Diese soll eine klare Strukturierung und vereinfachte Organisation ermöglichen. Die technischen Metadaten sind dabei direkte Informationen aus den WAV-Dateien, wie beispielsweise Länge, Größe, Kanalanzahl und Abtastrate und sollen grundsätzlich automatisch ermittelt werden.
Die organisatorischen Metadaten dagegen sollen relevanter für den tatsächlichen Workflow sein und manuell pflegbar sein, wie beispielsweise Tags, Kategorien und Kurzbeschreibungen.

Jede Audio-Datei soll dabei genau einer Kategorie zugeordnet sein, um vorab eine grobe Einordnung ihres Charakters beziehungsweise der möglichen Verwendung zu ermöglichen. Die Einordnung in Kategorien geschieht automatisch über die Struktur des Verzeichnisses. Werden Dateien nicht in einen dafür vorgesehenen Unterordner gelegt, werden sie zunächst als „nicht zuordenbar“ eingestuft, um sie trotzdem nutzbar zu machen, ohne die Konsistenz zu beschädigen. Für eine detaillierte Eingrenzung stehen die Tags zur Verfügung.
Die Anzahl der möglichen Tags bleibt dagegen beliebig, sie sind außerdem kombinierbar und unterstützen damit eine flexible Suche oder Filterung.

Als eindeutige Referenz auf die Datei gilt der jeweilige Dateipfad, was auch für den Import und das Playback relevant ist. Zur inhaltsbasierten Identifikation der Datei wird ein Hash-Wert generiert, der unabhängig von Dateiname und Speicherort ist. Dieser dient der Erkennung verschobener Dateien sowie der Vermeidung von Duplikaten. Um die Nachvollziehbarkeit von Änderungen zu gewährleisten und sichtbar zu machen, werden für die Erstellung und mögliche Änderungen Zeitstempel angelegt.

Da durch die automatischen Prozesse Überschreibungen stattfinden können, existiert ein Mechanismus, um manuelle Kategorien zu schützen. Diese Prozesse sollen nicht gegen den Nutzer arbeiten, sondern den Workflow unterstützen.

Das Modell wird auf Datenbankseite so gestaltet, dass Entitäten getrennt voneinander liegen, um sie konsistent, wartbar und erweiterbar zu halten und außerdem unnötige Redundanzen zu vermeiden.

Aus diesen Anforderungen ergibt sich die Wahl eines relationalen Datenmodells. Die Beziehung zwischen Audiodateien und Tags ist grundsätzlich vom Typ viele-zu-viele, da ein Sound mehreren Tags zugeordnet sein kann und ein Tag mehreren Sounds zugeordnet sein kann. Ebenso besteht eine klare Beziehung zwischen Sounds und Kategorien. Relationale Datenbanken sind für genau diese Art von strukturierten Beziehungen konzipiert und ermöglichen komplexe und kombinierbare Abfragen,

wie sie für die geplanten Such- und Filterfunktionen notwendig sind. Zudem unterstützt das relationale Modell die angestrebte Trennung der Entitäten, vermeidet Redundanzen und bleibt durch seine klare Struktur erweiterbar. Die konkrete Umsetzung des Datenmodells in Tabellenstruktur, Attribute und Constraints erfolgt in Kapitel 4.5.

3.5 Import-/Scan-Konzept

Im Folgenden werden die konzeptionellen Überlegungen für den Import- und Scan-Prozess beschrieben. Dieser soll als Bindeglied zwischen Dateisystem und Sound-Library dienen.

Das Ziel dieses Prozesses ist das automatische Übernehmen neuer Dateien aus dem Basisverzeichnis, sowie die Aktualisierung bereits bekannter Dateien. Das soll durch minimale manuelle Schritte des Benutzers im Workflow geschehen, nach dem Export wird das Verzeichnis gescannt und die Dateien sind sofort verfügbar. Der Fokus liegt auf Zuverlässigkeit und Wiederholbarkeit.

Das Dateisystem ist wie angedacht die Quelle der Dateien, der Import erzeugt und aktualisiert nur Metadaten-Einträge, die Audio-Dateien an sich werden nicht verändert. Die Dateisystemstruktur kann dabei als Orientierung dienen.

Der Scan-Prozess soll das Basisverzeichnis vollständig erfassen und dabei ausschließlich relevante Dateitypen berücksichtigen. Neue Dateien sollen automatisch in die Library aufgenommen, bestehende Einträge aktualisiert werden, ohne dabei organisatorische Metadaten zu verwerfen. Das Ergebnis ist eine stets aktuelle Datenbasis, die unmittelbar nach dem Scan nutzbar ist.

Auch der Schutz vor der Überschreibung von Kategorien muss hier sichergestellt werden. Manuell gesetzte Kategorien werden bei erneutem Scannen nicht aufgrund ihres Dateipfades überschrieben und Tags, Beschreibungen und Ratings bleiben erhalten. Die konkrete technische Umsetzung des Scan-Prozesses erfolgt in Kapitel 4.9.

3.6 GUI-Konzept

Die GUI dient als zentrale Schnittstelle zur Sound-Library und unterstützt den Workflow. Die folgenden Funktionen ergeben sich direkt aus den in Abschnitt 3.2 beschriebenen Anforderungen.

Die GUI verfolgt das Ziel, möglichst effizient auf den Inhalt der Sound-Library zuzugreifen. Der Fokus liegt dabei auf Geschwindigkeit und einem Workflow in wenigen, unkomplizierten Schritten. Es geht dabei nicht um Audio-Bearbeitung, sondern primär um Verwaltung, Auswahl und Vergleich.

Die Kernfunktionen umfassen ein Such-Feld zur Textsuche, Filter für Kategorien und Tags, eine übersichtliche Ergebnisliste, Playbacks sowie die Möglichkeit der Metadatenbearbeitung und Bewertung.
Pro Treffer einer Suche sollen die Ergebnisse kompakt mit den wichtigsten Informationen bereitgestellt werden, wie Name, Kategorie, Bewertung und gegebenenfalls Dauer. Dabei wird eine klare Trennung von Trefferliste und Detailbereich anvisiert.

Bei den Bedienprinzipien steht klar im Fokus, dass die Such- und Filter-Funktionen immer sichtbar sind und schnelle Interaktionen möglich sind. Zudem sollen mögliche Fehler grundlegend minimiert werden und konsistentes Verhalten, wie dass Filter immer gleich wirken, ermöglicht werden. Editing, Schneiden und Effekte spielen ebenso wie eine DAW-Integration oder Session-Import keine Rolle. Auch komplexe Rechte-/Nutzer-Verwaltung wird nicht berücksichtigt.

Abbildung 3: Konzeptionelle Systemarchitektur der Sound-Library (eigene Darstellung)

Die vorliegenden Konzeptionsentscheidungen bilden die Grundlage für die Implementierung in Kapitel 4. Abweichungen, die sich im Verlauf der Umsetzung ergaben, werden in den jeweiligen Abschnitten begründet.

4. Implementierung

4.1 Aufnahme der Audiodateien

Die Aufnahme der Audiodateien diente dazu, eine praxisnahe Grundlage für den Prototypen der Sound-Library zu schaffen. Dieser Datensatz soll als praxisnaher Test- und Evaluierungsdatensatz für die Funktionen des Prototyps dienen. Das Ziel ist nicht der Aufbau einer vollständigen kommerziellen Sound-Library, sondern ein Funktionsnachweis des Prototyps, sowie die Möglichkeit den dazugehörigen Workflow zu bewerten.

Hierzu wurde vorab geplant, welche Kategorien abgedeckt werden sollten. Diese umfassten Atmosphären (im Folgenden kurz Atmos, in der Sound-Library als „AMBIENCE“ bezeichnet, siehe Kapitel 3.3), Türen, Foleys, Schritte, Autos und Geräusche aus dem Haushalt. Dabei waren je Kategorie 40 beispielhafte und typische Geräusche aus dem Alltag vorgesehen. Während Schritte,

Foleys, Autos und Haushaltsgeräusche als kurze Aufnahmen geplant waren, stellten insbesondere die kontinuierlichen Atmos die längsten Aufnahmen am Stück dar. Musikalische Inhalte oder Dialoge wurden in den Aufnahmen nicht berücksichtigt.

Die Mischung aus kurzen Audios sowie längeren Atmos wurde bewusst gewählt, um unterschiedliche Dateilängen und typische Workflows abzudecken. Innerhalb der Kategorien wurde eine gewisse Varianz, durch beispielsweise unterschiedliche Umgebungen, Materialien oder Intensitäten angestrebt, um eine ausreichende Heterogenität für spätere Filter- und Tagging-Funktionen zu erzeugen.

Trotz der vergleichsweise großen Anzahl der geplanten Aufnahmen für diese Arbeit war der Umfang bewusst begrenzt, da der Fokus darauf lag, einen möglichst praxisnahen Workflow darzustellen. Während es in kommerziellen Sound-Libraries häufig eine größere Varianz von einzelnen Geräuschen gibt, lag die Priorität der Aufnahmen in der Abdeckung eines möglichst großen Spektrums innerhalb der Sound-Library. Das aufgenommene Material diente in erster Linie zur Konzeption und Implementierung der Sound-Library, sowie zur Evaluation der Software.

Die Auswahlbegründung der Geräusche und Kategorien wurde bereits in Kapitel 3 vertieft, die folgenden Abschnitte beschäftigen sich mit der praktischen Umsetzung der Aufnahmen und Bearbeitungen.

4.1.1 Mobile Field-Recordings

Die mobilen Field-Recordings umfassten die Kategorien Schritte, Türen, Autos, Haushaltsgeräusche und Atmos. Als Aufnahmegerät wurde der Zoom H1n genutzt, ein mobiler Audio-Recorder mit integrierten (XY-)Stereo-Mikrofonen. Dieser empfahl sich aufgrund seiner flexiblen Einsatzmöglichkeiten. Diese entstehen durch die kompakte und portable Bauweise, sowie die Möglichkeit der Anpassung durch verschiedene Griffe oder Stative, wie beispielsweise einem Dreibein. Außerdem gibt es die Möglichkeit den Recorder mit externen Mikrofonen einzusetzen, worauf allerdings bei dieser Arbeit verzichtet wurde, um ein möglichst einheitliches Klangbild der Field-Recordings zu gewährleisten.

Das Vorgehen beim Recording vor Ort variierte je nach äußeren Begebenheiten und Zweck der Aufnahmen, wobei ein gewisser Grundworkflow meist bestehen blieb.
So wurde das Stereo-Mikrofon meist direkt zur Schallquelle ausgerichtet. Vereinzelt wurde die Ausrichtung leicht Off-Axis (seitlich zur Mikrofonachse) vorgenommen, um Peaks zu reduzieren.

Bei Aufnahmen für Details oder Handling-Geräusche wurde mit kürzerer Distanz aufgenommen, um Umgebungsgeräusche zu minimieren, während für natürlichere Perspektiven, insbesondere bei Atmos oder auch Autos, eine größere Distanz gewählt wurde.
Je nach Aufnahmebedingung wurde entweder der Griff des Recorders oder das Dreibein-Stativ genutzt, um Griffgeräusche zu minimieren.

Abbildung 4: Field-Recording-Aufnahmesituation mit dem Zoom H1n auf Dreibein-Stativ an einem teilweise zugefrorenen Wasserfall (eigene Aufnahme)

Vor den meisten Aufnahmen wurden zusätzliche Testaufnahmen durchgeführt, um den eingestellten Pegel zu überprüfen und eventuelle Störquellen zu identifizieren.

In den Field-Recordings wurde bis auf wenige Ausnahmen zu Testzwecken mit einer Abtastrate von 48 kHz aufgenommen. Diese stellt eine gängige Praxis für Film, Games und allgemein einen Großteil der audiovisuellen Medien dar. Für die Sound-Library ist der Frequenzbereich ausreichend, um die geplanten Alltags- und Umgebungsgeräusche adäquat aufzunehmen. Es wurde sich bewusst gegen höhere Abtastraten, wie beispielsweise 96 kHz entschieden, da diese keinen relevanten Mehrwert für den Anwendungsfall bieten und gleichzeitig einen höheren Speicherbedarf erfordern, was eine höhere Datenmenge bei gleichbleibendem Nutzen darstellt. Die Priorität lag auf Effizienz und Kompatibilität.
Anders zu bewerten wäre es, falls in der Postproduktion offensichtlich mit extremen Pitch-Shifting oder Time-Stretching gearbeitet werden würde. Da in der Sound-Library aber primär realistische Geräusche genutzt werden, ergab sich hierbei kein Mehrwert.

Als Bittiefe wurde 24 Bit gewählt. Diese bietet einen theoretischen Dynamikumfang von etwa 146 dB (Kester, 2009, S.19) und stellt einen gängigen Standard für Audioaufnahmen und Film- und Game-Audio dar. Dies gewährleistet einen ausreichend großen Headroom während der Aufnahmen, sowie geringes Quantisierungsrauschen und somit die Möglichkeit einer sicheren Pegelwahl, auch bei wechselnden Lautstärken. Eine alternative Überlegung stellte eine Bittiefe von 32 Bit Float dar. Bei 32 Bit Float entsteht ein sehr großer digitaler Headroom (Sound Devices, 2020), wodurch eine flexible nachträgliche Pegelanpassung möglich ist, solange es zu keiner analogen Übersteuerung kam. Da der Zoom H1n allerdings nur Aufnahmen bis 24 Bit unterstützt (Zoom Corporation, 2017), wurde 32 Bit Float für diese Arbeit nicht eingesetzt.

Auch das Dateiformat lässt sich im Zoom H1n einstellen. Da im Prototyp der Sound-Library WAV als Format angestrebt wurde (vgl. Kapitel 4.9.1), wurde sich auch hier für WAV entschieden. Aufgrund ihrer hohen Kompatibilität und zur Gewährleistung einer robusten Weiterverarbeitung im Prototyp wurden dabei PCM-WAV-Dateien verwendet. Dieses Format bietet eine unkomprimierte Aufnahme und somit die verlustfreie Speicherung der Rohdaten und findet flächendeckend in DAWs, Game-Engines und Audio-Tools Verwendung und eignet sich vollumfänglich für spätere Bearbeitungen und Archivierungen.

Die Möglichkeit zur Automatischen Pegelregelung (AGC) des Recorders wurde testweise eingesetzt, wodurch subjektiv wahrnehmbare Nachteile festgestellt wurden. Dazu zählten neben hörbaren Pegelsprüngen auch das unnatürliche Anheben leiser Passagen sowie generell instabile Lautheitsverläufe. Insbesondere bei kurzen oder transientenreichen Geräuschereignissen waren diese Probleme vermehrt wahrnehmbar. Daher wurde auf AGC während der Aufnahmen verzichtet

und die manuelle Einstellung des Aufnahmepegels gewählt. Durch die bewusste Kontrolle war es möglich, den Pegel aktiv an Aufnahmesituationen anzupassen, ausreichend Headroom zur Vermeidung von Clipping sicherzustellen und unkontrollierte Eingriffe in das Signal zu vermeiden.

Außen- und insbesondere Atmo-Aufnahmen stellten sich als besonders abhängig von nicht kontrollierten Bedingungen wie beispielsweise Verkehr, Fluglärm oder Wetter dar. Entsprechend wurde iterativ vorgegangen. Nach kurzen Testaufnahmen wurde die Nutzbarkeit bewertet, anschließend die Aufnahme fortgesetzt, wiederholt oder verworfen.

Aus der Praxis wurden einige Störquellen vermehrt festgestellt. So war Straßenlärm selbst bei größerer Distanz oft sehr klar wahrnehmbar und dominant auf den Aufnahmen. Auch Fluglärm trat wiederholt auf, was sich als besonders problematisch herausstellte, da dieser sich nicht sinnvoll herausschneiden ließ. Zudem waren die von Wind erzeugten Artefakte relevant. Selbst durch den Einsatz des Windschutzes kam es in einigen Fällen zum Verwerfen und Abbruch von Aufnahmen. Des Weiteren unterschied sich in einigen Fällen die erwartete Geräuschkulisse deutlich von den Erwartungen. Beispielsweise wurden bei nächtlichen Aufnahmen im Wald kaum Tiergeräusche oder Rascheln im Unterholz festgestellt, stattdessen traten trotz der Tageszeit und Entfernung entfernte Straßen primär in den Vordergrund. Ähnlich verhielt es sich bei Atmo-Aufnahmen an Seen, die Wassergeräusche waren weniger präsent als erwartet, während Verkehrsgeräusche im Hintergrund deutlich wahrnehmbar blieben.
Subjektiv wurde beobachtet, dass die Jahreszeit möglicherweise eine Auswirkung auf die Hörbarkeit entfernter Lärmquellen haben könnte, beispielsweise durch die geringere Abschirmung durch fehlende Belaubung im Winter. Schlussendlich wurde als Konsequenz festgestellt, dass Aufnahmebedingungen nicht vollständig planbar sind, identische Orte können je nach Wetterbedingungen, Tages- und auch Jahreszeit deutlich unterschiedlich klingen.
Um entsprechende Störquellen möglichst zu reduzieren, wurden verschiedene Maßnahmen getestet. Bei zeitlichen Anpassungen, wie der Variation der Tageszeit zu beispielsweise verkehrsärmeren Zeitfenstern, blieb Lärm dennoch oftmals präsent. Ein Standortwechsel innerhalb eines Gebiets, um das natürliche Signal-Rausch-Verhältnis zu verbessern, beispielsweise durch einen erhöhten Abstand zu Straßen oder die Suche nach einer natürlichen Abschirmung durch Geländebegebenheiten, brachten in den meisten Fällen keine signifikanten Verbesserungen.

Als Konsequenz wurde für besonders ruhige Atmo-Aufnahmen mehrtägig in abgelegeneren Regionen aufgenommen, um Verkehrs- und Fluglärm zu reduzieren. Trotz dieser Maßnahme konnten nicht alle

geplanten Atmo-Szenarien realisiert werden, was die praktischen Grenzen von Field-Recordings unter realen Umweltbedingungen unterstreicht.

Als hilfreich für das spätere Editing erwies sich eine kurze Referenzaufnahme zur Unterstützung späterer Crossfades oder als Referenz für Grundrauschen.

Es zeigte sich, dass das Field-Recording ein iteratives und insbesondere flexibles Vorgehen erfordert. Während die Planung durchaus relevant für Struktur und Orientierung ist, wird die praktische Umsetzung jedoch maßgeblich durch Umweltbedingungen beeinflusst. Rückblickend lieferte das iterative Vorgehen trotz der Einschränkungen verwertbares Material für den Prototyp, auch wenn nicht alle geplanten Szenarien vollständig realisiert werden konnten.
Die mobilen Field-Recordings bildeten damit den ersten Teil der Aufnahmebasis, für präzisere und kontrollierte Geräusche wurden ergänzend Foley-Aufnahmen durchgeführt.

4.1.2 Foley-Aufnahmen

Die Aufnahmesituation der Foleys war kontrollierbarer als die der Field-Recordings und erlaubte die Ergänzung der Sound-Library um präzisere und reproduzierbare, klar definierte Geräusche.

Während sich der Zoom H1n für mobile oder spontane Aufnahmen eignet, bringt er für Foley-Aufnahmen bestimmte Einschränkungen. So gibt es nur eine begrenzte Kontrolle über die Mikrofonposition, außerdem können Handling- und Körperschallanteile sowie wechselnde Abstände die Aufnahme stärker beeinflussen. Da auf reproduzierbare Bedingungen und eine konstante Klangcharakteristik abgezielt wurde, war eine gezielte Platzierung notwendig.
Stattdessen wurde für die Foley-Aufnahmen das RØDE NT1-A verwendet. Das Großmembran-Kondensatormikrofon zeichnet sich durch ein sehr geringes Eigenrauschen aus (Equivalent Noise Level: 5 dBA, A-weighted) (RØDE Microphones, n.d.) und eignet sich dadurch auch für leise Geräusche, feine Texturen und Handling-Geräusche. Als häufig in Home-Recording-Setups eingesetztes Mikrofon eignet es sich als praxisnahes Setup.

Bei der Aufnahme wurde auf externe Hardware wie beispielsweise Vorverstärker verzichtet. Stattdessen wurde das Mikrofon über ein Steinberg UR22-Interface direkt angeschlossen und in Nuendo abgenommen. Das Gain-Staging erfolgte somit ausschließlich über das UR22, hierfür wurde vor jeder Aufnahme grob eingepegelt und darauf geachtet, ausreichend Headroom für Transienten zu lassen und Clipping somit vorzubeugen.
Während der Aufnahme kam es nur bedingt zu weiteren Vorkehrungen, falls notwendig wurde ein leichter Tiefpassfilter eingesetzt.

