**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra Informačních technologií**

# Systém pro vizuální reprezentaci MIDI souborů s interaktivním osvětlením

Bakalářská práce

Autor: Lukáš Kovář
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Jaroslav Langer

Hradec Králové duben 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 26.4.2024 Lukáš Kovář

Poděkování:
Děkuji vedoucímu bakalářské práce Ing. Jaroslavu Langerovi za metodické vedení práce a firmě Spectoda za poskytnutí technologií k vizualizaci.

# Abstrakt

## Název: Systém pro vizuální reprezentaci MIDI souborů s interaktivním osvětlením

Tato bakalářská práce popisuje vývoj a implementaci webové aplikace pro přehrávání MIDI souborů s vizualizací na připojených světelných instalacích a zobrazováním titulků pro karaoke. Aplikace je navržena tak, aby uživatelům poskytovala flexibilní a pohodlný přístup k jejich MIDI souborům odkudkoliv bez nutnosti instalace jakéhokoliv softwaru.

Klíčovým aspektem aplikace je schopnost vizualizovat přehrávané MIDI soubory na světelných instalacích podle aktuálně hrajících nástrojů. Toho je dosaženo pomocí konfigurovatelných zvukových map a tematických módů optimalizovaných pro různé typy světel, zejména RGB LED pásky. Vizualizace je implementována pro systém Spectoda s možností budoucího rozšíření na další systémy.

Webové rozhraní aplikace nabízí uživatelům intuitivní ovládání a možnost přepínat mezi různými hudebními nástroji, zvukovými mapami a vizuálními módy. Díky tomu mohou uživatelé přizpůsobit svůj zážitek z přehrávání MIDI souborů podle svých preferencí.

Tato aplikace představuje inovativní řešení, které kombinuje audio přehrávání, vizuální efekty a karaoke titulky v jediném snadno přístupném webovém rozhraní. Nabízí tak uživatelům jedinečný a interaktivní multimediální zážitek při přehrávání jejich oblíbených MIDI souborů.



**Klíčová slova**: MIDI, JavaScript, TypeScript, Spectoda, Vizualizace hudby, MIDI přehrávač, Karaoke, Next.js, React, tRPC

# Abstract

## Title: System for visual representation of MIDI files with interactive lighting

This bachelor thesis describes the development and implementation of a web application for playing MIDI files with visualization on connected light installations and displaying subtitles for karaoke. The application is designed to provide users with flexible and convenient access to their MIDI files from anywhere without the need to install any software.

A key aspect of the app is the ability to visualize MIDI files being played on light installations according to the instruments currently playing. This is achieved through configurable sound maps and thematic modes optimized for different types of lights, especially RGB LED strips. Visualization is implemented for the Spectoda system with the possibility of future expansion to other systems.

The application's web interface offers users intuitive controls and the ability to switch between different musical instruments, sound maps and visual modes. This allows users to tailor their MIDI file playback experience to their preferences.

This app is an innovative solution that combines audio playback, visual effects and karaoke subtitles in a single easy-to-use web interface. It offers users a unique and interactive multimedia experience when playing their favorite MIDI files.



**Key words:** MIDI, JavaScript, TypeScript, Spectoda, Music Visualization, MIDI Player, Karaoke, Next.js, React, tRPC

# Obsah

# Seznam obrázků



# Seznam tabulek

# Seznam ukázek kódu

# Seznam zkratek

| | |
|---|---|
| DAW | Digital Audio Workstation |
| MIDI | Musical Instrument Digital Interface |
| ORM | Object-Relational Mapping |
| MIT | Massachusetts Institute of Technology (licence) |
| JS | JavaScript |
| TS | TypeScript |
| BLE | Bluetooth Low Energy |
| IDE | Integrated Development Environment |
| DB | Database |
| SQL | Structured Query Language |
| GM | General MIDI |
| PG | PostgreSQL |
| HOC | Higher Order Component |
| JSON | JavaScript Object Notation |
| tRPC | Remote Procedure Call |

# 1 Úvod

Hudba je všudypřítomná a její vizualizace představuje fascinující oblast zájmu. Protokol MIDI (Musical Instrument Digital Interface) nabízí jedinečný způsob reprezentace hudebních dat, který umožňuje analyzovat, přehrávat a vizualizovat hudbu na základě informací o hře jednotlivých nástrojů.

Výsledná aplikace umožní uživatelům vytvářet vizuální efekty reagující na přehrávanou hudbu, jako například „svítící piano“ či „pulzující bubny“. Webová aplikace bude zahrnovat na front-endovou a back-endovou část, přičemž back-endová část bude zodpovědná za ukládání MIDI skladeb a nastavení vizualizace pod uživatelskými účty.

Práce vychází z poznatků získaných z odborné literatury, z online zdrojů a ze znalostí načerpané z praxe ve firmě Spectoda. Při implementaci budou využity moderní technologie a frameworky pro vývoj webových aplikací.

# 2 Cíl a metodika práce

Cílem této bakalářské práce je prozkoumat možnosti využití MIDI pro vizualizaci hudby s interaktivním osvětlením. V praktické části bude implementován webový MIDI přehrávač s podporou načítání MIDI souborů, ovládání přehrávání, výběru nástrojů a zobrazení vizualizací. Bude integrována syntetizace zvuku pomocí WebMIDI a soundfontů. Aplikace bude obsahovat back-endovou část pro ukládání a správu MIDI souborů v databázi PostgreSQL.

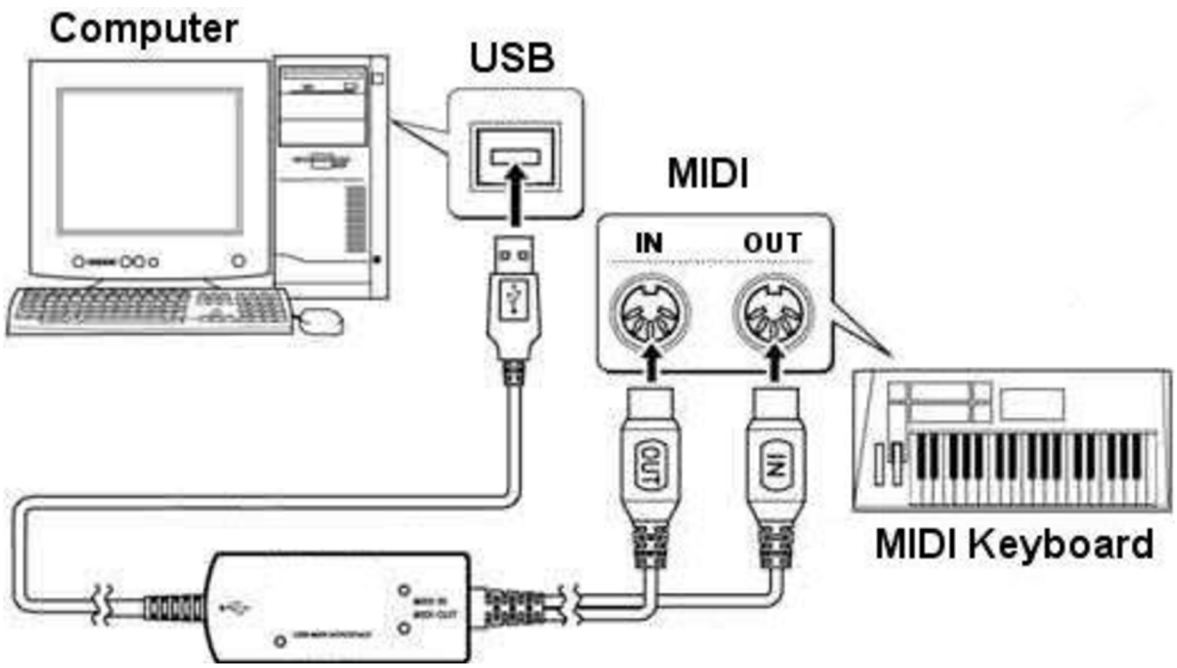
Dále bude navržen algoritmus pro vizualizaci MIDI dat na světelných instalacích Spectoda s uživatelským rozhraním pro přepínání efektů a módů. Aplikace bude mít funkci zobrazování synchronizovaných titulků pro karaoke a koncept uživatelských účtů. Při implementaci budou využity moderní webové technologie jako React, Next.js, TypeScript, Tailwind CSS, Node.js a tRPC.

# 3 Teoretické poznatky

V této části se zabýváme teoretickou částí zvoleného tématu.
Budou zde vysvětleny pojmy, nástroje a popis použitých technologií.

## *3.1 MIDI*

MIDI neboli (Musical Instrument Digital Interface) je široce akceptován hudebníky již od jeho vzniku v roku 1982. Jedná se o velice efektivní a úsporný formát dokumentující prezentaci hudby, a to primárně hudebních nástrojů. Využití je nejen v hudbě, ale také ve starších hrách, které používali zvukové karty umožňující zvukové projevy za pomocí předání informace skrze MIDI. [1]
Jeho standardizovaná publikace GM (General MIDI) je nejpopulárnější a pracuje s ní tato práce.




**Obrázek 1 - MIDI konektor s převodníkem do USB – převzato [2]**

Pro připojení MIDI zařízení k počítači používáme v dnešní době převodníky na USB konektor (viz obrázek s pianem výše). Ten nám umožňuje oboustrannou komunikaci mezi PC a připojeným zařízením. Novější modely keyboardů jej mají přímo v sobě a můžeme jej tedy připojit k PC skrze USB nebo bluetooth rozhraní napřímo.

