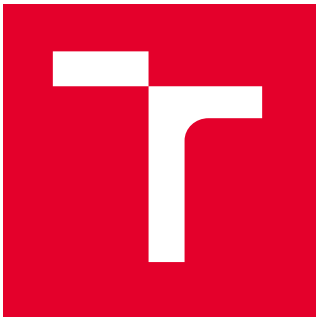


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

WEBOVÁ APLIKACE PRO VIZUALIZACI PARAMETRŮ HUDEBNÍCH NAHRÁVEK

WEB APPLICATION FOR VISUALIZATION OF MUSIC RECORDING PARAMETERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Klimeš**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Štěpán Miklánek**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Bc. Martin Klimeš ***ID:*** 203733
Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Webová aplikace pro vizualizaci parametrů hudebních nahrávek

POKYNY PRO VYPRACOVÁNÍ:

V teoretické části práce se zaměřte na zmapování současného stavu v oblasti vývoje webových aplikací nejdříve v obecné rovině a následně ve spojení s vědní oblastí Music Information Retrieval (MIR). V praktické části práce implementujte webovou aplikaci, pomocí které bude možné pracovat s databází hudebních nahrávek. Kromě běžné práce s audio soubory, se zaměřte hlavně na přehlednou vizualizaci zvukových parametrů. Webové rozhraní bude určeno pro porovnávání více interpretací vybrané skladby klasické hudby a bude možné provádět analýzu vybraných hudebních úseků za pomoci časové anotace začátků taktů. Aplikace bude umožňovat základní práci s audio soubory, vizualizaci časového průběhu nahrávek a zobrazení spektrogramů získaných pomocí výpočtu krátkodobé Fourierovy transformace. Všechna zobrazení bude možné omezit na uživatelem zvolený počet taktů nebo ručně zvolený časový rozsah. Aplikace bude take umožňovat vizualizaci dalších parametrů, které popisují hudební nahrávky z hlediska rytmické struktury, harmonického vývoje a dynamiky.

DOPORUČENÁ LITERATURA:

[1] Vue.js - The Progressive JavaScript Framework [online]. [cit. 2022-07-13]. Dostupné z: https://vuejs.org/
[2] MÜLLER, Meinard. Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications. Cham: Springer International Publishing, 2015. ISBN 978-3-319-21945-5.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 19.5.2023

Vedoucí práce: Ing. Štěpán Miklánek

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato diplomová práce se zaměřuje na vývoj webové aplikace pro vizualizaci hudebních parametrů. Cílem je poskytnout uživateli prostředí, ve kterém může snadno vizualizovat parametry libovolné hudební nahrávky a v případě různých interpretací skladby tyto parametry mezi sebou porovnat. Hudební parametry vizualizované v aplikaci vycházejí z oblasti Music Information Retrieval. Pro každou z těchto vizualizací jsou v aplikaci implementována různá nastavení, která se ukládají do databáze pro přihlášeného uživatele a umožňují tak upravit zobrazení vizualizací podle individuálních potřeb uživatele. Pro vývoj byl použit reaktivní framework Vue.js pro klientskou část, Flask framework pro serverovou část a relační databázový systém PostgreSQL pro ukládání dat.

KLÍČOVÁ SLOVA

webová aplikace, vizualizace, hudební parametry, Music Information Retrieval, Vue.js, Flask, PostgreSQL

ABSTRACT

This thesis focuses on the development of a web application for visualizing musical parameters. The goal is to provide users with an environment where they can easily visualize parameters of any music recording and compare these parameters across different interpretations of the composition. The musical parameters visualized in the application are based on the field of Music Information Retrieval. For each of these visualizations, the application implements various settings that are saved to a database for the logged-in user, allowing them to adjust the visualization display according to their individual needs. The reactive Vue.js framework was used for the client-side, Flask framework for the server-side, and the PostgreSQL relational database system for data storage.

KEYWORDS

web application, music visualization, Music Information Retrieval, Vue.js, Flask, PostgreSQL

Vysázeno pomocí balíčku `thesis` verze 4.07; `http://latex.feec.vutbr.cz`

KLIMEŠ, Martin. *Webová aplikace pro vizualizaci parametrů hudebních nahrávek*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 61 s. Diplomová práce. Vedoucí práce: Ing. Štěpán Miklánek

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Martin Klimeš
VUT ID autora:	203733
Typ práce:	Diplomová práce
Akademický rok:	2022/23
Téma závěrečné práce:	Webová aplikace pro vizualizaci parametrů hudebních nahrávek

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Štepánu Miklánkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Cílem této práce je vytvořit webovou aplikaci sloužící k vizualizaci parametrů hudebních nahrávek. Důvod pro implementaci této aplikace je poskytnout uživateli prostředí, ve kterém má možnost snadno vizualizovat parametry libovolné hudební nahrávky a v případě různých interpretací skladby tyto parametry mezi sebou porovnat. Další motivací byla absence podobného nástroje ve webovém prostředí.

V úvodní části práce je obecně popsána problematika webových aplikací. Je zmíněna stručná historie jejich vývoje a dříve používaných technologií. Dále jsou popsány technologie, které se v současnosti využívají pro tvorbu frontendu a backendu. Následně jsou představeny využité technologie pro tvorbu této aplikace

Hudební parametry vizualizované v aplikaci budou vycházet z oblasti Music Information Retrieval (MIR). Ta si klade za cíl objektivně zkoumat parametry nahrávek, které nelze jednoznačně vyhodnotit pouze poslechem. Tato oblast bude krátce představena v následující kapitole. Poté bude uveden přehled již existujících nástrojů pro extrakci a vizualizaci hudebních parametrů. Další kapitola je pak věnována samotným parametrům, které je aplikace schopna vizualizovat. Jsou zde popsány nástroje a knihovny, pomocí kterých jsou vizualizace implementovány. Dále pak všechna nastavení, které je možné na vizualizace aplikovat. V poslední kapitole se práce zaměří na popis architektury aplikace.

Výstupem této diplomové práce bude webová aplikace určená pro práci s hudebními nahrávkami a přehlednou vizualizaci zvukových parametrů. Aplikace bude umožňovat vizualizaci parametrů, mezi které patří např. časový průběh zvukového signálu, spektrogram, pianoroll, tempo, efektivní hodnota zvukového signálu, detekované začátky not a dob a intervaly mezi takty a detekovanými začátky not i dob. Pro každou z těchto vizualizací budou v aplikaci implementována různá nastavení, která se pro přihlášeného uživatele ukládají do databáze a umožňují tak upravit zobrazení vizualizací podle individuálních potřeb uživatele.

1 Vývoj webových aplikací

Při popisu webové aplikace lze mluvit o programu, který je uživatelům poskytován prostřednictvím webového serveru. K webové aplikaci uživatel přistupuje pomocí webového prohlížeče. Webový prohlížeč tedy slouží k získání požadovaného obsahu z celosvětové sítě (World Wide Web), což zahrnuje stahování textů, obrázků, videí a jiných dat z internetu. Dále pak zobrazování obsahu na zařízení uživatele a odesílání dat uživatele zpět na server [11].

1.1 Vlastnosti webových aplikací

Nároky na dnešní webové aplikace jsou poměrně vysoké. Je od nich očekáváno, že budou k dispozici kdykoliv a kdekoliv, a to na většině zařízení s různou velikostí a rozlišením obrazovky. Webové aplikace by měly být také snadno použitelné a intuitivní, aby se uživatelé mohli rychle a efektivně dostat k požadovaným funkcím. Díky své univerzálnosti se těší čím dál větší popularitě [12]. Už jen díky absenci instalace mohou být pro uživatele mnohem atraktivnější než klasické desktopové aplikace.

K zajištění interaktivity a dynamického chování webových aplikací slouží programovací jazyk JavaScript, který je interpretován ve webovém prohlížeči klienta a umožňuje manipulaci s obsahem stránky, reakce na události a interakci s uživatelem. Po rozšíření webových aplikací se JavaScript stal i serverovým jazykem na straně klienta v prostředí Node.js viz 1.3.9. Díky tomu, že je přístup k aplikaci poskytován pomocí webového prohlížeče, je její vývoj i následná údržba poměrně snadná záležitost. Nová aktualizace aplikace se uživatelům projeví prakticky okamžitě bez nutnosti celou aplikaci přeinstalovat [11].

1.2 Nástroje pro vývoj webových aplikací

Vzhledem k dřívějším nárokům na obsah internetu si uživatelé bohatě vystačili se statickými stránkami. S rostoucím počtem uživatelů a rychlejšímu připojení se však tvůrci webu více a více snažili stránky rozvíjet a internet se tak začal stávat dynamičtějším místem [12].

Webové aplikace se skládají ze dvou částí: backend (serverová část) a frontend (klientská část). Frontend určuje uživatelské rozhraní, se kterým uživatel přímo interaguje. Naproti tomu backend slouží ke generování dynamického obsahu podle konkrétních požadavků každého uživatele.

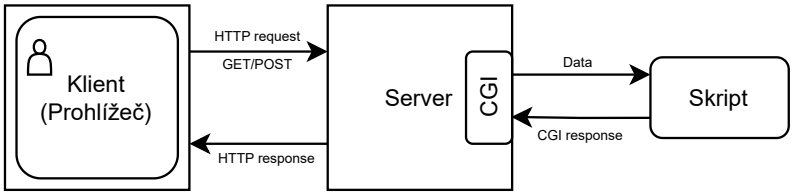
1.2.1 Backend

Server Side Includes (SSI)

Server Side Includes (SSI) byla první technologie, která přišla s tím, že by bylo možné kromě statických stránek generovat i dynamický obsah. V principu se jedná o jednoduchý skriptovací jazyk, který se provádí přímo na serveru těsně před odesláním HTML kódu zpět uživateli do prohlížeče. Syntakticky se SSI do HTML kódu zapisuje pomocí HTML komentáře `<!-- příkaz -->`. Možnosti příkazů SSI bylo například zjištění velikosti souboru, načtení externího souboru nebo spuštění externího programu.

Common Gateway Interface (CGI)

Pro psaní webových aplikací však bylo třeba zasílat data z aplikace přímo na server. K tomu dopomohla technologie Common Gateway Interface (CGI), na jejíž principech staví programovací jazyky, které na serveru obsluhují požadavky přicházející z prohlížeče a zpracovávají data z formuláře viz obr. 1.1.



Obr. 1.1: Schéma fungování CGI protokolu.

Princip je takový, že uživatel vyplní formulář ve webovém prohlížeči a po jeho odeslání na server se pošle požadavek obsahující data z formuláře. Rozhraní CGI pak definuje způsob, jakým webový server předává data nějaké externí aplikaci (skript). Tato aplikace následně zpracuje data ze serveru a uživateli navrátí statickou stránku.

Na počátku 90. let neexistovaly specializované programovací jazyky pro tvorbu webu, a tak bylo možné CGI skripty programovat v jakémkoliv jazyce [19]. Pro předávání dat pomocí rozhraní CGI se používají dvě metody – GET/POST. U metody GET se zakódovaná data přidají za otazník v URL požadavku. Otazník slouží jako oddělovač jména skriptu a parametrů. V metodě POST se data předají v těle HTTP požadavku viz 1.3.7.

Nevýhodou rozhraní CGI byl velký nárok na výpočetní výkon, jelikož se při každém požadavku od uživatele musel skript opětovně načítat do paměti. Tento problém vyřešila varianta FASTCGI, kdy skript zůstane načtený v paměti, a pak postupně obsluhuje další požadavky od uživatele [19, 20].

Hypertextový preprocesor (PHP)

Jedná se o skriptovací programovací jazyk, který byl vyvinut primárně pro dynamické webové stránky. Odlišnost oproti JavaScriptu je ta, že se PHP skripty provádí výhradně na straně serveru. Zde se vygeneruje HTML viz 1.2.3, které se následně pošle zpět klientovi.

Za zmínku některých PHP frameworků stojí Laravel nebo Symfony. PHP mimo jiné pohání nejpopulárnější systém určený pro tvorbu webových stránek WordPress [21].

Ruby

Ruby je objektově orientovaný skriptovací jazyk. Byl navržený pro efektivní a jednoduchou práci (má poměrně snadnou syntaxi). Tvůrcem tohoto jazyka je japonský programátor Yukihiro Matsumot. Mimo Japonsko se Ruby nesetkal s významnou popularitou až do vydání webového frameworku Ruby on Rails, který umožnil rychlý vývoj mnoha webových aplikací [19].

1.2.2 Backend frameworky v programovacím jazyce Python

Obecně smyslem existence frameworků je ulehčení vývoje aplikací. Správný framework bychom měli zvolit podle typu projektu, dostupnosti knihoven, svobodě při jeho používání, kvalitní dokumentaci, rychlosti a uživatelské komunitě. Vzhledem k menšímu rozsahu aplikace, a ne příliš velkým nárokům na backend, byl zvolen framework Flask. Tento framework je popsán v kapitole 1.3.3.

Django

Django je open source komplexní webový framework napsaný v jazyce Python. Zakládá se na architektuře Model-View-Controller (MVC). V jednoduchosti se jedná o oddělení logiky do tří komponent: logika (Model), pohled (View) a kontroler (Controller). V modelu probíhají výpočty, dotazy na databázi, validace dat atd. Pohled vstupní data pouze zobrazí uživateli a kontroler funguje jako zprostředkovatel mezi všemi komponentami a řídí tok dat mezi nimi.

1.2.3 Frontend

Klasické desktopové aplikace existují již několik desetiletí. Naproti tomu vývoj webových aplikací začal nabývat na významu až v průběhu 90. let a rapidně se rozvíjel v průběhu posledních dvou desetiletí. Kromě samotného JavaScriptu, který získal

nové funkce se široce rozšířila škála nejrůznějších frontend frameworků [19]. Následující odstavce ve stručnosti popíší jazyky, se kterými se frontend webových aplikací běžně vytváří, a které byly použity pro vývoj aplikace v této práci.

HTML

HTML je zkratka pro Hypertext Markup Language. Jedná se o značkovací jazyk, který se využívá pro tvorbu webových stránek a webových aplikací. Hypertext znamená propojení dvou či více stránek mezi sebou pomocí odkazů. Markup Language (značkovací) označuje počítačový jazyk, který se stará o strukturu a formátování textového dokumentu.

Jazyk HTML se skládá z množství značek tzv. tagů a jejich vlastností neboli atributů. Vytvořené stránky pouze pomocí jazyka HTML by byly statické a uživatel by s nimi nemohl nijak interagovat. K zajištění interaktivity stránek se v prohlížečích používá skriptovací jazyk JavaScript a k tvorbě designu CSS.

CSS

CSS je zkratka pro Cascading Style Sheets (kaskádové styly). Použití CSS umožňuje webovým stránkám přidat vizuální prvky a vylepšit jejich vzhled a uspořádání. Díky CSS je možné specifikovat rozvržení a styly různých HTML elementů, jako jsou nadpisy, odstavce, obrázky a tlačítka. Toho je docíleno pomocí přiřazení CSS tříd k HTML elementům. Díky CSS třídám je možné opakované využití předdefinovaných stylů na různých elementech. CSS může být využit rovněž i pro animace [11].

Pro účely této práce byl využit framework Windi CSS. Jedná se o framework pro stylování webů přes tzv. utility třídy, což jsou jednoduché předpřipravené CSS třídy. Díky němu je možné aplikovat příslušné třídy přímo na HTML elementy [13].