Da der für die Aufnahmen genutzte Raum keine relevanten akustischen Optimierungen besaß, wurden je Aufnahme-Session kurze Referenzaufnahmen durchgeführt, um das Grundrauschen dokumentiert zu erfassen und im Editing berücksichtigen zu können. Während die fehlende akustische Optimierung für Aufnahmen nahe des Mikrofons kaum negative Auswirkung hatte, so stellte die Raumakustik eine klare Grenze dar, insbesondere bei impulsiven Geräuschen mit kurzen Transienten. Entsprechende Aufnahmen eignen sich nur eingeschränkt für eine hochwertige Postproduktion und wurden hier nur teilweise berücksichtigt, da sie dennoch ausreichend als Test- und Demonstrationsmaterial für den Prototyp der Sound-Library sind, da der Funktionsnachweis übergeordnet wurde.

4.2 Editing der Audiodateien

Ziel des Editings war die Aufbereitung der Rohaufnahmen für die weitere Verarbeitung. Dabei wurden technische Störfaktoren entfernt und gleichzeitig darauf geachtet, einen möglichst natürlichen Klangcharakter der Aufnahmen zu erhalten. Entsprechend kam es hier zu keinem kreativen Einsatz von Sounddesign, um eben dieses bei der Nutzung der Sound-Library im vollen Umfang zu ermöglichen. Das Editing stellte die Vorbereitung der Dateien für die Loudness-Normalisierung, Metadatenerfassung und Datenbankintegration dar.

Hierfür wurde auf Software-Seite Nuendo gewählt, die Bearbeitung erfolgte ausschließlich innerhalb der DAW und es kam zu keiner externen Offline-Bearbeitung, um einen konsistenten Workflow für alle Audiodateien sicherzustellen.

Die Arbeit erfolgte in mehreren Sessions, in denen jeweils Batches von 20 bis 40 Audiodateien importiert wurden. Dies ermöglichte übersichtliche DAW-Sessions und verhinderte gleichzeitig die Überlastung des RAMs des Rechners durch zu große Sessions.
Es wurden verschiedene Workflows getestet, zunächst wurden alle Dateien auf einer Spur bearbeitet und in späteren Sessions eine Spur je Datei angelegt. Mit der ersten Variante ergab sich ein schnellerer Import und Setup der Dateien und ein geringerer Session-Overhead. Allerdings waren präzise Plugin-Eingriffe pro Datei schwerer und mehr manuelle Offline-Schritte wurden nötig. Dadurch entstand ein höheres Fehlerrisiko durch beispielsweise den Export von falschen Regionen, zudem wurden Orientierung und Organisation schneller unübersichtlich. In der zweiten Variante dagegen gab es eine klarere Zuordnung, A/B-Vergleiche wurden einfacher, zudem waren Plugins und Automationen pro Datei sauber trennbar. Abschließend war der Batch-Export einfacher und weniger fehleranfällig. Ein Nachteil der Variante ist die größere Anzahl an Spuren, wobei sich diese in Batches von 20 bis 40 Audiodateien noch als gut handhabbar erwiesen. Entsprechend wurde der Workflow

mit einer Spur pro Datei zum Standard, die Arbeit wurde schneller, reproduzierbarer und weniger fehleranfällig.

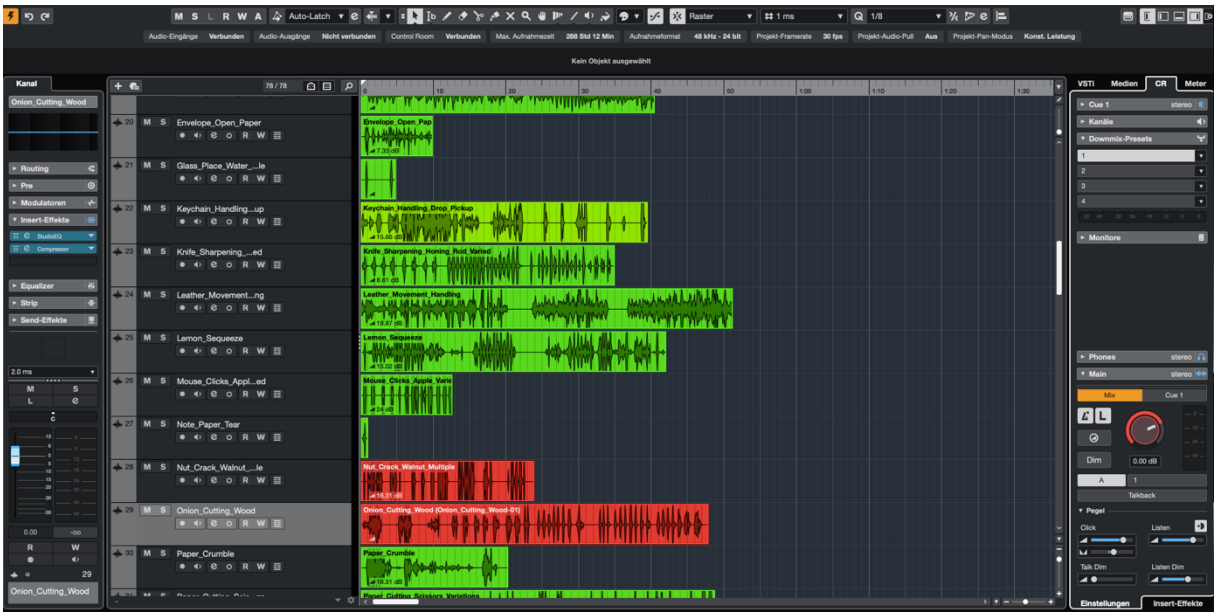

Abbildung 5: Nuendo-Session mit Spur-pro-Datei-Workflow beim Editing der Foley-Aufnahmen. Rot markierte Spuren kennzeichnen während des Editings aussortierte Aufnahmen, grüne Spuren die weiterverarbeiteten Dateien (eigener Screenshot)

Zunächst wurden geeignete Aufnahmen ausgewählt und unbrauchbare direkt entfernt. Bei den als brauchbar bewerteten Dateien wurden unerwünschte Nebengeräusche entfernt und saubere Start- und Endpunkte gesetzt, welche mit kurzen Fades zur Vermeidung von Klicks versehen wurden. Störgeräusche wurden konsistent nur dann entfernt, wenn sie nicht unmittelbar zum Klangereignis gehörten und die Nutzbarkeit der Audio-Dateien einschränkten. Dazu gehörten beispielsweise Atemgeräusche oder Take-Ansagen, insbesondere bei den Field-Recording-Aufnahmen kam es außerdem zu Griff- und Handling-Geräuschen sowie unbeabsichtigten Bewegungen, welche mit aufgenommen wurden. Störstellen oder Handling-Geräusche wurden zumeist komplett herausgeschnitten und mit Crossfades überblendet, vereinzelt auch durch kurze Überblendungen entkoppelt, um Diskontinuitäten zu vermeiden. Die Länge der Crossfades wurde situativ definiert, so waren sie bei transientenreichen Audios kurzgehalten, bei Atmos länger. Die Crossfades wurden situativ, jedoch überwiegend mit konstanter Leistung statt konstantem Pegel gesetzt. Dadurch wurden eher wahrgenommene Löcher in der Mitte der Crossfades verhindert, wie es häufiger bei solchen mit konstantem Pegel auftritt. Der Fokus lag auf subjektiv gleichmäßigen Übergängen ohne hörbares Abfallen der Lautheit.
Das allgemeine Ziel in diesen Editing-Schritten war es, keine hörbaren Sprünge zu produzieren.

Neben dem reinen Schneiden kam es auch zum moderaten Einsatz von Filtern, wobei Hochpassfilter zur Reduktion tieffrequenter Störanteile vermehrt zum Einsatz kamen. Diese wurden typischerweise zwischen 50 Hz und 100 Hz angelegt, wobei die Entscheidung situativ und abhängig von der Art des Klangereignisses war. Auch Tiefpassfilter wurden gelegentlich eingesetzt, um hochfrequente Störanteile zu reduzieren und unnötiges Rauschen zu verhindern. Der Einsatz von Filtern diente ausschließlich der Reduktion von Störanteilen und nicht zur klanglichen Verfremdung.
Ergänzend zu den Filtern wurde häufig ein Gate eingesetzt, um leise, irrelevante Inhalte wie Hintergrundrauschen zwischen den eigentlichen Klangereignissen auszublenden. Die Threshold-Einstellung erfolgte dabei stets vorsichtig und wurde so gewählt, dass das eigentliche Klangereignis vollständig erhalten blieb, während nur die wirklich stillen Passagen betroffen waren. Relevant waren neben dem Threshold auch die Attack- und Release-Zeiten des Gates, um harte Ein- und Ausblendungen zu vermeiden, die als Artefakte hörbar aufgetreten wären. Bei transientenreichen Aufnahmen wurden kurze Attack-Zeiten gewählt, während bei längeren, kontinuierlichen Sounds weichere Release-Zeiten für natürliche Übergänge sorgten.

Neben Filtern und Gates kam auch das Plugin „Spectral De-noise“ von Izotope vermehrt zum Einsatz. Dieses dient zur spektralen Rauschminderung und kann tonale sowie breitbandige Störanteile reduzieren. Entstehende Artefakte werden minimiert, sodass der Klangcharakter erhalten bleibt (iZotope, n.d.).
Durch den Einsatz des Plugins konnte sowohl bei den Field-Recordings als auch bei den Foley-Aufnahmen häufig Hintergrundgeräusche stark reduziert und eine Verbesserung des Signal-Rausch-Verhältnisses erreicht werden. Insbesondere die Eigenschaft, das Plugin anzulernen und ihm „stille“ Stellen als Referenz zu geben erwies sich dabei als besonders effektiv. Auch diese Bearbeitung wurde möglichst dezent eingesetzt, um die natürliche Klangstruktur zu erhalten.

Das Editing stellte also einen reinen technischen Vorbereitungsschritt dar, ohne kreative Klanggestaltung. Der Fokus lag auf Natürlichkeit, Konsistenz und Weiterverwendbarkeit. Der gewählte Workflow erwies sich dabei als praktikabel und reproduzierbar, die Entscheidung für eine Spur pro Datei hat sich insbesondere bei größeren Batches klar bewährt.

4.3 Loudness-Normalisierung

Im nächsten Schritt wurde die Loudness-Normalisierung durchgeführt. Diese dient zur Vereinheitlichung der wahrgenommenen Lautheit und ermöglicht eine Vergleichbarkeit der einzelnen Audiodateien. In der Sound-Library sollte es zu keiner klangästhetischen Veränderung

durch unterschiedliche Lautheit-Wahrnehmung kommen. Das ermöglicht die flexible Weiterverwendung und spätere Mischung der Audio-Dateien.

Als zentrale Messgröße wurden Integrated LUFS-Werte (Loudness Units relative to Full Scale) verwendet. Diese beschreiben die gemittelte Lautheit über die gesamte Dauer einer Messung. Während RMS-Werte den tatsächlichen Energiepegel eines Signals messen, lässt sich durch diese Methode die wahrgenommene Durchschnittslautheit beschreiben, was für diese Arbeit geeigneter ist und auch unterschiedlich lange Audiodateien direkt vergleichbar macht.
Die Ziel-Lautheitsbereiche wurden je nach Klangart angepasst. Während Atmo-Aufnahmen grundsätzlich zwischen -20 bis -24 LUFS angepeilt wurden, wurden bei „One-Shots“ oder sehr kurzen Audios auch höhere Loudness-Werte bis zu -16 LUFS toleriert. Dabei handelt es sich um interne Referenzwerte für diesen Prototyp, um eine allgemeine Vergleichbarkeit zu ermöglichen. Relevant bei den „One-Shots“ ist die Tatsache, dass diese im Rahmen des Workflows häufig mehrere Varianten desselben Sounds in einer gemeinsamen Datei als Variationsserien enthalten. Dadurch ist die Integrated-Loudness-Messung als Referenzwert aussagekräftiger, als wenn ausschließlich einzelne, isolierte One-Shots vorliegen würden. Die stillen Zwischenbereiche beeinflussen den Messwert durch das Loudness-Gating nur gering. Zusätzlich wurde zur Kontrolle auf die True-Peak-Werte geachtet.

Auf Peak-Normalisierung wurde bewusst verzichtet, da diese ausschließlich den höchsten Momentanwert eines Signals berücksichtigt. Als Konsequenz hätte sich voraussichtlich eine stark unterschiedlich wahrgenommene Lautheit trotz identischer Peaks in der Sound-Library ergeben. Stattdessen wurden Peak-Werte im Rahmen des True-Peak-Managements berücksichtigt und nicht als Normalisierungsziel verwendet. Hierbei wurde True Peak als harte Grenze definiert und ein maximaler True-Peak-Wert von -1 dBTP eingehalten.
Um zu garantieren, dass dieser Wert nicht überschritten wird, wurde vor der finalen Messung ein Limiter mit einem Ceiling von -1 dB eingesetzt, die Einhaltung des Grenzwerts wurde anschließend über die True-Peak-Messung überprüft. Audio-Dateien, bei denen der Limiter greifen musste, wurden anschließend nochmals bearbeitet, um durch den Limiter verursachte Artefakte zu vermeiden.

Die Messwerte dienten als Orientierung und Grundlage, die Bewertung der Ergebnisse wurde durch eine Kombination aus den Messwerten und dem direkten Vergleich beim Abhören erreicht. Hierbei wurden mehrere Audios unterschiedlicher Art hintereinander verglichen und die subjektive Konsistenz beurteilt.

Zur neutralen Beurteilung wurde auf dem Abhörweg das Steinberg Plugin Headphone Match eingesetzt, um den Frequenzgang der verwendeten Kopfhörer (AKG K371) zu entzerren. Das Plugin war ausschließlich im Monitoring aktiv und wurde beim Export deaktiviert, damit die Entzerrung nicht die Audiodateien beeinflusst.
Audios, die nach der Normalisierung nicht zufriedenstellend waren, wurden bewusst markiert und ausgeschlossen.
Um die entsprechenden Werte zu erreichen, wurde im Masterbus der jeweiligen Session nach dem Limiter das WLM Plus Stereo Loudness Meter Plugin von Waves genutzt.
Dieses nutzt Messungen nach ITU-R BS.1770-3 und unterstützt verschiedene Standards wie EBU R128 und ATSC A/85 (Waves, n.d.). Auch wenn diese Standards die Messung nach BS.1770 als Grundlage verwenden, haben sie dennoch unterschiedliche Zielkontexte und Vorgaben (Lund, 2011, S.8).

Abbildung 6: Loudness-Messung im WLM Plus Loudness Meter bei einer Atmo-Aufnahme (eigener Screenshot)

Die Loudness-Anpassung wurde eventbasiert durchgeführt, sodass jede Audiodatei einzeln betrachtet wurde. Nach Messung, gegebenenfalls Anpassung und erneuter Messung wurde das Plugin vor jeder neuen Messung zurückgesetzt.

Als primäres Werkzeug um die Standards zu erfüllen, wurde der Clip-Gain der einzelnen Events genutzt. So konnten effektiv laute Passagen abgesenkt und leise gegebenenfalls angehoben werden. Im Falle extrem lauter Einzelereignisse wurde mit gezieltem Schneiden, Gain-Reduzierung und Crossfading gearbeitet. Dabei wurde stets das Ziel eines relativen Lautheitsverlaufs sowie der Vermeidung unnötiger Dynamikbearbeitung durch Kompressoren verfolgt.

Der Einsatz von Kompressoren ließ sich jedoch nicht vollständig vermeiden, wurde allerdings vornehmlich in Fällen mit starkem Dynamikumfang oder einzelnen, sehr dominanten Peaks eingesetzt. Teilweise wurde auch vergleichsweise starke Kompression eingesetzt, wenn die Anpassung des Clip-Gains allein nicht ausreichte, keine andere sinnvolle Loudness-Anpassung möglich war, oder die manuelle Bearbeitung bei vereinzelten Aufnahmen zu aufwändig wurde. Das Ziel war ein kontrollierter Pegel und der Erhalt der Nutzbarkeit der Audios.

Die Loudness-Normalisierung stellte damit den letzten technischen Verarbeitungsschritt vor der Integration in die Sound-Library dar und brachte Konsistenz und Nachvollziehbarkeit in die Audios. Die Kombination aus Messwerten und direktem Abhörvergleich hat sich dabei als sinnvoller Ansatz erwiesen, da rein numerische Zielwerte die wahrgenommene Konsistenz allein nicht ausreichend beschreiben. Es wurde keine vollständige Gleichmachung durchgeführt, sondern bewusst zwischen Zielwerten, klanglicher Qualität und direkten Vergleichen abgewogen.

Nach Abschluss der Loudness-Normalisierung einer Session wurden die Audiodateien als Batch-Export in einen vorliegenden WAV_EXPORTS Ordner exportiert und in Unterordner entsprechend ihrer Kategorie in der Sound-Library sortiert. Mit diesem Schritt lagen technisch bereinigte und standardisierte Audiodateien vor, welche klanglich final und nicht weiter Gegenstand weiterer Bearbeitung waren. Hiermit wechselt der Fokus von der Audiobearbeitung zur Organisation, Strukturierung und Verwaltung, wobei ab hier Metadaten eine zentrale Bedeutung einnehmen. Die systematische Erfassung der Audiodaten und die eindeutige Zuordnung entsprechender Metadaten werden notwendig, woraus sich der Einsatz einer relationalen Datenbank und die Entwicklung einer strukturierten Datenhaltung ergibt.

4.4 Datenbankkonzept

Die Datenbank ist von elementarer Bedeutung für die Sound-Library. Sie verwaltet alle Audio-Dateien und trennt zwischen Audiodateien und Metadaten. Dabei werden die Audiodateien nicht direkt in der Datenbank gespeichert, sondern bleiben als WAV-Dateien im Dateisystem.

Die Aufgabe der Metadaten ist in erster Linie die Beschreibung der eigentlichen Dateien. So werden hier Inhalt, Kontext und die Eigenschaften der jeweiligen Datei beschrieben. In diesem Fall können das beispielsweise die Kategorie, verschiedene Tags, eine Beschreibung, die Dauer und der Dateipfad sein. Dadurch werden gezielte Filterungen, thematische Gruppierungen und effiziente Wiederverwendungen unterstützt.
Entsprechend wird zwischen Inhalt und Verwaltung getrennt. Die Audiodatei bleibt stets ein unverändertes Medienobjekt, während die Metadaten jederzeit, unabhängig vom Audioinhalt, anpassbar und erweiterbar sind.

Wie in Kapitel 3.4 begründet, ermöglicht ein relationales Datenmodell die für dieses System notwendigen strukturierten Beziehungen sowie kombinierbare Abfragen.

Für diese Arbeit stellt die Datenbank die sogenannte *Single Source of Truth* dar und dient als Backend für die GUI. Diese greift auf die Datenbank zu, um die Audiodaten zu finden, die Metadaten zu filtern und nach Tags zu suchen. Änderungen in der GUI wirken sich direkt auf die Datenbank aus und umgekehrt.

Die Datenbank stellt als zentrales Bindeglied zwischen Audio und Software ein funktionales Organisationswerkzeug dar und existiert nicht als reiner Selbstzweck. Der Fokus in der Implementierung lag auf Verständlichkeit, Erweiterbarkeit und Praxisnähe.

4.5 Datenbankschema

Die Konzeption des Datenbankschemas verfolgte mehrere Ziele. So soll die eindeutige Identifikation jeder Audiodatei gewährleistet werden und gleichzeitig alle Metadaten der Sound-Library strukturiert abgebildet werden. Zudem sollen die Filter- und Suchfunktionen, die GUI-Anbindung und der automatische Import neuer Dateien unterstützt werden.
Gleichzeitig werden redundante Daten vermieden, was die Grundlage für eine verlässliche und erweiterbare Datenverwaltung gewährleistet.

Das Schema besteht aus vier zentralen Entitäten:

1. `sounds`: Diese Entität repräsentiert einzelne Audiodateien und enthält die Dateireferenz sowie technische und beschreibende Metadaten.
2. `categories`: Hier werden die Audios inhaltlich kategorisiert, beispielsweise in Foleys oder Atmos.

3. `tags`: Entität, die zur flexiblen Beschreibung der Audios dient, wobei mehrere Tags pro Datei möglich sind.
4. `sound_tags`: Die Entität fungiert als Zuordnungstabelle zur Abbildung der viele-zu-viele-Beziehung zwischen *sounds* und *tags*.