### 3.1.1 MIDI soubor

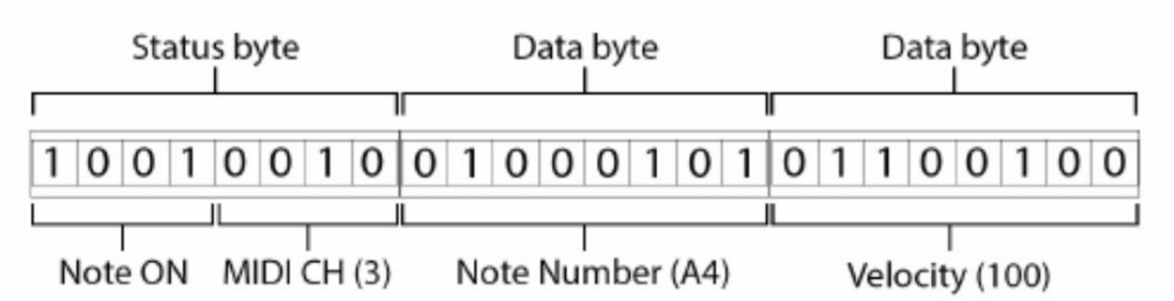
Účelem MIDI souboru je poskytnout výměny časové označených dat typu MIDI mezi různými program nebo i vícero počítačích.
Cílem je kompaktní reprezentace, díky čemuž zabírá minimum místa na disku.
MIDI soubory obsahují jeden nebo vícero proudů s časovými údaji pro danou událost.
Je zde podpora struktury skladeb, sekvencí, stop, informací o tempu a časové značce.
S daty mohou být uloženy názvy stop a další popisné informace.
Specifikace definuje 8bitový binární tok použitý v souboru, z nichž 7 bitů se používá pro přenos MIDI, případně lze přepsat do hexadecimální podoby.
Samotná specifikace se nezabývá tím, jak jsou MIDI instrukce přenášeny. [3]

### 3.1.2 MIDI versus Digitální hudba

Protokol MIDI byl původně určený pro propojení hudebních syntezátorů, nyní se hojně využívá v hudebních editorech jako alternativa k digitalizovanému zvuku. Hlavní výhodou MIDI je jeho malá velikost souborů ve srovnání s velkými soubory vzorkovaného zvuku, jako jsou WAV nebo MP3 soubory. MIDI soubory jsou malé, protože obsahují pouze instrukce pro syntezátor, jaké tóny má hrát, nikoliv samotné zvukové záznamy, takzvané **soundfonty**.
Tato efektivita znamená menší zatížení počítačové paměti při přehrávání zvuku, v minulosti toto bylo velmi důležité, jelikož počítače na reprodukci vzorkovaného zvuku neměli výkon, ale díky MIDI mohli posílat instrukce zvukové kartě s MIDI syntetizátorem a mohli reprodukovat zvuk piana, bubnu nebo třeba výstřelu z pistole, toto bylo užitečné zejména ve starých počítačových hrách.
MIDI oproti vzorkovanému zvuku umožňuje snadné úpravy a nezávislou změnu rychlosti a tónu zvuků, což je ideální pro aplikace například karaoke nebo digitální skladbu hudebních skladeb.

### 3.1.3 MIDI zprávy

Zpráva MIDI se skládá z osmibitového stavového bajtu, za kterým obvykle následuje jeden nebo dva datové bajty. Existuje řada různých typů zpráv MIDI.
Na nejvyšší úrovni jsou zprávy MIDI klasifikovány jako zprávy kanálu nebo systémové zprávy. [3]




**Obrázek 2 – MIDI struktura zprávy – [4]**

Zpráva MIDI se skládá ze série bajtů, kde každý bajt je osmi bitů. Typická MIDI zpráva obsahuje jeden status bajt, následovaný jeden nebo dvěma datovými bajty. Status bajt definuje typ zprávy (např. note on, note off, kontrolní změna atd.) a MIDI kanál, přes který je zpráva poslána (existuje 16 možných MIDI kanálů). Datové bajty poskytují specifické informace, jako je výška tónu noty nebo její dynamika (velocity), což odkazuje na intenzitu, s jakou byla nota zahrána.

Níže je základní přehled hlavních typů MIDI zpráv:

| **Typ zprávy** | **Status Byte** | **Popis** |
|---|---|---|
| Note Off | 0x80 - 0x8F | Signál k ukončení hraní noty. |
| Note On | 0x90 - 0x9F | Signál k zahájení hraní noty. Pokud má rychlost (velocity) hodnotu 0, funguje jako Note Off. Rozsah hodnot je 0-127. |
| Polyphonic Key Pressure (Aftertouch) | 0xA0 - 0xAF | Úprava citlivosti pro jednotlivé noty (již hrající). |
| Control Change | 0xB0 - 0xBF | Změna ovládacího signálu (např. hlasitost). |
| Program Change | 0xC0 - 0xCF | Změna programu (nástroje) na daném kanálu. |
| Channel Pressure | 0xD0 - 0xDF | Změna citlivosti jednotlivých not na sílu stisku kláves. |
| Pitch Wheel Change | 0xE0 - 0xEF | Změna výšky tónu noty. |

**Tabulka 1 - Typy základních zpráv – převzato, upraveno [3]**

Toto jsou základní zprávy a hlavní informace se kterýma budeme pracovat v praktické části při vizualizaci. Za pomocí těchto zpráv poznáme, kdy daný nástroj v daném kanálu zahrál tón, ukončil či byl nějakým způsobem modifikován. S těmito daty lze dále pracovat kromě reálného přehrání tónu také v jeho vizualizaci.

Kromě základních MIDI zpráv, které nesou informace o hraných notách a jejich vlastnostech, existují i systémové zprávy v reálném čase. Tyto zprávy umožňují ovládat přehrávání, synchronizaci tempa a výběr skladeb z externích zařízení. Systémové zprávy také řídí komunikaci mezi MIDI zařízeními a dokáží restartovat celý MIDI systém. V praktické části této práce budeme pracovat jak se základními MIDI zprávami pro vizualizaci hraných tónů, tak se systémovými zprávami pro ovládání a synchronizaci přehrávání MIDI sekvencí. [3]

### 3.1.4 Meta zprávy

Meta zprávy v MIDI slouží primárně k sdělení extra informací. Má ale i praktické užití, například zprávy typu lyrics, se používají v programech pro titulky pro karaoke a využijeme je i v této práci v kapitole.

MIDI meta-eventy jsou speciální typy zpráv vložené do MIDI souborů, které slouží k řízení různých aspektů MIDI sekvence, ale přímo nereprezentují hudební noty nebo výkonnostní data. Tyto eventy poskytují dodatečné informace a pokyny, které ovlivňují přehrávání nebo MIDI souborů. [3]

### 3.1.5 Hudební nástroje

MIDI standard GM (General MIDI) je dělí do 8 rodin/kategorií a celkem jich je 128 (díky tomu můžeme nástroje adresovat pomocí 7 bitů).

| PC# | Family | PC# | Family |
|---|---|---|---|
| 1-8 | Piano | 65-72 | Reed |
| 9-16 | Chromatic Percussion | 73-80 | Pipe |
| 17-24 | Organ | 81-88 | Synth Lead |
| 25-32 | Guitar | 89-96 | Synth Pad |
| 33-40 | Bass | 97-104 | Synth Effects |
| 41-48 | Strings | 105-112 | Ethnic |
| 49-56 | Ensemble | 113-120 | Percussive |
| 57-64 | Brass | 121-128 | Sound Effects |

**Tabulka 2 - Základní typy nástrojů – převzato [3]**

V každém souboru by mělo být uvedeno tempo a časová signatura. Pokud se tak neučiní, předpokládá se časová signatura 4/4 a tempo 120 BPM (beats per minute) úderů za minutu. [3]

## *3.2 JavaScript*

JavaScript je dynamický interpretovaný jazyk, který byl vyvinut v roce 1995 Brendanem Eichem, který v té době pracoval v Netscape Communications Corporation. Původně byl nazýván Mocha, ale brzy byl přejmenován na LiveScript a nakonec na JavaScript. [5]

První verze JavaScriptu byla vydána v roce 1995 a byla určena pro zpracování akcí ve webových prohlížečích. Poskytovala programátorům možnost přidávat do webových stránek interaktivní prvky, jako jsou například tlačítka, menu nebo animace.

JavaScript byl od počátku navržen jako klientský skriptovací jazyk, který běží výhradně ve webovém prohlížeči. To znamená, že kód JavaScriptu je interpretován a spuštěn prohlížečem na straně klienta, tedy na počítači uživatele. [5]

V roce 1997 byl JavaScript standardizován organizací ECMA International. Standard ECMAScript definuje syntaxi a sémantiku JavaScriptu.

Od svého vzniku se JavaScript stal jedním z nejpopulárnějších programovacích jazyků na světě. Používá se k vývoji webových aplikací, mobilních aplikací, serverů a dalších aplikací.

*„Každá aplikace, která může být napsána v JavaScriptu, nakonec bude napsána v JavaScriptu.“* Jeff Antwood, zakladatel platformy Stack Overflow [6]

Vývoj JavaScriptu probíhá pod záštitou organizace ECMA International. Tato organizace vydává nové verze standardu ECMAScript, který definuje syntaxi a sémantiku JavaScriptu. [5]

### 3.2.1 Oblasti použití JavaScriptu

JavaScript lze použít v mnoha různých oblastech, včetně:

**Vývoj webových aplikací** – JavaScript je jedním z nejdůležitějších jazyků pro vývoj webových aplikací. Používá se k přidávání interaktivních prvků, animací a dalších funkcí do webových stránek.

**Vývoj mobilních aplikací –** JavaScript lze použít také pro vývoj mobilních aplikací. V tomto případě se JavaScript spouští na mobilním zařízení uživatele.

**Serverové aplikace –** JavaScript lze použít také pro vývoj serverových aplikací. V tomto případě se JavaScript spouští na serveru.

V této práci bude jeho verze Node.js využita k tvorbě backendu (API) rozhraní.

#### 3.2.1.1 Vytváření interaktivního obsahu

JavaScript lze použít také pro vytváření interaktivního obsahu, jako jsou například hry, animace nebo interaktivní prvky pro webové stránky.

#### 3.2.1.2 Výhody JavaScriptu;

JavaScript má řadu výhod:

- je relativně snadno naučitelný (na rozdíl o staticky typovaných jazyků)
- je multiplatformní a je nejpopulárnějším skriptovacím programovacím jazykem na světě
- je podporován ve všech moderních webových prohlížečích, tzn., že webové aplikace a další aplikace, které používají JavaScript, jsou kompatibilní se širokou škálou zařízení. [7]

### 3.2.2 Typescript “superset JavaScriptu”

TypeScript, vyvinutý Microsoftem pod vedením Anderse Hejlsberga, je rozšířením JavaScriptu. Přináší řadu dodatečných funkcí, zatímco zápis kódu se kompiluje zpět do JavaScriptu. Tento jazyk se stal populární díky svému vyspělému typovému systému a podpoře nejnovějších funkcí JavaScriptu. Vývojářům nabízí vynikající podporu

nástrojů v integrovaných vývojových prostředích IDE (Integrated development environment). [8]

Podle výsledků průzkumu StackOverflow z roku 2019 byl TypeScript třetím nejoblíbenějším programovacím jazykem. Jeho oblíbenost pramení z toho, jak řeší některé nejasnosti a potenciální chyby, které mohou vzniknout při používání dynamického JavaScriptu. Samotný JavaScriptový kód je kompilován až za běhu, což může vést k odhalení chyb až v příliš pozdní fázi, zvláště v produkčním prostředí. [8]

TypeScript přináší řešení těchto problémů díky možnosti definování pevných typových konstruktů. Použití tříd, rozhraní a implicitních návratových typů metod zvyšuje předvídatelnost kódu a snižuje pravděpodobnost chyb. Vývojáři tak mohou okamžitě detekovat některé typy chyb přímo při psaní kódu. [8]

Projekt k této práci bude je psán v TypeScriptu, kvůli jeho bezpečnosti a minimalizování chyb.