JavaScript

JavaScript je skriptovací programovací jazyk, který se ve velké míře používá k zajištění interaktivity webových stránek. Umožňuje na webové stránky implementovat dynamické vlastnosti, které by s pouhým HTML a CSS nebyly možné. Pro pohodlnější vývoj frontendu webových aplikací se využívají JavaScriptové frameworky. Mezi nejznámější patří React, Angular a Vue. Kromě použití na frontendu je možné JavaScript využít také pro vývoj backendu pomocí technologie Node.js viz 1.3.9.

1.3 Použité technologie

1.3.1 Vývojářské prostředí

Integrated Development Environment (IDE) je vývojářské prostředí sloužící k pohodlnějšímu psaní kódu, debugování[1] a automatizaci[2]. Jako IDE pro psaní frontendu aplikace byl zvolen Visual Studio Code (VSC) vyvinutý společností Microsoft. Editor je naprogramovaný v TypeScriptu a JavaScriptu. Pro rychlejší a efektivnější vývoj aplikace byly do VSC doinstalovány následující rozšíření [26]:

- **ESLint** – nástroj, který umožňuje provádět analýzu kódu v jazyce JavaScript a identifikovat problematické části kódu, které by mohly způsobovat chyby,
- **i18n Ally** – tento nástroj usnadňuje tvorbu webových stránek v různých jazycích,
- **Iconify IntelliSense** – rozšíření, které usnadňuje práci s ikonami tím, že je umožňuje upravovat a vkládat přímo v kódu a zobrazovat jejich náhledy,
- **Vue Language Features (Volar)** – jedná se o jazykové rozšíření pro podporu syntaxe Vue.js.
- **WindiCSS IntelliSense** – toto rozšíření pomáhá s automatickým doplňováním kódu, zvýrazňováním syntaxe a vytvářením kódu pro CSS framework WindiCSS, což zrychluje a usnadňuje psaní stylů pro webové stránky.

Naproti tomu backend, psaný v programovacím jazyce Python a frameworku Flask, byl tvořen v IDE s názvem PyCharm. Jedná se o vývojové prostředí pro Python napsané v kombinaci Javy a Pythonu. Pro účely aplikace postačila free verze (Pycharm Community Edition) [28].

Jako vývojářský nástroj pro práci při vývoji aplikace byl zvolen JavaScriptový Vite. Vytvořil jej Evan You, který stojí rovněž i za vytvořením frameworku Vue.js. Hlavním důvodem pro zvolení právě vývojářského serveru Vite byla jeho rychlost při vývoji, díky které jsou změny v kódu ihned viditelné v prohlížeči [23].

1.3.2 Vue.js

Pro psaní frontendu aplikace byl zvolen framework Vue.js. Jedná se o JavaScriptový reaktivní framework pro budování uživatelského rozhraní. Reaktivní znamená, že když dojde ke změně datové proměnné v kódu, Vue.js změnu detekuje a automaticky aktualizuje odpovídající část uživatelského rozhraní bez potřeby manuálního překreslení stránky. Je ideální na vytváření jednostránkových aplikací (viz 1.3.5).

[1] Debugování je proces hledání a opravování chyb v softwaru.

[2] Automatizace v IDE je proces automatického provádění určitých úkolů, které by jinak musel programátor provádět ručně např. dokončování kódu, nápovědu při psaní, optimalizace kódu atd.

Z hlediska MVC architektury (viz 1.2.2) tvoří pouze View vrstvu. Vue.js je jeden ze tří v současnosti nejoblíbenějších a nejpoužívanějších JavaScriptových frameworků. Vue.js je nejmladší a oproti zmíněným starším frameworkům za ním nestojí žádná velká korporace. Svoji cestu si tak Vue.js musel vybudovat svoji kvalitou. Jeho hlavní předností oproti ostatním je jeho jednoduchost a přímočarost. Má vynikající přehlednou dokumentaci, rozsáhlou komunitu a rychle rostoucí ekosystém [22].

Vue.js zobrazuje obsah na stránce pomocí šablon. Šablony jsou v podstatě HTML kódy, které umožňují párovat renderovaný Document Object Model (DOM) s daty ve Vue instanci. DOM je rozhraní umožňující přistupovat k jednotlivým prvkům v HTML stránce pomocí JavaScriptu. Vue.js kompiluje šablony do virtuálního DOM, který umožňuje, aby byly komponenty renderovány předtím, než se aktualizuje prohlížeč.

Vue.js rozšiřuje HTML pomocí takzvaných direktiv. Direktivy poskytují funkcionality do HTML aplikace. Jsou zabudované přímo do Vue.js (zapisují se s předponou `v-` nebo si uživatel může definovat své vlastní.

Jak již bylo zmíněno, při vývoji se aplikace ve Vue.js rozdělí na části. Těmto částem se říká komponenty. Komponenta je vlastně další Vue instance, která má předem dané vlastnosti [25]. Reprezentuje část aplikace, která má vlastní data, JavaScript a často i vlastní styl. Komponenty mohou obsahovat další podkomponenty a komunikovat mezi sebou. Komponenta může být menší část kódu (např. jedno tlačítko), nebo i větší úsek, jako například celý formulář. Každá komponenta může být použita na různých místech v rámci aplikace. Komponenta ve Vue.js je „self-contained“, tzn. soběstačná. Díky tomu máme jistotu, že kód uvnitř jedné komponenty neovlivní kód uvnitř dalších komponent a naopak.

Pro komunikaci mezi komponentami se používají `props` a `emits`. Pomocí `props` je možné předávat data a konfigurace z rodičovské komponenty do dalších komponent. Jako `emits` se poté označují události vyvolávané potomkem rodičovské komponenty, na které mohou rodičovské komponenty reagovat.

Lifecycle hooks

Lifecycle hooks ve Vue.js jsou speciální metody, které jsou volány v různých fázích životního cyklu komponenty. Tyto metody umožňují vývojářům vykonávat určité akce nebo manipulovat se stavem komponenty v přesně definovaných okamžicích. Jedním z příkladů lifecycle hook ve Vue.js je metoda `onMounted()`. Tato metoda je volána, když je komponenta úspěšně vložena do DOM. V této fázi je komponenta již připravena a má přístup k DOM elementům. Tato metoda se často volá například k načtení dat ze serveru.

Composition API

Při vývoji aplikace byl využit způsob zápisu, kterému se říká Composition API. Jedná se o nejnovější a doporučený způsob psaní kódu, který poskytuje několik výhod. Napomáhá lepší organizaci kódu, což umožňuje psát komponenty více modulárně. Díky tomu je správa, rozvoj a testování aplikace snadnější a efektivnější.

Pinia

Pinia je moderní a jednoduchý nástroj pro správu stavu ve Vue.js aplikacích. Stav v tomto případě zahrnuje všechny proměnné a hodnoty, které se mohou v průběhu používání aplikace měnit a ovlivňovat tak zobrazení uživatelského rozhraní. Pinia poskytuje způsob, jak spravovat globální stav aplikace, který umožňuje oddělit stav do menších a snadno spravovatelných částí. Pinia podporuje modulární architekturu, což umožňuje jednoduché řízení stavu v celé aplikaci [22].

1.3.3 Flask

Pro psaní backendu aplikace byl zvolen framework Flask. Flask je webový framework napsaný v jazyce Python. Oproti Djangu se hodí spíše ke zpracování menších projektů. Flask postrádá vlastní ORM (Object Relational Mapping neboli Objektově relační zobrazení), které slouží k automatické konverzi dat mezi objektově orientovaným programovacím jazykem a relační databází (data jsou v databázi sdružována do tabulek). Pro práci s daty je tak nutné použít externí ORM knihovnu, jako například SQLAlchemy, která umožňuje snadnou práci s databázovými dotazy a mapování výsledků na objekty. Flask rovněž nezahrnuje zabudované funkce pro administraci nebo validaci formulářů. Avšak podporuje množství rozšiřujících funkcí, které může programátor implementovat až během vývoje. Flask se tak stává velmi flexibilním nástrojem.

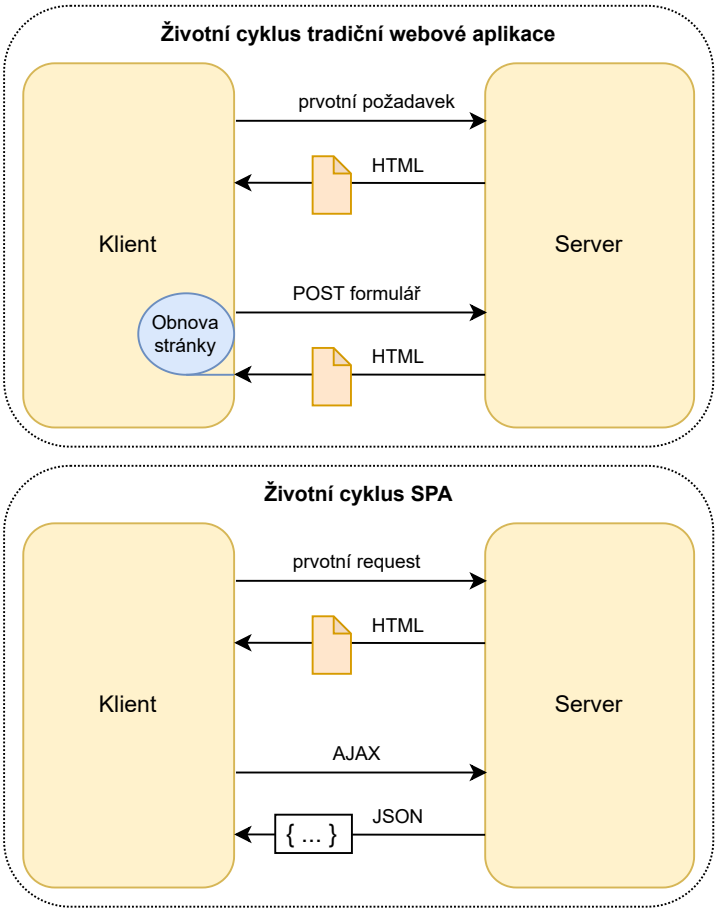
1.3.4 Použitá databáze – PostgreSQL

Databáze slouží jako úložiště, se kterým aplikace komunikuje a do které ukládá data. V případě této aplikace to jsou informace o uživateli a nahraném audio souboru. Flask komunikuje s databází pomocí SQL[3] jazyka. Pro účely této aplikace byl využit databázový systém PostgreSQL. Jak již bylo zmíněno, Flask nemá implementované vlastní ORM. Proto do něj bylo nainstalováno rozšíření SQLAlchemy, které slouží ke komunikaci mezi Flask serverem a databází PostgreSQL. Pro práci s PostgreSQL databází bylo použito grafické prostředí pgAdmin [31].

[3]SQL (Structured Query Language) je dotazovací jazyk, který se používá pro práci s daty v relačních databázích.

1.3.5 Single Page Application

Pro webovou aplikaci vyvíjenou v rámci této práce byl zvolen typ tzv. Single Page Application (SPA). Již podle názvu je zřejmé, že se jedná o jednostránkové aplikace. Princip je takový, že celý program je založený na jediné stránce, která se společně s veškerým JavaScript a HTML kódem načte pouze při prvním spuštění. Důraz tedy klade na klientskou část aplikace a minimalizuje tu serverovou. Se serverem komunikuje pomocí AJAX požadavků viz 1.3.8, které přenášejí data ze serveru ve formátu JSON nebo XML. Výsledkem je plynulá spolupráce aplikace s uživatelem bez potřeby neustále obnovovat stránku, jelikož získaná data se dynamicky zacílí jen na požadované komponenty, a pouze ty se překreslí. U tradičních webových aplikací tomu bylo tak, že kdykoliv klient zavolal požadavek na server, ten vytvořil novou HTML stránku, která se po obnově prohlížeče zobrazila [24]. Rozdíl mezi tradiční webovou aplikací a zmíněnou SPA je uveden na obrázku 1.2.



Obr. 1.2: Rozdíl průběhu SPA a tradiční aplikace.

Nevýhodou SPA je, že vzhledem k velkému množství výpočtů na straně klienta je k hladkému provozu aplikace bezpochyby nutný dostatečný výpočetní výkon. Další nevýhodou je také pomalejší počáteční načtení aplikace. Toto řeší technika zvaná Server-Side Rendering (SSR), která přináší některé výhody tradičních webových aplikací do SPA. SSR umožňuje vykreslování webových stránek na straně serveru před odesláním do prohlížeče. Tím se minimalizuje nutnost provádět výpočty na straně klienta a zlepšuje se rychlost zobrazení obsahu pro uživatele [14].

1.3.6 API

API (Application Programming Interface) je sada definovaných pravidel a postupů, které umožňují komunikaci a interakci mezi různými softwarovými aplikacemi. REST (Representational State Transfer) je architekturální styl pro návrh webových služeb, který se často používá při tvorbě API. REST API je specifický typ API, které následuje principy REST a využívá HTTP protokol pro komunikaci mezi klientem a serverem viz 1.3.7. Pomocí REST API může klient požadovat data nebo provádět operace na serveru, a to jednoduše pomocí standardních HTTP požadavků a odpovědí.

1.3.7 HTTP

HTTP označuje zkratku pro „Hypertext Transfer Protocol“. Jedná se o internetový protokol určený pro přenos hypertextových dokumentů ve formátu HTML, XML, JSON a dalších. HTTP se skládá ze dvou částí: požadavek (request) a odpověď (response). Při zadání URL adresy do prohlížeče se vytvoří požadavek s danou adresou, který se pošle na server. Server následně odesílá odpověď obsahující informace o požadavku a data z požadované URL adresy (v případě, že existují) zpět na stranu klienta [14].

V HTTP protokolu existuje několik různých typů požadavků, ale pro použití v této aplikaci postačí čtyři základní: **GET**, **POST**, **PUT** a **DELETE**. Metoda GET se používá pro získání dat ze serveru. Pokud je třeba předat nějaká data, jsou tyto data přidána na konec URL adresy za symbol otazníku. Tento způsob přenosu dat však není z hlediska bezpečnosti příliš vhodný, protože data jsou veřejně viditelná v URL adrese.

Oproti tomu metoda POST je bezpečnější, protože data jsou součástí samotného HTTP požadavku, nikoliv jen součástí URL adresy. Tato metoda se používá pro odesílání dat na server. Data jsou v požadavku zakódována a mohou být přenášena například ve formátu JSON nebo XML.

Metody PUT a DELETE se používají pro aktualizaci nebo smazání dat na serveru. Metoda PUT umožňuje nahradit existující data novými daty, zatímco metoda DELETE slouží ke smazání existujících dat [14].

1.3.8 AJAX

Jedná se o zkratku pro Asynchronous JavaScript and XML. Pojem „asynchronous“ znamená, že prohlížeč nečeká na odpověď ze serveru, ale je schopen ji zpracovat až tehdy, kdy k ní skutečně dojde. Jinak řečeno, přenosy dat se odehrávají na pozadí, aniž by musel prohlížeč zablokovat běh aplikace. Při spojení se serverem a při zpracování dat přijatých ze serveru hraje klíčovou roli JavaScript, který řídí všechny úkony odehrávající se v prohlížeči.