Jede Entität besitzt einen Primärschlüssel als künstliche ID. In Kombination mit Fremdschlüsseln entstehen eindeutige Referenzen, die unabhängig vom Dateinamen sind, was zu referenzieller Integrität führt.

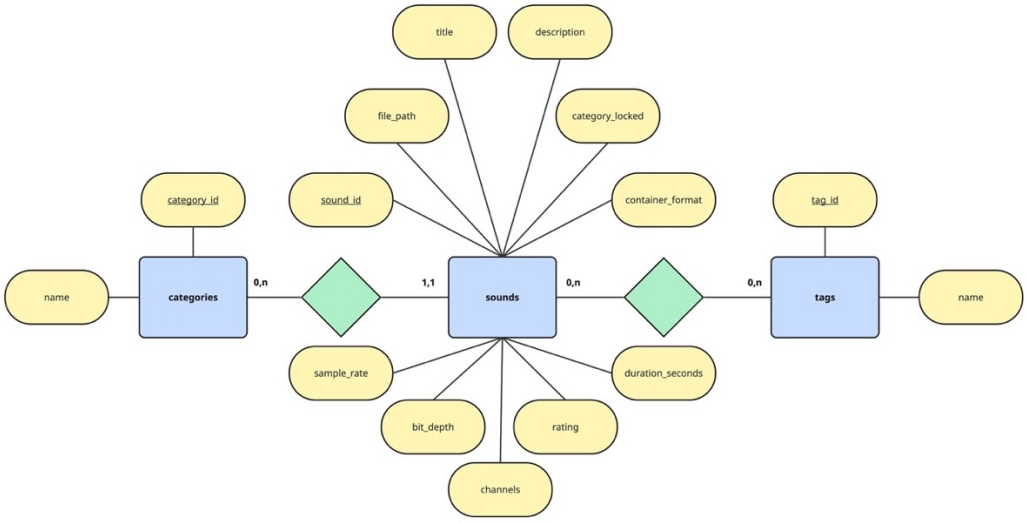

Abbildung 7: Logisches ER-Modell der Datenbank in Chen-Notation (eigene Darstellung)

Das Schema ist in der dritten Normalform (3NF), es gibt keine redundanten Metadaten und klare Verantwortlichkeiten je Tabelle, so sind beispielsweise Kategorienamen nicht in *sounds* gespeichert. Die Normalisierung ist essenziell für die Pflege, die Erweiterung und der zuverlässigen Suche in der Datenbank.
Mögliche Erweiterungen umfassen zusätzliche Metadatenfelder, neue Entitäten und Bewertungssysteme, dafür muss die bestehende Struktur nicht mehr grundlegend verändert werden. Das Schema ist bewusst offen gestaltet und entspricht damit dem Prototyp-Character. Der Fokus liegt auf Praxistauglichkeit und Verständlichkeit, auf komplexe Vererbungsstrukturen oder optionale Zusatzentitäten wurde bewusst verzichtet.

4.6 Wahl des Datenbankmanagementsystems

Die initiale Umsetzung der Datenbank erfolgte mit PostgreSQL. Die Gründe hierfür waren vorhandene Vorkenntnisse, sowie die gute Eignung für die Anforderung der Anwendung, durch mögliche relationale Modelle, einheitliche Datenhaltung und effiziente Abfragen. Der Fokus der frühen Entwicklungsphase lag in erster Linie auf dem Schema-Design und der Abfrage-Logik.

Während des Projektverlaufs wurde ersichtlich, dass die Anwendung zunächst als lokaler Prototyp, Einzelplatz-Anwendung und ohne Mehrbenutzerbetrieb konzipiert wird. Daher ergibt sich kein Bedarf für eine Client-Server-Architektur, eine Benutzerverwaltung oder parallele Zugriffe. Zudem entsteht durch die Nutzung von PostgreSQL ein zusätzlicher, jedoch nicht nötiger Aufwand durch Konfigurationen und laufende Serverprozesse.
Stattdessen sollte der Prototyp einfach in Betrieb genommen werden und möglichst portabel sein. Priorisiert wurde entsprechend die reine Funktionalität, nicht die Infrastruktur.

Als Konsequenz wurde das System auf SQLite umgeschrieben. Durch den Wechsel ist keine separate Server-Installation mehr notwendig und die Datenbank liegt als einzelne Datei vor, wodurch die Portabilität stark erhöht wird, da eine einfache Verteilung mit der Anwendung möglich ist.
Die logische Struktur blieb dadurch unverändert, es kam lediglich zu Anpassungen des SQL-Dialekts und einzelner Datentypen. Es kam zu keinem Verlust an Funktionalität und das Schema blieb weiterhin normalisiert, erweiterbar und konsistent.

Der Wechsel stellt also keine Korrektur dar, sondern eine Anpassung an die realen Anforderungen des Prototypen. Diese Entscheidung erhöhte die Praxistauglichkeit und machte die Zugänglichkeit deutlich einfacher und erhöhte somit die Reproduzierbarkeit, da SQLite sich schlicht besser für prototypische Anwendungen und lokale Medienverwaltung eignet.
Dennoch kann die Verwendung eines PostgreSQL-Servers weiterhin sinnvoll sein, wenn das System durch Mehrbenutzer-Systeme, verteilte Anwendungen oder große Datenmengen erweitert werden soll. Dies ist für den vorliegenden Anwendungsfall jedoch funktional nicht erforderlich.

Entsprechend beziehen sich alle Implementierungsdetails auf die SQLite-Umsetzung.

4.7 Implementierung der Datenbank

Ziel der Implementierung der Datenbank war die technische Umsetzung des zuvor definierten Datenbankschemas und somit die Erstellung einer lauffähigen, lokalen Datenbank, die als Grundlage

für Import-Skripte, GUI-Zugriffe und Metadatenpflege dient und den Fokus auf Stabilität, Einfachheit und Nachvollziehbarkeit legt.

In der Frühphase der Arbeit wurde, wie im letzten Kapitel erwähnt, die Datenbank mit PostgreSQL über pgAdmin als Administrationswerkzeug erstellt.
Durch den Wechsel auf SQLite wird die Datenbank programmatisch über ein Python-Skript erstellt, es erfolgte keine manuelle Erstellung über grafische Tools. Die Initialisierung wurde direkt in der Entwicklungsumgebung Visual Studio Code umgesetzt, wo auch der Anwendungscode geschrieben wurde, wodurch eine enge, vollständige und leicht nachvollziehbare Struktur entstand.

Ein zentrales Entwurfsziel war, die Datenbank jederzeit vollständig neu aufsetzen zu können. Das Initialisierungsskript ist entsprechend voll restartfähig, es löscht zu Beginn alle Tabellen und Beziehungen, um das Schema vollständig neu aufsetzen zu können. Das ist eine wichtige Entscheidung für die Entwicklung, Tests und saubere Ausgangszustände. Diese Restartfähigkeit bezieht sich dabei klar auf den Neubau der Struktur, nicht auf den Erhalt von Daten, bestehende Einträge werden beim erneuten Ausführen des Skriptes bewusst verworfen. Daher ist die klare Trennung zwischen Initialisierung und dem produktiven Betrieb essenziell.

Die Initialisierung kann über das dedizierte Python-Skript `create_sqlite_db.py` erfolgen. Hier wird die SQLite-Datenbank `soundlibrary.db` erzeugt, wobei die Definition des vollständigen Schemas sowie das Einfügen notwendiger Initialdaten inbegriffen ist. Die Datenbank wird im Zielordner `database` abgelegt, falls der Ordner noch nicht existiert wird er automatisch angelegt. Dadurch ist das Projekt ohne weitere manuelle Vorbereitung lauffähig und gleichzeitig existieren keine externen Abhängigkeiten und keine globalen Pfade sind notwendig. Optional kann auch die bestehende Datenbankdatei vor der Neuerstellung gelöscht werden, was sich in der Entwicklungsphase als hilfreich erwies.

Zusätzlich zu `create_sqlite_db.py` stellt die Zugriffsschicht in `db.py` zur Laufzeit über `ensure_db_exists()` sicher, dass bei Bedarf eine Datenbank angelegt, bzw. bereitgestellt wird.

Nach dem Anlegen der Datenbankdatei wird eine Verbindung zur SQLite-Datenbank aufgebaut. Essenziell für die referenzielle Integrität der Datenbank ist es, explizit die Durchsetzung von Foreign-Key-Constraints zu aktivieren, da diese Funktion in SQLite nicht standardmäßig aktiv ist (SQLite Consortium, 2024). Das wird über `PRAGMA foreign_keys = ON` erreicht, wobei diese Einstellung pro Datenbankverbindung aktiviert wird.

Ohne die Funktion würden Fremdschlüsseldefinitionen nur syntaktisch existieren, konsistente Beziehungen zwischen den Entitäten nicht möglich sein und die `ON DELETE`-Regeln nicht fehlerfrei möglich sein.

Die Erstellung des Schemas erfolgt innerhalb einer expliziten Transaktion, dadurch wird sichergestellt, dass das Schema entweder vollständig angelegt wird oder bei Fehlern kein inkonsistenter Zustand erreicht wird. Entsprechende Fehler würden so schon frühzeitig in der Entwicklungsphase erkannt werden.

Im Anschluss wird das tatsächliche Schema angelegt. Um die Restartfähigkeit zu gewährleisten, werden zunächst bestehende Tabellen vor Neuerstellung in folgender Reihenfolge gelöscht:
sound_tags → tags → sounds → categories.
Diese Reihenfolge ist invers zu der, der Erstellung der Tabellen und essenziell, um Foreign-Key-Konflikte zu vermeiden.

Als Initialdaten wird ausschließlich die Kategorie `UNCATEGORIZED` in die Datenbank eingefügt. Diese dient als Fallback-Kategorie für Audiodateien, die keinem spezifischen Unterordner zugeordnet werden können, alle weiteren Kategorien werden beim Scan-Vorgang aus der Ordnerstruktur des Sound-Verzeichnisses generiert. So ist eine flexible Anpassung der Kategorienstruktur ohne manuelle Datenbankeingriffe möglich.

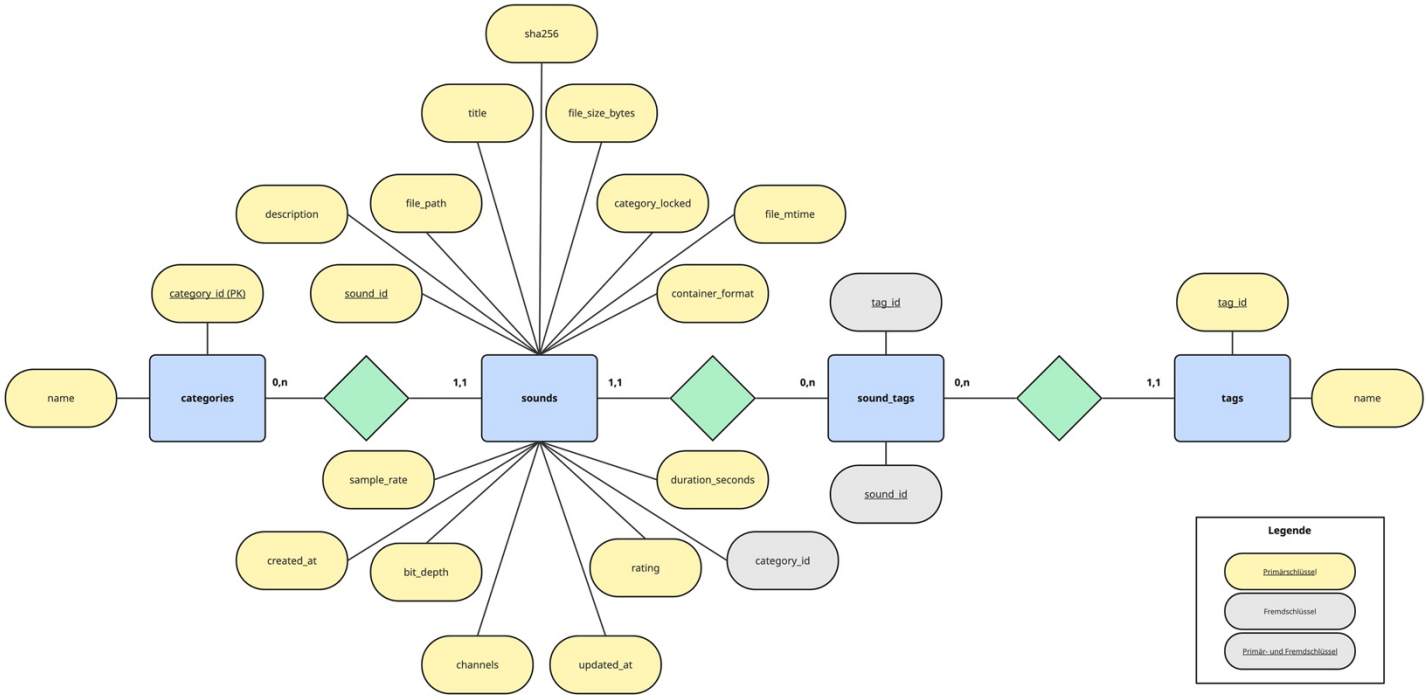

Abbildung 8: Physisches ER-Modell der Datenbank in Chen-Notation (eigene Darstellung)

4.7.1 Entität `categories`

Die erste erstellte Tabelle ist dementsprechend `categories`. Diese stellt eine eigene Entität dar und dient der zentralen Kategorisierung und zur Vermeidung redundanter Kategorienamen.
Das Attribut `category_id` ist der Primärschlüssel und vergibt durch die Eigenschaft `AUTOINCREMENT` automatisch jedem neuen Datensatz einen neuen, fortlaufenden Zahlenwert. Zudem existiert das Pflichtattribut `name`, in welchem der Name des entsprechenden Primary-Keys als Text festgehalten wird. Dadurch existiert der Name einer Kategorie später in der Tabelle `sounds` nicht als Textfeld.

4.7.2 Entität `sounds`

Als nächstes wird die Tabelle `sounds` definiert. Sie ist die zentrale Entität des Schemas und jeder Eintrag repräsentiert genau eine Audiodatei. Zudem verbindet sie Dateisystem, Metadaten und organisatorische Informationen.

Das Attribut `sound_id` fungiert als Primärschlüssel und vergibt die entsprechenden Werte automatisch über `AUTOINCREMENT`.
In `file_path` wird der Dateipfad gespeichert, entsprechend ist dieser Wert zur eindeutigen Identifikation `UNIQUE`. Dieses Attribut dient als direkte Referenz für Import, GUI und Playback.

In `sha256` wird der SHA-256-Hashwert der Audiodatei gespeichert. Dieser wird beim Import berechnet und ermöglicht eine inhaltsbasierte Identifikation unabhängig vom Dateinamen sowie die Erkennung von Duplikaten und verschobenen Dateien.

Nun kommen Attribute für die semantischen Metadaten zum Einsatz. Diese beschreiben die Audiodateien inhaltlich und sollen Such- und Filterfunktionen ermöglichen bzw. verbessern.
Im optionalen Attribut `title` wird ein kurzer, sprechender Name für den Sound vergeben, welcher unabhängig vom Dateinamen angelegt werden kann. In der aktuellen Implementierung wird dieses Attribut jedoch nicht aktiv genutzt, da der Dateiname als ausreichende Bezeichnung bewertet wurde.
Optional kann in `description` eine freitextliche Beschreibung der Audiodatei angelegt werden, um den Inhalt detailliert einzuordnen. Auch diese wird in die Suchfunktion eingegliedert.
Durch diese Attribute werden Dateiname und semantische Beschreibung voneinander getrennt. Die Attribute sind bewusst offengehalten, nicht jede Datei muss manuell beschrieben werden, so wird der Import- bzw. Scan-Prozess einfach gehalten und eine unterschiedliche Detailtiefe je Audio bleibt möglich.
Die nächsten Attribute dienen zur inhaltlichen Grundstrukturierung der Audios und ermöglichen eine sichere und konsistente Kategorisierung über eine Referenztabelle. Außerdem wird gesteuert, ob automatische Prozesse eine Kategorie überschreiben dürfen.
Das Pflichtattribut `category_id` ist ein Fremdschlüssel, der auf `category_id` in der Tabelle *categories* zeigt und gewährleistet, dass jedes Audio genau einer Kategorie zugeordnet ist.
Durch `ON DELETE RESTRICT` wird sichergestellt, dass Kategorien, die in `sounds` referenziert werden, nicht gelöscht werden dürfen.
Das Attribut `category_locked` steuert, ob eine Kategorie eines Audios durch automatische Prozesse verändert werden darf. Sein Datentyp ist `INTEGER`, wird allerdings als boolescher Wert verwendet. Bei `category_locked = 0` darf die Kategorie automatisch angepasst werden. Bei `category_locked = 1` gilt die Kategorie als manuell festgelegt. Die initiale Kategorie wird beim Import aus der Ordnerstruktur abgeleitet, falls keine passende Kategorie gefunden wird, wird die Default-Kategorie `UNCATEGORIZED` vergeben.
So entsteht eine Trennung zwischen automatischer Kategorisierung und manueller Kuration, gleichzeitig wird unbeabsichtigtes Überschreiben verhindert. Außerdem wird verhindert, dass Audios umklassifiziert in die Datenbank integriert werden.

Als nächstes folgen Attribute, um die technischen Metadaten der Audiodateien abzuspeichern. Diese gelten als Grundlage für Qualitätsbewertung, Filterfunktionen und DAW-Integration. Eine automatische Erfassung beim Import ist möglich.

Zunächst wird das Attribut `container_format` definiert, welches das Containerformat der Audiodateien übernimmt. Es ist durch den Default-Wert ‚WAV' aktuell auf WAV-Files ausgelegt, bereitet aber spätere Erweiterung für weitere Formate, z.B. AIFF oder FLAC vor.
Im Attribut `sample_rate` wird die Abtastrate der Audiodatei als Integer-Wert hinterlegt, was relevant für die technische Vergleichbarkeit, Filter in der GUI und DAW-Kompatibilität ist.
Analog dazu speichert `bit_depth` die Bittiefe der Audiodatei, um Aussagen über den Dynamikumfang und die Aufnahmequalität zu treffen.
In `channels` wird die Anzahl der Audiokanäle hinterlegt, beispielsweise 1 für Mono, 2 für Stereo.
Das Attribut `duration_seconds` speichert die Länge der Audiodateien in Sekunden. Durch den Datentyp `REAL` wird dabei ein kontinuierlicher Wert anstatt einer Ganzzahl verwendet.
Diese technischen Daten sind bewusst als optionale Felder gehalten, um auch Imports ohne eine vollständige Analyse durchführen zu können, was eine flexible Nutzung auch in frühen Projektphasen ermöglichte.

Im nächsten Schritt werden Dateieigenschaften unabhängig vom Audio-Inhalt abgebildet. Das unterstützt Änderungs- und Aktualitätsprüfungen beim Scan, bzw. Import. Außerdem wird unnötiges Lesen von Dateien beim GUI-Zugriff minimiert.
In `file_size_bytes` wird die Dateigröße in Bytes hinterlegt. Neben der Möglichkeit Dateien später zu sortieren oder zu filtern ist dies ein Indikator für Änderungen an der Datei.
Das Attribut `file_mtime` gibt den Zeitpunkt der letzten Dateiänderung an, die Speicherung erfolgt in einem ISO-String. Durch diesen Wert kann erkannt werden, ob die Datei seit dem letzten Import geändert wurde. Als Datentyp wurde `TEXT` gewählt, da SQLite keinen nativen `datetime`-Typ besitzt (SQLite Consortium, 2025). So ist auch ein plattformneutraler Vergleich und Debugging möglich.
Durch die Speicherung dieser Werte in der Datenbank, anstatt diese bei jeder GUI-Abfrage aus dem Dateisystem zu lesen entsteht sowohl ein Performance-Vorteil als auch eine konsistente Abfragebasis. Es kann zudem geprüft werden ob Dateien neuer sind als der letzte Datenbankstand und spätere Erweiterungen in Richtung Synchronisation sind möglich.

Das optionale Attribut `rating` ermöglicht die manuelle Bewertung der Nutzer innerhalb der GUI von 1 bis 5. Durch die `CHECK`-Constraint wird sichergestellt, dass nur Werte in dem entsprechenden Bereich gültig sind. Die Implementierung als optionales Attribut wurde getroffen, damit nicht jedes Audio bewertet werden muss und der Import-Prozess schlank bleibt. Die einfache Skala von 1 bis 5 ist intuitiv bedienbar und ausreichend differenziert für den Prototyp-Zweck.