### 3.2.3 TypeScript versus JavaScript

Při srovnání TypeScriptu a JavaScriptu vychází TypeScript jako jasnější a bezpečnější volba pro vývoj. Například v TypeScriptu můžete vynutit explicitní typ návratové hodnoty funkce, čímž se vyhneme nekonzistencím a chybám, které jsou běžné v JavaScriptu. Takový přístup přispívá k lepší srozumitelnosti, ladění a testování kódu.

```
function readPhoneNumberCountryCode(telephone : string) {
  return telephone.slice(0,3);
}

readPhoneNumberCountryCode("420800600400") // vrací 420
readPhoneNumberCountryCode(420800600400) // Uncaught TypeError: a.slice is not
a function
```

**Ukázka kódu 1 - ukázkové odchycení chyby v TypeScriptu – vlastní zpracování**

Zatímco v Javascriptu bychom na tuto chybu přišli až při kompilaci, díky typům v TypeScriptu nás na to upozorní IDE již při psaní.

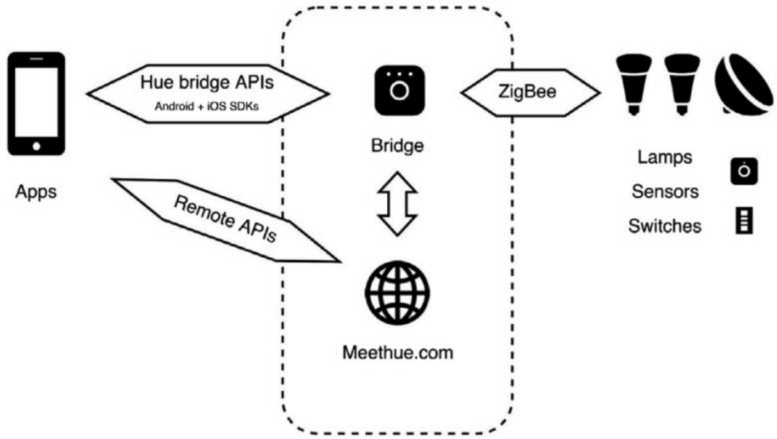
V praxi to znamená, že například chyba, kdy je proměnné přiřazen špatný typ (jako přiřazení čísla místo textového řetězce telefonnímu číslu), bude v TypeScriptu odhalena ihned při kompilaci. V JavaScriptu by taková chyba mohla zůstat nepovšimnuta až do chvíle, kdy by došlo k pokusu o zpracování telefonního čísla v produkčním prostředí. [8]

## 3.3 Systémy řízení světel

Tato kapitola se zabývá vybranými systémy pro řízení světel s podporou RGB led pásků, s kterými by mohla být vizualizace aplikace integrována.

### 3.3.1 Philips Hue

Philips Hue je systém chytrého osvětlení, který využívá LED žárovky, jež lze ovládat pomocí aplikace v chytrém telefonu. Systém funguje prostřednictvím centrálního "mostu", který se připojuje k místní síti Wi-Fi pomocí komunikačního protokolu Zigbee. To umožňuje integraci světel Philips Hue s dalšími zařízeními chytré domácnosti.
Způsob fungování je popsán názorně v obrázku:




**Obrázek 3 - Philips Hue architektura – převzato [9]**

Kromě ovládání po z mobilní aplikace z lokální sítě je možné komunikovat přes most, který zpřístupňuje komunikaci přes internet a připojení na chytrou domácnost.
Pro ovládání skrze API existuje několik knihoven, v případě tohoto projektu padá v úvahu knihovna **huejay** pro Node.js.

```
import huejay from "huejay"

huejay.discover()
  .then(bridges => {
    for (let bridge of bridges) {
      console.log(`Id: ${bridge.id}, IP: ${bridge.ip}`);
    }
  })
  .catch(error => {
    console.log(`An error occurred: ${error.message}`);
  });
```

**Ukázka kódu 2 - skenování Philips Hue v síti – převzato, upraveno [10]**

Po naskenování Philips Hue bridge v síti se můžeme na něj připojit. To zahrnuje inicializaci klienta a následně provedení požadovaných akcí na něm.

```
// Inicializace klienta
const client = new huejay.Client({
  host: bridge.ip,
  // Uživatelské jméno účtu zde. Musíte jej vytvořit v mobilní aplikaci
  username: "YOUR_USERNAME_HERE",
});

// Získání všech světel připojených k Philips Hue Bridge
client.lights
  .getAll()
  .then((lights) => {
    // Projde všechna nalezená světla a vypíše jejich informace
    for (let light of lights) {
      console.log(`Light [${light.id}]: ${light.name}`);
      console.log(`State: ${light.on ? "On" : "Off"}`);
      console.log(`Brightness: ${light.brightness}`);

      // Změna očekávaného stavu světla na "zapnuto"
      light.on = true;
    }
    // Provedení změn na všech světlech
    client.lights.save()
})
```

**Ukázka kódu 3 - Provedení akce na Philips Hue světlech – vlastní zpracování**

Tento kód demonstruje základní interakci s Philips Hue pomocí knihovny huejay v prostředí Node.js. Nicméně, tato práce nebude zahrnovat přímou podporu pro Philips Hue z důvodu omezení komunikace z webového prohlížeče.

Knihovna huejay vyžaduje, aby klient běžel na stejné lokální síti jako Philips Hue bridge, což znamená, že by bylo nutné mít spuštěný Node.js server na stejné síti. To vyžaduje tvorbu a instalaci hostitelské aplikace do PC, nebo na externím zařízení, například typu Raspberry PI. Webové prohlížeče mají omezení pro přímou komunikaci s lokálními zařízeními z bezpečnostních důvodů, což znemožňuje přímou integraci Philips Hue do webové aplikace bez použití dodatečného serveru jako prostředníka.

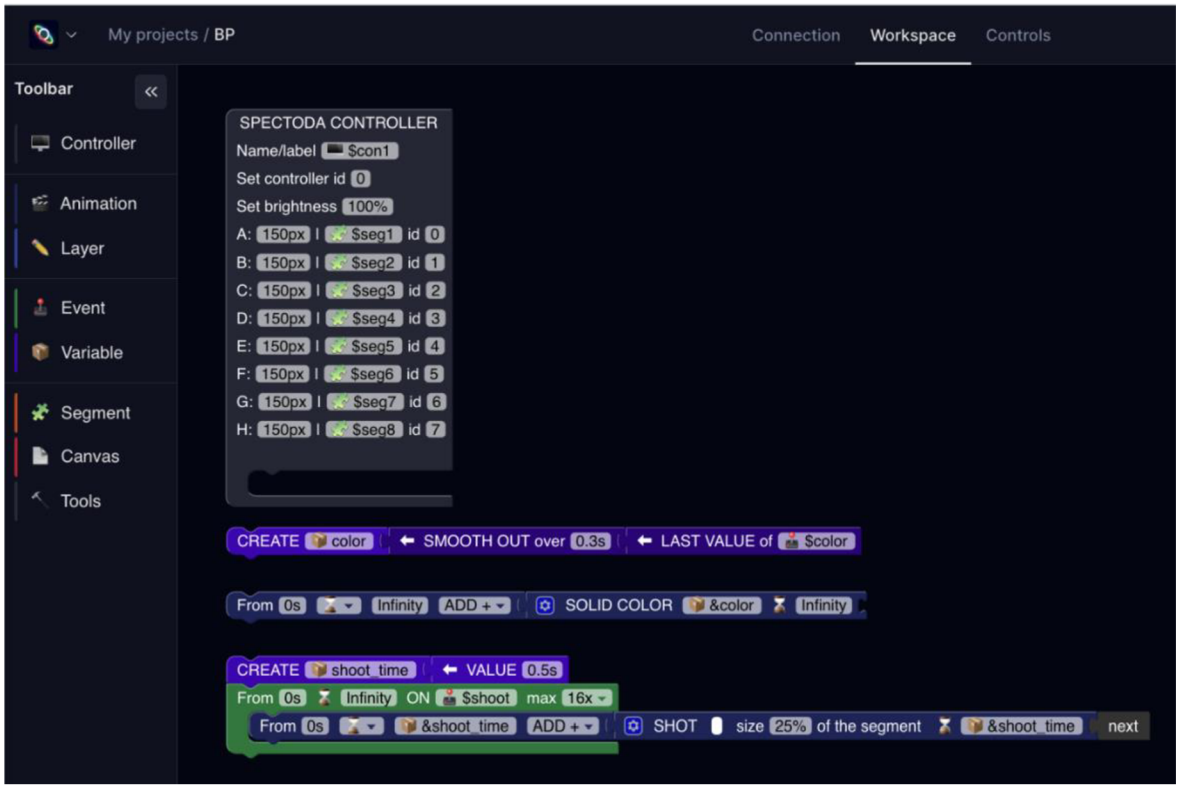
### 3.3.2 Spectoda

Spectoda (dříve Tangle) je komunikační protokol pro bezdrátové ovládání světelných produktů a instalací. Díky nim lze tvořit vlastní produkty poskytovat ovládání komerčním prostorům, bez nutnosti kabelů. Technologii vyvíjí firma Light Seekers s.r.o. sídlící v Česku.