Poslední písmeno X označuje jazyk XML, který se stal základním komunikačním prostředkem umožňující webu odesílání dat v textové podobě skrze internet. Nicméně brzy se zjistilo, že posílání dat pomocí XML vyžaduje mnohem větší objem dat než je nutné. A tak Douglas Crockford přišel s návrhem nového datového typu založeného na JavaScriptu – JSON (JavaScript Object Notation). JSON je velmi jednoduchý datový formát založený na podmnožině syntaxe JavaScriptu (konkrétně na poli a objektech).

Oproti XML je JSON mnohem více kompaktní a neobsahuje přebytečné množství informací. Protože není nutné, aby koncové značky odpovídaly úvodním značkám, je množství bajtů potřebných k přenosu informací značně zredukováno [14]. Rozdíl mezi JSON a XML je znázorněn ve výpisech 1.1 a 1.2.

Výpis 1.1: Příklad jednoduchého XML kódu.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <student>
    <id>01</id>
    <name>Tom</name>
    <lastname>Novák</lastname>
  </student>
  <student>
    <id>02</id>
    <name>Pavel</name>
    <lastname>Majzlík</lastname>
  </student>
</root>
```

Výpis 1.2: Ekvivalentní kód ve formátu JSON.

```
  "student": [
     {
        "id":"01",
        "name": "Tom",
        "lastname": "Novák"
     },
     {
        "id":"02",
        "name": "Pavel",
        "lastname": "Majzlík"
     }
  ]
}
```

1.3.9 Node.js

Node.js vytvořil Ryan Dahl v roce 2009. Jedná se o platformu postavenou na běhovém prostředí JavaScriptu z prohlížeče Chrome umožňující spouštět JavaScriptový kód mimo webový prohlížeč. Jeho hlavní účel je tvorba backendu webové aplikace, podobně jako tomu je např. u PHP. Oproti PHP je u Node.js kladen důraz na *škálovatelnost*, tzn. schopnost více připojených klientů naráz. Jedním z klíčových prvků Node.js je jeho asynchronní model programování, což umožňuje serveru zpracovávat mnoho požadavků najednou bez zdržování. To znamená, že když Node.js server zpracovává požadavek, může současně pracovat na dalších požadavcích. Díky této

schopnosti může Node.js efektivně obsluhovat více připojených klientů najednou [15].

Node.js používá neblokující vstupně-výstupní model řízený událostmi. To znamená, že Node.js přidělí všechny uživatelské požadavky jednotlivým nezávislým vláknům. Vlákno si je možné představit jako proces, který provádí určité úlohy. Díky tomu je Node.js malá a efektivní platforma pro zpracování datově náročných aplikací, které mají běžet napříč různými zařízeními [15].

Součástí instalace Node.js je NPM – správce knihoven pro JavaScript. Lze jej spustit pomocí příkazové řádky. Pro účely této aplikace byl použit správce balíčků PNPM, což je zkratka pro „Performant NPM“. Jedná se o alternativu k NPM. Hlavní účel PNPM je uchovat všechny balíčky v globálním (centralizovaném) úložišti a poskytovat k nim přístup formou odkazu. Oproti NPM tak šetří množství místa na disku a poskytuje rychlejší instalaci [27].

1.3.10 Axios

Axios je promise-based HTTP klient pro Node.js viz 1.3.9. Promise-based znamená, že konkrétní funkce nevrací okamžitě výsledek, ale vrátí tzv. Promise (slib). Promise je objekt, který slibuje vrácení výsledku někdy v budoucnu, až bude dostupný. Promise se zpracovává pomocí metody `then()`. Pokud dojde k chybě, Promise se zamítne (reject) a chyba se může zpracovat pomocí metody `catch()`. Pomocí metody `finally()` může být vykonán kód, který se musí spustit bez ohledu na to, zda Promise byl splněn nebo zamítnut.

Axios je rovněž izomorfní, tzn. může zasílat požadavky jak z prohlížeče, tak pomocí Node.js. Podporuje automatickou transformaci na JSON data a chrání stranu klienta před CSRF[4] [30].

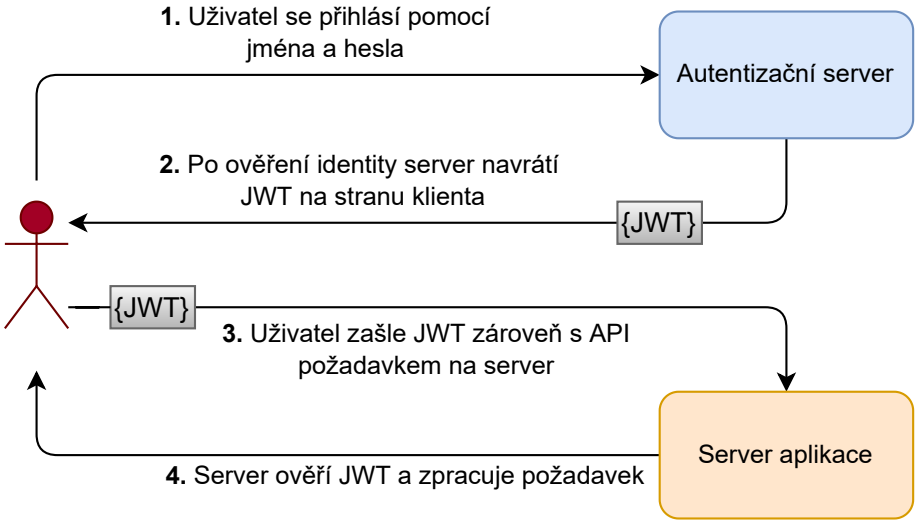
1.3.11 JWT

JSON Web Token (JWT) je standart definující způsob bezpečného přenosu informací mezi stranami ve formě JSON objektů. JWT byl v aplikaci použit za účelem ověření uživatele, což je taky nejčastější způsob využití JWT. Po ověření jsou JWTs typicky uloženy v prohlížeči do cookies nebo localStorage. Bezpečnější je ukládání do cookies, protože cookies jsou chráněny proti útokům typu CSRF a jsou přístupné pouze pro doménu, která je uložila [32].

Délka platnosti JWT je určena při vytváření. Platnost tokenu může být nastavena na krátkou dobu, například několik minut, nebo na delší dobu, například

[4]CSRF (Cross-site Request Forgery) je typ útoku na webové aplikace, sloužící k provedení nežádoucího příkazu pocházejícího z nelegitimního zdroje za účelem získání přístupu do aplikace [29].

několik hodin nebo dnů. Správná volba délky platnosti zajišťuje bezpečnost tokenu, ale zároveň minimalizuje počet požadavků na obnovení tokenu. JWTs jsou složené z hlavičky (header), dat (payload) a podpisu (signature) [32]. Následující diagram 1.3 zobrazuje získání JWT a jeho použití při přístupu k API serveru.



Obr. 1.3: Ukázka použití JWT.

1. Nejprve se uživatel (pomocí validního jména a hesla) přihlásí (autentizuje) na autentizačním serveru.
2. Jakmile server ověří identitu uživatele, vytvoří JWT a navrátí jej na stranu klienta.
3. Uživatel pak může používat tento token třeba pro přístup k API serveru [32].

Jakmile je uživatel úspěšně přihlášen, každý jeho požadavek obsahuje i JWT, který uživateli umožňuje přístup k jednotlivým adresám, službám a zdrojům, kde je přístup umožněn díky tomuto platnému tokenu. Pro vytvoření, autentizaci a práci s tokenem JWT bylo na Flask serveru doinstalováno rozšíření Flask-JWT-Extended[5].

[5] `https://flask-jwt-extended.readthedocs.io/en/stable/`

2 Vizualizace hudebních parametrů

2.1 Music Information Retrieval

Music Information Retrieval (MIR) je vědní oblast zabývající se získáváním informací o hudbě. Jak již bylo zmíněno v úvodu, cílem této práce bude pomocí poznatků z vědní oblasti MIR vizualizovat parametry hudebních nahrávek. Tyto parametry mohou být extrahovány z audio dat nebo symbolických dat (např. MIDI soubor) [1].

I přes význam hudby v naší společnosti je výzkumné pole MIR relativně mladé, avšak od svých počátků před méně než dvěma desetiletími zažívá stálý vzestup. K tomu přispěly faktory jako vývoj technik pro kompresi audia v 90. letech, rostoucí výkon osobních počítačů, což umožnilo extrahovat vlastnosti hudby v rozumném čase, a široká dostupnost mobilních hudebních přehrávačů [2].

MIR využívá technik z oblasti zpracování signálů, strojového učení, hudební teorie a psychoakustiky k extrakci různých parametrů z hudebních nahrávek. MIR má významné aplikace v oblasti digitálního zpracování hudby a hudebního průmyslu. Komerční služby jako Spotify, Deezer nebo Apple Music využívají MIR technologie pro doporučování hudby svým uživatelům a usnadňují tak navigaci v obrovském množství hudebních nahrávek. Další významnou aplikací MIR je rozpoznávání melodie na základě pouhého zpěvu nebo hraní na hudební nástroj [1].

2.2 Nástroje pro výpočet hudebních parametrů

Většina moderních algoritmů popisujících parametry vycházející z MIR se skládá ze dvou částí. Nejprve jsou jednotlivé parametry extrahovány a následně jsou parametry analyzovány. Následovat bude výčet a stručný popis jendotlivých nástrojů.

Marsyas

Marsyas[1] je zkratkou pro Music Analysis, Retrieval and Synthesis for Audio Signals. Jedná se o open-source knihovnu pro digitální zpracování zvuku, která umožňuje vytvářet a testovat algoritmy pro zpracování zvuku a hudby. Marsyas vznikl jako projekt doktorského studenta George Tzanetakise na Princeton University. Zpočátku to byla sada tříd napsaných v jazycích C++ a Java, které usnadňovaly různé úkoly analýzy a syntézy zvuku. Postupem času se Marsyas vyvinul v rozsáhlou sbírku C++ modulů pro zpracování zvuku, které jsou snadno propojitelné a intuitivní v používání [18].

[1] `https://github.com/marsyas/marsyas`

Yaafe

Yaafe[2] je zkratkou pro Yet Another Audio Feature Extractor. Jedná se o knihovnu napsanou v jazyce C++ s podporou Pythonu a MATLABu. Yaafe se často používá v programech pro rozpoznávání řeči, klasifikaci zvuků a hudby a v mnoha dalších zvukových aplikacích [17].

openSMILE

Knihovna openSMILE[3] umožňuje extrahovat vlastnosti z audiosignálů a klasifikovat řeč a hudbu. Často se využívá pro automatické rozpoznávání emocí. Původně byla vyvinuta v roce 2008 na Technické univerzitě v Mnichově jako součást systému pro analýzu řeči a emocí v reálném čase. V roce 2013 byl openSMILE převzat společností audEERING, ale stále je k dispozici zdarma pro akademické účely [4].

Essentia

Essentia[4] je open-source knihovna v jazyce C++ sloužící pro analýzu zvuku a získávání hudebních informací. Tato knihovna nabízí širokou škálu algoritmů pro analýzu zvukových dat. Obsahuje nástroje pro zpracování zvuku, včetně analýzy spektra. Je optimalizovaná pro robustnost, výpočetní rychlost a nízkou paměťovou náročnost, což ji dělá vhodnou pro průmyslové aplikace. Její implementace je dostupná v jazycích Python a JavaScript. To umožňuje snadnou integraci do již existujících projektů [16].

Librosa

Librosa[5] je knihovna určená pro jazyk Python. Slouží k extrakci a vizualizaci hudebních parametrů. Jedná se o populární knihovnu oblíbenou mezi programátory pracujícími s hudebními daty. Librosa je strukturovaná jako několik submodelů [8]. Níže je uveden seznam modulů využitých v této práci.

- **librosa.beat** – funkce pro odhad tempa a detekování dob v nahrávce
- **librosa.core** – hlavní funkce, sloužící k načtení audio souboru a výpočtu základních parametrů pro hudební analýzu
- **librosa.onset** – používá se k výpočtu časového údaje začátku not

[2] `https://yaafe.sourceforge.net/`
[3] `https://www.audeering.com/research/opensmile/`
[4] `https://essentia.upf.edu/`
[5] `https://librosa.org/`

MIRtoolbox

MIRtoolbox[6] je integrovaný soubor funkcí napsaných v prostředí Matlab sloužící k extrakci hudebních parametrů z audio nahrávek. Je organizovaný do objektově orientované architektury. Nabízí extrakci přibližně padesáti hudebních parametrů. Pro správný chod MIRtoolboxu je v Matlabu doporučena instalace Signal Processing Toolbox. MIRtoolbox využívá zejména funkce dostupné ve veřejných nástrojích jako je Auditory Toobox, NetLab nebo SOM-toolbox. Parametry extrahované díky MIRtoolboxu lze dále přímo analyzovat pomocí sad nástrojů, jako jsou Statistics toolbox nebo Neural Network toolbox [5].

Aubio

Aubio[7] je soubor algoritmů a nástrojů sloužící k detekci hudebních událostí. Je napsaný v jazyce C. Aubio poskytuje následující funkce: digitální filtry, FFT a fázový vokodér, detekci začátku not, detekci tempa a taktů, detekci výšek tónů, spektrální filtraci a další matematické nástroje používané při zpracování audio signálu [7].

Meyda

Meyda[8] je knihovna pro JavaScript. Jedná se o první knihovnu zpracovávající hudební parametry z audio nahrávek na straně klienta. Její vývoj byl ovlivněn nástrojem Yaafe sloužící k extrakci hudebních parametrů v jazyce Python [6].