Abschließend werden noch zwei Pflichtattribute zur Nachvollziehbarkeit von Änderungen an Metadaten implementiert. Dies unterstützt Filter- und Sortierfunktionen und dient als Grundlage für Debugging, Auswertungen und spätere Synchronisationslogik.
In `created_at` wird der Zeitpunkt der erstmaligen Erstellung eines Datenbankeintrags festgehalten. Dies geschieht durch den `DEFAULT`-Wert `CURRENT_TIMESTAMP`, der Wert ist unabhängig vom Erstellungsdatum der Audiodatei selbst.
Im Attribut `updated_at` wird der Zeitpunkt der letzten Änderung des Datensatzes hinterlegt. Beim Anlegen wird `CURRENT_TIMESTAMP` als `DEFAULT` gesetzt, bei Änderungen wird `updated_at` in den Update-Operationen explizit auf `CURRENT_TIMESTAMP` aktualisiert.
Durch diese Attribute kann eine Sortierung nach zuletzt bearbeiteten Audios vorgenommen werden, kürzlich geänderte Einträge können identifiziert werden und inkrementelle Scan-Strategien werden unterstützt.
Analog zu `file_mtime` werden die Attribute aufgrund der SQLite-Spezifikationen als `TEXT` gespeichert.

4.7.3 Entität `tags`

Die Entität `tags` ermöglicht die flexible Zuordnung von Audios zu Tags, hierbei werden mehrere Tags je Audio unterstützt. Der Fokus liegt auf konsistenten Tag-Namen; Dubletten oder Tippfehler-Varianten werden nicht unterstützt.

Das Attribut `tag_id` fungiert analog zu den anderen Entitäten als Primärschlüssel mit `AUTOINCREMENT`.
In `name` wird der Tag-Name vergeben, dieser muss eindeutig sein. Dadurch werden doppelte Tags verhindert und stabile Verknüpfungen ermöglicht. Die Verwendung eines künstlichen Schlüssels entkoppelt die Identität von möglicher Umbenennung und erleichtert spätere Erweiterungen, wie beispielsweise Tag-Gruppen.

4.7.4 Relationstabelle `sound_tags`

Die Relationstabelle `sound_tags` dient zur Abbildung der viele-zu-viele-Beziehung zwischen den Entitäten *sounds* und *tags*. Dadurch entsteht eine klare Trennung von Entitäten und Beziehungen gemäß des relationalen Modells.
Die Tabelle besteht aus den Pflichtattributen `sound_id` und `tag_id`, welche beide einen Integer-Wert abbilden und gemeinsam einen zusammengesetzten Primärschlüssel bilden. So wird eine doppelte Zuordnung desselben Tags zu einem Sound verhindert, gleichzeitig ist keine zusätzliche künstliche ID notwendig. Beide Attribute sind mit `ON DELETE CASCADE` versehen, sodass beim

Löschen eines Sounds alle zugehörigen Tag-Zuordnungen automatisch entfernt und beim Löschen eines Tags alle betroffenen Zuordnungen entfernt werden. Da es sich bei `sound_tags` um eine reine Beziehungstabelle handelt, sind keine weiteren Attribute notwendig, es bleibt bei einer klaren und verständlichen Struktur.

4.7.5 Beziehungen zwischen den Entitäten

Bisher wurden die Entitäten isoliert betrachtet, die Vollständigkeit des Schemas ergibt sich allerdings erst durch deren Verknüpfung. Die Beziehungen zwischen den Entitäten definieren die entsprechenden Datenabhängigkeiten und stellen referenzielle Integrität sicher. Dies ist die Grundlage für konsistente Abfragen und Joins.

Die Beziehung zwischen `categories` und *`sounds`* ist vom Typ `1:n`: Eine Kategorie kann mehreren Sounds zugeordnet sein, jeder Sound gehört in eine Kategorie. Die Umsetzung erfolgt über den Fremdschlüssel `category_id` in `sounds`. Durch `ON DELETE RESTRICT` wird sichergestellt, dass Kategorien nicht gelöscht werden, solange sie noch referenziert werden. Dies schützt vor inkonsistenten Datenzuständen.

Der Beziehungstyp zwischen `sounds` und `tags` ist `n:m`: Ein Sound kann mehreren Tags zugeordnet sein, ein Tag kann ebenso mehreren Sounds zugeordnet sein. Die Umsetzung auf Datenbankebene erfolgt durch die explizite Zuordnungstabelle `sound_tags`.

Zwischen `sounds` und `sound_tags` besteht eine `1:n`-Beziehung: Ein Sound kann mehreren Tag-Zuordnungen zugeordnet sein, jede Zuordnung in `sound_tags` gehört zu einem Sound.
Analog dazu besteht zwischen `tags` und `sound_tags` eine `1:n`-Beziehung: Ein Tag kann mehreren Sound-Zuordnungen zugeordnet sein, jede Zuordnung in `sound_tags` verweist genau auf einen Tag.

4.7.6 Indizes, Constraints und Integritätsregeln

Im Folgenden werden verschiedene Gestaltungsentscheidungen in Hinblick auf Indizes, Constraints und Integritätsregeln erläutert. Diese dienen zur Sicherstellung von Datenqualität, der Durchsetzung konsistenter Beziehungen zwischen den Entitäten, eine Verbesserung der Abfrage-Performance, sowie der Entlastung der Anwendung durch eine Validierung auf Datenbankebene.

Indizes sind zusätzliche Datenstrukturen zur Beschleunigung von Abfragen. Diese sind besonders relevant bei Filteroperationen, Join-Abfragen und Sortiervorgängen. In der Sound-Library kommt es

zu häufigen Lesezugriffen und interaktiven GUI-Abfragen. Indizes wurden gezielt dort eingesetzt, wo sie einen praktischen Nutzen bringen, ohne unnötige Speicher- und Schreibkosten zu verursachen.

Der Index `idx_sounds_category_id` ist auf dem Attribut `category_id` der Entität *sounds* definiert. Durch ihn werden `WHERE category_id = x`-Anfragen beschleunigt und Full-Table-Scans bei wachsenden Datenbestand reduziert. Er wurde implementiert, da Filterungen nach Kategorien wahrscheinlich zu den häufigsten Filter-Operationen gehören.

In der Entität `sounds` wurde der Index `idx_sounds_sha256` auf dem Attribut `sha256` definiert. Durch ihn wird eine schnelle Überprüfung möglich, ob eine Audiodatei bereits in der Datenbank vorhanden ist. Es entsteht eine robuste Erkennung von Duplikaten unabhängig vom Dateinamen, mehrere Datenbankeinträge für identische Daten werden vermieden. Außerdem werden effiziente `WHERE sha256 = x`-Abfragen unterstützt. Auch wenn es sich dabei um ein optionales Attribut handelt, können entsprechende Abfragen bei Nutzung schnell performancekritisch werden.
Der Index `idx_sounds_updated_at` ist auf dem Attribut `updated_at` der Entität `sounds` definiert und wird zur GUI-Darstellung und der Sortierung von Sound-Einträgen genutzt. Er ermöglicht eine effiziente Sortierung nach den zuletzt geänderten Datensätzen, beispielsweise wenn nach zuletzt bearbeiteten Audios oder neu importierter, bzw. aktualisierter Daten sortiert werden soll. Es werden aufwendige Sortieroperationen vermieden und die Reaktionszeit bei interaktiven Nutzung verbessert. Der Index wurde implementiert, da Zeitstempel häufig gelesen, aber selten geschrieben werden, so verursacht er nur geringe Zusatzkosten bei hohem Nutzen.

Auf dem Attribut `sound_id` der Entität `sound_tags` wurde nun der Index `idx_sound_tags_sound_id` zur Auflösung der Sound-zu-Tag-Beziehungen definiert. Er dient zur effizienten Ermittlung aller Tags eines bestimmten Sounds. In einer GUI-Abfrage zur Anzeige der Tags eines Sounds entsteht eine komplexe Join-Operation (`sounds → sound_tags → tags`). Durch den Index wird diese beschleunigt und verhindert wiederholte vollständige Scans der Zuordnungstabelle.

Analog dazu wurde der Index `idx_sound_tags_tag_id` auf `tag_id` in `sound_tags` definiert. Dieser dient zur effizienten Ermittlung aller Sounds, die mit einem bestimmten Tag verknüpft sind. Typischerweise wird es in der Sound-Library häufig zur Suche nach Sounds mit einem bestimmten Tag oder zur Kombination mehrerer Tags über Join-Abfragen kommen. Durch den Index werden

diese Operationen direkt unterstützt. Durch ihn und `idx_sound_tags_sound_id` werden effiziente Abfragen in beide Richtungen der `sounds`-zu-`tags`-Beziehung ermöglicht.

Allgemein wurden Indizes gezielt anhand realer Nutzungsszenarien definiert, mit dem Fokus auf häufigen Leseoperationen, GUI-Interaktionen und Import-/Scan-Prozesse. Pauschale oder vorsorgliche Indexierung wurde bewusst vermieden.

Durch die eingesetzten Constraints und Integritätsregeln wird die Datenqualität innerhalb der Datenbank sichergestellt und fachliche Regeln auf Datenbankebene durchgesetzt. Dies entlastet den Anwendungscode von grundlegender Validierungslogik.

Die Primärschlüssel dienen zur eindeutigen Identifikation aller Datensätze und sind stabile Referenzen, unabhängig von fachlichen Attributen. Die einzige Ausnahme stellt der zusammengesetzte Primärschlüssel der Zuordnungstabelle `sound_tags` dar.
Die Fremdschlüssel dagegen erzwingen gültige Referenzen zwischen den Entitäten und stellen somit die Grundlage für referenzielle Integrität dar.
Mit `RESTRICT` werden fachlich zentrale Entitäten geschützt, durch `CASCADE` werden abhängige Zuordnungen automatisch bereinigt.
Die verwendeten `UNIQUE`-Constraints dienen zur Vermeidung fachlicher Redundanzen sowie zur Absicherung eindeutig zu interpretierenden Attribute, beispielsweise Dateipfade.
Durch die `NOT NULL`-Constraints werden Pflichtinformationen definiert und unvollständige Datensätze vermieden.
`CHECK`-Constraints validieren eingeschränkte Wertebereiche und stellen fachlich sinnvolle und erlaubte Eingaben sicher.
`DEFAULT`-Werte ermöglichen eine konsistente Initialisierung neuer Datensätze und vereinfachen Insert-Operationen.
Die Constraints ergänzen das Schema sinnvoll und verlagern grundlegende Validierungslogik auf die Datenbankebene.

4.8 Anbindung der Datenbank an die Anwendung

Die Datenbank wird in diesem Modell nicht direkt aus der GUI oder Import-Skripten angesprochen. Stattdessen erfolgt der Zugriff über die zentrale Python-Modellstruktur `db.py`. Das Ziel ist eine Kapselung aller Datenbankoperationen, die Schaffung eines einheitlichen Zugriffpunkts und die Reduktion von Redundanzen im Anwendungscode. Außerdem erfolgt eine strikte Trennung von Datenhaltung, Anwendungslogik und Benutzeroberfläche.

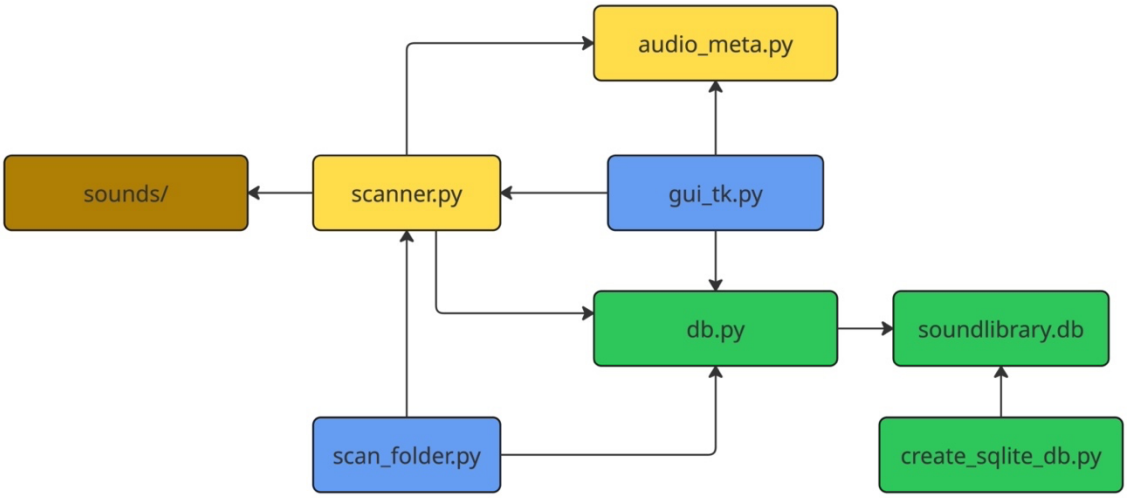

Abbildung 9: Technische Systemarchitektur des Prototyps mit Modulabhängigkeiten zwischen GUI, Scan-Logik, Datenbankzugriffsschicht und Dateisystem (eigene Darstellung)

Da jede Datenbankoperation eine eigene Verbindung nutzt, wird `PRAGMA foreign_keys = ON` zentral beim Verbindungsaufbau genutzt, sodass die Durchsetzung von Fremdschlüssel-Constraints sichergestellt wird.
Durch die Nutzung von `row_factory` erfolgt der Zugriff auf Ergebnisse über Spaltennamen und eine bessere Lesbarkeit im Code wird ermöglicht. Es wird konsistentes Verhalten, eine geringere Fehleranfälligkeit und eine einfache Wiederverwendbarkeit sichergestellt.

Da die Dateipfade relativ zum Projektverzeichnis gespeichert werden sollen, ist eine Umsetzung über Hilfsfunktionen notwendig. So wird die Portabilität der Datenbank, unabhängig von absoluten Systempfaden ermöglicht.

Die Dateien werden über eine kombinierte Insert-/Update-Logik verarbeitet. Hierzu wird ein `ON CONFLICT`-Mechanismus mit dem eindeutigen Schlüssel `file_path` verwendet. Bei neuen Dateien entsteht ein neuer Eintrag, bei bestehenden Dateien werden relevante Metadaten aktualisiert. Es entsteht ein *idempotenter* Import, bei wiederholten Scans entstehen keine Dubletten, eine robuste Aktualisierung anhand der gefundenen Dateien ist möglich.

Des Weiteren gibt es separate Funktionen für die Aktualisierung von Titel und Beschreibung, dem Ändern der Kategorie und dem Setzen einer Bewertung, jede Funktion erfüllt genau eine Aufgabe. Durch die automatische Aktualisierung des `updated_at`-Zeitstempels sind Änderungen nachvollziehbar und der Datenstand bleibt synchron.

Tags werden bei Bedarf angelegt. Die Normalisierung der Tag-Namen entfernt Duplikate durch einen case-insensitiven Vergleich, wobei die ursprüngliche Schreibweise des Tags erhalten bleibt.
Die Zuordnung erfolgt idempotent, es gibt keine doppelten Einträge in `sound_tags`. Wenn Tags entfernt werden sollen, werden entsprechende Zuordnungen gezielt gelöscht, was eine robuste Tag-Verwaltung ermöglicht, außerdem kommt es zu keinen Inkonsistenzen zwischen GUI und Datenbank.

Diese Zugriffschicht stellt die Basisfunktionen bereit, welche von Import- und Scan-Skripten genutzt werden. Schreib- und Updateoperationen der GUI erfolgen ausschließlich über diese Schnittstelle. Leseabfragen für die Darstellung werden bewusst in der GUI-Komponente selbst umgesetzt, da sie eng mit GUI-spezifischen Parametern wie Filterlogik, Sortierung und Ergebnismenge verzahnt sind.

4.8.1 Query-Logik

Um eine Trennung von SQL-Abfragen zur Anwendungslogik sicherzustellen, werden Lese- und Suchabfragen zentral in der GUI-Komponente mit der Funktion `fetch_sounds` umgesetzt. Hier werden die Textsuche über mehrere Metadatenfelder, die Filterung nach Kategorien und Tags, sowie die Sortierung der Ergebnislisten umgesetzt. Dabei werden parametrisierte Abfragen zur Übergabe von Such- und Filterwerten verwendet, die Durchführung der Filter- und Suchlogik erfolgt direkt auf Datenbankebene.

Während in `db.py` Schreib- und Update-Operationen durchgeführt werden, werden die für die Darstellung benötigten `SELECT`-Abfragen in der GUI-Funktion `fetch_sounds` vorgenommen. Durch diese Trennung entsteht eine bessere Möglichkeit zur Wartung, eine einfachere Erweiterbarkeit der Suchlogik, sowie ein übersichtlicherer GUI-Code.

4.9 Import- und Scan-Skript

Das Import-/Scan-Skript überführt automatisch die Audiodateien aus dem Dateisystem in die Datenbank und stellt den initialen Aufbau der Sound-Library dar, außerdem erfolgt hier die spätere Aktualisierung bestehender Einträge.
Das Skript bildet die Schnittstelle zwischen dem Dateisystem mit den Audiodaten und der Datenbank zur Metadatenverwaltung.
Dadurch, dass ohne Import keine nutzbaren Daten vorhanden wären und die Grundlage für GUI-Funktionen nicht vorhanden wäre, nimmt das Skript eine zentrale Rolle im Gesamtsystem ein.
Auch die Import-Logik wird bewusst von der GUI getrennt, um Wiederverwendbarkeit und eine bessere Wartbarkeit zu ermöglichen.

Die Anforderungen an den Scan-Prozess wurden im Kapitel 3.5 beschrieben. Die folgende Implementierung setzt diese Anforderungen um und dokumentiert die dabei getroffenen Abwägungen.
Die Grundidee besteht darin, dass ein definiertes Basisverzeichnis rekursiv durchsucht wird, wobei auch alle relevanten Unterordner berücksichtigt werden. Jede gefundene Audiodatei wird einzeln verarbeitet, das Dateisystem dient dabei als Quelle der Mediendateien. In der Datenbank werden neue und geänderte Daten ergänzt oder aktualisiert. Gelöschte Dateien werden beim Scan-Vorgang automatisch aus der Datenbank entfernt, sofern der Parameter `cleanup_missing` aktiv ist.

Die Abgrenzung zur Zugriffsschicht besteht darin, dass das Import-Skript den Ablauf steuert und entscheidet, wann was passiert. Die Zugriffsschicht dagegen kapselt wie Daten gespeichert werden und führt konkrete SQL-Operationen aus. Durch diese klaren Verantwortlichkeiten existieren keine SQL-Statements im Import-Skript und keine Dateisystemlogik in der Zugriffsschicht.

4.9.1 Ablauf des Scan-Prozesses

In diesem Unterkapitel erfolgt der detaillierte Ablauf des Scan-Prozesses.

Zunächst wird ein festes Basisverzeichnis für die Sound-Library definiert. Das Verzeichnis repräsentiert die physische Struktur der Audiodateien, entsprechende Unterordner werden rekursiv berücksichtigt. Es wird auf der Annahme gearbeitet, dass sich jede relevante Audiodatei innerhalb dieses Verzeichnisbaums befindet. So entsteht eine klare Trennung zwischen verwalteten und nicht verwalteten Dateien, zudem ist eine einfache Erweiterung durch das Hinzufügen neuer Dateien im Dateisystem möglich.

Bevor die eigentliche Verarbeitung der Audiodateien beginnt, erfolgt eine Synchronisation der Kategorien mit der Ordnerstruktur des Dateisystems. Dabei werden alle im Sound-Verzeichnis vorhandenen Unterordner erster Ebene als Kategorien erkannt und in der Datenbank angelegt, sofern sie noch nicht existieren. Dies erlaubt eine dynamische Erweiterung der Kategorienstruktur durch einfaches Hinzufügen neuer Ordner, ohne dass manuelle Eingriffe in die Datenbank erforderlich sind. Nach Abschluss des Scans erfolgt eine erneute Synchronisation, bei der ungenutzte Kategorien, also solche, denen keine Audiodateien mehr zugeordnet sind, automatisch aus der Datenbank entfernt werden. So wird verhindert, dass verwaiste Kategorien angesammelt werden und die Datenbank wird synchron mit der tatsächlichen Nutzung gehalten. Die automatische

Kategorie-Verwaltung trennt klar zwischen der physischen Ordnerstruktur als Datenquelle und der Datenbank als Verwaltungsschicht, wobei beide Seiten vollständig synchron gehalten werden.

Nach der Definition des Verzeichnisses erfolgt eine rekursive Iteration über alle Unterordner, jede gefundene Datei wird nun einzeln verarbeitet. Es werden keine Vorabannahmen über die Anzahl der Dateien oder die Ordnerstruktur getroffen. Das Ziel dieses Schrittes ist die vollständige Erfassung aller vorhandenen Audiodateien.