Spectoda je vyvíjená na čipu ESP32 s nahraným bezdrátovým komunikačním protokolem Spectoda, nazývané jako Spectoda Controllers. K čipu může být připojený jeden nebo více LED pásků, či jiný typ světla a ten je zdrojem osvětlení. Zařízení tvoří tzv. mesh topologii, ta je vyvíjena nad komunikačním protokolem ESP-NOW (Jedná se o protokol vyvinutý společností Espressif, který umožňuje vzájemnou komunikaci více zařízení bez nutnosti Wi-Fi centrálního bodu. Protokol je podobný bezdrátovému připojení s nízkou spotřebou energie v pásmu 2,4 GH.). [11]

#### 3.3.2.1 Definice animací a světel

Spectoda Studio je vývojové prostředí, které umožňuje uživatelům deklarativně tvořit vlastní animace a definice světel pro světelné instalace. Toto prostředí je k dispozici online na webové adrese **studio.spectoda.com**. Umožňuje programovat projekty pro světla ve speciálním jazyce zvaném TNGL, který slouží k nastavení chování světel a definici API pro komunikaci s projektem. V Spectoda Studio je možné navrhovat a ovládat dynamické světelné instalace, což zahrnuje tvorbu proměnných a reakcí na události (eventy), které ovlivňují chování instalace. Tímto způsobem lze vytvářet složité a plynulé světelné efekty a animace, které reagují na externí podněty v reálném čase. [12] [13]




**Obrázek 4 - projekt ve Spectoda Studio - vlastní zpracování**

Tento kód v blocích se následně překládá do speciálního jazyka zvaného TNGL. Vygenerovaný kód následně vypadá takto:

```
defDevice($con1, 0x00, 0xff, 150px, $seg1, 0x00, 150px, $seg2, 0x01, 150px,
$seg3, 0x02, 150px, $seg4, 0x03, 150px, $seg5, 0x04, 150px, $seg6, 0x05, 150px,
$seg7, 0x06, 150px, $seg8, 0x07, {});

var color = genSmoothOut(genLastEventParam($color), 0.3s);
addDrawing(0s, Infinity, animFill(Infinity, &color));

var shoot_time = 0.5s;

interactive<0x10>(0s, Infinity, $shoot, {
  addDrawing(0s, 0.5s, animPlasmaShot(&shoot_time, #ffffff, 25%));
});
```

**Ukázka kódu 4 – vygenerovaný kód z bloků – vlastní zpracování**

V tomto příkladu je nadefinované zařízení pojmenované *con1*, tzn. definice pro controller pojmenovaný Controller 1. Máme zde 8 segmentů, pojmenované seg1-8 a přidělené ID jednotlivým portům.

Dále definujeme proměnnou color, který přijímá event *$color*, tu následně vykreslujeme v čase od 0 sekund do nekonečna a přidáváme v ní solidní barvu tzn. neustále svítící barvu.

Následně definujeme čas výstřelu 0.5 sekund jako proměnnou *shoot_time* a reakci na event *$shoot*, kterých může být maximálně 16 aktivních naráz. V reakci na *$shoot* definujeme výstřel od času 0 s, tzn. že začne ihned s obdržením eventem $shoot po dobu 0.5 s a dobu daného animačního prvku.

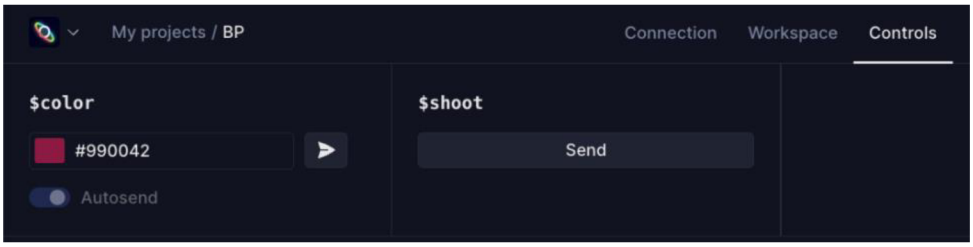
### 3.3.2.2 Ovládání animací a světel

Po vytvoření projektu ve Spectoda Studio můžeme otestovat jeho funkčnost a reakce na eventy pomocí nástroje "Spectoda Preview".

**Spectoda Preview** – náhled na virtuální LED pásky běžící na technologii Spectoda. Využívá WebAssembly pro spuštění vykreslovací části firmwaru Spectoda přímo ve webovém prohlížeči. WebAssembly umožňuje kompilaci nativního kódu z programovacích jazyků jako C/C++ a dalších do podoby spustitelné přímo ve webovém prostředí. Tento nástroj poskytuje tvůrcům možnost rychle otestovat a ladit své projekty pro světelné instalace.


**Obrázek 5 - Spectoda Preview – vlastní zpracování**

Náhled je černý, jelikož jsme animace nemá nadefinovanou počáteční hodnotu, můžeme jej ale nastavit zasláním Spectoda eventu $color, který jsme si nadefinovali výše. To lze provést přes Controls záložku ve Spectoda Studiu.



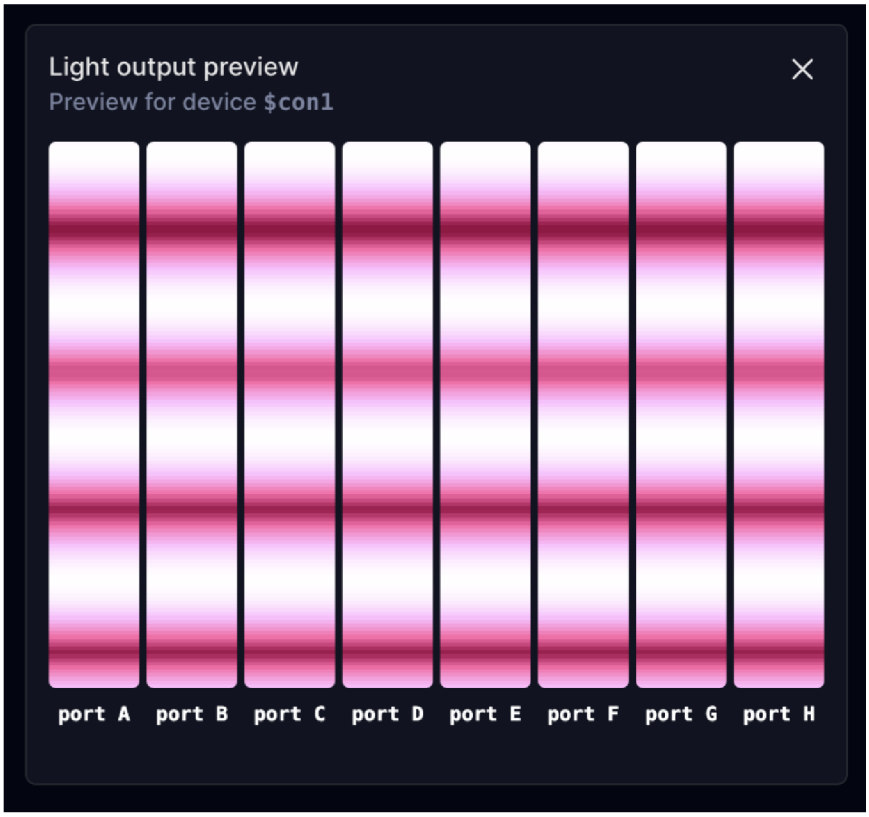

**Obrázek 6 - Controls tab Spectoda Studio – screenshot**

Spectoda eventy lze také zasílat z naší aplikace pomocí knihovny spectoda-js.

```
const barva = "#990042"

// periodické posílání eventu „shoot“ každou 1s
setInterval(() => {
  spectoda.emitEvent("shoot");
}, 1000);

spectoda.emitColorEvent("color", barva);
```

**Ukázka kódu 5 - posílaní eventů z aplikace – vlastní zpracování**

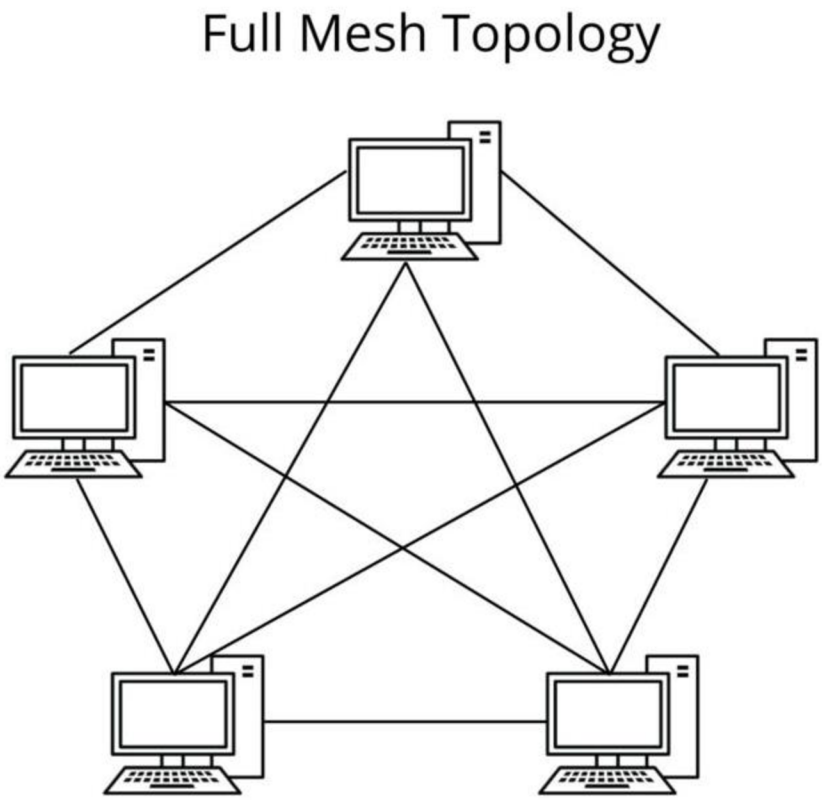
Následující kód můžeme následně vložit přímo do konzole Spectoda Studio po zkompilování a nahrání TNGL do náhledu a vznikla animace simulující výstřely linující ze spoda náhledu nahoru.




**Obrázek 7 - Ukázka pravidelných zobrazovaných výstřelů SpectodaPreview - vlastní zpracování**

### 3.3.2.3 Princip komunikace se Spectodou

Pro ovládání Spectoda instalací se používá aplikace SpectodaConnect, která komunikuje se Spectodou pomocí BLE (Bluetooth Low Energy). Aplikace je dostupná pro operační systémy IOS, Android a Mac OS. Pro speciální použití je dostupná knihovna pro JavaScript **spectoda-js**, která umožnuje připojení skrze technologii Web Bluetooth nebo Web Serial.




**Obrázek 8 - Mesh topologie – převzato [14]**

Připojit se stačí pouze k jednomu z controllerů a uživatel ovládá celou přiřazenou síť, není zde nutný most, který tvoří centrální jednotku a zároveň potenciální centrální bod selhání.