2.2.1 Existující nástroje pro vizualizaci hudebních parametrů

Jak již bylo zmíněno v úvodní kapitole práce, jedna z motivací pro implementace této webové aplikace byl fakt, že podobných nástrojů pro extrakci a vizualizaci hudbeních parametrů je poměrně malé množství. Co se týče webového prostředí, tak nebyla nalezena žádná podobná webová aplikace, která by byla dostupná a umožňovala vizualizaci a porovnání hudebních parametrů různých interpretací dané skladby.

Nejpodobnějším nástrojem je v tomto případě Sonic Visualizer. Jedná se o desktopovou aplikaci, která umožňuje uživatelům analyzovat a zobrazovat různé parametry hudebních nahrávek. Umožňuje zobrazení a porovnání libovolného počtu různých oken s parametry, které jsou zarovnané v časové ose [9].

[6] `https://www.mathworks.com/matlabcentral/fileexchange/24583-mirtoolbox`
[7] `https://aubio.org/`
[8] `https://meyda.js.org/`

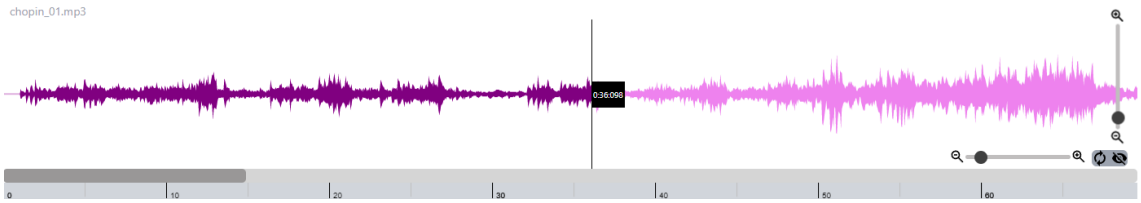
2.3 Zvukové parametry

Webová aplikace nabízí uživatelům možnost vizualizovat a porovnávat různé hudební parametry více nahrávek současně. Tato funkce umožňuje uživatelům například porovnat různé interpretace stejné skladby a lépe porozumět rozdílům mezi nimi. Aplikace disponuje souborem nastavení, které je možné pro každou vizualizaci upravit dle vlastních preferencí. Jakékoli změny v nastavení se automaticky ukládají do databáze, což zajišťuje, že uživatelé neztratí svá data o nastavení vizualizací pro jednotlivé nahrávky.

2.3.1 Časový průběh zvukového signálu

Časový průběh zvukového signálu se využívá k zobrazení závislosti okamžité hodnoty signálu na čase. Pro vytvoření časového průběhu byla využita JavaScriptová knihovna Wavesurfer.js[9]. Jedná se o knihovnu postavenou na rozhraní Web Audio API a HTML5 Canvas. HTML5 Canvas je HTML element `<canvas>` určený pro dynamické vykreslování grafiky na webu pomocí JavaScriptu. Web Audio API pak poskytuje výkonný a všestranný systém pro ovládání zvuku na webu. Umožňuje vývojáři vybrat zdroje zvuku, pracovat se zvukovými efekty a vytvářet zvukové vizualizace [10].

Před vytvořením nové Wavesurfer instance bylo nutné vytvořit nový `<div>` element, s ID odkazem na danou nahrávku. Instance WaveSurferu se následně vytvoří pomocí metody `WaveSurfer.create()`. Při vytvoření instance je možné vkládat různé parametry a pluginy. V případě časového průběhu zvukového signálu byl využit TimelinePlugin pro zobrazení časové osy, Cursor pro zobrazení kurzoru, RegionsPlugin pro možnost výběru časového úseku a Markers pro vizualizaci začátků taktů, začátků tónů a dob. Po vytvoření instance je možné načíst audio ze serveru. Ukázka vytvořené vizualizace časového průběhu zvukového signálu je znázorněna na obr. 2.1.



Obr. 2.1: Časový průběh zvukového signálu ve výchozím nastavení.

[9] `https://wavesurfer-js.org/`

Nastavení vizualizace

Ve vizualizaci je k dispozici horizontální posuvník umístěný v pravé části, který umožňuje přibližování a roztahování vizualizace. Při změně přiblížení se automaticky upraví šířky spektrogramu a pianorollu tak, aby odpovídaly šířce časového průběhu zvukového signálu, což vyžaduje přerenderování těchto vizualizací.

Vertikální posuvník pak slouží ke změně výšky zvukové vlny. Oba posuvníky je možné skrýt pomocí pravého tlačítka na panelu, který se po najetí myši objeví v dolním rohu vizualizace. Levé tlačítko na panelu umožňuje vrátit zobrazení vizualizace do výchozího stavu. Rozbalený panel určený k nastavení vizualizace časového průběhu zvukového signálu je zobrazen na obr. 2.2.

Obr. 2.2: Panel nastavení vizualizace časového průběhu zvukového signálu.

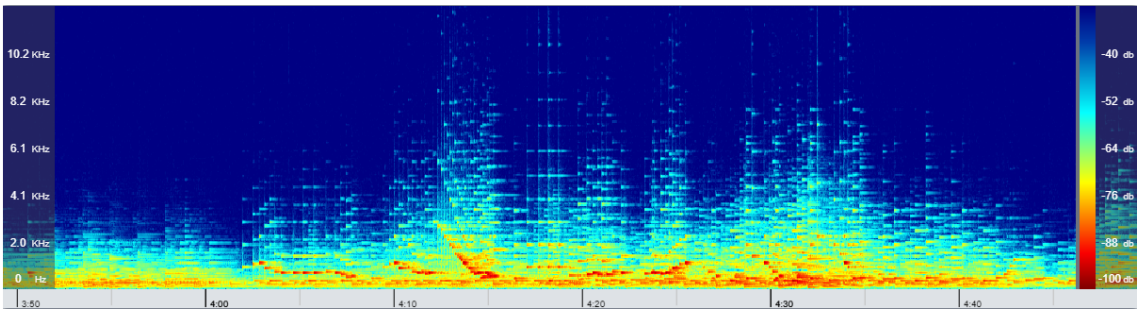
V pravém horním rohu se nachází tlačítko pro export a uložení časového průběhu zvukového signálu ve formátu PNG (1). Tento export je možný u všech vizualizací. Pro jeho implementaci byla využita knihovna html2canvas[10]. Ve formě nastavitelných čar je do vizualizace možné přidat zobrazení časových anotací začátků taktů (2) a detekované začátky not (3) a dob (4). Uživatel pak může nastavovat jejich barvu (5), tloušťku (6), zobrazení před/za časovým průběhem zvukového signálu (7) a možnost zapnout a zvolit zvukový signál, který zazní během přehrávání při kontaktu s kurzorem (8).

Vizualizace také umožňuje vybrat časový úsek přímo zadáním časových údajů (10), nebo pomocí zadaného rozsahu taktů (9). Ve spodní části se pak nachází možnost nastavení vzhledu vizualizace (11). Je možné měnit barvu (12), a to včetně barevného rozdílu mezi přehraným a zbývajícím úsekem. Dále pak výšku celé vizualizace (13) a rozdělení časového průběhu zvukového signálu do levého a pravého kanálu (14).

[10] `https://html2canvas.hertzen.com/documentation`

2.3.2 Spektrogram

Spektrogram je jednou z možností vizualizace hudební nahrávky. Spektrogram lze interpretovat jako třírozměrný graf, kde na ose x je čas, na ose y jsou frekvence a intenzita spektrálního obsahu signálu je reprezentována intenzitou barvy. Tedy čím více energie je v určité frekvenci v určitém okamžiku času, tím intenzivnější barva se na tomto místě v grafu objeví. Frekvenční osa se často zobrazuje lineárně nebo logaritmicky.



Obr. 2.3: Spektrogram ve výchozím nastavení.

Pro výpočet spektrogramu byla opět využita knihovna Wavesurfer.js a konkrétně modul `SpectrogramPlugin`. Tato knihovna provádí výpočet spektrogramu pomocí krátkodobé Fourierovy transformace (STFT), která umožňuje analyzovat signál postupně po krátkých úsecích. Signál se nejdříve rozdělí na segmenty o určité délce (ve vzorcích) a potom je tento segment vynásobený váhovacím oknem. V praxi se často používají různé typy váhovacího okna, jako jsou například Hammingovo okno, Hannovo okno, Blackmanovo okno a další. Váhovací okna pomáhají potlačit nežádoucí efekty na okrajích této části signálu. Na tyto úseky je pak aplikována rychlá Fourierova transformace (FFT). Díky tomuto procesu je možné získat informaci o frekvenčním obsahu signálu v daném časovém úseku. STFT je definována následujícím způsobem

$$X(m,k) = \sum_{n=0}^{N-1} x(n+mH)w(n)e^{-2\pi ikn/N}, \tag{2.1}$$

kde $X(m,k)$ je výstupní spektrum pro časový úsek m a frekvenční složku k, $x(n)$ je vstupní signál, $w(n)$ váhovací funkce, N délka okna a H představuje počet vzorků mezi jednotlivými úseky.

Spektrogram se poté z STFT spočítá následovně

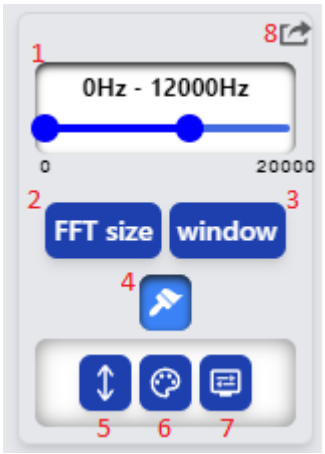
$$Y(m,k) = |X(m,k)|^2, \tag{2.2}$$

kde hodnota spektrogramu $Y(m,k)$ je reprezentována různou intenzitou barvy na souřadnicích (m,k)[1].

Knihovna Wavesurfer.js vypočítává data spektrogramu pomocí knihovny DSP.js[11]. Získaná data následně vykreslí do `ImageData` objektu. Tento objekt umožňuje reprezentaci pixelových dat obrazu v HTML canvas. Wavesurfer.js ovšem neumožňuje zobrazení spektrogramu v logaritmickém měřítku. A proto byla v této diplomové práci pro vizualizaci dat spektrogramu využita knihovna Plotly.js[12]. Ta umožňuje vykreslení frekvenční osy spektrogramu v logaritmickém zobrazení.

Nastavení vizualizace

Vizualizace spektrogramu umožňuje zobrazení a skrytí popisu frekvenční osy a colorbaru. Colorbar slouží k interpretaci intenzity signálu. Díky němu lze ze spektrogramu vyčíst, jaké frekvence jsou v signálu nejvýraznější a k jaké změně intenzity signálu dochází v závislosti na čase. Zobrazení a skrytí popisu frekvenční osy a colorbaru je možné po najetí myši na dané místo s umístěnými panely. Panel sloužící k nastavení jednotlivých parametrů spektrogramu je zobrazen na následujícím obrázku 2.4.



Obr. 2.4: Nastavení vizualizace spektrogramu.

Při změně parametrů ve vizualizaci časového průběhu zvukového signálu se změny projeví okamžitě, zatímco při vizualizaci spektrogramu může být nutné několik vteřin pro jeho překreslení. Do vizualizace byla přidána komponenta signalizující probíhající výpočet, aby byl uživatel informován o probíhajícím přepočtu vizualizace spektrogramu při změně parametrů. Uživatel vidí tuto komponentu jako animovaný symbol na obrazovce.

Pro export spektrogramu jako obrázku ve formátu PNG je k dispozici tlačítko umístěné v pravém horním rohu (8). K výběru frekvenčního rozsahu pro vykreslení spektrogramu slouží posuvník v horní části (1). Níže se nachází tlačítko pro zvolení

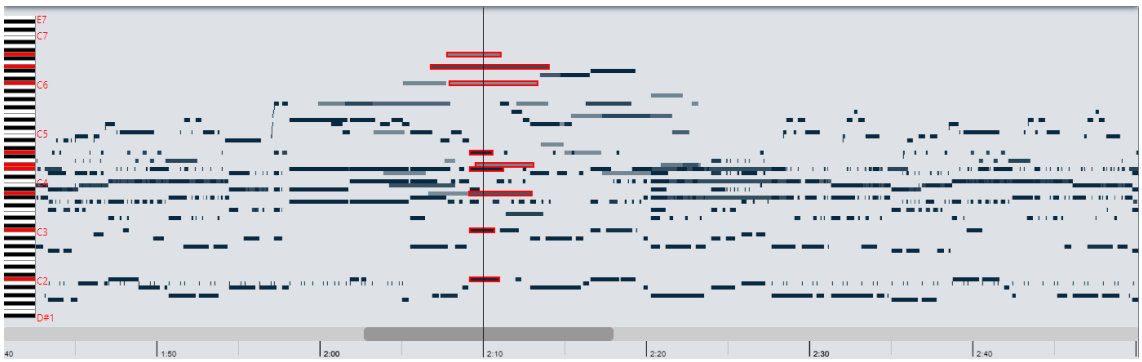
[11] `https://github.com/corbanbrook/dsp.js`
[12] `https://plotly.com/javascript/`

počtu vzorků, které určují délku jednotlivých úseků při výpočtu STFT (2). Toto nastavení umožňuje uživateli upravit rozlišení spektrogramu. Dále se zde nachází možnost vybrat typ váhovacího okna, která se aplikuje na rámce před výpočtem Fourierovy transformace. Ve výchozím nastavení je zvoleno Hannovo okno.

V dolní části aplikace se nachází rozbalovací panel, který umožňuje uživateli přizpůsobit vzhled vizualizace spektrogramu (4). V této sekci má uživatel možnost nastavit výšku spektrogramu (4), zvolit typ colormapy, která definuje, jak budou vykresleny intenzity jednotlivých frekvencí (6), a také zvolit mezi lineárním a logaritmickým zobrazením spektrogramu (7).

2.3.3 MIDI pianoroll

Musical Instrument Digital Interface (MIDI) je standardizovaný protokol a formát pro komunikaci a záznam hudebních informací. MIDI pianoroll je pak vizualizace, sloužící k zobrazení MIDI dat, která reprezentují hudební skladbu nebo melodii. Pomocí tohoto nástroje je možné graficky zobrazit noty do tabulky tak, že každý řádek odpovídá jedné notě a každá časová pozice pak zobrazuje, zda se daná nota má hrát nebo ne, případě s jakou délkou. Intenzita zvuku se v MIDI udává pomocí velocity, což je číslo mezi 0 a 127, kde vyšší číslo znamená hlasitější zvuk. Vizualizace pianorollu v aplikaci ve výchozím nastavení je zobrazena na obr. 2.5.



Obr. 2.5: Pianoroll ve výchozím nastavení.

Pro vytvoření pianorollu byla využita knihovna html-midi-player[13]. V této webové aplikaci slouží pouze k vykreslení MIDI informací do elementu `<canvas>`. Bylo však nutné zajistit, aby šířka pianorollu odpovídala šířce ostatních vizualizací. Z tohoto důvodu byl využit parametr `pixelsPerTimeStep`, který určuje, kolik pixelů v pianorollu odpovídá jednomu časovému kroku. Rovněž bylo nutné vizualizaci doplnit o popis jednotlivých not. Za tímto účelem byla vytvořena jednoduchá klaviatura z černých a bílých `<div>` HTML elementů. Rozsah této klaviatury odpovídá nejnižší a nejvyšší notě ve skladbě. Klaviatura byla rovněž doplněna o popis jednotlivých oktáv. Vizualizace je dále vylepšena tím, že barevně ohraničuje MIDI noty, které se při přehrávání nachází v aktuální pozici kurzoru, viz následující kód 2.1.

Výpis 2.1: Kontrola, zda se MIDI nota nachází v pozici kurzoru.