Im nächsten Schritt werden die relevanten Audiodateien herausgefiltert. Wie in Kapitel 3.3 konzeptionell offengelassen, wurde im Implementierungsprozess entschieden, ausschließlich WAV-Dateien zu berücksichtigen. Diese Entscheidung ergab sich, da die Python-Standardbibliothek mit dem `wave`−Modul eine direkt und unkomplizierte WAV-Unterstützung bietet. Die Einbindung weiterer Formate wie AIFF oder FLAC hätte zusätzliche externe Bibliotheken und eine erweiterte Parsing-Logik erfordert, was den Rahmen des Prototyps gesprengt hätte. Irrelevante Dateien, beispielsweise andere Datentypen aber auch temporäre Dateien oder Metadaten-Dateien, werden entsprechend ausgeschlossen. So wird unnötige Verarbeitung vermieden und es entsteht ein klar definierter Importumfang.

Nun werden grundlegende Dateisysteminformationen der relevanten Dateien erfasst, wie die Dateigröße und der letzte Änderungszeitpunkt. Zudem wird der relative Dateipfad zum Projektverzeichnis ermittelt um die eindeutige Identifikation, spätere Änderungsdetektion und allgemein die Portabilität der Datenbank zu ermöglichen.
Das Auslesen der technischen Metadaten wird durch die Datei `audio_meta.py` ermöglicht, welche dazu vom Scan-Skript aufgerufen wird. Diese Datei liest die technischen WAV-Eigenschaften automatisch aus, unabhängig von Dateisystem- und Datenbanklogik.
Dieses Auslesen der technischen Eigenschaften beinhaltet die Abtastrate, Bittiefe, Kanalanzahl, die Dauer sowie einen SHA-256-Hashwert zur Datei-Identifikation.

Nun werden alle ermittelten Informationen an die Zugriffsschicht übergeben. Über die `UPSERT`-Logik der Zugriffsschicht werden neue Dateien angelegt und bestehende Einträge aktualisiert, wobei technische Metadaten, Dateisysteminformationen und der Zeitstempel (`updated_at`) erneuert werden. Dadurch ist der Scan-Prozess idempotent und ohne unerwünschte Nebeneffekte mehrfach ausführbar.

Werden die Dateien ausschließlich über den Dateipfad identifiziert, können Dateien, die verschoben oder umbenannt werden, nicht mehr als solche identifiziert werden. Das Programm hält die Dateien für gelöscht und findet stattdessen eine vermeintlich neue Datei. Dabei würden alle manuell hinzugefügten Informationen wie Tags oder Beschreibungen verloren gehen und müssten neu angelegt werden. Der beim Einlesen berechnete SHA-256-Hash dient daher als unveränderliche, inhaltsbasierte ID der Datei, die unabhängig von Dateiname und Speicherort ist, solange der Audio-Inhalt gleichbleibt.

Beim Scan-Vorgang wird der Hash-Wert jeder Datei geprüft. Falls dieser bereits bekannt ist, wird geprüft, ob der bisherige Dateipfad noch gültig ist. Ist das nicht der Fall, interpretiert das System dies als Verschiebung und der bestehende Datenbankeintrag zu dieser Datei wird aktualisiert, so bleibt die interne `sound_id` bestehen. So bleiben Tags und Kategorien auch bei Neustrukturierung der Sound-Library erhalten, zudem werden Duplikate verhindert.

Der Umgang mit manuellen Metadaten muss differenziert betrachtet werden. Die automatischen Prozesse überschreiben keine manuell erstellten Titel, Beschreibungen oder gesperrten Kategorien. So entsteht eine Trennung zwischen automatisch ermittelten Informationen und manuell kuratierten Metadaten. Es wird sichergestellt, dass der Scan Daten ergänzt, die manuelle Benutzerarbeit jedoch nicht zerstört.

Neben dem Hinzufügen und Aktualisieren von Dateien erfolgt auch eine automatische Bereinigung der Datenbank. Beim Scan-Vorgang werden alle Dateipfade mit dem aktuellen Dateisystem abgeglichen und Einträge zu nicht mehr existierenden Dateien entfernt. Dieses Verhalten wird durch den Parameter `cleanup_missing` gesteuert, welcher standardmäßig aktiv ist. Der Scan-Prozess gibt Statistiken über erfolgreich verarbeitete, fehlgeschlagene und gelöschte Einträge zurück, was eine nachvollziehbare Protokollierung ermöglicht. So wird sichergestellt, dass die Datenbank stets den tatsächlichen Bestand des Dateisystems widerspiegelt und keine verwaisten Einträge enthält. Diese automatische Bereinigung ist insbesondere bei größeren Reorganisationen der Sound-Library essenziell.

Nach dem Ausführen des Import-/Scan-Skripts wird die Datenbank vollständig synchronisiert: Neue Dateien sind sofort verfügbar und geänderte Dateien werden aktualisiert, während bestehende Metadaten erhalten bleiben und fehlende Dateien aus der Datenbank entfernt werden.

Das Import-/Scan-Skript bietet optional einen *Watch-Modus*, der das Sound-Verzeichnis kontinuierlich überwacht. Im Watch-Modus („Auto-Scan“) wird der Scan-Vorgang in regelmäßigen Intervallen wiederholt, wobei die Datenbank automatisch mit dem Dateisystem synchronisiert wird. Beim Aktivieren erfolgen die Scans standardmäßig alle 10 Sekunden, wobei die Frequenz konfigurierbar ist. Diese Möglichkeit kann gerade bei aktiven Sessions oder bei häufigen Änderungen im Dateibestand hilfreich sein. Der Modus wird über das `--watch` Flag aktiviert und läuft als Hintergrundprozess bis zur manuellen Bedienung. Der Watch-Modus bleibt dabei stets optional, falls Nutzer auf Hintergrundprozesse der Sound-Library bewusst verzichten möchten.

Der Scan-Prozess steht zusätzlich über das eigenständige Skript `scan_folder.py` zur Verfügung. Dadurch kann der Scan unabhängig von der GUI ausgeführt werden, wodurch Szenarien abgedeckt werden, in denen die Sound-Library selbst nicht gestartet werden soll. Ein praktisches Beispiel wäre eine Sound-Library die mit einem Netzwerkspeicher synchronisiert wird. Hier könnte der Scan nun automatisiert und regelmäßig im Hintergrund ausgeführt werden, ohne dass der Nutzer die Anwendung aktiv öffnen muss. Insbesondere der Watch-Modus lässt sich so als dauerhafter Hintergrundprozess betreiben.

4.10 Implementierung der grafischen Benutzeroberfläche (GUI)

Die GUI stellt die oberste Anwendungsschicht dar, Benutzer interagieren nach dem Setup ausschließlich über die GUI, Datenbank und Import-Skript arbeiten im Hintergrund. Die GUI soll eine visuelle und interaktive Nutzung sowie den Zugriff auf importierte Audiodateien ohne direkte Datenbankkenntnisse ermöglichen. Die Implementierung erfolgt mit `Tkinter`, dem Standard-GUI-Framework der Python-Standardbibliothek. Dieses wurde gewählt, da es plattformübergreifend verfügbar ist, keine zusätzlichen Abhängigkeiten erfordert und sich gut für Desktop-Anwendungen eignet.

Die GUI fokussiert sich auf Funktionalität, Übersichtlichkeit und Praxistauglichkeit. Für den Prototyp wurde auf ein komplexes UI-Design mit visuellen Effekten oder Styling verzichtet. Es dient ausschließlich als Arbeitswerkzeug, Prototyp und funktionaler Nachweis des Gesamtsystems.

Wie beschrieben erfolgen Schreib- und Updateoperationen ausschließlich über die Zugriffsschicht (`db.py`). Leseabfragen für die GUI-Darstellung werden direkt in der GUI-Komponente umgesetzt. Das Import/-Scan-Skript erzeugt und aktualisiert die Daten, während die GUI Metadaten liest, filtert und modifiziert. Auch hier werden also Verantwortlichkeiten klar getrennt, die Datenhaltung erfolgt in der Datenbank, Schreib- und Updateoperationen in der Zugriffsschicht und die Leseabfragen sowie Nutzer-Interaktionen in der GUI.

Die Anforderungen an die GUI wurden in Kapitel 3.6 beschrieben. Die folgenden Abschnitte dokumentieren die konkrete Umsetzung.

4.10.1 Anzeige und Grundlayout der GUI

Durch das Layout soll ein schneller Zugriff auf große Mengen an Audiodateien stattfinden. Hierbei ist eine klare Trennung zwischen der Übersicht, also der Liste der Dateien, der Detailansicht für die jeweiligen Metadaten und Aktionen, wie beispielsweise Buttons oder Controls notwendig. Der Fokus liegt auf einem übersichtlichen Workflow statt visuellem Design.

Die zentrale GUI besteht aus mehreren Bereichen: Die Sound-/Ereignis-Liste, dem Detailbereich für ausgewählte Sounds, eine Suchleiste mit Filterfunktionen und den Bereich zur Playback-Steuerung.

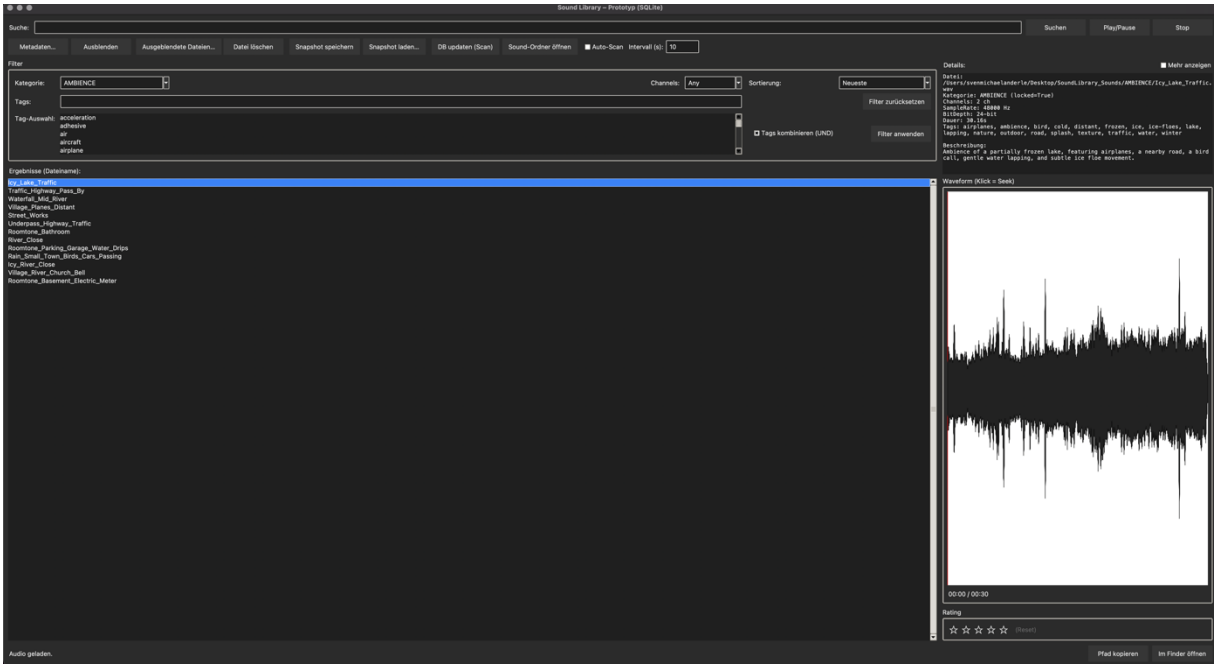

Abbildung 10: Hauptansicht des Prototyps mit aktivem Kategorie-Filter (AMBIENCE), Tag-Auswahl, Suchergebnisliste, Detailbereich und Wellenform (eigener Screenshot)

Die Sound-Liste stellt einen Datensatz pro Zeile dar, die angezeigten Kerninformationen sind Dateipfad oder Dateiname und eine Kurzinfo. Standardmäßig wird diese nach dem Zeitpunkt der letzten Änderung sortiert (`updated_at`). Durch Klicken auf den Listeintrag wird die entsprechende Datei ausgewählt und die Metadaten erscheinen im Detailbereich.
Dieser umfasst typischerweise den Dateipfad, Tags und die Kurzbeschreibung. Durch ein Dropdown-Menü kann die Kategorie gewählt werden, Tags können hinzugefügt oder geändert werden und die Bewertung von 1 bis 5 kann vorgenommen werden. So kann Metadatenpflege ohne Kontextwechsel betrieben werden und entsprechende Änderungen sind direkt nachvollziehbar.

Die GUI lädt initial die Liste der Sounds, die Kategorienliste und die Tag-Liste. Bei der Auswahl werden die dazugehörigen Metadaten und Tags geliefert. Änderungen im Detailbereich werden über die Zugriffsschicht in die Datenbank geschrieben, wobei die Aktualisierung von `updated_at` als Änderungsnachweis dient.

Die Listenansicht ist also das zentrale UI-Element, skalierbar für viele Einträge und ermöglicht eine schnelle Navigation. Durch die Trennung der Detailansicht von der Liste wird eine Überladung der Ergebnisliste verhindert und Fehler in der Bearbeitung reduziert. In der Liste selbst liegt der Fokus auf einer Auswahl, statt den vollständigen Metadaten.

4.10.2 Such- und Filterfunktionen

Durch die Such- und Filterfunktionen soll eine schnelle Auffindbarkeit von Sounds in großen Datenbeständen ermöglicht werden. Es werden zwei verschiedene Suchstrategien unterstützt: Eine textbasierte Suche über das freie Suchfeld, sowie eine strukturierte Suche über Kategorien und Tags. Durch kombinierbare Filter kann die Treffermenge gezielt eingegrenzt werden.
Das Suchfeld für einen beliebigen Suchbegriff umfasst die Bereiche `file_path`, den Kategorienamen, den Dateinamen und `description`. Bei einer leeren Suche werden alle Dateien angezeigt, ist ein Suchbegriff vorhanden erscheinen in der Trefferliste nur passende Einträge.
In der Abfrage wird ein `LIKE`-Muster verwendet, sodass auch unvollständige Suchbegriffe zu Treffern führen können. Außerdem erfolgt die Suche nicht case-sensitive, sodass Groß-und Kleinschreibung keinen Einfluss auf das Suchergebnis haben.
Die Trefferliste wird auch standardmäßig nach dem Attribut `updated_at` sortiert, da kürzlich importierte oder bearbeitete Audios möglicherweise häufiger von Interesse sind. Gleichzeitig bleibt die Ergebnisliste auch bei größeren Datenbeständen übersichtlich und nachvollziehbar.
Die Anzahl der angezeigten Treffer wird bewusst begrenzt, um zukünftig mögliche sehr große Ergebnismengen zu vermeiden, welche die Reaktionszeit der GUI negativ beeinflussen könnten, eine flüssige Bedienung wird im Prototyp einer vollständigen Anzeige vorgezogen.

Die Kategorie-Filter können über ein Dropdown-Menü ausgewählt werden. Dieses wirkt auf `category_id` in der Entität `sounds`. Wird eine Kategorie ausgewählt, so werden nur Dateien dieser Kategorie angezeigt, wird keine Kategorie ausgewählt gibt es keine Einschränkungen. Dieser Filter dient nur zur groben thematischen Eingrenzung und als zentrale Navigationsstruktur der Library.

Es gibt die Möglichkeit einen oder mehrere Tags in der GUI auszuwählen. Diese Filterung geschieht über die `n:m`-Beziehung zwischen den Entitäten `Sounds` und `Tags`, wobei die technische Umsetzung über die Zuordnungstabelle `sound_tags` stattfindet. Es können entweder alle Audios mit mindestens einem ausgewählten Tag, oder alle Audios mit allen ausgewählten Tags angezeigt werden. Dies ermöglicht eine feinere inhaltliche Differenzierung, als es nur mit Kategorien möglich wäre. Durch die Tags kann flexibel nach Eigenschaften und Merkmalen gesucht werden.

Des Weiteren ist die Kombination von Textsuche, Kategorie- und Tag-Filtern möglich. Die Filter reduzieren dabei zunächst die Gesamtmenge der Audios, während die Textsuche innerhalb der gefilterten Ergebnismenge wirkt. So kann die Trefferliste schrittweise eingegrenzt werden und lange, unübersichtliche Ergebnisse vermieden werden.

Die angezeigten Treffer werden bewusst begrenzt, da der Fokus auf der schnellen Reaktionszeit der GUI liegt. Typische Abfragen werden zusätzlich durch Datenbank-Indizes unterstützt, welche im Kapitel *4.7.6* bereits beschrieben wurden.
Die Trefferliste wird automatisch aktualisiert, um dem Nutzer eine Reaktion auf Änderungen von Suchbegriffen oder Filtern zu geben und eine visuelle Rückmeldung bei leeren Ergebnissen zu liefern.

4.10.3 Metadatenbearbeitung in der GUI

Wie bereits erwähnt, ist die Möglichkeit zur manuellen Ergänzung und Pflege von Metadaten, sowie die Korrektur automatisch gesetzter Informationen eine wichtige Kernfunktion der GUI und entsprechende Operationen werden direkt in die Datenbank übertragen.

Die bearbeitbaren Metadaten umfassen vier Bereiche. Die Beschreibung ist ein optionales Freitextfeld zur inhaltlichen Einordnung der Audiodatei. Die Kategorie basiert auf den in der Datenbank vorhandenen Einträgen und ist über ein Dropdown-Menü auswählbar, wobei jede Datei genau einer Kategorie zugeordnet ist. Ergänzend kann eine Kategorie-Sperre gesetzt werden, die verhindert, dass spätere Scan-Vorgänge die manuell gewählte Kategorie automatisch überschreiben. Schließlich können Tags als flexible, nicht-hierarchische Beschreibungen hinzugefügt und entfernt werden.

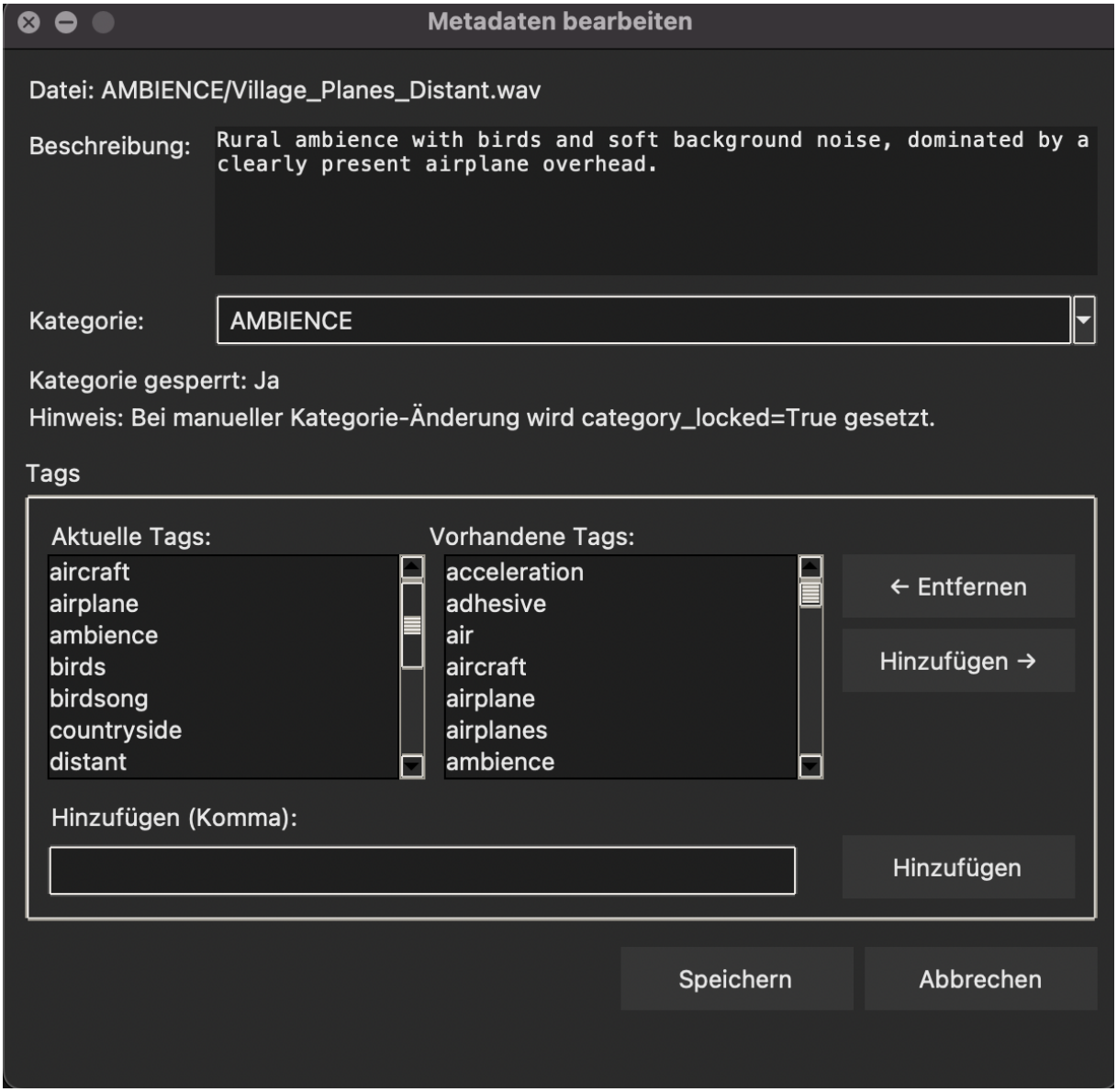

Abbildung 11: Metadaten-Bearbeitungsdialog mit ausgefüllter Beschreibung, aktiver Kategorie-Sperre und gesetzten Tags (eigener Screenshot)

Die Bearbeitungen geschehen ohne zusätzlichen Dialog, durch das Speichern erhält der Nutzer unmittelbares Feedback nach Änderung. Es ist keine separate Speicher-Phase notwendig, der Fokus liegt auf einem schnellen Workflow statt komplexer Validierung.