Pro inicializaci knihovny musíme inicializovat třídu SpectodaJS. Máme zde několik možných parametrů určující typ připojení. My zde použijeme typ „default“, který zvolí typ připojení, který podporuje náš prohlížeč.

```
const spectoda = new Spectoda("default");
```

Nyní se chceme připojit. Pokud chceme využívat co nejmenší latence, a proto využijeme připojení skrze „webserial“. A následným zavoláním funkce connect()

```
spectoda.assignConnect(“webserial”)
```

Na výběr máme webbluetooth, webserial, dummy a jeho alternativní verze nodebluetooth a nodeserial pro NodeJS, případně „flutter“, který reprezentuje implementaci pro mobilní aplikaci **SpectodaConnect**.
Ke komunikaci s hardwarem z vlastního softwaru je veřejně vyvíjená knihovna Spectoda-js vyvíjená právě touto společnosti.

```
import { Spectoda } from “lib/spectoda-js”

const spectoda = new Spectoda("webbluetooth", 0);

// key a signature jsou přihlašovací údajemi pro danou síť
const key = “00000000000000000000”;
const signature = “00000000000000000”;

// button reprezentuje abstraktni tlacitko značící “připojit se”
button.addEventListener(() => {
   spectoda.connect(null, true, signature, key);
})
```

**Ukázka kódu 6 - Připojení ke Spectodě skrze Web Bluetooth**

### 3.3.3 Portály a zdroje pro stahování MIDI souborů

Existuje mnoho portálů specializovaných na stahování MIDI souborů. V této práci jsou uvedeny některé z nich včetně ukázkových MIDI skladeb, které si můžete stáhnout a ihned vyzkoušet na přehrávači. Tyto skladby jsou získány ze služby BitMidi[1].

V České republice jsou populární portály svetkaraoke.cz, midistage.cz a spousta dalších, které rovněž nabízí MIDI soubory jako jeden z podporovaných formátů ke stažení. Skladby na těchto portálech jsou však placené a chráněné autorskými právy, což je důvod implementace oddělených účtů pro naši aplikaci, pozn. autora.

[1] bitmidi.com - populární platforma, umožňuje poslouchat a stahovat MIDI skladby zdarma. Nabízí také open source implementaci syntetizátoru a disponuje archivem přes 100 000 skladeb.

### 3.3.4 Programy pro práci s MIDI

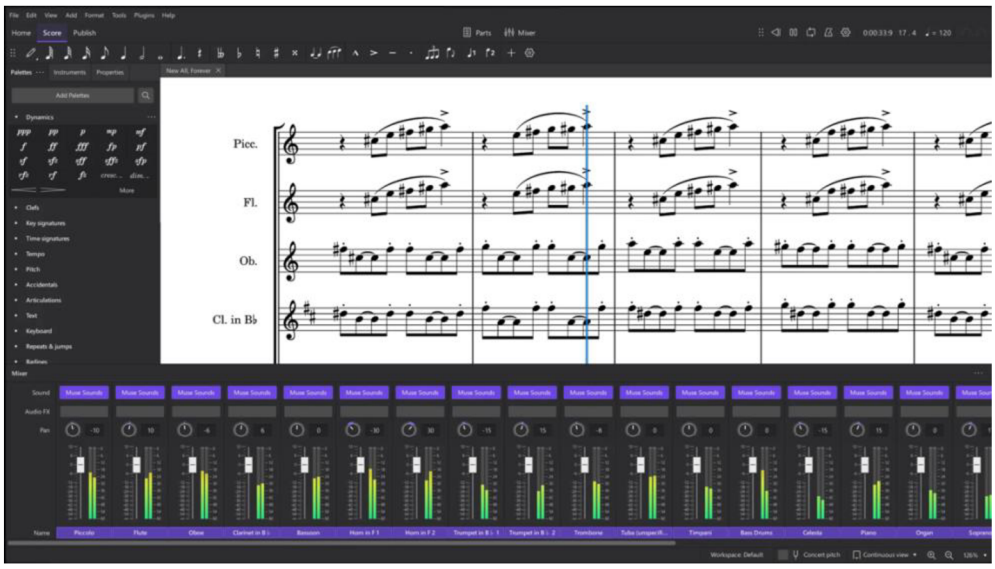
Pro práci s MIDI existují různé editory, ale také audio editory, takzvané DAW, které MIDI standard buď rozšiřují, či umožnují přidávání vlastních zvukových stop, které za pomocí MIDI nelze dosáhnout (např lidský hlas).

Zde je seznam několika populárních nástrojů podporující práci s MIDI:

- **Ableton Live:** Oblíbené DAW (Digital Audio Workstation) s širokým využitím pro produkci, editaci, mix a mastering na vysoké úrovni
- **Garage Band:** Pro uživatele Apple zařízení je populární volbou díky své jednoduchosti a široké škále virtuálních nástrojů.
- **Avid Pro Tools:** je vynikajícím nástrojem pro práci s MIDI, vhodným pro profesionální hudebníky i začínající producenty. Poskytuje širokou škálu funkcí pro tvorbu, úpravu a přehrávání MIDI hudby. [16]

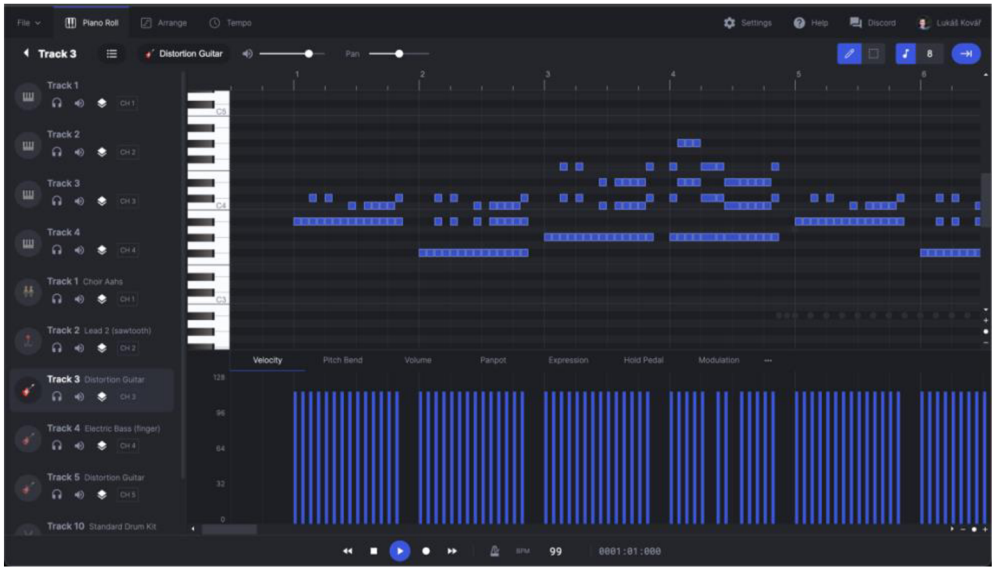
### 3.3.5 Open Source nástroje pro práci s MIDI

V této části budou představeny 2 editory s otevřeným zdrojovým kódem, kterým se lze inspirovat.




**Obrázek 9 - MuseScore prostředí programu – převzato [17]**

**MuseScore:** je oblíbeným nástrojem pro hudebníky, školy, konzervatoře a orchestry díky své jednoduchosti použití a široké škále funkcí. Pracuje na bázi notového zápisu a nabízí možnost importovat a exportovat MIDI soubory a podporuje různé další formáty zápisů. Kromě toho existují mobilní aplikace pro prohlížení a přehrávání notových zápisů. [17]




**Obrázek 10 - Prostředí programu Signal – vlastní zpracování**

**Signal:** je uživatelsky přívětivý online MIDI editor, který umožňuje uživatelům vytvářet, upravovat a přehrávat MIDI hudbu přímo v prohlížeči. Tato aplikace běží ve webových prohlížečích jako Google Chrome, Firefox či Safari a v omezené podobě i na mobilních telefonech s Androidem a IOS. Je plně open-source s MIT licencí. Vyvíjí ji Japonec Ryohei Kameyama (dále označován pod přezdívkou ryohey). [18]

## 3.4 Použité technologie

V této části bakalářské práce je představena sada technologií, které byly vybrány pro vývoj webové aplikace. Tyto technologie jsou klíčové pro různé aspekty projektu, včetně front-end a back-end vývoje, stylování, validace dat a manipulace s daty. Každá z těchto technologií byla zvolena na základě její schopnosti zvýšit efektivitu vývoje, udržitelnost kódu a zlepšit uživatelskou zkušenost.

### 3.4.1 Bun

Bun je rychlý, vše-v-jednom JavaScript runtime, který obsahuje vestavěné nástroje jako bundler, transpiler, task runner a lze ho používat i náhradu za balíčkovací systém npm. Byl navržen s cílem nabídnout vývojářům snadnější a efektivnější způsob práce s JavaScriptem a TypeScriptem. Je založen na moderních technologiích a optimalizacích, které mu umožňují dosahovat vynikajících výkonů při zpracování JavaScriptového kódu. [19]

Díky integraci různých nástrojů do jediného runtime umožňuje Bun vývojářům rychlejší a pohodlnější vývoj aplikací bez nutnosti konfigurace a instalace mnoha externích závislostí. A můžeme se tím vyhnout jindy nutným transpilátorů jako je např. Babel a TypeScript kompilátor.

### 3.4.2 React

React je knihovna pro tvorbu uživatelských rozhraní za pomocí komponent, pro tvorbu Single Page Aplikací tzv. SPA, kde se aplikace načte jen jednou a poté reaguje bez pře načítání. Pro React existuje od komunity spoustu knihoven implementující se přímo s ním, jako je např. React Router. [20]

### 3.4.3 NEXT.js

Next.js rozšiřuje možnosti Reactu o automatické rozdělení kódu pro rychlejší načítání stránek, integrované nástroje pro routování bez nutnosti dalších knihoven a podporu API routes, což umožňuje snadné vytváření API endpointů.

### 3.4.4 Zustand

Zustand je malá a rychlá knihovna pro správu stavu aplikací v Reactu. Díky svému jednoduchému a efektivnímu API, které je ve srovnání například s populárním Reduxem mnohem přímější, se Zustand stává vhodnou volbou pro efektivní správu stavu ve webových aplikacích. V tomto projektu je Zustand použit k ukládání a správě informací o přehrávaném MIDI souboru a aktivních hudebních nástrojích

### 3.4.5 PostgreSQL

PostgreSQL (dále v práci označován zkratkou PG), je open-source relační databázový systém, který se používá pro ukládání a správu dat. Jedná se o jednu z nejvíce pokročilých a rozšířených relačních databází, která je známa pro svou spolehlivost, výkonnost a širokou škálu funkcí. [26]

### 3.4.6 Create T3 App

Konzolový nástroj umožňující generovat šablony kódu pro typově bezpečné Next.js aplikace s připravenou implementací Next.js, tRPC, Drizzle a Tailwind CSS, Zod a SuperJSON. Nejedná o framework, ale čistě generátor startovací šablony projektu. Přináší nám vysoce efektivní strukturu připravenou na to abychom mohli co nejrychleji a udržitelně začít vyvíjet naši aplikaci.