```
let trackCursor

wavesurfer[props.track.id].on('play', () => {
    trackCursor = setInterval(setPianorollCurrentPosition,100)
})
```

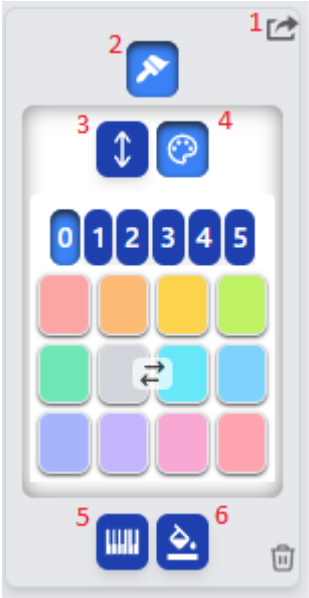
[13] `https://github.com/cifkao/html-midi-player`

```
wavesurfer[props.track.id].on('pause', () => {
    clearInterval(trackCursor)
})
```

Tento kód vytvoří při spuštění přehrávání interval, který bude každých 100 ms volat funkci `setPianorollCurrentPosition()`. Tato funkce iteruje přes každou notu a zjišťuje, jestli se noty nachází na pozici, kde se nachází kurzor. Pokud je to pravda, obarví se okraje noty uživatelem zvolenou barvou. Rovněž se obarví i příslušná klávesa na klaviatuře. Stejně jako u ostatních vizualizací i zde je možné skrýt klaviaturu přímo z vizualizace.

Nastavení vizualizace

Na následujícím obrázku 2.6 je zobrazeno nastavení vizualizace pianorollu s rozbaleným oknem pro zvolení barev jednotlivých nástrojů (4). V pravém horním rohu



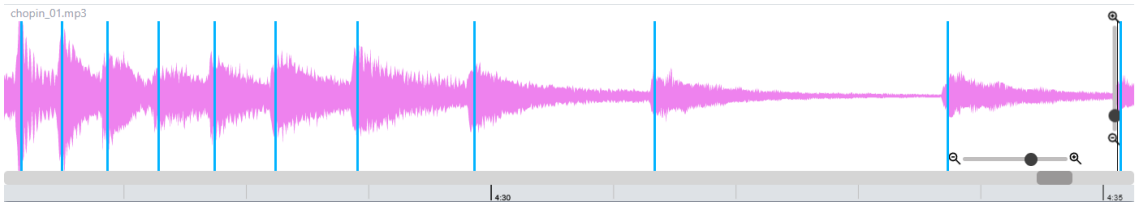
Obr. 2.6: Nastavení vizualizace pianorollu.

je opět možnost exportovat vizualizaci včetně klaviatury (1). Ve vrchní části se pak nachází tlačítko pro nastavení vzhledu vizualizace (2). Zde je možné nastavit výšku vizualizace (3) a barvu jednotlivých hudebních nástrojů (4). V dolní části je možné nastavit dynamickou změnu popisků klaviatury v závislosti na přehrávaných notách (5). Výchozí nastavení obsahuje popisky pouze pro oktávy a nejvyšší nebo nejnižší noty. Vedle toho je možné nastavit barvu ohraničení právě přehrávaných not (6).

2.3.4 Detekce začátků tónů

Pro výpočet detekce začátků tónů byla využita knihovna Librosa pro Python. Ta využívá metodu spectral flux, při které se sleduje změna spektrálního obsahu mezi sousedními časovými rámci. Konkrétně se vypočítává rozdíl mezi spektry po sobě jdoucích rámců, což vytváří funkci nazvanou spectral flux. Pokud se výrazně změní spektrální obsah signálu, například při nástupu nového tónu, bude mít tato funkce vysokou hodnotu [1].

Detekované začátky tónů lze v aplikaci vizualizovat přímo do časového průběhu zvukového signálu pomocí nastavitelných čar. Před jejich zobrazením se nejprve zavolá požadavek na server, kde proběhne výpočet pomocí knihovny Librosa (pokud již výpočet proběhl, soubor s detekovanými začátky not je uložen na serveru a rovnou se tak navrátí jeho obsah ve formátu JSON na stranu klienta). Následně se pomocí pluginu `Markers` zobrazí do časového průběhu zvukového signálu, viz následující obrázek 2.7.

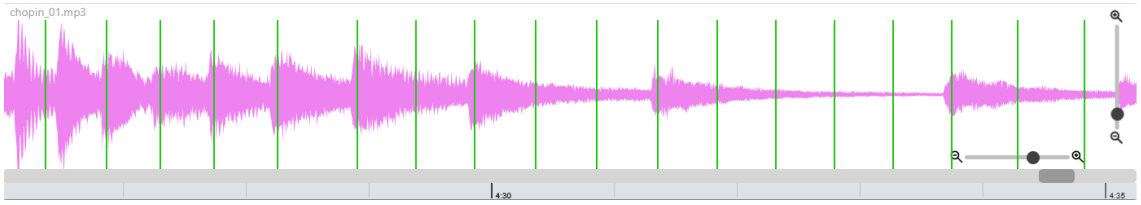


Obr. 2.7: Ukázka použití vizualizace detekce začátků not v časovém průběhu zvukového signálu.

2.3.5 Detekce dob

Detekce tempa nebo taktů při analýze hudby může být obtížná v případech, kdy dochází k lokálním změnám tempa nebo rytmu. Tyto změny mohou být nepředvídatelné a nepravidelné. V případě popu nebo rockové hudby, které mají silný periodický rytmický základ, je detekce poměrně snadná. Ale u klasické hudby, kde dochází k častým změnám tempa, je situace náročnější.

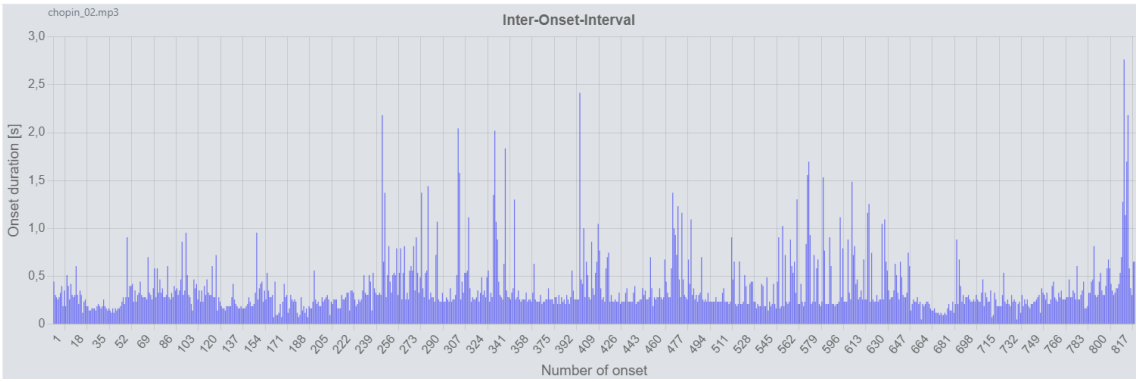
Tempo a doby jsou základní aspekty hudby. Doba (anglicky beat) je často popisována jako sekvence vnímaných pulzů, které jsou typicky rovnoměrně rozložené v čase a jsou specifikované dvěma parametry – fází a periodou. Tempo se určuje podle rychlosti pulzu a jeho číselné označení se provádí zápisem udávajícím počet úderů za minutu tzv. BPM (beats per minute) [1]. Stejně jako v případě detekovaných začátků tónů, i zde byla na výpočet detekovaných dob využita knihovna Librosa a jejich vizualizace přímo do časového průběhu zvukového signálu, viz následující obrázek 2.8.



Obr. 2.8: Ukázka použití vizualizace detekce dob v časovém průběhu zvukového signálu.

2.3.6 Časové intervaly mezi detekovanými událostmi

Detekované tóny a doby lze vizualizovat také pomocí grafu, který ukazuje trvání intervalů mezi jednotlivými detekovanými událostmi neboli Inter-Onset-Intervals a Inter-Beats-Intervals. Stejným způsobem lze vizualizovat i dobu trvání mezi jednotlivými takty v průběhu skladby – Inter-Measure-Intervals. Implementace vizualizace je provedena následujícím postupem. Po načtení komponenty se nejprve získají data časových pozic událostí, a to stejným způsobem, jako v předchozích případech 2.3.4 a 2.3.5. Následně je vytvořena instance třídy `ChartManager`, kdy se provede výpočet intervalů mezi událostmi a tyto data se následně vloží do grafu. K vytvoření grafu byla využita knihovna Chart.js[14]. Ukázka vizualizace trvání intervalů mezi detekovanými tóny je zobrazena na následujícím obrázku 2.9.



Obr. 2.9: Ukázka použití vizualizace Inter-Onset-Intervals.

[14] `https://www.chartjs.org/docs/latest/`

2.3.7 RMS

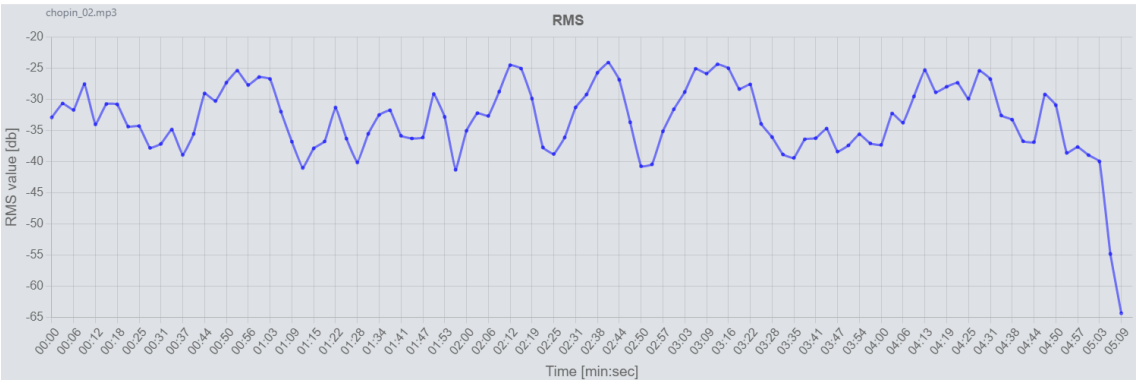
Root Mean Square (RMS) je metoda výpočtu efektivní hodnoty signálu. V kontextu MIR aplikací se RMS hodnota často používá k vyjádření energie jednotlivých segmentů zvukového signálu. Při analýze hudby může být RMS využíváno k detekci úseků s vysokou nebo nízkou energií. Hodnotu RMS lze vypočítat jako

$$\mathrm{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}, \tag{2.3}$$

kde x_i je hodnota signálu v čase i a N je celkový počet hodnot signálu [33]. Hodnoty RMS jsou pak převedeny na hodnoty v decibelech pomocí vzorce

$$\mathrm{dB} = 20 \cdot \log_{10}(\mathrm{RMS}). \tag{2.4}$$

V aplikaci je vizualizace RMS nahrávky zobrazena pomocí knihovny Chart.js, kdy se nahrávka rozdělí na uživatelem zadaný počet segmentů a pro každý segment se pak vypočítá hodnota RMS. Ta se následně vizualizuje do spojnicového grafu, viz obrázek 2.10.

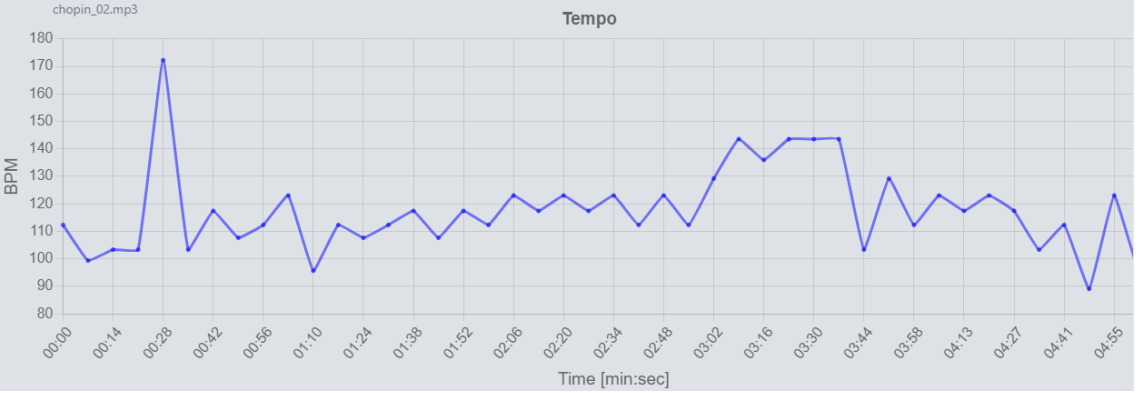


Obr. 2.10: Vizualizace RMS.

Samotná realizace v aplikaci vypadá tak, že po načtení komponenty se nejprve získá pole dat obsahující hodnoty RMS v decibelech. Toto pole se získá zavoláním funkce `calculateRMSForSegments()`, která přijímá jako vstupní parametry `audioBuffer`, který obsahuje nahrávku, a zvolený počet segmentů, ze kterých bude hodnota vypočítána. Funkce nejprve rozdělí nahrávku na požadovaný počet segmentů a následně spočítá RMS pro každý segment podle vzorce 2.3. Pole s těmito hodnotami následně navrátí do komponenty, kde se vytvoří nová instance třídy `ChartManager`, která data vizualizuje do spojnicového grafu.

2.3.8 Tempo

Další možností vizualizace parametrů hudební nahrávky je pomocí zobrazení vývoje tempa. Tempo je jedním z klíčových atributů, které ovlivňují celkový dojem z hudby a je často měněno hudebníky k tomu, aby vyjádřili svou vlastní interpretaci skladby [1]. V této aplikaci je tempo vizualizováno pomocí spojnicového grafu, který ukazuje změny tempa v průběhu skladby, viz obrázek 2.11.



Obr. 2.11: Vizualizace vývoje tempa hudební nahrávky.

Realizace v aplikaci vypadá tak, že se po načtení komponenty zavolá požadavek na server, kde se spustí funkce `get_tempo()`, která má dva vstupní parametry. První z nich je název nahrávky uložené na serveru a druhý počet segmentů, na které se má nahrávka rozdělit. Funkce nejprve načte nahrávku pomocí knihovny Librosa a vypočítá délku jednoho segmentu na základě délky nahrávky a počtu segmentů. Poté funkce pro každý segment spočítá tempo pomocí funkce `beat.tempo()` z knihovny Librosa a výsledné hodnoty vrátí jako odpověď ve formátu JSON. Tyto hodnoty jsou následně vizualizovány pomocí knihovny Chart.js.

Je však důležité mít na paměti, že tempo je subjektivní a citlivé na interpretaci, a že získání přesného tempa je obtížné kvůli nesrovnalostem v měření a synchronizaci hudebních nástrojů. Proto je nutné použít robustní numerické metody k extrakci tempa a vhodně volit velikost časového okna. Při vizualizaci vývoje tempa také vzniká problém určení, zda je změna v tempu způsobena chybami synchronizace hudebních nástrojů nebo skutečnou změnou tempa v interpretaci [1].

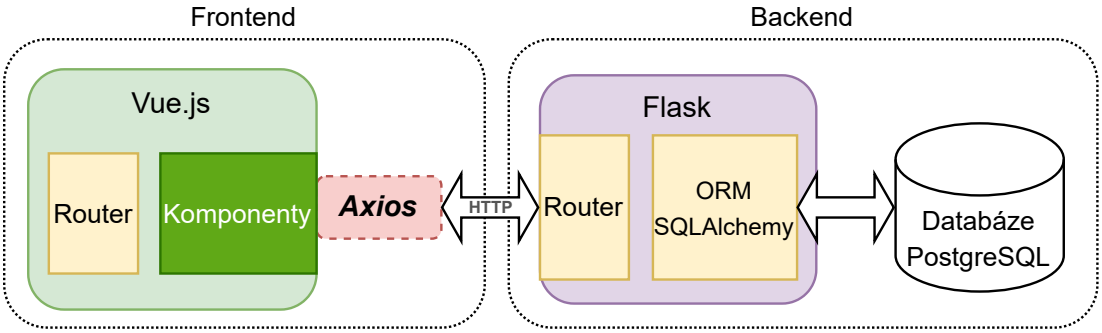
Obr. 2.12: Nastavení vizualizace

Pro všechny vizualizace vyobrazené pomocí knihovny Chart.js (RMS, tempo, Inter-Onset-Intervals, Inter-Beats-Intervals a Inter-Measure-Intervals) je k dispozici jeden společný panel sloužící k nastavení vizualizace. Mezi jednotlivými typy vizualizací lze přepínat pomocí tlačítek (2). V případě zobrazené vizualizace RMS a Tempa je možné přepínat i mezi nimi. Export vybrané vizualizace je možný pomocí tlačítka v pravém horním rohu (1). Dále je možné vizualizaci přepnout do zobrazení histogramu (3). V tomto režimu se nahrávka rozdělí na zvolený počet časových úseků a pomocí histogramu se zobrazí počet detekovaných událostí pro jednotlivé úseky. Druhý způsob, jak zobrazit histogram, je pomocí rozdělení intervalů mezi detekovanými událostmi podle jejich délky.

Zobrazení histogramu je možné pouze u vizualizací IOI, IBI a IMI. Tempo a RMS má namísto toho možnost měnit počet úseků, na které se nahrávka rozdělí a ze kterých budou data vypočítána. Vpravo se pak nachází možnost přidat do vizualizace průměrnou hodnotu ve formě horizontální čáry a také klouzavého průměru s nastavitelným počtem rámců, ze kterých bude klouzavý průměr vypočítán (4). Vizualizace má opět možnost nastavení vlastního stylu (5). Je možné měnit barvu (6), výšku (7) a přepínat mezi sloupcovým a spojnicovým grafem (8).

3 Implementace aplikace

Následující kapitola popisuje praktickou realizaci webové aplikace. Nejprve je popsán backend aplikace z pohledu celkové architektury, způsobu propojení s databází a funkcí, které poskytuje Flask server. Následuje popis stromové struktury frontend části aplikace. Jsou zde popsány jednotlivé vztahy mezi komponentami a způsob komunikace se serverem. Architektura aplikace je znázorněna na následujícím obrázku 3.1.



Obr. 3.1: Architektura aplikace.

3.1 Backend

Backend byl napsán ve frameworku Flask. Ten ze strany klienta přijímá HTTP požadavky. Klient zasílá požadavky díky knihovně Axios. Router následně nalezne adresu příslušného požadavku. Ten se dále zpracuje případně požádá databázi o data. Po zpracování požadavku navrací na stranu klienta odpověď ve formátu JSON.

3.1.1 Propojení s databází

Do Python prostředí, ze kterého se spouští Flask server, bylo doinstalováno rozšíření SQLAlchemy. Toto rozšíření poskytuje možnost komunikace mezi Flask serverem a databází bez nutnosti přímo pracovat s jazykem SQL. V hlavním souboru `_init_.py` byla nakonfigurována databáze PostgreSQL. Toho bylo dosaženo pomocí příkazu `app.config['SQLALCHEMY_DATABASE_URI']`. Pro interakci s databází bylo třeba vytvořit databázový objekt a uložit jej do proměnné `db`. K tomu byla využita třída `SQLAlchemy`, do které byla vložena instance aplikace: `db = SQLAlchemy(app)`.

Na následující straně je na obrázku 3.2 zobrazen Entity-Relationship Diagram (ERD). Tento diagram slouží k vizualizaci struktury databáze. Poskytuje grafickou reprezentaci tabulek, sloupců a vztahů mezi nimi. V našem případě ERD popisuje strukturu databáze aplikace, která byla propojena s databází. Na diagramu jsou znázorněny dvě hlavní třídy, `User` a `Recording`, s popisem atributů, které obsahují. Třída `User` reprezentuje uživatele a obsahuje jejich ID, uživatelské jméno, e-mail a heslo. Třída `Recording` reprezentuje nahrávku a obsahuje informace jako název souboru, cestu k souboru, zda je tato nahrávka označena, barvu pozadí, jméno souboru s nahranými takty a jméno MIDI souboru. Kromě toho jsou zde také další třídy, které slouží k uchování informací o vizualizacích a jsou propojeny s třídou `Recording` pomocí cizího klíče. Vztah mezi třídami `Recording` a ostatními třídami je typu „one-to-many“, což znamená, že každá nahrávka může mít více vizualizací, ale každá vizualizace patří pouze k jedné nahrávce. Stejným způsobem jsou také propojeny třídy `Recording` a `User`, kdy každému uživateli může být přiřazeno více nahrávek.