4.10.4 Weitere Funktionen der GUI

Neben den bereits beschriebenen Kernfunktionen bietet die GUI weitere Funktionen zur Dateiverwaltung und Datenbanksicherung.

Zu den Dateioperationen gehören das vollständige Löschen einer Datei, wobei diese sowohl aus der Datenbank als auch aus dem Dateisystem entfernt wird, sowie das Ausblenden einer Datei. Beim Ausblenden wird die Datei lediglich aus der Ansicht der GUI entfernt, bleibt jedoch im Dateisystem erhalten. Ausgeblendete Dateien können über die Funktion „Ausgeblendete Dateien“ eingesehen und bei Bedarf wieder eingeblendet werden. Ergänzend gibt es die Möglichkeit, den Speicherort einer Datei direkt im Finder zu öffnen oder den Dateipfad in die Zwischenablage zu kopieren, was eine schnelle Weiterverwendung außerhalb der Anwendung ermöglicht.

Zudem stehen Snapshot-Funktionen zur Datenbankverwaltung bereit. Über „Snapshot speichern“ wird eine versionierte Kopie der aktuellen Datenbank erstellt, die als Wiederherstellungspunkt dient. Über „Snapshot laden“ kann eine gespeicherte Version wiederhergestellt werden, wobei die aktuelle Datenbank vor dem Überschreiben automatisch gesichert wird. Diese Funktion ist besonders vor größeren Reorganisationen der Sound-Library sinnvoll.

4.10.5 Audio-Playback

Der nächste zentrale Punkt der GUI ist die akustische Vorschau der Audios mit einer visuellen Orientierung über den zeitlichen Verlauf. Eine Bearbeitung der Audiodateien ist hierbei nicht vorgesehen.

Das Audio-Playback verfügt über eine einfache Steuerung mit der Option, die Audios zu starten und zu stoppen. Die Wiedergabe erfolgt dabei direkt aus dem Dateisystem, wobei der in der Datenbank hinterlegte Dateipfad genutzt wird. Es kann nur ein Audio zeitgleich wiedergegeben werden, ein Wechsel der ausgewählten Datei beendet die laufende Wiedergabe.

Zusätzlich erscheint eine grafische Darstellung des Signalverlaufs über die Zeit als Wellenform. Diese wird automatisch beim Auswählen einer Datei geladen. Die Darstellung wird auf Peaks reduziert, um eine ausreichende Performance zu gewährleisten, die Darstellung passt sich auf die aktuelle Fenstergröße an und bleibt unabhängig von der Metadatenbearbeitung.
Die Wellenform ist zeitlich mit dem Playback synchronisiert und es existiert eine visuelle Markierung an der aktuellen Abspielposition („Playhead“). Diese wird regelmäßig während des Playbacks aktualisiert. Eine wichtige Funktion ist das Verändern der Abspielposition durch das Klicken auf eine entsprechende Stelle in der Wellenform, wodurch auch eine direkte Rückmeldung durch die aktualisierte Anzeige entsteht.

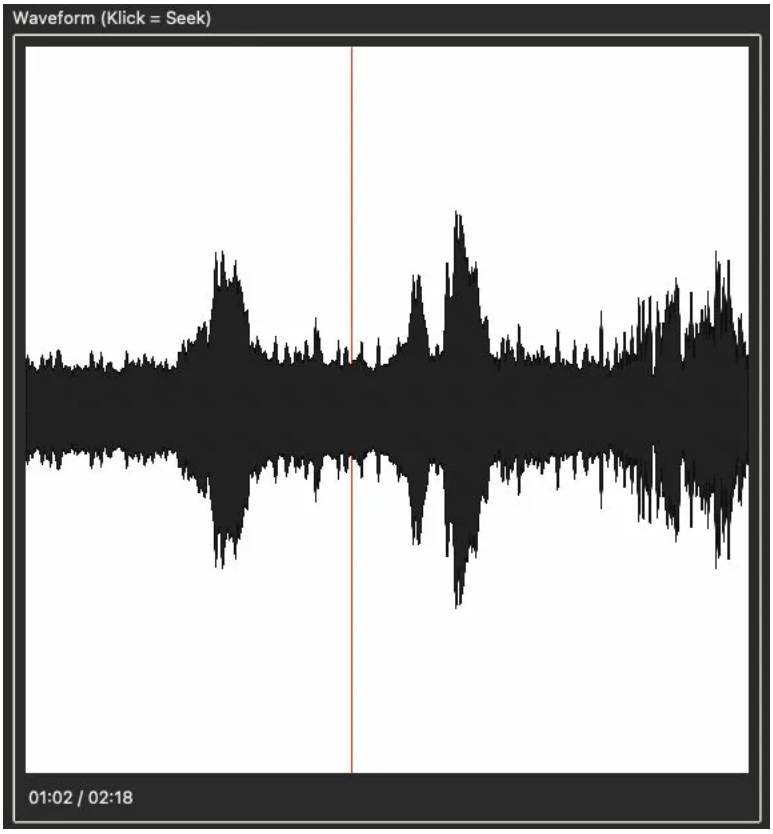

Abbildung 12: Wellenformdarstellung mit Playhead (1:02 / 2:18) im Audio-Playback; Klick auf die Wellenform setzt die Abspielposition (eigener Screenshot)

Das Playback wird über Python-Libraries umgesetzt. Dabei werden mit `wave` die Dateien eingelesen, inklusive bestimmter Grundparameter wie Abtastrate, Bittiefe und die Anzahl der Kanäle. Die Peak-Berechnung erfolgt über `numpy`, die eigentliche Wiedergabe läuft dagegen über *`sounddevice`*.

Die Audio-Vorschau dient als reine Hilfsfunktion, um relevante Stellen im Audiomaterial zu finden und diese zu bewerten. Sie dient nicht als Ersatz für eine DAW, es gibt keine Funktionen zum Schneiden, Zoomen, Editieren oder zur Effektbearbeitung. Es wurde sich bewusst auf die Kernfunktionalität beschränkt und der Fokus auf Übersichtlichkeit, Stabilität und Praxistauglichkeit gesetzt. So entsteht ein schneller Eindruck der Dynamik, Struktur und Länge einer Audio-Datei, was besonders beim Vergleich von ähnlichen oder kurzen Inhalten hilfreich ist. Somit stellt diese Funktion eine sinnvolle Ergänzung zur Suche, Filterung und Metadatenbearbeitung dar.

Mit der fertiggestellten GUI ist die Implementierung des Kernsystems abgeschlossen. Der letzte Schritt bestand in der Bereitstellung des Prototyps als eigenständige, nutzbare Anwendung.

4.11 Bereitstellung des Prototyps

Die Bereitstellung des Prototyps erfolgt als eigenständige Anwendung ohne die Notwendigkeit einer Python-Installation. Um die Nutzbarkeit für Tester ohne technischen Hintergrund zu gewährleisten, wurde der Prototyp mittels PyInstaller als plattformspezifische Anwendung entwickelt. PyInstaller ermöglicht die Umwandlung von Python-Anwendungen in ausführbare Dateien, die alle notwendigen Abhängigkeiten, einschließlich des Python-Interpreters, der verwendeten Bibliotheken und der Anwendungslogik, in sich bündeln (Cortesi (n.d.)).

Der Prototyp wurde primär für macOS entwickelt und getestet, entsprechend stellt die macOS-Version die Hauptversion der Anwendung dar und wurde als natives App-Bundle erstellt. Da jedoch nicht alle potenziellen Nutzer oder Tester Zugriff auf ein macOS-System haben, wurde zusätzlich eine angepasste Windows-Version erstellt, die als Alternative dient.
Durch plattformspezifische Unterschiede im Verhalten bei Dateipfaden, Betriebssystem-Integration und GUI-Rendering ergab sich dabei die Notwendigkeit von Anpassungen in der Windows-Version, wobei die Kernfunktionalität beider Versionen identisch blieb.

Um eine sofortige Funktionsprüfung zu ermöglichen, wird die Anwendung mit vorinstallierten Demo-Inhalten zur Verfügung gestellt. Dabei handelt es sich um 77 Audiodateien aus den Kategorien Ambience, Cars, Doors, Foley, Footsteps und Household. Diese Zahl liegt unter den ursprünglich geplanten 40 Sounds je Kategorie, was auf die in Kapitel 4.1 beschriebenen Aufnahmebedingungen zurückzuführen ist, wobei der Umfang für den angestrebten Funktionsnachweis des Prototyps als ausreichend bewertet wurde. Diese Dateien repräsentieren den identischen Bestand, der auch für die Entwicklung und Evaluierung der Sound-Library verwendet wurde. Zusätzlich wird eine vorbefüllte Datenbank mitgeliefert, die bereits technische und manuelle Metadaten der Demo-Sounds enthält. So können Tester die Funktionalität der Anwendung sofort erkunden, einschließlich Suche, Filterung, Tag-Verwaltung und Audio-Playback, ohne zunächst eigene Audiodateien importieren zu müssen.
Beim ersten Start der Anwendung wird der Nutzer aufgefordert, einen Ort für das Sound-Verzeichnis zu wählen. Im Anschluss werden die eingebetteten Demo-Sounds automatisch hierhin kopiert. Durch diesen Mechanismus wird sichergestellt, dass die Anwendung unmittelbar nach dem Start funktionsfähig ist, während gleichzeitig verhindert wird, dass das Demo-Material ungewollt eigene Sounds überschreibt.

Die fertig paketierte Anwendung wird als komprimiertes Archiv (ZIP) bereitgestellt, das neben der ausführbaren Datei eine systemspezifische `README`-Datei mit grundlegenden Nutzungshinweisen

enthält. Nach dem Entpacken kann die Anwendung ohne Installationsroutine direkt gestartet werden. Die Datenbank wird automatisch im Anwendungsverzeichnis angelegt und bleibt zwischen Sessions beständig. Die Anwendung verhält sich grundsätzlich wie kommerzielle Software und erfordert kein technisches Verständnis von Python, Paketverwaltung oder Kommandozeilen-Befehlen. Dies entspricht dem Ziel, einen praxisnahen Prototyp zu liefern, der als Grundlage für Nutzerfeedback und Workflow-Evaluierung dienen kann.

5. Evaluation

Im Folgenden wird die Evaluation des Prototyps beschrieben. Dieser wird dabei zunächst im Rahmen eines Vergleichs mit bekannten bestehenden kommerziellen Lösungen eingeordnet. Anschließend wird ein informeller Funktionstest durch externe Tester beschrieben und ausgewertet. Abschließend erfolgt ein Abgleich der Testergebnisse mit den in 3.2 formulierten Anforderungen.

5.1 Ziel und Kriterien

In der Evaluation werden zwei übergeordnete Ziele verfolgt. So soll der Prototyp zum einen mit bestehenden Lösungen verglichen werden, um seinen Mehrwert und seine Abgrenzung zu diesen nachvollziehbar zu machen. Durch die praktischen Funktionstests soll überprüft werden, ob der konzipierte Workflow in der Praxis wie vorgesehen funktioniert.

Bei den externen Tests handelt es sich ausdrücklich nicht um einen kontrollierten Usability-Test im wissenschaftlichen Sinne. Es wurden keine standardisierte Testaufgaben definiert, keine quantitativen Metriken erhoben und keine statistische Auswertung durchgeführt. Stattdessen wurde ein informeller Funktionstest durchgeführt, bei dem externe Tester den Prototyp auf ihren eigenen Rechnern installiert und ausprobiert haben. Dieser Ansatz ist für den Umfang und das Ziel dieser Arbeit angemessen, da der Prototyp als funktionaler Nachweis des Konzepts fungiert und keine marktreife Anwendung darstellen soll.

Die Evaluation orientiert sich entsprechend an folgenden Kriterien: Die konzeptionelle Positionierung gegenüber bestehenden Lösungen, die erfolgreiche Installation und der Start der Anwendung ohne technische Vorkenntnisse, die plattformübergreifende Lauffähigkeit auf macOS und Windows, die grundlegende Nutzbarkeit der Kernfunktionen sowie die Praxistauglichkeit des Gesamtworkflows aus Nutzersicht.

5.2 Vergleich mit bestehenden Lösungen

Um den entwickelten Prototyp einzuordnen, werden im Folgenden vier im Bereich der Sound-Libraries etablierte Lösungen verglichen: Soundly, SoundQ, BaseHead und Explorer (Sound Particles). Diese Tools wurden ausgewählt, da sie denselben Kernbereich abdecken, nämlich die Organisation und Nutzung von Audiodateien. Allerdings setzen sie auf unterschiedliche Schwerpunkte und stellen damit einen repräsentativen Querschnitt des Marktes dar. Es handelt sich hierbei um einen qualitativen Vergleich auf Basis öffentlich zugänglicher Produktinformationen und nicht um systematische Funktionstests.

Ein zentraler Unterschied ist der Umgang mit Cloud-Inhalten und Benutzerkonten. Soundly und SoundQ kombinieren lokale Libraries explizit mit Cloud-basierten Inhalten (Soundly, n.d.; Pro Sounds Effects, n.d.). Soundly beschreibt dabei sowohl lokale als auch cloudbasierte Libraries. SoundQ bietet eine kostenlose Cloud-Library sowie unbegrenzte lokale Dateien. BaseHead ist stärker als lokal-professionelle Lösung positioniert, während Explorer ebenfalls ohne Cloud-Abhängigkeit auskommt (BaseHead, n.d.; Sound Particles, n.d.). Alle vier kommerziellen Tools setzen jedoch eine Account-Erstellung oder Lizenzierung voraus.

Im Bereich Such- und Metadatenfunktionen zeigen sich ebenfalls deutliche Unterschiede. Soundly hebt eine intelligente Suche mit Funktionen wie AI-gestützter Ähnlichkeitssuche und Auto-Complete hervor (Soundly, n.d.). SoundQ betont Boolean-Suche, Smart Search sowie explizite Read- und Write-Unterstützung für iXML-Metadaten (Pro Sound Effects, n.d.). BaseHead ist besonders stark im Datenbankbereich und bietet ein dediziertes Details Panel zur Metadatenanzeige und Bearbeitung (BaseHead, n.d.). Explorer ist in diesem Bereich öffentlich weniger ausführlicher dokumentiert als die anderen drei Tools. Alle vier bieten damit deutlich detailliertere Suchfunktionen als der im Rahmen dieser Arbeit entwickelte Prototyp.

Hinsichtlich der Produktions-Workflow-Integration ist SoundQ besonders klar positioniert. Drag-and-Drop sowie eine „Spot to Timeline“-Funktion ermöglichen die direkte Übergabe von Sounds in DAWs wie Pro Tools, Adobe Premiere und Reaper. Soundly bietet ebenfalls Workflow-orientierte Funktionen für lokale und cloudbasierte Nutzung. BaseHead richtet sich mit erweiterbarem Funktionsumfang in höheren Editionen deutlich an professionelle Postproduktion-Workflows. Explorer wirkt im Vergleich als schlankeres Werkzeug ohne tiefe DAW-Integration.
Bezüglich der Zugänglichkeit sind SoundQ und Explorer kostenlos nutzbar (Pro Sound Effects, n.d.; Sound Particles, n.d.), Soundly bietet eine kostenlose Variante mit eingeschränkten Umfang an (Soundly, n.d.) und BaseHead stellt eine 15-Tage-Trial sowie eine kostenlose Edition bereit (BaseHead, n.d.). Alle vier Tools sind damit grundsätzlich kostenlos testbar.

Der im Rahmen dieser Arbeit entwickelte Prototyp lässt sich funktional im Kernbereich lokaler Sound-Library-Verwaltung einordnen. Dateien können erfasst, Metadaten gespeichert, gesucht, gefiltert und wiedergegeben werden. Im Vergleich zu den kommerziellen Lösungen fehlen Cloud-Integration, DAW-Anbindung und erweiterte Suchfunktionen. Diese Unterschiede sind jedoch keine unbeabsichtigte Lücken, sondern entsprechen der in Kapitel 3.1 definierten Abgrenzung. Der Prototyp verfolgt bewusst einen anderen Ansatz, er ist vollständig Lokal, kommt ohne Account-Pflicht und externe Abhängigkeiten aus und besitzt einen offenen, erweiterbaren Code. Damit richtet er sich an Nutzer, die ihren eigenen Workflow ohne Cloud-Zwang und ohne Bindung an den Funktionsumfang kommerzieller Lösungen abbilden möchten.

5.3 Test-Setup

Der Prototyp wurde im Rahmen der in Kapitel 4.11 beschriebenen Bereitstellung als komprimiertes Archiv (ZIP) zur Verfügung gestellt. Dieses enthielt die ausführbare Anwendung sowie eine systemspezifische README-Datei mit grundlegenden Nutzungshinweisen. Die Tester erhielten keine weiterführende Einweisung in die Bedienung, sondern wurden lediglich gebeten, die Anwendung zu entpacken, zu starten und die grundlegenden Funktionen zu erkunden.

An diesen Tests nahmen 10 Personen teil. Die Testgruppe setzte sich sowohl aus Personen mit Vorkenntnissen im Bereich der Tontechnik als auch Personen ohne entsprechende Vorkenntnisse zusammen. Ein Teil der Tester nutzten macOS-Geräte, der andere Teil Windows-Rechner. Da es sich um einen informellen Test handelt, wurden keine detaillierten demografischen Daten erhoben.
Das Feedback wurde in persönlichen Gesprächen im Anschluss an den Test gesammelt. Es gab keine standardisierten Fragen oder Bewertungsskalen. Die Rückmeldungen bezogen sich auf die Erfahrungen in den Bereichen der Installation, dem Funktionsumfang, wahrgenommene Probleme, sowie den allgemeinen Eindruck der Anwendung.

5.4 Externer Test

Der Test verlief bei der überwiegenden Mehrheit der Tester ohne nennenswerte Probleme. Die Anwendung ließ sich auf den getesteten Systemen in der Regel direkt nach dem Entpacken starten, ohne dass eine manuelle Installation von Abhängigkeiten oder Python-Kenntnisse erforderlich waren. Dieser Aspekt wurde von mehreren Testern positiv hervorgehoben. Die Möglichkeit eine funktionsfähige Anwendung direkt auszuführen, ohne eine vorherige Installation oder gar Programmcode ausführen zu müssen, wurde als klarer Mehrwert wahrgenommen.

Auch der Workflow wurde aus Nutzersicht positiv bewertet. Die grundlegenden Schritte wie Sounds auffinden, filtern Vorhören und die Metadatenbearbeitung wurden als nachvollziehbar und praxistauglich eingeschätzt. Die Trennung zwischen der Dateiablage im Dateisystem und der Verwaltungsschicht der Sound-Library wurde dabei als schlüssig empfunden.

Im Verlauf der Tests traten zwei Auffälligkeiten auf. Ein Tester mit einem Windows-Rechner berichtete von Problemen bei Entpacken des ZIP-Archivs. Da das Problem ausschließlich bei diesem einen System auftrat und die übrigen Windows-Tester keine vergleichbaren Schwierigkeiten hatten, ist davon auszugehen, dass es sich um eine systemspezifische Konfiguration handelt und kein generelles Bereitstellungsproblem vorliegt.