#### 3.4.6.1 Drizzle

Drizzle je ORM (Object-Relational Mapping) knihovna pro TypeScript, která slouží k usnadnění práce s databázemi v aplikacích. Hlavními výhodami Drizzle jsou jeho jednoduchost, výkon a kvalitní typová kontrola díky využití TypeScriptu, která se přenáší dále skrze tRPC. [27]

#### 3.4.6.2 TRPC

Použití knihovny tRPC (označovaného také jako TypeScript RPC) přináší několik výhod oproti přímému volání REST API endpointů:

1. Typová bezpečnost: tRPC zajišťuje typovou kontrolu mezi frontendem a backendem, což snižuje riziko chyb a usnadňuje vývoj.
2. Automatické generování klienta: tRPC automaticky generuje klienta na základě definovaných procedur, což zjednodušuje volání API na frontendové části aplikace.
3. Jednodušší údržba: Díky centralizovanému definování procedur v tRPC routeru je údržba a úpravy API snazší a přehlednější.

V případě sdíleného kódu mezi frontendem a backendem nám používání tRPC nám hlídá validnost vstupních i výstupních dat hlídá přímo TypeScript na svojí nativní úrovni.

#### 3.4.6.3 Zod

Zod je knihovna určená pro validaci a parsování dat v TypeScriptu a JavaScriptu. Umožňuje vytvářet schémata pro kontrolu dat, která lze efektivně využívat jak na straně klienta, tak na straně serveru, díky unifikaci použitého programovacího jazyka. API, které Zod poskytuje, je navrženo pro intuitivní definici schémat s využitím konstruktorů Zod pro různé datové typy a jejich omezení. Tyto definice umožňují následně validovat a analyzovat příchozí data. V případě, že data nesplňují požadavky schématu, Zod generuje chybové hlášení specifikující jádro problému. [22]

#### 3.4.6.4 SuperJSON

SuperJSON je nástroj určený pro bezproblémovou serializaci a deserializaci složitějších datových struktur, které standardní JSON formát nedokáže přímo zpracovat. Toto zahrnuje datové typy jako jsou Date (datum), Map, Set, nebo dokonce i Error objekty, které běžný JSON formát neumí správně serializovat nebo deserializovat bez ztráty informací nebo struktury. [23]

SuperJSON tak rozšiřuje možnosti standardního JSONu tím, že umožňuje serializaci těchto složitějších typů, čímž usnadňuje přenos komplexních dat mezi různými částmi aplikace, například mezi serverem a klientem ve webové aplikaci. Díky SuperJSON můžeme snadno ukládat a přenášet data mezi backendem a frontendem bez nutnosti manuální transformace nebo ztráty datových typů.

#### 3.4.6.5 Tailwind CSS

Tailwind CSS je progresivní utility-first CSS framework, který se zaměřuje na využití atomických CSS tříd pro tvorbu komponent a stylování webových stránek. Místo předpřipravených komponent nabízí nízko úrovňové utility třídy pro jednotlivé CSS vlastnosti, což umožňuje vytvářet jakýkoliv design. [21]

# 4 Návrh a implementace aplikace

Tato kapitola se zabývá návrhem a implementací webové aplikace pro přehrávání a vizualizaci MIDI souborů. Jsou zde popsány části tvorby frontendové a následně backendové části.

## *4.1 Tvorba frontendového rozhraní*

Pro implementaci si tato práce NEXT.js, TypeScript, TailwindCSS s rozšířením Daisy UI na stylování a Shadcn pro speciální komponenty typu notifikační toast nebo modál pro náhled.

### 4.1.1 Implementace jednoduchého MIDI přehrávače

Pro otestování validity MIDI předtím, než se přesuneme do implementace vlastního přehrávače můžeme využít existujícího řešení. Knihovnu přímo pro umožňující implementaci ve stylu React komponenty se najít nepodařilo. Existuje ale projekt ***html-midi-player***, který umožnuje jednoduše MIDI soubory spouštět přímo v prohlížeči s integrovaným syntetizérem @magenta/music, který vzniknul jako laboratorní projekt v Google AI divizi. [24]

Použití html-midi-player v samotném JavaScriptu je velice jednoduché. Stačí naimportovat potřebné skripty a vložit webkomponentu do HTML kódu.

```
<!DOCTYPE html>
<html>
<head>
  <title>Ukázka html-midi-player</title>
  <!-- Import potřebných skriptů -->
  <script
src="https://cdn.jsdelivr.net/combine/npm/tone@14.7.58,npm/@magenta/music@1.23.
1/es6/core.js,npm/focus-visible@5,npm/html-midi-player@1.4.0"></script>
</head>
<body>
  <!-- Vložení html-midi-player komponenty -->
  <midi-player
    src="cesta/k/souboru.mid"
    sound-font
    visualizer="#myVisualizer">
  </midi-player>

  <!-- Volitelná vizualizace (piano-roll) -->
  <midi-visualizer type="piano-roll" id="myVisualizer"></midi-visualizer>

  <script>
    // Případná další inicializace či ovládání komponenty
  </script>
</body>
</html>
```

**Ukázka kódu 7 – Implementace webkomponenty html-midi-player – vlastní zpracování**


**Obrázek 11 – Ukázka vizualizace MIDI nástroje – vlastní zpracování**

S touto implementací bychom se mohli spokojit a stavět dále na ní, ale vzhledem k tomu že MIDI chceme přímo analyzovat postavíme přehrávač vlastní na udržovaných knihovnách u kterých bude jednoduší tvorba rozšíření pro vizualizační účely.

### 4.1.2 Klíčové knihovny tvořící přehrávač MIDI

Pro stavbu přehrávače použijeme kombinaci několika knihoven a vlastní logiky, která tyto komponenty využívá pro vyčítaní dat z MIDI, zpracování, časování MIDI eventů a následné přehrávání skrze MIDI syntetizér a alternativně WebMIDI Api

- **midifile-ts** – určený pro parsování a manipulaci s MIDI soubory. Umožňuje načítat MIDI soubory do formátu JSON, procházet jejich strukturu, přistupovat k jednotlivým MIDI eventům a provádět různé operace s MIDI daty. Tato knihovna je klíčová pro načtení a interpretaci MIDI souborů v naší aplikaci.
- **wavelet** – v našem přehrávači použijeme wavelet pro generování zvuku na základě MIDI dat pomocí na základně soundfontů namísto nutnosti mít MIDI syntetizér instalovaný v počítači.
- **sf2parser** – slouží pro parsování a načítání Soundfontů (SF2) souborů a integruje se přímo s knihovnou wavelet.

Ačkoliv existuje mnoho open-source knihoven pro práci s MIDI v JavaScriptu, kvalitních a aktivně udržovaných knihoven v TypeScriptu není mnoho. Knihovny výše zmíněných knihoven jsou primárně vyvíjena a udržována vývojářem s přezdívkou Ryohey, který je také autorem populárního open-source webového MIDI editoru Signal.

## 4.2 Tvorba přehrávače MIDI

Pro přehrávání MIDI souborů v prohlížeči potřebujeme přehrávač, který ve webových prohlížečích není. Budeme jej implementovat tedy sami s pomocí knihoven.

### 4.2.1 Parsování MIDI

Pro parsování MIDI použijeme knihovnu *midifile-ts.*

V jeho TypeScript definičním souboru můžeme nalézt metodu read(), která přijímá datový typ StreamSource a vrací typ MidiFile což je rozparsovaný formát MIDI převedený do JSON struktury.

```
type StreamSource = DataView | number[] | ArrayBuffer | Uint8Array;
// MidiFile je JSON struktura, obsahující hlavičku a eventy jednotlivých zvukových stop
declare function read(data: StreamSource): MidiFile;
```

**Ukázka kódu 8 – knihovna midifile-ts, typy podporovaných vstupních dat – převzato**

Nad funkcí read() vytvoříme vlastní statickou třídu MidiParser, která kromě eventů, poskytne agregovaný seznam nástrojů, titulků a metadat obsažených ve skladbě.

```
// přidání indexů k jednotlivým eventům pro jednoduchou adresaci v aplikaci
const addIndexToEvents = (midi: MidiFile) => {
  midi.tracks.forEach((track, trackIndex) => {
    track.forEach((event: AnyEvent & WithIndex, eventIndex) => {
      // Přidání indexů k jednotlivým událostem v MIDI souboru
      event.index = eventIndex;
      event.trackIndex = trackIndex;

      // Kontrola, zda se jedná o meta událost typu "lyrics"
      if (event.type === "meta" && event.subtype === "lyrics") {
        const text = event.text.replace("\r", "\n");
        lyrics.push({ text, index: eventIndex, trackIndex });
      }
      if (event.type === "channel" && event.subtype === "programChange") {
        instruments.push(event);
      }
    });
  });
};
```

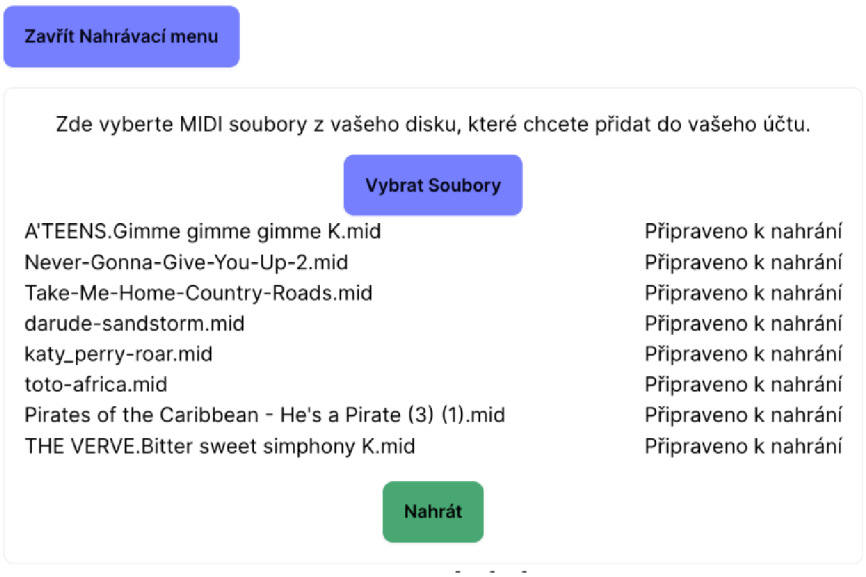
```
export class MidiParser {
  static parse(midiData: Uint8Array) {
    try {
      // Inicializace polí pro uložení různých typů dat z MIDI souboru
      const lyrics: LyricsWithIndex[] = [];
      const introText: string[] = [];
      const instruments: InstrumentWithIndex[] = [];
      let endEvent = null;

      // Parsování MIDI souboru pomocí funkce read z knihovny midifile-ts
      const parsedMidi = read(midiData);
      addIndexToEvents(parsedMidi!);

// Vytvoření objektu s rozparsovanými daty z MIDI souboru pro další části aplikace
      const data = {
        introText: introText,
        lyrics: lyrics,
        endEvent,
        instruments: instruments,
        data: parsedMidi,
        totalTime: calculateTotalTime(parsedMidi),
      };

      return data;
    } catch (err) {
      console.log(err);
    }
  }
 }
```

**Ukázka kódu 9 - funkce pro parsování MIDI s tvorbou vlastních metadat- vlastní zpracování**

Třída MidiParser obsahuje statickou metodu parse, která přijímá MIDI data ve formátu Uint8Array. Uvnitř této metody se nejprve inicializují pole pro ukládání různých typů dat z MIDI souboru, jako jsou lyrics (texty), introText (úvodní text), instruments (nástroje) a endEvent (událost konce skladby). Tyto informace využijí další komponenty aplikace.