Obr. 3.2: ERD vytvořené databáze.

3.1.2 Implementované funkce ve Flask serveru

Webová aplikace pro vizualizaci parametrů hudebních nahrávek poskytuje uživateli několik funkcionalit. Při vývoji byla snaha jich co největší množství implementovat na stranu klienta. Některé z nich ovšem bylo vhodnější počítat na straně serveru a zasílat zpět odpověď ve formátu JSON.

Při implementaci těchto funkcí byl využit dekorátor `jwt_required()`. Tento dekorátor zajišťuje, že pro volání dané funkce musí být uživatel autentizován pomocí JWT tokenu viz 1.3.11.

Funkce sloužící ke správě uživatelů

Po vyplnění registračního formuláře ve frontendu v `RegisterForm.vue` komponentě se data odešlou na server a zavolá se funkce `register()`. Tato funkce nejprve zkontroluje, zda už není uživatel v databázi zaregistrovaný. Stejným způsobem se zkontroluje i zadaný email. V případě již zaregistrovaného emailu nebo uživatelského jména se zpět na stranu klienta odešle chybová zpráva.

Dále se pomocí příkazu `bcrypt.generate_password_hash` z rozšíření Flask-Bcrypt[1] vytvoří hash vyplněného hesla[2]. Následně se nový uživatel uloží do databáze. Pro uživatele se na serveru vytvoří vlastní složka, kam se budou ukládat nahrávky, sestříhané nahrávky, časové anotace začátků taktů, MIDI soubory a vypočítané data detekovaných událostí.

Jak je zmíněno v kapitole 3.2.3, popisující komponentu `LoginForm.vue`, kontrola přihlašovacích údajů probíhá na straně serveru. Po zavolání funkce `login()` se nejprve pomocí příkazu `User.query.filter_by(user=user).first()` ověří, zda je uživatelské jméno uložené v databázi a uživatel je tedy řádně zaregistrován. Pomocí funkce `check_password_hash()` se ověří platnost hesla a v případě validních údajů se uživateli pošle odpověď obsahující nový token JWT.

Při pokusu o vstup do hlavního okna aplikace je na serveru zavolána funkce `login_check()`. Ta slouží ke kontrole platnosti tokenu JWT. Ukázka kódu požadavku na straně klienta se nachází v kapitole popisující frontend aplikace 3.1. O kontrolu tokenu se stará funkce `verify_jwt_in_request()` importovaná z knihovny Flask-JWT-Extended. V případě úspěšné verifikace odešle zpět na stranu klienta zprávu o platnosti tokenu ve formátu JSON. K převodu do formátu JSON je použita funkce `jsonify()`.

[1] `https://flask-bcrypt.readthedocs.io/en/1.0.1/`

[2] Hashing - hesla se kvůli bezpečnosti neukládají v čitelné podobě, ale pouze jako otisk (tzv. hash). Z otisku poté nelze zpětně původní podobu hesla zrekonstruovat.

Při zavolání požadavku o odhlášení se zavolá funkce `unset_jwt_cookies`, která modifikuje odpověď serveru tak, aby odstranila JWT cookies[3], které obsahují přístupový token JWT.

Funkce sloužící ke správě souborů a nahrávek

Při pokusu o nahrání souborů na server se zavolá funkce `upload_file()`. Tato funkce nejprve zjistí, o jaký typ souboru se jedná a na základě toho se určí, do které složky se soubor uloží. Pokud jde o audio soubor, tak se nejprve zkontroluje, zda daný soubor pro daného uživatele již neexistuje v databázi. Pokud neexistuje, tak se uloží a vytvoří se nový záznam v tabulce `Recording` v databázi. V případě, že se jedná o textový nebo MIDI soubor, uloží se do příslušných složek. Funkce vrací JSON objekt s informací o tom, zda se soubor podařilo nahrát na server.

Funkce `rename_track()` slouží k přejmenování zvukového souboru uživatele a příslušného záznamu v databázi. V případě pokusu o odstranění souboru uživatele se zavolá funkce `delete_file()`. Kromě odstranění samotného souboru ze serveru smaže i veškeré záznamy o nahrávce z databáze. Při zavolání funkce `get_track_list()` je klientovi vrácena odpověď ve formátu JSON, která obsahuje informace o nahrávkách z databáze a seznam nahraných souborů s časovými anotacemi taktů a MIDI souborů.

Funkce `get_file()` slouží k získání cesty k souboru na serveru, který má být odeslán klientovi. Funkce bere název souboru jako argument a na základě přípony určuje složku, ve které se soubor nachází. Poté používá metodu `send_from_directory` z modulu Flask k odeslání souboru klientovi.

V případě, že uživatel změní parametry nahrávky nebo nastavení vizualizací, zavolá se funkce `update_recording()`. Ta slouží k aktualizaci záznamu nahrávky v databázi podle poskytnutých dat v JSON formátu v požadavku. Nejprve jsou získána potřebná data z požadavku, jako jsou název parametru, který chce uživatel změnit, název vizualizace, novou hodnotu a ID nahrávky, pro kterou se má změna provést. Poté funkce používá SQLAlchemy dotazy k načtení dat pro relevantní vizualizace z databáze. U nich změní vybraný parametr na novou hodnotu a změny uloží do databáze.

K výběru úseku ze zvukové nahrávky slouží na serveru funkce `trim_audio()`. Vstupními parametry jsou jméno nahrávky, počáteční a koncový čas a informace o tom, zda se má časový úsek vybrat na základě taktových úseků nebo časového intervalu. Pokud již z nahrávky existuje předem vybraný časový nebo taktový úsek, funkce vrátí tuto existující nahrávku. V případě, že takový úsek neexistuje, funkce

[3]Cookies jsou textové soubory vytvořené na straně serveru. Ukládají se ve webovém prohlížeči, ze kterého se následně mohou zasílat zpět na server pro další zpracování.

provede výběr úseku nahrávky podle zadaného rozsahu. Tento vybraný úsek nahrávky je poté uložen na server a informace o něm jsou uloženy do databáze.

Funkce sloužící k výpočtu parametrů

Funkce `get_event_detection()` slouží k získání souboru ve formátu JSON, který obsahuje informace o detekovaných začátcích dob a not. Ty jsou spočítány jen v případě, že soubor na serveru dosud nebyl pro danou nahrávku vytvořen. Jejich výpočet je proveden pomocí příkazu `onset.onset_detect()` a `beat.beat_track()` z knihovny Librosa.

V případě požadavku na výpočet tempa nahrávky se na serveru zavolá funkce `get_tempo()`. Tato funkce přijímá jako parametry název souboru a počet segmentům, na které se má nahrávka rozdělit. Nejprve se spočítá délka jednoho segmentu a následně se pro každý segment spočítá tempo pomocí příkazu `beat.tempo()`. Výsledná data jsou poté odeslána zpět klientovi ve formátu JSON.

3.2 Frontend

Základní kostru aplikace tvoří soubor `index.html`. Ten obsahuje základní HTML elementy včetně `<div>` elementu, který slouží jako kontejner pro hlavní komponentu aplikace `App.vue`. Soubor dále obsahuje odkaz na hlavní JavaScriptový soubor aplikace `main.js`. Ten obsahuje kód pro inicializaci Vue.js a připojení `App.vue` ke kořenovému elementu v `index.html`.

3.2.1 Composables

Composables jsou důležitou součástí Vue.js a umožňují vývojářům psát znovupoužitelný a čistý kód. Jedná se v podstatě o funkce vytvořené ve frameworku Vue.js, které umožňují sdílení logiky mezi různými komponentami. V aplikaci se poté mohou importovat a používat jako běžné funkce, což zjednodušuje a zrychluje vývoj [25].

Tyto funkce byly v aplikaci vytvořeny celkem čtyři. Jedna slouží k vytvoření Axios instance, která bude použita pro všechny požadavky z různých komponent. Tím se zjednoduší práce s API a zlepší se přehlednost kódu. Dále pak funkce `getCookie()` umožňuje získání hodnoty cookie, která je následně přidána do hlavičky API požadavku, aby bylo možné ověřit totožnost uživatele na serverové straně.

Funkce `showAlert()` slouží k zobrazení upozornění a jako argument přijímá zprávu, která se má zobrazit. Funkce nejprve získá globální stav upozornění z `alertState()` uloženého v Pinia store, který obsahuje aktuální stav upozornění (zda je zobrazeno nebo ne) a zprávu. Pomocí `$patch` metody nastaví zprávu na požadovaný text a nastaví hodnotu stavu na true. Funkce `closeAlert()` slouží k uzavření upozornění a nastaví hodnotu stavu na false a zprávu na prázdný řetězec. V aplikaci je upozornění zprostředkováno pomocí komponenty `AlertModal.vue`, která podle globálního stavu z `alertState()` zobrazuje modální okno s ikonou upozornění a zprávou. Celkově tedy komponenta `AlertModal.vue` a funkce `showAlert()` a `closeAlert()` spolu tvoří systém pro zobrazování upozornění v aplikaci, který je řízen globálním stavem v Pinia store.

Funkce `useHideElement()` slouží k zobrazení určitého prvku po najetí myší na jiný prvek a následnému skrytí po určité době. Funkce přijímá jeden argument, a to reaktivní proměnou, která určuje, zda má být prvek skrytý nebo ne.

3.2.2 Stores

Stores umožňují ukládat data z různých komponent a modulů aplikace na jednom místě, což usnadňuje jejich správu a aktualizaci. Stores také umožňují vytvářet jednoduché kódy pro manipulaci se stavem aplikace. V této aplikaci byl pro správu

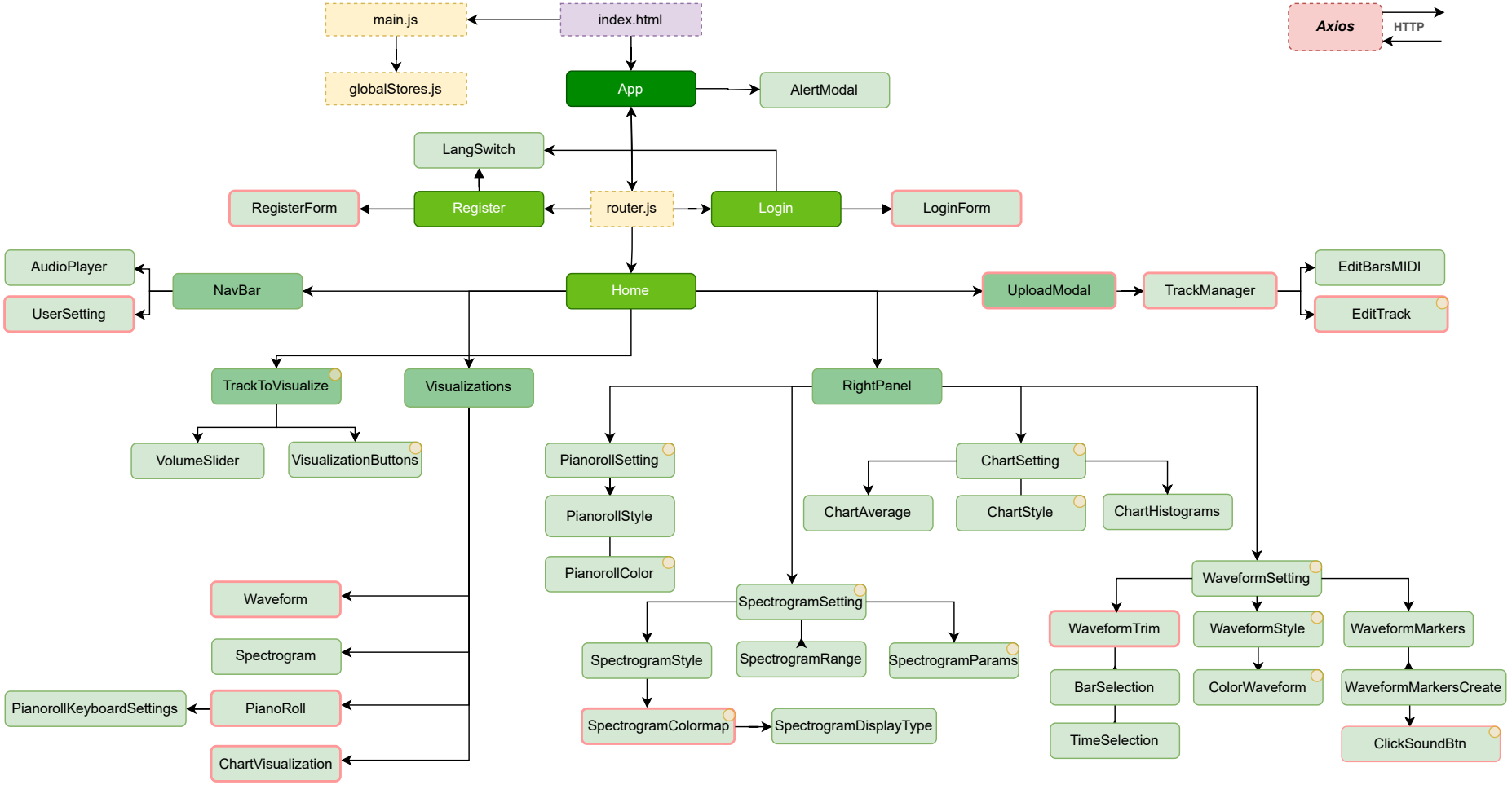
stavu využit nástroj Pinia viz 1.3.2.

Byly vytvořeny celkem čtyři definice store, kdy každý store je definován funkcí `defineStore()`. První store `alertState()` ukládá stav týkající se zobrazování upozornění, viz předchozí kapitola 3.2.1. Druhý store `userInfo()` ukládá informace o přihlášeném uživateli. Nejobsáhlejší store je `trackList()`, který obsahuje všechny informace o nahrávkách pro přihlášeného uživatele. Tento store má také metodu (action) `fetchRecordings()`, která umožňuje získání dat o nahrávkách z databáze pomocí funkce `get_track_list()`, jak bylo popsáno v kapitole 3.1.2. Poslední je pak `trackIndex()`. Ten slouží k ukládání ID aktuálně zvolené nahrávky.

3.2.3 Struktura aplikace

Hlavní komponentou aplikace je `App.vue`. Tato kořenová komponenta obsahuje další komponenty, které mohou být dále vnořeny do dalších komponent a tvořit tak hierarchickou stromovou strukturu.

Komponenty s růžovým ohraničením značí, že v rámci těchto komponent je vytvořena instance Axios knihovny, která slouží k odesílání a přijímání požadavků ze serveru. Některé komponenty obsahují také funkci `updateRecording()`, což je naznačeno žlutě vyplněným kolečkem v pravém horním rohu. Tato funkce je využitelná napříč celou aplikací a slouží k aktualizaci nastavení vizualizací v databázi pro konkrétního uživatele. Celková struktura aplikace, vyvíjené v rámci této práce, je zobrazena na následujícím obrázku 3.3.



Obr. 3.3: Stromová struktura aplikace.

Směrování na straně klienta

Soubor route.js obsahuje definice cest (routes), které jsou použity v aplikaci pro navigaci mezi různými stránkami a komponentami. Tyto cesty jsou definovány jako objekty a každý objekt obsahuje několik vlastností, jako je cesta, název cesty a komponenta, která se má zobrazit.

Při prvotním spuštění aplikace je uživatel směrován na cestu `'/'`, která odkazuje na komponentu `Home.vue`. Před vstupem se provede kontrola `beforeEnter`, kdy funkce `loggedIn()` zavolá požadavek na server a zkontroluje platnost tokenu JWT, viz kapitola popisující backendovou část aplikace 3.1.2. V případě úspěšné autorizace navrátí server odpověď obsahující textový řetězec `'jwt is valid'`, pomocí kterého je uživateli umožněn vstup do části `Home.vue`, viz ukázka kódu 3.1. V opačném případě je uživatel přesměrován do komponenty `Login.vue`, kde se musí přihlásit, nebo zaregistrovat přesměrováním do komponenty `Register.vue`.

Výpis 3.1: Ověření před vstupem na hlavní stránku aplikace.