Die zweite Auffälligkeit betraf einen macOS-Tester, bei dem die Anwendung während der Nutzung gelegentlich einfror. Da dieses Verhalten nicht reproduzierbar war und bei andern macOS-Testern nicht auftrat, lässt sich die Ursache nicht abschließend klären. Eine mögliche Erklärung ist eine Inkompatibilität mit unterschiedlichen macOS-Versionen, da der Prototyp primär auf einer definierten Systemkonfiguration entwickelt und getestet wurde. Dieses Verhalten stellt einen offenen Punkt dar, der im Rahmen einer Weiterentwicklung gezielt untersucht werden sollte.

5.5 Ergebnisse und Auswertung

Die Ergebnisse des Tests lassen sich in zwei Bereiche gliedern, nämlich die technische Lauffähigkeit und die Praxistauglichkeit des Workflows.

Hinsichtlich der technischen Lauffähigkeit zeigt der Test, dass der Prototyp auf den getesteten System grundsätzlich funktionsfähig ist. Die Bereitstellung als eigenständige Anwendung ohne Installationsroutine hat sich bewährt. Die beschriebenen Auffälligkeiten, ein systemspezifisches ZIP-Problem unter Windows, sowie vereinzeltes Einfrieren unter macOS betreffen Randfälle und schränken den allgemeinen Funktionsnachweis nicht wesentlich ein. Sie weisen jedoch auf Optimierungsbedarf in der plattformübergreifenden Kompatibilität hin, der bei einer Weiterentwicklung adressiert werden sollte.

Hinsichtlich der Workflow-Tauglichkeit fällt das Feedback positiv aus. Die Tester konnten die Kernfunktionen der Sound-Library ohne Einweisung nutzen, was auf eine ausreichende Verständlichkeit der GUI hindeutet. Der konzipierte Workflow mit dem Ablegen von Audiodateien im Dateisystem, dem automatischen Einlesen durch den Scan-Prozess, sowie dem Suchen und Vorhören wurde als praxistauglich bewertet.

Einschränkend muss festgehalten werden, dass die Aussagekraft dieser Evaluation aufgrund der geringen Testgröße, des informellen Rahmens und des Fehlens strukturierter Aufgaben begrenzt ist. Die Ergebnisse erlauben keine verallgemeinerbaren Schlussfolgerungen über die Usability der Anwendung, sondern liefern qualitative Hinweise auf den Funktionsstand des Prototyps. Für eine fundierte Bewertung der Benutzerfreundlichkeit wäre ein strukturierter Usability-Test mit definierten Aufgaben notwendig, was den Rahmen dieser Arbeit übersteigen würde.

5.6 Abgleich mit den Anforderungen

Im letzten Schritt der Evaluation werden die Ergebnisse des Tests mit den in Kapitel 3.2 formulierten Anforderungen verglichen, um zu bewerten, inwieweit der Prototyp die gesetzten Anforderungen erfüllt.

Die Anforderung nach einer automatischen Erkennung und Übernahme neuer Audiodateien ohne manuellen Import wurde durch den Scan-Prozess umgesetzt und im Rahmen des Tests als funktionierend bestätigt. Dateien konnten nach dem Ablegen im Basisverzeichnis direkt in der Sound-Library genutzt werden.

Die Anforderung nach einer Textsuche sowie der Filterung nach Kategorien und Tags wurde in der GUI implementiert und stand den Testern zur Verfügung. Das positive Feedback zum Workflow legt nahe, dass diese Funktionen grundsätzlich nutzbar sind. Eine vertiefte Bewertung ihrer Treffgenauigkeit oder Effizienz war im Rahmen des informellen Tests jedoch nicht möglich.

Die Anforderung nach einem direkten Vorhören in der Anwendung wurde durch die implementierte Playback-Funktion mit Wellenformdarstellung erfüllt. Diese Funktion wurde im Test genutzt und nicht als problematisch genannt.

Die Anforderung nach einer manuellen Metadatenpflege, also dem Setzen von Tags, Beschreibungen, Kategorien und Bewertungen direkt in der GUI war implementiert und zugänglich, wurde jedoch im Rahmen des informellen Tests jedoch nicht systematisch evaluiert, da die Tester primär die Installations- und Grundfunktionalitäten prüften.

Zusammenfassend lässt sich festhalten, dass der Prototyp die in Kapitel 3.2 formulierten Kernanforderungen im Rahmen des durchgeführten Tests erfüllt. Der Funktionsnachweis ist damit erbracht. Der Vergleich mit bestehenden Tools in Kapitel 5.2 bestätigt zudem, dass der Prototyp die in Kapitel 3.1 definierte Abgrenzung einhält. Er ist keine Konkurrenz zu kommerziellen Lösungen,

sondern eine schlanke, lokal kontrollierbare Alternative für einen spezifischen Workflow. Offene Punkte betreffen die Kompatibilität mit unterschiedlichen Betriebssystemversionen sowie Funktionsbereich, die im informellen Test nicht gezielt geprüft wurden und einer weiterführenden Evaluation bedürften.

6. Diskussion

Im Folgenden werden die Ergebnisse der Arbeit kritisch reflektiert. Dabei werden sowohl konzeptionelle und technische Stärken des Prototyps als auch sein Grenzen und Schwächen diskutiert. Ergänzend wird der Entstehungsprozess selbst bewertet, insbesondere die zeitliche und inhaltliche Schwerpunktsetzung. Ziel ist keine nachträgliche Rechtfertigung getroffener Entscheidungen, sondern einer sachlichen Einordnung.

6.1 Konzeptionelle Stärken

Der Kerngedanke des Prototyps, die Trennung von Dateisystem und Verwaltungsschicht, hat sich als tragfähig erwiesen. Audiodateien bleiben als unveränderte Objekte im Dateisystem, während Metadaten flexibel und unabhängig vom Dateiinhalt in der Datenbank gepflegt werden können. Dieses Prinzip ist konzeptionell sauber und hat sich sowohl in der Implementierung als auch im externen Test als verständlich erwiesen.

Besonders durchdacht ist der category_locked-Mechanismus, der das Spannungsfeld zwischen automatisierten Prozessen und manueller Kontrolle adressiert. Automatische Scans können Kategorien vorschlagen, ohne manuell kuratierte Einträge zu überschreiben. Dieses Prinzip, dass automatische Prozesse den Nutzer unterstützen, zieht sich konsistent durch das gesamte Konzept und spiegelt sich auch in der UPSERT-Logik des Scan-Prozesses wider.

Der Wechsel von PostgreSQL zu SQLite hat sich für den Prototyp-Kontext als richtige Entscheidung erwiesen. Die daraus resultierende dateibasierte Datenbank ohne Serverabhängigkeit hat die Bereitstellung und den externen Test deutlich vereinfacht. Dass die Anwendung ohne Installation direkt ausführbar ist und die Datenbank als einzelne Datei mitgeliefert wird, wurde von Testern explizit positiv hervorgehoben. Diese Portabilität wäre mit PostgreSQL nicht ohne erheblichen Mehraufwand erreichbar gewesen.

6.2 Grenzen des Prototyps

Die Bedienlogik der GUI weist Schwächen auf. Das explizite Bestätigen von Filtern statt einer automatischen Aktualisierung bei jeder Änderung entspricht nicht der erwarteten Interaktionslogik moderner Such-Interfaces und erzeugt unnötige Reibung im Workflow. Ähnlich verhält es sich in anderen Bereichen. Verschachtelte Bereiche und nicht sofort offensichtliche Funktionen machen die Oberfläche insgesamt komplexer als notwendig. Diese Probleme sind lösbar, wurden aber im Rahmen des Prototyps nicht mehr adressiert, da der Fokus auf der Funktionalität lag.

Ein konzeptionell interessant Erweiterung wäre das Zurückschreiben von Metadaten direkt in die Audiodatei, also nicht nur in die Datenbank, sondern auch in die eingebetteten Metadaten der Audiodatei selbst. Aktuell ist die Datenbank die einzige Quelle für organisatorische Metadaten, was bedeutet, dass diese Informationen bei einem Verlust der Datenbankdatei nicht wiederherstellbar sind. Eine bidirektionale Metadatenverwaltung würde die Robustheit des Systems deutlich erhöhen und wäre ein sinnvoller nächster Schritt.

Der bewusst eingeschränkte Funktionsumfang ist keine konzeptionelle Schwäche, sondern eine definierte Abgrenzung, die in Kapitel 3.1 begründet wurde und durch den Vergleich mit kommerziellen Lösungen, in Kapitel 5.2 eingeordnet ist. Diese Einschränkungen sind bekannt und gewollt, sie stellen keine Defizite dar, solange der Prototyp in seinem definierten Anwendungsbereich funktioniert, was der externe Test bestätigt hat.

6.3 Reflexion des Entstehungsprozesses

Rückblickend war die inhaltliche Schwerpunktsetzung im Bereich der Audioaufnahmen nicht optimal. Die Entscheidung, pro Kategorie eine möglichst große Anzahl an Aufnahmen zu erstellen, hat erheblichen Zeitaufwand verursacht, der im späteren Verlauf der Arbeit nicht vollständig zum Tragen kam. Ein Teil des aufgenommenen Materials wurde letztlich nicht in den Prototyp integriert, da der Umfang die ursprünglich geplanten 40 Sounds je Kategorie nicht erreichte und viele Aufnahmen durch Umgebungsbedingungen wie Straßen- oder Fluglärm beeinträchtigt wurden.

Im Nachhinein wäre ein stärkeres Prinzip von Qualität vor Quantität sinnvoller gewesen. Weniger, aber sorgfältiger ausgewählte Aufnahmen hätten denselben Funktionsnachweis ermöglicht, bei gleichzeitig besserem Klangbild und geringerem Zeitaufwand. Zudem hätte eine breitere Kategorisierung, etwa durch eine zusätzliche Kategorie für allgemeine Soundeffekte, die Varianz des Datensatzes und damit die Aussagekraft der Filter- und Tagging-Funktionen erhöht.

Das 50/50-Verhältnis zwischen Audio- und Softwareteil war in der Konzeption sinnvoll und hat dem Prototyp einen praxisnahen Charakter gegeben. In der Durchführung hat der Audioteil jedoch mehr Zeit beansprucht, als sein späterer Anteil in der Arbeit widerspiegelt. Die aufwendige Feldaufnahme und das Editing stehen in einem gewissen Missverhältnis zu dem vergleichsweise kurzen Abschnitt, den sie in der schriftlichen Arbeit einnehmen. Der Softwareteil hätte vor dem freigewordenen Zeitbudget profitieren könne, etwa durch eine tiefere Auseinandersetzung mit der Datenbankarchitektur, eine strukturierte Evaluation oder einen ausgearbeiteten Usability-Test.

Dennoch war der Audioteil nicht wertlos, die eigenen Aufnahmen haben ein tieferes Verständnis für die Anforderungen an eine Sound-Library erzeugt, als es ein vorgefertigter Datensatz ermöglicht hätte. Probleme wie inkonsistente Lautheit, unterschiedliche Aufnahmequalitäten und die Notwendigkeit, einer strukturierten Ordnerablage sind erst durch die praktische Erfahrung konkret greifbar geworden und haben die Konzeptionsentscheidung direkt beeinflusst.

6.4 Grenzen der Evaluation

Die in Kapitel 5 durchgeführte Evaluation liefert einen qualitativen Funktionsnachweis, keine wissenschaftlich belastbaren Aussagen über Usability oder Effizienz. Die Testgruppe war klein, nicht repräsentativ für die eigentliche Zielgruppe aus der Ton-Postproduktion und wurde ohne strukturierte Aufgaben oder Bewertungsskalen eingesetzt. Das Feedback ist damit als Indiz, nicht als Beleg zu verstehen.

Für eine fundierte Weiterentwicklung wäre ein strukturierter Usability-Test mit Personen aus dem Bereich der Ton-Postproduktion notwendig, ergänzt durch Langzeittests unter realen Arbeitsbedingungen mit wachsenden Dateibeständen. Insbesondere die Frage, ob die GUI auch bei Hunderten oder Tausenden von Einträgen noch performant und übersichtlich bleibt, konnte im Rahmen dieser Arbeit nicht beantwortet werden.

Das vereinzelte Einfrieren der Anwendung unter macOS bleibt ein offener Punkt. Da das Problem nicht reproduzierbar war, konnte keine Ursachenanalyse durchgeführt werden. Es ist nicht auszuschließen, dass es sich um ein tieferliegendes Kompatibilitätsproblem zwischen dem verwendeten Tkinter-Framework und bestimmten macOS-Versionen handelt, ein bekanntes Problem, das in der Tkinter-Dokumentation und Community diskutiert wird. Eine gezielte Untersuchung unter verschiedenen macOS-Version wäre ein sinnvoller erster Schritt.

7. Fazit und Ausblick

Ziel dieser Arbeit war die Konzeption und Implementierung einer lokalen, SQL-basierten Sound-Library mit grafischer Benutzeroberfläche. Der Ausgangspunkt war ein konkretes praktisches Problem. Audiodateien liegen in der Ton-Postproduktion oft verteilt in undurchsichtigen Ordnerstrukturen, sind ohne Metadatenpflege schwer auffindbar und erfordern für eine strukturierte Verwaltung üblicherweise die Abhängigkeit von kommerziellen Tools mit festem Funktionsumfang und externen Update-Zyklen. Die Sound-Library sollte dieses Problem durch einen tool-unabhängigen, lokal kontrollierbaren Workflow lösen.

Das Ergebnis ist ein funktionsfähiger Prototyp, der die wesentlichen Anforderungen erfüllt: Audiodateien werden automatisch durch einen Scan-Prozess erfasst und in eine relationale SQLite-Datenbank überführt, ohne dass ein manueller Import notwendig ist. Über eine Tkinter-basierte GUI können Sounds gesucht, gefiltert, vorgehört und mit Metadaten versehen werden. Der Prototyp ist plattformübergreifend für macOS und Windows verfügbar und ohne technische Vorkenntnisse direkt ausführbar. Der externe Funktionstest hat bestätigt, dass Workflow und Bereitstellung in der Praxis funktionieren.
Der Vergleich mit kommerziellen Sound-Libraries zeigt, dass der Prototyp im Kernbereich der lokalen Dateiverwaltung und Metadatenpflege funktioniert, dabei aber bewusst auf Funktionen wie Cloud-Integration, DAW-Anbindung oder erweiterte Suchfunktionen verzichtet. Diese Einschränkungen sind keine Schwäche, sondern eine definierte Abgrenzung, die dem Ziel eines schlanken, lokal kontrollierbaren Werkzeugs entspricht.

Gleichzeitig zeigt die Diskussion in Kapitel 6, dass der Entstehungsprozess Optimierungspotenzial bietet. Die zeitliche Investition in den Audioaufnahmen stand nicht im optimalen Verhältnis zu ihrem späteren Nutzen in der Arbeit, und die GUI weist in ihrer aktuellen Form Bedienlogiken auf, die in einer Weiterentwicklung überarbeitet werden sollten. Der Prototyp ist damit das, was er sein soll: Ein funktionaler Nachweis des Konzepts, kein marktreifes Produkt.

7.1 Ausblick

Die vorliegende Arbeit legt eine Grundlage, auf der in verschiedene Richtungen aufgebaut werden kann. Die naheliegendsten Erweiterungen betreffen dabei sowohl die technische Schwächen des aktuellen Stands als auch konzeptionelle Erweiterungen, die den Anwendungsbereich ausbauen würden.

Auf technischer Ebene wäre die Überarbeitung der GUI-Bedienlogik ein sinnvoller erster Schritt. Insbesondere die automatische Aktualisierung der Ergebnisliste bei Filteränderungen ohne manuelles Bestätigen sowie eine vereinfachte Gesamtstruktur der Oberfläche würde die Nutzbarkeit im Workflow deutlich verbessern. Auch die Wellenformdarstellung sollte hinsichtlich Ladezeit und Interaktionsmöglichkeiten überarbeitet werden, etwa durch eine optional zuschaltbare Darstellung oder eine performantere Implementierung.

Eine konzeptionell wertvolle Erweiterung wäre das zusätzliche Schreiben von Metadaten direkt in die Audiodatei. Aktuell existieren organisatorische Metadaten ausschließlich in der Datenbank, bei einem Verlust der Datenbankdatei gehen sie verloren. Eine bidirektionale Metadatenverwaltung, bei der Änderungen in der GUI auch in die eingebetteten Metadaten der Audiodateien geschrieben werden, würde die Robustheit und Portabilität des gesamten Systems erhöhen.

Im Bereich der Audioformate wäre die Unterstützung weiterer Formate wie AIFF, FLAC oder MP3 eine naheliegende Erweiterung. Da der aktuelle WAV-only-Ansatz auf der Nutzung des Python-Standardmoduls `wave` basiert, würde eine Erweiterung den Einsatz externer Bibliotheken erfordern, konzeptionell ist das lösbar, ohne die bestehende Architektur grundlegend zu verändern.

Für eine breitere Nutzbarkeit wäre zudem ein echter File-System-Watcher sinnvoll, der neue Dateien im Basisverzeichnis automatisch und ohne manuellen Scan-Aufruf erkennt. Die bestehende Watch-Modus-Implementierung mit festem Intervall ist ein funktionaler Workaround, kein vollwertiger Ersatz für eine ereignisbasierte Überwachung, wie sie beispielsweise über die Python-Bibliothek watchdog realisiert werden könnte.

Langfristig bietet die bestehende Architektur auch die Grundlage für mögliche weitergehende Erweiterungen. So wären eine DAW-Integration über Drag-and-Drop, ein Mehrbenutzer-Betrieb durch den Wechsel auf ein serverbasiertes Datenbanksystem oder ein automatisches Tagging durch Machine-Learning-basierte Klassifikation von Umgebungsgeräuschen denkbar. Diese denkbaren Schritte könnten konzeptionell auf dem hier entwickelten System aufbauen, hätten allerdings den Rahmen dieser Arbeit überschritten.

Insgesamt zeigt die Arbeit, dass der Ansatz einer tool-unabhängigen, lokal kontrollierbaren Sound-Library tragfähig ist. Der Prototyp liefert einen funktionalen Nachweis dafür, dass sich die Kernfunktionen eines Sound-Library-Workflow mit überschaubaren Mitteln und ohne externe

Abhängigkeiten realisieren lassen. Die identifizierten Schwächen sind bekannt, lösbar und bieten einen klaren Ausgangspunkt für eine Weiterentwicklung.

Literaturverzeichnis

Ableton. (2016, 20. Juni). *Sounds outside: The art of field recording*. https://www.ableton.com/en/blog/art-of-field-recording/ (aufgerufen am 30.01.2026).

Adobe. (2020). *Adobe XMP Specification Part 3: Storage in Files* (Januar 2020). https://dl.photoprism.app/pdf/specifications/20120101-Adobe_XMP_Specification_Part_3.pdf

BaseHead. (n.d.). basehead - The Ultra SFX Search Engine. https://baseheadinc.com/ (aufgerufen am 20.02.2026).

Brandenburg, K., & Popp, H. (2000). MPEG Layer-3: An introduction to the basic technology and some of the special features of MPEG Layer-3. *EBU Technical Review*, Juni, 1–15. http://www.mp3-tech.org/programmer/docs/trev_283-popp.pdf

Bray, T. (Hrsg.). (2017). *The JavaScript Object Notation (JSON) Data Interchange Format* (RFC 8259). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc8259

Chalmers, R. (1997). The Broadcast Wave Format – An introduction. *EBU Technical Review*, Winter, 1–6. https://tech.ebu.ch/files/live/sites/tech/files/shared/techreview/trev_274-chalmers.pdf

Cortesi, D. (n.d.). *PyInstaller Manual*. https://pyinstaller.org/en/stable/ (aufgerufen am 18.02.2026).

Dang, Q. H. (2012). *Recommendation for applications using approved hash algorithms* (NIST Special Publication 800-107 Revision 1). National Institute of Standards and Technology. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf

Day, M. (2005). *Metadata*. In *DCC Digital Curation Manual*. University of Edinburgh. https://era.ed.ac.uk/bitstream/1842/3321/1/Day%20metadata.pdf

Eastlake, D., 3rd, & Hansen, T. (2006). *US secure hash algorithms (SHA and HMAC-SHA)* (RFC 4634). RFC Editor. https://www.rfc-editor.org/rfc/rfc4634

Edge, R. (2016). Embedding metadata in large-scale legacy digital audio collections. In *Archiving 2016 Final Program and Proceedings* (S. 156–160)

Eisma, Y. B., Hancock, P. A., & de Winter, J. C. (2023). On Sander's models of visual sampling behavior. *Human Factors*, 65(5), 723–736. https://doi.org/10.1177/0018720820959956

Electron Documentation. (n.d. *Docs (latest)*. https://www.electronjs.org/docs/latest (aufgerufen am 14.02.26).