### 4.2.2 Serializace a ukládání MIDI skladby do databáze

Pro nahrávání MIDI souborů na server a jejich ukládání do databáze je vytvořeno frontendové rozhraní s možností výběru souborů. Po výběru souborů jsou data serializována do formátu Base64 a odeslána na backend pomocí mutace *trpc.midi.upload*.




**Obrázek 12 – ukázka nahrávacího rozhraní MIDI skladeb do DB – vlastní zpracování**

Kliknutí na tlačítko *Vybrat Soubory,* zobrazí okno s výběrem souborů, těch umožnuje aplikace vybrat několik naráz. Po vybrání se uloží do pole a po kliknutí na tlačítko *Nahrát,* se stane samotné parsování MIDI a jeho upload do databáze i s jeho metadaty.

```
const handleUpload = async () => {
  for await (const file of files) {
    const reader = new FileReader();
    reader.onload = async () => {
        const arrayBuffer = reader.result as ArrayBuffer;
        const base64 = btoa(new Uint8Array(arrayBuffer).reduce((data, byte) =>
data + String.fromCharCode(byte), ""));
        await upload();

        async function upload() {
            const result = await uploadMidi.mutateAsync({
              name: file.name,
              description: headerText,
              midi: base64,
              text: plainLyrics,
            });
        }
      }
```

```
    };
    reader.readAsArrayBuffer(file);
  }
};
```

**Ukázka kódu 10 – Implementace frontend MIDI upload funkce do DB – vlastní zpracování**

### 4.2.3 Rozhraní pro výběr přehrávaných nástrojů

General MIDI má standard 128 nástrojů, každý tento nástroj má svoje ID, pro implementaci názvů byla přejata implementace ze zdrojového kódu z open-source projektu Signal a následně proveden překlad do češtiny pomocí překladače s umělou inteligencí Deepl.

```
export const InstrumentName: FC<{ programNumber: number | undefined }> = ({
  programNumber,
}) => {
  switch (programNumber) {
    case 0:
      return (
        <Localized default="Acoustic Grand Piano">
          Akustický klavír
        </Localized>
      );
    case 1:
      return (
        <Localized default="Bright Acoustic Piano">
          Jasný akustický klavír
        </Localized>
      );
…
    case 64:
      return <Localized default="Soprano Sax">Sopránový saxofon</Localized>;
…
    case 127:
      return <Localized default="Gunshot">Výstřel</Localized>;
  }
  return <></>;
};
```

**Ukázka kódu 11 – knihovna se jmény nástrojů – převzato, upraveno [25]**

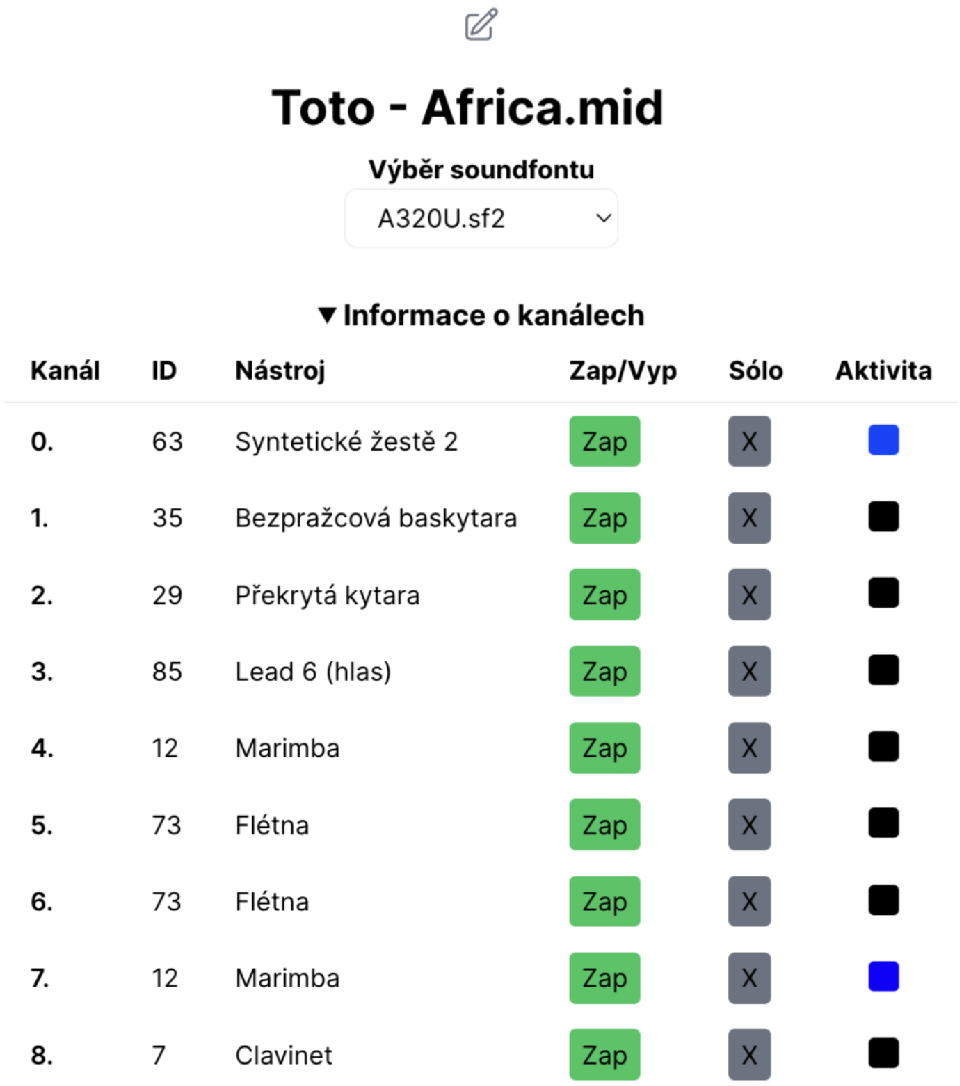
Komponenta *InstrumentName* převádí identifikátory nástrojů na textovou podobu, jsou zaobaleny v *Localized* tagu, aby byl možný zobrazit anglický název nástrojů.

```
function ChannelList({ channelStates, setChannels, isAnySoloChannelEnabledValue
}: ChannelProps) {
  return (
    <div className="text-left dark:text-white">
      <table className="w-full border-collapse">
        {/* ... */}
        <tbody>
          {Array.from(channelStates).sort((a,b)=> {
            return a[0] - b[0];
          }).map(([channel_id, channel], index) => (
            <tr key={index} className={channel.enabled ? "" : "bg-gray-800
text-gray-300"}>
              {/* ... */}
              <td className="flex items-center justify-center">
                <InstrumentPulser channelId={channel.channel}
enabled={isAnySoloChannelEnabledValue ? channel.solo : channel.enabled} />
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
}
```

**Ukázka kódu 12 - Zobrazení seznamu stavu a přepínání nástrojů - vlastní zpracování**

Komponenta *ChannelList* renderuje tabulku s informacemi o jednotlivých kanálech, včetně čísla kanálu, ID nástroje, názvu nástroje (pomocí komponenty *InstrumentName*), tlačítka pro zapnutí/vypnutí kanálu a tlačítka pro sólový režim. Kanály jsou seřazeny podle čísla kanálu.

Speciální vnořenou komponentou je následně *InstrumentPulser*, který barevně indikuje kdy hraje daný nástroj.

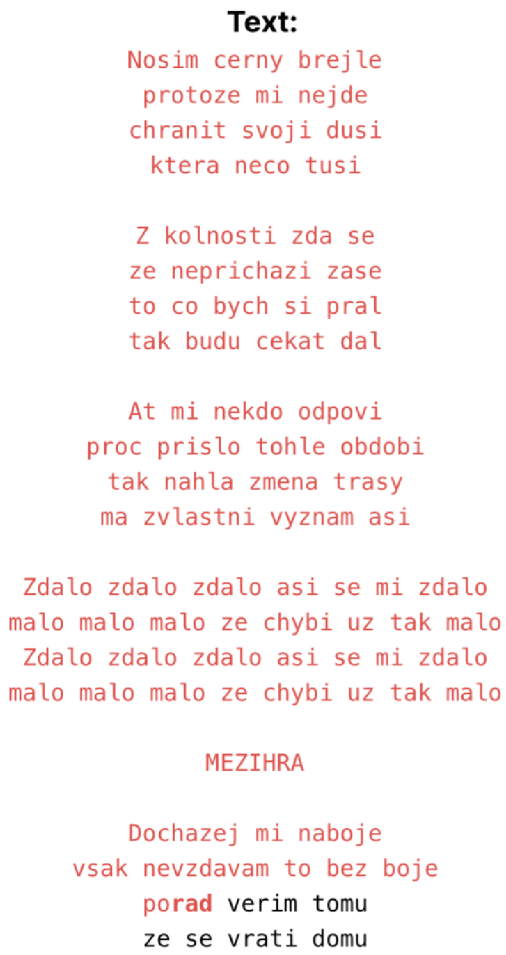



**Obrázek 13 – Ukázka seznamu kanálů s přiřazenými nástroji k dané skladbě – vlastní zpracování**

Na obrázku můžeme vidět jaké nástroje MIDI jsou v daný moment přehrávány v políčku (Aktivita) a můžeme jej dále ovlivnit přepnutím skladby do sólo režimu (kdy jsou upozaděny ostatní nástroje), či jednoduchým vypnutím kliknutím na tlačítko Zap (jako Zapnuto).

### 4.2.4 Implementace zobrazování titulků

V této části bude popsána implementace titulků, která se integruje s přehrávačem a označuje text aktuální části přehrávané skladby.