```
const routes = [
    {
        path: '/',
        name: 'Home',
        component: Home,

        beforeEnter: async () => {

            await loggedIn()
            .then(response => {
                if (response.data.message === 'jwt is valid')

                // při splnění této podmínky je uživatel
                // přesměrován na  obsah komponenty Home.vue
```

Registrační formulář

`RegisterForm.vue` je potomek komponenty `Register.vue`. Obsahuje registrační formulář a systém validace zadaných vstupních údajů, které následně pošle na server a uloží uživatele do databáze. Druhým potomkem je komponenta `LangSwitch.vue`. Ta obsahuje plugin vue-i18n sloužící k překladu obsahu webové aplikace.

Registrační formulář je vytvořen pomocí klasického `<form>` elementu. Jednotlivé `<input>` elementy jsou pomocí direktivy `v-model` propojeny s reaktivním objektem `formData`, který obsahuje reaktivní proměnné, kam se ukládá zadaný email, uživatelské jméno a heslo.

Při registraci je nutné zadat heslo dvakrát kvůli kontrole. K ověření shodnosti hesel je využita knihovna Vuelidate[4], která kontroluje, zda nebyly použity nedovolené znaky a zda zadané údaje splňují minimální nebo maximální délku. Díky reaktivitě parametrů vstupujících do funkce `useVuelidate()` probíhá ověření okamžitě a uživatel tak hned vidí informace u chybně vyplněného pole.

V případě korektně vyplněného formuláře je uživateli umožněno odeslat data na server. Data se zasílají pomocí funkce `registerNewUser()` a v případě úspěšné registrace na straně serveru dojde k přesměrování na komponentu `Login.vue`.

Přihlašovací formulář

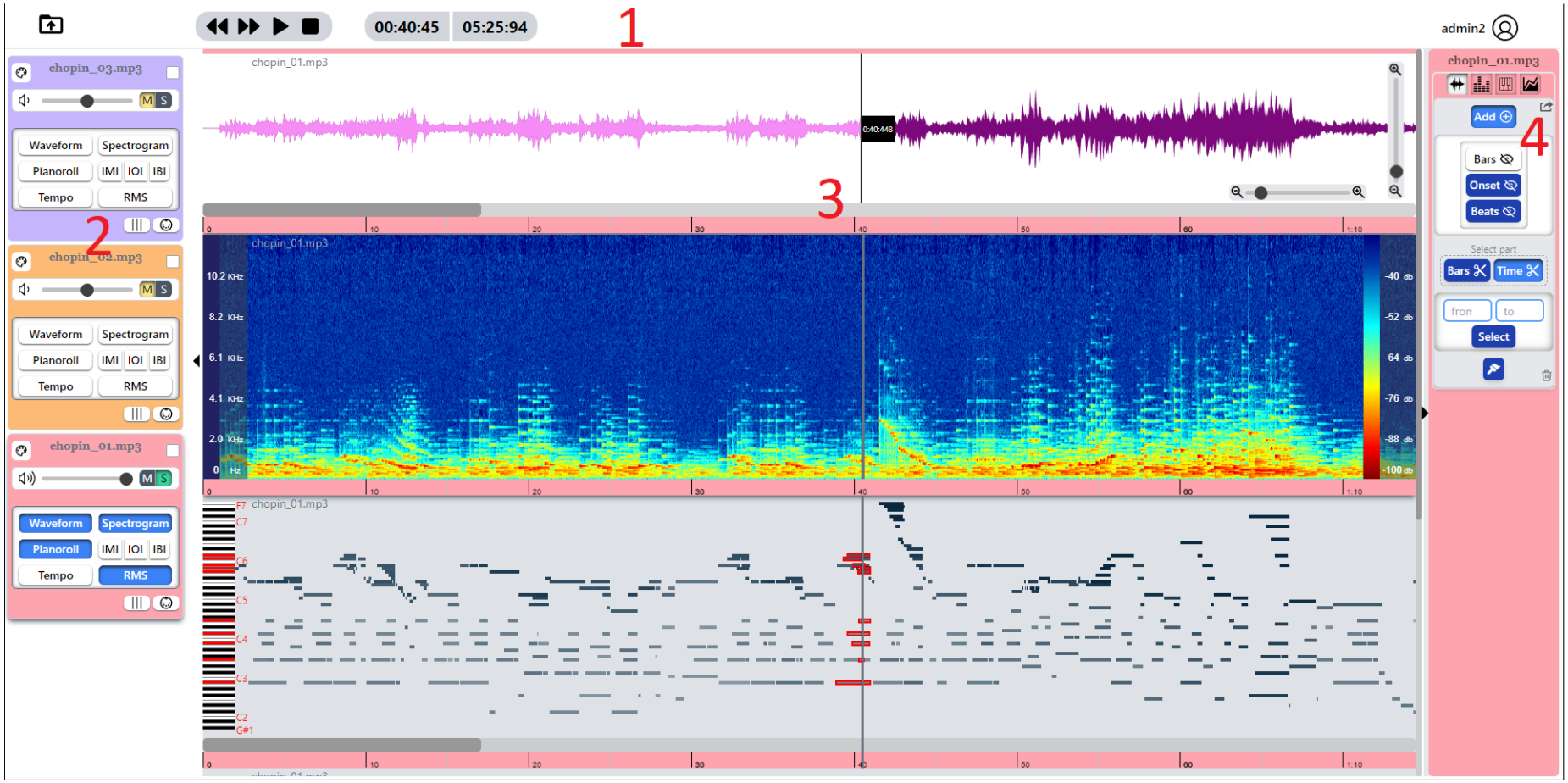
`LoginForm.vue` je potomek komponenty `Login.vue`. Tato komponenta stejně jako `RegisterForm.vue` obsahuje formulář a systém validace zadaných údajů. Kontrola zadání správných přihlašovacích údajů probíhá na straně serveru viz 3.1.2. V případě úspěšné autorizace uživatele dojde k jeho přesměrování na komponentu `Home.vue`.

`Hlavní stránka aplikace`

Vzhledem k tomu, že tato aplikace je postavená na principech SPA 1.3.5, je celé dění postaveno na jediné stránce bez nutnosti dalšího směrování. Tato stránka je vygenerována komponentou `Home.vue`. Po načtení této komponenty nejprve dojde k zavolání funkce `fetchRecordings()` ze store `trackList()`. Tím se získají všechna potřebná data z databáze. Tyto data jsou dále poskytnuta pomocí `props` dalším komponentám. V případě potřeby dojde k výpočtu a vykreslení vizualizací podle uloženého nastavení.

Podle stromové struktury viz obr. 3.3 lze komponentu `Home.vue` rozdělit na pět základních částí – správce souborů, hlavní panel, správce vizualizací, vizualizace a nastavení vizualizací. Kromě správce souborů, který se zobrazí po stisknutí tlačítka v levém horním rohu, jsou všechny části patrné z hlavní stránky aplikace zobrazené na následujícím obrázku 3.4.

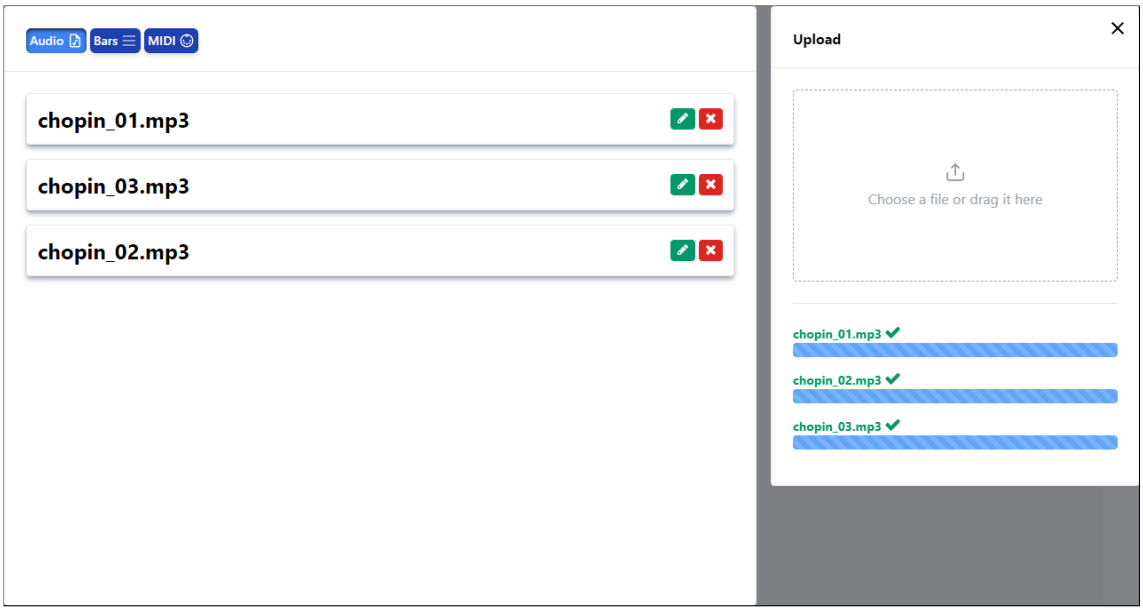
[4] `https://vuelidate-next.netlify.app/`



Obr. 3.4: Hlavní stránka aplikace.

Správce souborů

Komponenta `UploadModal.vue` umožňuje uživatelům nahrávat audio soubory, časové anotace začátků taktů a MIDI soubory do aplikace. Všechny nahrávané soubory jsou ukládány na serveru a jsou přístupné pouze uživatelům, kteří dané soubory nahráli. Tato komponenta je rozdělena na dvě části. Pravá část uživatelům umožňuje nahrát soubory na server. Toho lze dosáhnout buď výběrem konkrétního souboru z počítače nebo jeho přetažením pomocí funkce drag-and-drop. Uživatel zde vidí informace o stavu nahrávání souboru. Levá část slouží ke kontrole nahraného obsahu. Uživatel má možnost nahrávku přejmenovat případně smazat z databáze. Funkce, které tyto úlohy zajišťují na serveru jsou popsány v kapitole 3.1.2. Komponenta z aplikace je zobrazena na následujícím obrázku 3.5.



Obr. 3.5: Správce souborů.

Hlavní panel

Jedná se o horní lištu sloužící k zobrazení správce souborů, audio přehrávače, časových údajů a nastavení uživatele. Tato lišta je implementována pomocí komponenty `NavBar.vue`, která využívá globálního store `userInfo()` k zobrazení přihlášeného uživatele.

Součástí horní lišty je také komponenta `UserSetting.vue`, která umožňuje uživatelům odhlásit se a přepínat aplikaci mezi angličtinou a češtinou. Další komponentou je `AudioPlayer.vue`. Ta umožňuje základní práci s přehrávačem nahrávek a zobrazuje aktuální pozici ve skladbě a celkovou délku nahrávky. Komponenta `AudioPlayer.vue` využívá globálního store `trackIndex()` k získání seznamu všech

nahrávek, které se mají současně přehrávat. Tyto nahrávky jsou přidávány do tohoto globálního seznamu z komponenty `TrackToVisualize.vue`, která se nachází v části správce vizualizací. Globální store byl využit v tomto případě proto, že komponenty, které potřebují sdílet data, jsou od sebe v aplikaci vzdálené. Použití `props` by tedy v tomto případě bylo nepřehledné a nepraktické.

Správce vizualizací

Tato část se nachází na levé straně hlavního okna aplikace a skládá se z komponenty `TrackToVisualize.vue`. Tato komponenta se zobrazuje pro každou vybranou nahrávku z části správce souborů a umožňuje uživatelům různé nastavení.

Uživatelé mohou měnit barvu pozadí pro každou nahrávku a také přidat nahrávky do globálního seznamu současně přehrávaných nahrávek. Dále je možné přiřadit soubory s takty a MIDI soubory k nahrávce, měnit hlasitost nahrávky a případně nahrávku zcela ztlumit. Režim sólo pak umožňuje přehrávat pouze vybranou nahrávku.

Tato část je dále složena z komponenty `VisualizationButtons.vue`. Díky ní má uživatel přehled o tom, které vizualizace jsou již spočítány a zobrazeny, které jsou skryté a které ještě nebyly vypočítány.

Vizualizace

Část aplikace zodpovědná za vizualizace tvoří většinu rozložení aplikace. Kořenovou komponentou je `Visualizations.vue`, která zajišťuje zobrazení jednotlivých vizualizací. V šabloně této komponenty se pomocí direktivy `v-for` iteruje seznam všech zobrazených nahrávek a pro každou z nich se zobrazí příslušné vizualizace. Jednotlivé vizualizace zprostředkovávají následující komponenty: `Waveform.vue`, `Spectrogram.vue`, `Pianoroll.vue` a `ChartVisualization.vue`. Každá z těchto komponent má vlastní funkci životního cyklu `onMounted()`, která se spustí po načtení komponenty. V této funkci je následně spuštěn kód určený k výpočtu a vykreslení příslušné vizualizace podle nastavení získané z databáze přihlášeného uživatele.

Každá z uvedených komponent importuje komponentu `LoadingOverlay.vue`, která zajišťuje zobrazení animace načítání v případě, že probíhá výpočet dané vizualizace. Dále pak komponentu `NameVisualization.vue`. Ta slouží k zobrazení jména nahrávky v levém horním rohu pro všechny vizualizace. Tento název je rovněž možné skrýt pomocí funkce `useHideElement()`, která je dostupná v composables, viz kap. 3.2.1. Komponenty `Spectrogram.vue` a `Pianoroll.vue` pak využívají komponentu `Cursor.vue`. Tato komponenta zajišťuje zobrazení kurzoru v obou vizualizacích a jejich synchronizaci s kurzorem ve vizualizaci `Waveform.vue`.

Nastavení vizualizací

Tato část je tvořena komponentou `RightPanel.vue`. Slouží k nastavení příslušných vizualizací. Možné nastavení pro jednotlivé vizualizace je popsané v kapitole 2.3. Tato komponenta umožňuje přepínání mezi nastavením těch vizualizací, které jsou pro danou nahrávku vypočítané. Pro každou vizualizaci je vytvořena vlastní komponenta s nastavením – `WaveformSetting.vue`, `SpectrogramSetting.vue`, `PianorollSetting.vue` a `ChartSetting.vue`. Ve Vue.js aplikaci bylo přepínání realizované pomocí dynamických komponent. Za tímto účelem bylo nutné vytvořit `<component>` element se speciálním atributem `is`, který může obsahovat buď název registrované komponenty nebo objekt importované komponenty. Pomocí elementu `<KeepAlive>` je zajištěno, že neaktivní komponenty lze uchovat v paměti, takže není třeba je znovu vytvářet při každém přepnutí, což sníží nároky na výkon.

Všechny využívají společné komponenty jako např. `SizeSetter.vue`. Tato komponenta umožňuje uživatelům měnit výšku příslušných vizualizací. Pro implementaci tohoto prvku byla využita knihovna s názvem vue-slider[5]. Tento prvek slouží jako posuvník (slider), který uživatelé mohou použít ke změně hodnoty výšky vizualizace. Dále pak společná komponenta `ColorsPicker.vue` slouží ke změně barev vizualizací. Ke změně barvy časového průběhu zvukové stopy však byla využita knihovna ColorPicker [6], která umožňuje použití širšího spektra barev.

[5] `https://nightcatsama.github.io/vue-slider-component/#/`

[6] `https://github.com/kleinfreund/vue-accessible-color-picker`

Závěr

Cílem této práce byla implementace webové aplikace umožňující výpočet a vizualizaci hudebních parametrů. Před samotnou implementací aplikace bylo třeba provést rešerši vhodných technologií pro vývoj této aplikace. Největší pozornost byla kladena na popis frameworku Vue.js, použitého pro tvorbu frontendové části aplikace v jazyce JavaScript, a frameworku Flask, použitého pro tvorbu backendové části aplikace v jazyce Python.