European Broadcasting Union. (2023a). *EBU R 128: Loudness normalisation and permitted maximum level of audio signals* (Version November 2023). https://tech.ebu.ch/docs/r/r128.pdf

European Broadcasting Union. (2023b). *EBU Tech 3341: Loudness metering: "EBU Mode" metering to supplement EBU R 128 loudness normalization* (Version November 2023). https://tech.ebu.ch/docs/tech/tech3341.pdf

Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., & Hallam-Baker, P. (2013). *Naming things with hashes* (RFC 6920). RFC Editor. https://www.rfc-editor.org/rfc/rfc6920

Federal Agencies Audio-Visual Working Group. (2012). *Embedding metadata in digital audio files: Guideline for Federal Agency use of Broadcast WAVE files* (Version 2, 23. April 2012). https://sustainableheritagenetwork.org/system/files/atoms/file/Audio_Embed_Guidelines.pdf

Griscom, R. (2006). *Digital audio in the library* (Version 1.11). University of Pennsylvania ScholarlyCommons. https://repository.upenn.edu/bitstreams/cf5ad04b-a665-49a9-b150-c68785f60246/download

Grundström, E. (2013). *Rounding errors in floating point audio: Investigating the effects of rounding errors on the fixed point output format of a simulated digital audio chain, using fixed point input, and floating point intermediate storage* [Bachelorarbeit, Luleå University of Technology]. https://www.diva-portal.org/smash/get/diva2:1022814/FULLTEXT02.pdf

Gustafsson, H., Nordholm, S., & Claesson, I. (1999). Spectral subtraction with adaptive averaging of the gain function. In Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH'99). https://doi.org/10.21437/Eurospeech.1999-655

International Association of Sound and Audiovisual Archives (IASA). (2009). Guidelines on the production and preservation of digital audio objects (2nd ed., IASA-TC 04). https://www.iasa-web.org/tc04/audio-preservation

International Telecommunication Union. (2017). *Recommendation ITU-R BS.1770-4: Algorithms to measure audio programme loudness and true-peak audio level*. https://www.itu.int/rec/R-REC-BS.1770-4-201510-I/en

iZotope. (o. D.). *Spectral De-noise*. https://www.izotope.com/en/products/rx/features/spectral-denoise (aufgerufen am 12.02.2026).

Maher, R. C. (2003). Lossless compression of audio data. In K. Brandenburg & M. Kahrs (Eds.), Applications of Digital Signal Processing to Audio and Acoustics. Springer.

Marks, A. (2010, October 22). Aaron Marks Special: A Practical Guide to Field Recording [Part 1]. Designing Sound. https://designingsound.org/2010/10/22/aaron-marks-special-a-practical-guide-to-field-recording-part-1/

Kester, W. (2009). Taking the Mystery out of the Infamous Formula, "SNR = 6.02N + 1.76dB," and Why You Should Care (MT-001). Analog Devices. https://docs.ampnuts.ru/analog.com.datasheet/ad7671/related_data/MT-001.pdf

Kosonocky, S., & Xiao, P. (1999). Analog-to-digital conversion architectures. In V. K. Madisetti & D. B. Williams (Hrsg.), *Digital signal processing handbook* (S. 1). CRC Press.

Laroche, J., & Dolson, M. (2000). *New phase-vocoder techniques for real-time pitch-shifting, chorusing, harmonizing and other exotic audio modifications* (Preprint). https://ece.uvic.ca/~peterd/48409/Dobson1999.pdf

Library of Congress. (2021). ID3 metadata for MP3, version 2 (Format description). https://www.loc.gov/preservation/digital/formats/fdd/fdd000108.shtml

Lund, T. (2007). Level and distortion in digital broadcasting. *EBU Technical Review*, April. https://tech.ebu.ch/files/live/sites/tech/files/shared/techreview/trev_310-lund.pdf

Lund, T. (2011, April). ITU-R BS.1770 revisited. In *Proceedings of the NAB 2011 Convention*.

Matzik, A., & Anwar, S. (2016). Review of electrical filters. *IJISET – International Journal of Innovative Science, Engineering & Technology*, 3(4), 543–551.

Moller, H., & Pedersen, C. S. (2004). Hearing at low and infrasonic frequencies. *Noise and Health*, 6(23), 37–57.

Neuhaus, P., Shlezinger, N., Dörpinghaus, M., Eldar, Y. C., & Fettweis, G. (2021). Task-based analog-to-digital converters (arXiv:2009.14088). *arXiv*. https://arxiv.org/pdf/2009.14088.pdf

Nordström, J. (2017). *Real time digital signal processing using Matlab* [Bachelorarbeit, Uppsala University]. https://www.diva-portal.org/smash/get/diva2:1151549/FULLTEXT01.pdf

OpenStax. (2016). *University Physics Volume 1 (Chapter 16: Waves)*. Rice University. https://openstax.org/books/university-physics-volume-1/pages/16-introduction

Pro Sound Effects. (n.d.). SoundQ - Sound Library Software. https://www.prosoundeffects.com/soundq (aufgerufen am 20.02.2026).

Python Software Foundation. (2026a). *tkinter — Python interface to Tcl/Tk*. https://docs.python.org/3/library/tkinter.html (aufgerufen am 05.02.2026)

Python Software Foundation. (2026b). *tkinter.ttk — Tk themed widgets*. https://docs.python.org/3/library/tkinter.ttk.html (aufgerufen am 05.02.2026).

Riverbank Computing. (2026). *What is PyQt?* https://www.riverbankcomputing.com/software/pyqt/ (aufgerufen am 06.02.2026).

RØDE Microphones. (n.d.). *NT1-A data sheet (V03)* [PDF]. https://edge.rode.com/pdf/products/69/nt1a_ds_V03.pdf (aufgerufen am 01.02.2026).

Scharine, A. A., & Letowski, T. R. (2012). Auditory perception and cognitive performance. In T. R. Letowski (Ed.), Helmet-mounted displays: Sensation, perception and cognition issues (pp. 511-571). U.S. Army Aeromedical Research Laboratory.

Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1), 10–21. https://doi.org/10.1109/JRPROC.1949.232969

Smith, K. R., Saunders, S., & Kejser, U. B. (2014). Making the case for embedded metadata in digital images. In *Archiving 2014 Final Program and Proceedings* (S. 52–56). Society for Imaging Science and Technology.

Sound Devices. (2020). *How is a 32-bit float file recorded?* https://www.sounddevices.com/how-is-a-32-bit-float-file-recorded/ (aufgerufen am 23.02.2026).

Soundly. (n.d.). The Complete Sound Effect Platform. https://getsoundly.com/ (aufgerufen am 20.02.2026).

Sound Particles. (n.d.). Explorer | FREE Audio File Manager for Efficient Sound Organization. https://soundparticles.com/products/explorer (aufgerufen am 20.02.2026).

SQLite Consortium. (2024). *SQLite Foreign Key Support*. https://www.sqlite.org/foreignkeys.html (aufgerufen am 08.02.2026).

SQLite Consortium. (2025). *Datatypes in SQLite*. https://www.sqlite.org/datatype3.html (aufgerufen am 08.02.2026).

SQLite Documentation. (2024). *Frequently Asked Questions*. https://www.sqlite.org/faq.html (aufgerufen am 08.02.2026).

SQLite Documentation. (2025a). *Query Planning*. https://www.sqlite.org/queryplanner.html (aufgerufen am 08.02.2026).

SQLite Documentation. (2025b). *Appropriate Uses For SQLite ("When To Use SQLite")*. https://www.sqlite.org/whentouse.html (aufgerufen am 08.02.2026).

Terrell, M., Reiss, J. D., & Sandler, M. (2010). Automatic noise gate settings for drum recordings containing bleed from secondary sources. *EURASIP Journal on Advances in Signal Processing*, 2010, Article 465417. https://doi.org/10.1155/2010/465417

The Broadcast Bridge. (2025, 24. Februar). Standards: Part 16 - About MP3 audio coding & ID3 metadata. https://www.thebroadcastbridge.com/content/entry/20759/ standards-part-16-about-mp3-audio-coding-id3-metadata (aufgerufen am 15.02.2026).

Tomić, S., & Drljača, D. (2023). Digitalization of Sound Using Pulse Code Modulation (PCM). *JITA - Journal of Information Technology and Applications*, 13(1), 42-47. https://jita-au.com/wp-content/uploads/2024/03/9887-Article-Text-22677-1-10-20230715.pdf

van Beurden, M. Q. C., & Weaver, A. (2024). *Free lossless audio codec (FLAC)* (RFC 9639). RFC Editor. https://www.rfc-editor.org/rfc/rfc9639

Walter, P. L. (2014). *Guidance for the filtering of dynamic force, pressure, acceleration (and other) signals* (Technical Note 29). PCB Piezotronics, Inc. https://www.pcb.com/contentstore/mktg/LinkedDocuments/Technotes/PCB%201213_TechNote%20TN29_Lowres.pdf

Watt, A., & Eng, N. (2014). *Database design* (2nd ed.). BCcampus. https://opentextbc.ca/dbdesign01/

Waves. (o. D.). *WLM Plus Loudness Meter*. https://www.waves.com/plugins/wlm-loudness-meter (aufgerufen am 19.02.2026).

wxWidgets. (2026). *Overview (About wxWidgets)*. https://wxwidgets.org/about/ (aufgerufen am 07.02.2026).

Yost, W. A. (2015). Psychoacoustics: A brief historical overview. *Acoustics Today*, 11(3), 46–53. https://acousticstoday.org/wp-content/uploads/2015/08/Psychoacoustics-A-Brief-Historical-Overview.pdf

Zoom Corporation. (2017). *H1n Handy Recorder Specifications* [PDF]. https://zoomcorp.com/media/documents/E_H1n_Specifications.pdf (aufgerufen am 03.02.2026).

Anhang

A.1 Ziel und Auswahl der Code-Ausschnitte

Im Folgenden werden ausgewählte Code-Ausschnitte der Implementierung dokumentiert. Die vollständige Implementierung umfasst mehrere Python-Module und kann aus Platzgründen nicht vollständig abgebildet werden. Daher werden im Folgenden die für die Arbeit zentralen Teile dargestellt, die die wesentlichen Architekturentscheidungen und Kernfunktionen des Prototyps nachvollziehbar machen. Der vollständige Projektstand mit allen Quellcodedateien ist im zugehörigen Zenodo-Archiv dokumentiert und über den bereitgestellten Link abrufbar.

Die Auswahl orientiert sich an den folgenden Kriterien:

- Relevanz für die Kernfunktionen (Datenbankaufbau, Scan/Import, Suche/Filterung, Metadatenpflege)
- Nachvollziehbarkeit der Architektur (Trennung von GUI, Zugriffsschicht und Scan-Logik)
- Abbildung zentraler Entwurfsentscheidungen (z.B. Pfadnormalisierung, Schutz manueller Kategorien, Tag-Logik)

Die gezeigten Listings sind auszugsweise dargestellt und wurden teilweise gekürzt, sofern dies die Verständlichkeit nicht beeinträchtigt.

A.2 Erstellung und Initialisierung der SQLite-Datenbank

Dieses Listing zeigt die Erzeugung des Datenbankschemas (Tabellen, Fremdschlüssel, Indizes) sowie die initiale Anlage der Standardkategorie. Es bildet die Grundlage der gesamten Sound-Library. Die Schema-Erzeugung erfolgt in `db.py` über `create_empty_db(…)`.

```
def create_empty_db(db_path: Path, overwrite: bool = False) -> None:
    """Create a fresh DB with schema + initial categories."""
    db_path.parent.mkdir(parents=True, exist_ok=True)

    if db_path.exists() and overwrite:
        db_path.unlink()

    con = sqlite3.connect(db_path)
    try:
        con.row_factory = sqlite3.Row
        con.execute("PRAGMA foreign_keys = ON;")

        con.executescript(
            """
            BEGIN;

            DROP TABLE IF EXISTS sound_tags;
            DROP TABLE IF EXISTS tags;
            DROP TABLE IF EXISTS sounds;
            DROP TABLE IF EXISTS categories;

            CREATE TABLE categories (
              category_id INTEGER PRIMARY KEY AUTOINCREMENT,
              name        TEXT NOT NULL UNIQUE
            );

            CREATE TABLE sounds (
              sound_id          INTEGER PRIMARY KEY AUTOINCREMENT,

              file_path         TEXT NOT NULL UNIQUE,
              sha256            TEXT,

              title             TEXT,
              description       TEXT,

              category_id       INTEGER NOT NULL REFERENCES categories(category_id)
ON DELETE RESTRICT,
              category_locked   INTEGER NOT NULL DEFAULT 0,

              container_format  TEXT NOT NULL DEFAULT 'WAV',
              sample_rate       INTEGER,
              bit_depth         INTEGER,
              channels          INTEGER,
              duration_seconds  REAL,

              file_size_bytes   INTEGER,
              file_mtime        TEXT,

              rating            INTEGER CHECK (rating IS NULL OR (rating >= 1 AND
rating <= 5)),

              created_at        TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
              updated_at        TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP
            );
```

```
            CREATE INDEX idx_sounds_category_id ON sounds(category_id);
            CREATE INDEX idx_sounds_sha256     ON sounds(sha256);
            CREATE INDEX idx_sounds_updated_at ON sounds(updated_at);

            CREATE TABLE tags (
              tag_id  INTEGER PRIMARY KEY AUTOINCREMENT,
              name    TEXT NOT NULL UNIQUE
            );

            CREATE TABLE sound_tags (
              sound_id  INTEGER NOT NULL REFERENCES sounds(sound_id) ON DELETE
CASCADE,
              tag_id    INTEGER NOT NULL REFERENCES tags(tag_id)     ON DELETE
CASCADE,
              PRIMARY KEY (sound_id, tag_id)
            );

            CREATE INDEX idx_sound_tags_sound_id ON sound_tags(sound_id);
            CREATE INDEX idx_sound_tags_tag_id   ON sound_tags(tag_id);

            COMMIT;
            """
        )

        categories = [
            "UNCATEGORIZED",

        ]
        con.executemany(
            "INSERT OR IGNORE INTO categories (name) VALUES (?);",
            [(c,) for c in categories],
        )
        con.commit()
    finally:
        con.close()
```

A.3 Auslesen technischer WAV-Metadaten und Hash-Bildung

Dieses Listing zeigt die technische Metadatenextraktion aus WAV-Dateien (Kanäle, Abtastrate, Bittiefe, Dauer) sowie die Berechnung des SHA256-Hashes.

```
def sha256_file(path: Path, chunk_size: int = 1024 * 1024) -> str:
    """SHA-256 Hash über den Dateiinhalt (streaming, RAM-schonend)."""
    h = hashlib.sha256()
    with path.open("rb") as f:
        while True:
            chunk = f.read(chunk_size)
            if not chunk:
                break
            h.update(chunk)
    return h.hexdigest()


def read_wav_tech_meta(path: Path) -> dict:
    """Liest technische Metadaten aus einer WAV-Datei + SHA256."""
    if not path.exists():
        raise FileNotFoundError(path)
    if path.suffix.lower() != ".wav":
        raise ValueError("Keine WAV-Datei")

    with wave.open(str(path), "rb") as wf:
        channels = wf.getnchannels()
        sample_rate = wf.getframerate()
        frames = wf.getnframes()
        sample_width = wf.getsampwidth()  # bytes
        duration = frames / sample_rate
```

```
    return {
        "channels": int(channels),
        "sample_rate": int(sample_rate),
        "bit_depth": int(sample_width * 8),
        "frames": int(frames),
        "duration_sec": round(float(duration), 3),
        "sha256": sha256_file(path),
    }
```

A.4 Move-Detection im Scan-Prozess

Dieses Listing zeigt den relevanten Ausschnitt des Scan-Prozesses mit SHA-basierter Move-Detection. Wird beim Scannen eine Datei mit bereits bekanntem Hash gefunden, kann ein verschobener Datensatz erkannt und der Dateipfad aktualisiert werden, anstatt einen neuen Eintrag anzulegen.

```
def scan_once(root: Path | None = None, cleanup_missing: bool = True) -> tuple[int,
int, int]:
    """
    Scan WAV files under root and UPSERT into DB.

    Returns: (ok, failed, deleted_missing)
    """
    root = (root or sounds_root()).resolve()

    if not root.exists() or not root.is_dir():
        raise FileNotFoundError(f"Ordner nicht gefunden oder kein Ordner: {root}")

    # Sync categories from folders (create empty categories, prune missing unused)
    sync_categories_with_fs(root, prune_missing=True)

    deleted_missing = delete_missing_sounds(base_root=root) if cleanup_missing else
0
    ok = 0
    failed = 0

    for wav in iter_wavs(root):
        if is_ignored(wav):
            continue

        try:
            cat_name = category_from_folder(root, wav)
            cat_id = get_or_create_category_id(cat_name)

            meta = read_wav_tech_meta(wav)

            # Optional: Move-Detection über SHA256
            sha = meta.get("sha256")
            if sha:
                hits = find_sounds_by_sha256(sha)
                if len(hits) == 1:
                    sound_id, old_path_str = hits[0]
                    old_path = Path(old_path_str)
                    if not old_path.is_absolute():
                        old_path = (PROJECT_DIR / old_path).resolve()

                    if (not old_path.exists()) and (not sound_exists_for_path(wav,
base_root=root)):
                        update_sound_file_path(sound_id, wav)

            upsert_sound_minimal(wav, cat_id, meta)
            ok += 1

        except Exception as ex:
            failed += 1
            print(f"[FEHLER] {wav} -> {type(ex).__name__}: {ex}")
    # Final prune in case missing sounds were deleted
```

```
    sync_categories_with_fs(root, prune_missing=True)


    return ok, failed, deleted_missing
```

A.5 Pfadnormalisierung und UPSERT-Logik mit Schutz manueller Kategorien (`db.py`)

Das folgende Listing zeigt die zentrale Datenbankfunktion `upsert_sound_minimal(…)`, die Sound-Datensätze einfügt und aktualisiert. Besonders relevant ist die `UPSERT`-Logik mit `ON CONFLICT(file_path)` in Kombination mit dem Sperr-Flag `category_locked`.

```
def _to_rel_db_path(file_path: Path, base_root: Path | None = None) -> str:

    p = Path(file_path)

    # Normalize separators early (important on Windows)
    # (Path will keep backslashes, but our DB should always use '/')
    if not p.is_absolute():
        # Already a relative DB-style path; keep it relative.
        # Strip a leading './' if it ever sneaks in.
        return p.as_posix().lstrip("./")

    p = p.expanduser().resolve()
    root = (base_root or sounds_root()).expanduser().resolve()
    try:
        rel = p.relative_to(root)
        return rel.as_posix()
    except ValueError:
        return p.as_posix()


def upsert_sound_minimal(file_path: Path, category_id: int, tech_meta: dict[str,
Any], base_root: Path | None = None) -> None:
    """Insert or update a sound (UNIQUE: file_path).
    Important: category_id is only overwritten if category_locked=0.
    """
    sql = """
    INSERT INTO sounds (
        file_path,
        category_id,
        container_format,
        sample_rate,
        bit_depth,
        channels,
        duration_seconds,
        sha256,
        file_size_bytes,
        file_mtime,
        updated_at
    )
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP)
    ON CONFLICT(file_path)
    DO UPDATE SET
        category_id       = CASE
                                WHEN sounds.category_locked = 1 THEN
sounds.category_id
                                ELSE excluded.category_id
                            END,
        sample_rate       = excluded.sample_rate,
        bit_depth         = excluded.bit_depth,
        channels          = excluded.channels,
        duration_seconds  = excluded.duration_seconds,
        sha256            = excluded.sha256,
        file_size_bytes   = excluded.file_size_bytes,
        file_mtime        = excluded.file_mtime,
        updated_at        = CURRENT_TIMESTAMP;
```

```
    """

    stat = file_path.stat()
    file_mtime_iso =
datetime.fromtimestamp(stat.st_mtime).isoformat(timespec="seconds")

    con = get_conn()
    try:
        con.execute(
            sql,
            (
                _to_rel_db_path(file_path, base_root=base_root),
                int(category_id),
                str(tech_meta.get("container_format") or "WAV"),
                tech_meta.get("sample_rate"),
                tech_meta.get("bit_depth"),
                tech_meta.get("channels"),
                tech_meta.get("duration_sec"),
                tech_meta.get("sha256"),
                int(stat.st_size),
                file_mtime_iso,
            ),
        )
        con.commit()
    finally:
        con.close()
```