Následoval popis parametrů, které je webová aplikace schopna vizualizovat. V rámci této práce byl implementován registrační a přihlašovací formulář pro uživatele, možnost nahrávání souboru na server, základní práce s nahrávkami, vizualizace časového průběhu zvukového signálu, spektrogramu, pianorollu, vizualizace vývoje tempa, efektivní hodnoty zvukového signálu (RMS), detekovaných začátků not a dob a časových intervalů začátků taktů a detekovaných not i dob. Dále aplikace umožňuje omezení vizualizace na uživatelem zadaný časový rozsah. Po nahrání souboru s časovými anotacemi začátků taktů je možné takty zobrazit a rovněž vybrat úsek podle zvoleného rozsahu taktů. Každá vizualizace umožňuje několik možných nastavení, které se pro přihlášeného uživatele ukládají do databáze.

Poté byl proveden rozbor architektury aplikace. Bylo popsáno propojení s databází pomocí rozšíření SQLAlchemy, díky kterému je umožněna komunikace mezi Flask serverem a databází PostgreSQL bez nutnosti používání jazyka SQL. Pozornost rovněž byla věnována popisu funkcí, které jsou implementovány na straně serveru a stromové struktuře komponent, které tvoří frontendovou část na straně klienta.

V budoucnu by mohla být aplikace rozšířena o možnost přizpůsobit si nastavení pro přihlášeného uživatele a umožnit volnější práci s rozvržením vizualizací. Kromě toho by aplikaci bylo možné doplnit o další vizualizace, které popisují hudební nahrávky z hlediska harmonického vývoje, rytmické struktury a dynamiky. Výpočet všech vizualizací je poměrně rychlý, s výjimkou spektrogramu. Získání dat spektrogramu, zejména u delších nahrávek, totiž vyžaduje poměrně velký výpočetní výkon. Další časovou prodlevu pak způsobí vykreslení těchto dat pomocí knihovny Plotly.js. Řešení by mohlo být pomocí optimalizace algoritmu pro výpočet spektrogramu, případně nalezení vhodné knihovny, která umožňuje rychlejší výpočet spektrogramu.

Literatura

[1] MÜLLER, Meinard. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications* Springer International Publishing, 2015. ISBN 978-3-319-21945-5.

[2] M. Schedl , E. Gómez and J. Urbano. *Music Information Retrieval: Recent Developments and Applications.*, 2014, ISBN: 978-1-60198-807-2. [online]. [cit. 1.5.2023]. Dostupné z: `https://doi.org/10.1561/1500000042`

[3] BÖCK, Sebastian, Filip KORZENIOWSKI, Jan SCHLÜTER, Florian SEBASTIAN BÖCK, FILIP KORZENIOWSKI, JAN SCHLÜTER, FLORIAN KREBS, GERHARD WIDMER a Gerhard WIDMER. *Madmom: a new Python Audio and Music Signal Processing Library* [online]. [cit. 30.11.2022]. Dostupné z: `https://doi.org/10.48550/arXiv.1605.07008`

[4] *openSMILE 3.0 - audEERING.* [online]. [cit. 04.05.2023]. Dostupné z: `https://www.audeering.com/research/opensmile/`

[5] LARTILLOT, Olivier a Petri TOIVIAINEN. *MIR in Matlab (II): A Toolbox for Musical Feature Extraction from Audio* [online]. [cit. 4.12.2022]. Dostupné z: `https://www.researchgate.net/publication/220723440_MIR_in_Matlab_II_A_Toolbox_for_Musical_Feature_Extraction_from_Audio`

[6] RAWLINSON, Hugh, Nevo SEGA a Jakub FIALA. *Meyda: an audio feature extraction library for the Web Audio API* [online]. [cit. 4.12.2022]. Dostupné z: `https://wac.ircam.fr/pdf/wac15_submission_17.pdf`

[7] BROSSIER, Paul. *Aubio Documentation* [online]. [cit. 4.12.2022]. Dostupné z: `https://buildmedia.readthedocs.org/media/pdf/aubio/latest/aubio.pdf`

[8] *librosa — librosa 0.10.0 documentation* [online]. [cit. 4.12.2022]. Dostupné z: `https://librosa.org/doc/latest/index.html`

[9] CANNAM, Chris, Christian LANDONE a Mark SANDLER. *Sonic Visualiser: An Open Source Application for Viewing,Analysing, and Annotating Music Audio Files* [online]. New York, USA: ACM Press, 2010. ISBN 9781605589336. Dostupné z: `https://www.researchgate.net/publication/221573417_Sonic_visualiser_an_open_source_application_for_viewing_analysing_and_annotating_music_audio_files`

[10] *Web Audio API - Web APIs* [online]. Copyright ©1998 [cit. 06.12.2022]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

[11] ASLESON, Ryan a Nathaniel T. SCHUTTA. *Ajax: vytváříme vysoce interaktivní webové aplikace*. Brno: Computer Press, 2006. ISBN 80-251-1285-3.

[12] SABAH, Al-Fedaghi *Developing Web Applications* [online]. [cit. 22.11.2022]. Dostupné z: https://www.researchgate.net/publication/228849246_Developing_Web_Applications

[13] *Getting Started – Windi CSS* [online]. [cit. 29.11.2022]. Dostupné z: https://windicss.org/guide/

[14] HOLZNER, Steven. *Mistrovství v AJAXu*. Brno: Computer Press, 2007. ISBN 978-80-251-1850-4.

[15] NGUYEN, Don. *Node.js Okamžitě*. Přeložil Ondřej BAŠE. Brno: Computer Press, 2016. ISBN 978-80-251-4820-4.

[16] *Essentia 2.1-beta6-dev documentation* [online]. [cit. 03.05.2023]. Dostupné z: https://essentia.upf.edu/

[17] *Yaafe - audio features extraction* [online]. [cit. 04.05.2023]. Dostupné z: https://yaafe.sourceforge.net/

[18] *Marsyas - Music Analysis, Retrieval and Synthesis for Audio Signals* [online]. [cit. 04.05.2023]. Dostupné z: https://github.com/marsyas/marsyas

[19] KOPEC, David. *History of Web Programming*. [online]. 2014. Dostupné z: https://www.researchgate.net/publication/300357242_History_of_Web_Programming

[20] *Serverová řešení* [online]. [cit. 29.11.2022]. Dostupné z: https://is.muni.cz/el/1431/podzim2014/Z8188/um/Webkart_06_server.pdf

[21] *PHP – Introduction* [online]. [cit. 30.11.2022]. Dostupné z: https://www.php.net/manual/en/introduction.php

[22] MACRAE, Callum. *Vue.js: up and running : building accessible and performant web apps*. Sebastopol: O'Reilly, 2018. ISBN 978-1-491-99724-6.

[23] *Vite – Next Generation Frontend Tooling*. [online]. Copyright © 2019 [cit. 22.11.2022]. Dostupné z: https://vitejs.dev/

[24] *ASP.NET – Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET* [online]. Copyright © Microsoft 2022 [cit. 22.11.2022]. Dostupné z: https://learn.microsoft.com/en-us/archive/msdn-magazine/2013/november/msdn-magazine-november-2013

[25] *Vue.js – The Progressive JavaScript* [online]. [cit. 22.11.2022]. Dostupné z: https://vuejs.org/guide/introduction.html

[26] *Extensions for Visual Studio family of products – Visual Studio Marketplace* [online]. Copyright © 2022 Microsoft [cit. 22.11.2022]. Dostupné z: https://marketplace.visualstudio.com/VSCode

[27] *npm* [online]. [cit. 22.11.2022]. Dostupné z: https://www.npmjs.com/

[28] *PyCharm: the Python IDE for Professional Developers by JetBrains. JetBrains: Essential tools for software developers and teams* [online]. [cit. 22.11.2022]. Dostupné z: https://www.jetbrains.com/pycharm/

[29] *Cross Site Request Forgery (CSRF), the Open Source Foundation for Application Security* [online]. [cit. 23.11.2022]. Dostupné z: https://owasp.org/www-community/attacks/csrf

[30] *Getting Started – Axios Docs* [online]. [cit. 23.11.2022]. Dostupné z: https://axios-http.com/docs/intro

[31] *How to Use Flask-SQLAlchemy to Interact with Databases in a Flask Application* [online]. [cit. 29.11.2022]. Dostupné z: https://www.digitalocean.com/community/tutorials

[32] *JSON Web Token Introduction - jwt.io. JSON Web Tokens - jwt.io* [online]. [cit. 23.11.2022]. Dostupné z: https://jwt.io/introduction

[33] *What is RMS in the audio world?* [online]. [cit. 21.4.2023]. Dostupné z: https://majormixing.com/what-is-rms-in-audio-world

Seznam symbolů a zkratek

MIR	Music Information Retrieval
MIDI	Musical Instrument Digital Interface
RMS	Root Mean Square – efektivní hodnota zvukového signálu
FFT	Fast Fourier Transform – rychlá Fourierova transformace
STFT	Short-Time Fourier Transform – krátkodobá Fourierova transformace
BPM	Beats per minute – údery za minutu
WWW	World Wide Web
HTML	Hypertext Markup Language
SSI	Server Side Includes
CGI	Common Gateway Interface
MVC	Model-View-Controller
CSS	Cascading Style Sheets
VSC	Visual Studio Code
ORM	Object–relational mapping – objektivně relační zobrazení
SQL	Structured Query Language – standardizovaný strukturovaný dotazovací jazyk
SPA	Single Page Application
HTTP	Hypertext Transfer Protocol
AJAX	Asynchronous JavaScript and XML
CSRF	Cross-site Request Forgery
NPM	Node Package Manager
API	Application Programming Interface
JWT	JSON Web Token
JSON	JavaScript Object Notation

Obsah elektronické přílohy

Zdrojové soubory webové aplikace pro vizualizaci parametrů hudebních nahrávek jsou umístěny v GitHub repozitáři. Odkaz na repozitář se nachází na následující adrese `https://github.com/MartinKlimes/audio-feature-visualization/tree/master`. Postup pro spuštění aplikace se nachází v souboru `README.md`. Pro publikování aplikace byla využita platforma Render.com. Render.com je cloudová platforma pro nasazení a správu webových aplikací a služeb. Aplikace je umístěna na adrese `https://audio-feature-visualization.onrender.com/`.