**Obrázek 14 - Ukázka zobrazení titulků přehrávané skladby – vlastní zpracování**

V obrázku můžeme vidět, že červený text je již přehrání a část, kde je kurzor je označen tučně. Níže můžeme vidět přímo implementaci v kódu:

```
interface LyricsDisplayProps {
  lyrics: LyricsWithIndex[];
  midiIndex: number;
}

export function LyricsDisplayKaraoke({ lyrics, midiIndex }: LyricsDisplayProps)
{
  return (
    <>
      <h2 className="mt-5 text-xl font-bold">Text:</h2>
      <pre className="whitespace-pre-wrap">
        {lyrics.map((lyric, index) => (
```

```
            <span
              key={index}
              className={cn(lyric.index <= midiIndex && 'text-red-500',
lyric.index === midiIndex && 'font-bold')}
            >
              {lyric.text}
            </span>
          ))}
        </pre>
      </>
    );
}
```

**Ukázká kódu 13 - Komponenta LyricsDisplayKaraoke pro zobrazení synchronizovaného textu se skladbou – vlastní zpracování**

LyricsDisplayKaraoke komponenta má za úkol zobrazit text písně (lyrics) synchronizovaně s přehráváním MIDI souboru. Hlavní funkce této komponenty je zvýraznit aktuálně zpívanou část textu v závislosti na pozici v MIDI souboru.

Při vykreslování komponenta LyricsDisplayKaraoke prochází pole lyrics a pro každou část textu vytváří  element. Pokud je index dané části textu menší nebo roven midiIndex, aplikuje se na  TailwindCSS třída text-red-500, která zvýrazní aktuálně zpívanou část.

K tomu, aby karaoke komponenta věděla, kde je kurzor je třeba nadefinovat wrapper komponentu, v tomto případě HOC (higher order component), skrze kterou poskytujeme data dále. Naslouchám zde na event "midi-index-change", který reaguje na posílané eventy z přehrávače.

```
function LyricsIndexProvider({ children }) {
  const [midiIndex, setMidiIndex] = useState(0);

  useEffect(() => {
    return nanoevents.on("midi-index-change", setMidiIndex);
  }, []);

  return children(midiIndex);
}
```

**Ukázka kódu 14 – LyricsIndexProvider jako prostředník předávání stavu přehrávače – vlastní zpracování**

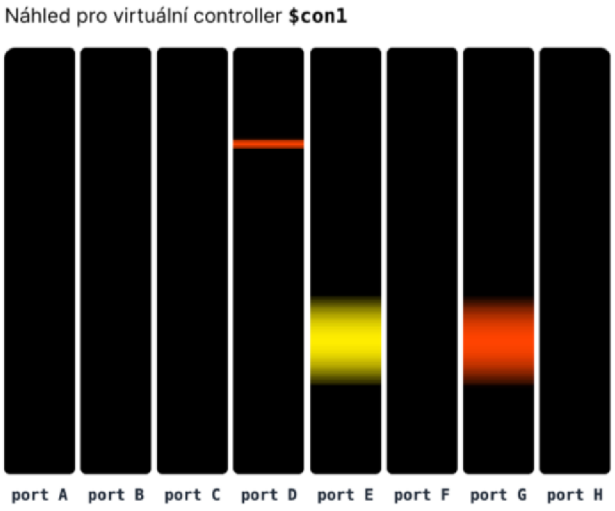
Komponenta LyricsIndexProvider funguje jako prostředník mezi přehráváním MIDI souboru a komponentou LyricsDisplayKaraoke. Jejím hlavním úkolem je získávat aktuální index (který přidělujeme v MidiParseru) k MIDI události (midiIndex) a předávat jej do vnořené komponenty, kde jednotlivé části textu interně řeší, zda má být zvýrazněna či nikoliv.

## 4.3 Tvorba algoritmu na zobrazování MIDI příkazů na světlech

V této kapitole se zaměříme na vývoj algoritmu, který dokáže převádět MIDI příkazy na vizuální efekty pomocí světel. Cílem je vytvořit interaktivní audiovizuální zážitek, kde jsou hudební události v reálném čase transformovány do světelných efektů. K dosažení tohoto cíle využijeme platformu Spectoda, která poskytuje rozhraní pro komunikaci mezi MIDI zařízeními a světelným hardwarem.

### 4.3.1 Implementace Spectoda Preview

Pro efektivní testování a ladění algoritmu převodu MIDI na světelné efekty byl implementován nástroj Spectoda Preview, který zobrazuje výsledky v reálném čase.
Tato komponenta je realizována jako iframe element, který zobrazuje virtuální controller pro náhled světelných efektů.




**Obrázek 15 – Ukázka implementace SpectodaPreview v projektu – vlastní zpracování**

Komunikace mezi hlavní aplikací a iframe probíhá prostřednictvím zasílání zpráv pomocí metody postMessage. Hlavní aplikace odesílá do iframe příkazy pro vykonání kódu (execute_bytecode), požadavky na bytecode (request_bytecode) a časové značky (clock_timestamp).

```
function PreviewButton() {
  const iframeRef = useRef<HTMLIFrameElement>(null);

  function emitToIframe(iframe: HTMLIFrameElement | null, data: any) {
    if (iframe && iframe.contentWindow) {
      iframe.contentWindow.postMessage(JSON.stringify(data), "*");
    }
  }

  function iframeOnloadHandler() {
      spectoda.on("wasm_execute", (command: Uint8Array) => {
        emitToIframe(currentIframe, {
          execute_bytecode: uint8ArrayToHexString(command),
        });
      });
  }

  return (
   <div className="fixed bottom-4 right-4 z-50 flex items-center space-x-2">
      <Dialog open={isOpen} modal={false}>
        <DialogContent>
          <SpectodaVisualization />
          <iframe onLoad={iframeOnloadHandler} ref={iframeRef} id="wasm-iframe"
height="330px" width="428px" className="overflow-auto rounded-lg"
src={"https://wasm.spectoda.com/main"} />
        </DialogContent>
      </Dialog>
      <button className={cn("btn  btn-sm", isOpen ? "btn-error" : "btn-
primary")} onClick={() => setIsOpen(!isOpen)}>
        {isOpen ? "Zavřít" : "Otevřít"} náhled
      </button>
    </ div>
  );
}
```

**Ukázka kódu 15 – Implementace SpectodaPreview, zjednodušeno – vlastní zpracování**

Komponenta je pojmenována jako PreviewButton. Kromě samotného tlačítka pro otevření/zavření náhledu obsahuje také logiku pro zobrazení dialogového okna s iframe elementem a vnořené komponenty.

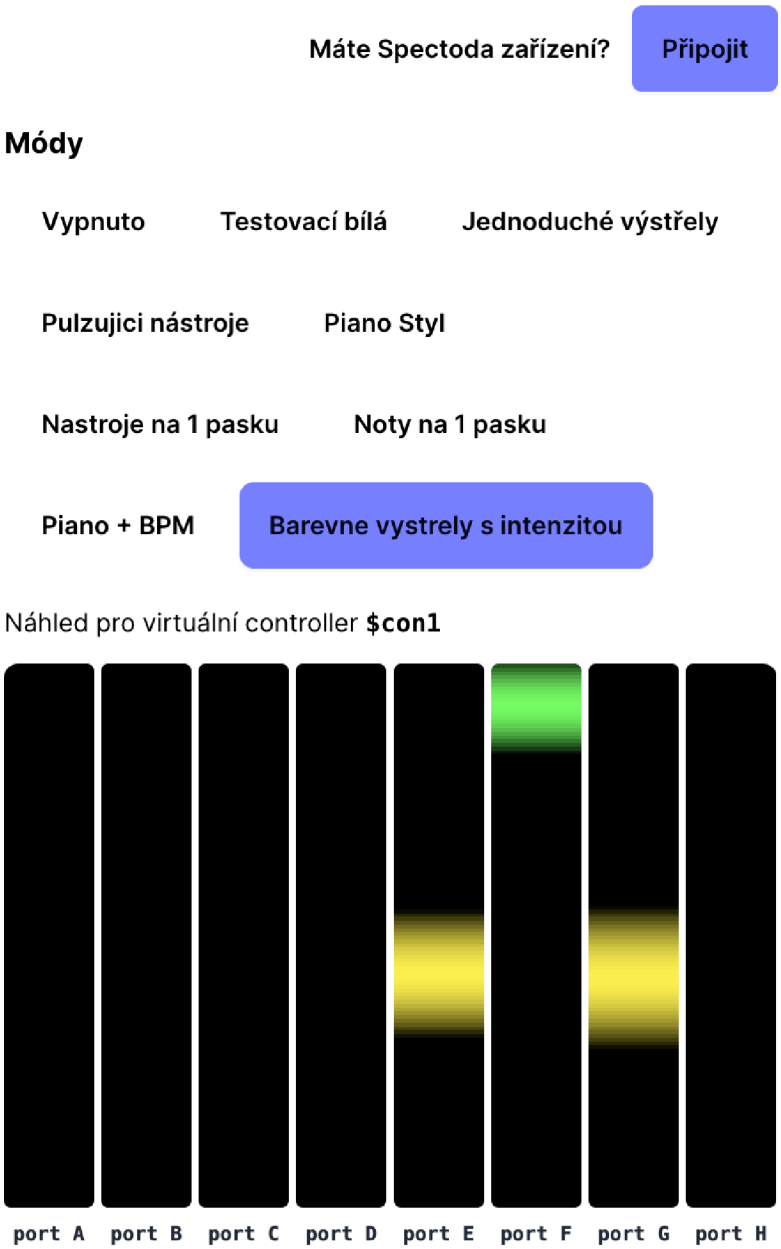
### 4.3.2 Přepínání módů a vizualizace eventů

K přepínání vizualizačních režimů byla implementována vlastní komponenta SpectodaVisualization. Ta implementuje tlačítka, metody a listenery, které posílají zprávy měnící nastavení světel. TNGL kód a nastavení všech vizualizací najdete ve zdrojovém kódu v příloze v souboru UploadTnglButtons.tsx.

```
const tnglList = [
  {
    name: "Vypnuto",
    code: ``,
    listener: () => {
      return () => {};
    },
    extraFunction: () => {},
  },
  {
    name: "Jednoduché výstřely",
    code: `
    // …
    var shoot_time = 1s;
    interactive<0x10>(0s, Infinity, $shoot, {
      addDrawing(0s, &shoot_time, animPlasmaShot(&shoot_time, #ffffff, 5%));
    });
  `,
    listener: () => {
      // Zde naslouchame na midi eventy a dle hrajícího kanálu posíláme výstřely
      return nanoevents.on("midichannelevent", event => {
        if (event.type === "channel" && event.subtype === "noteOn" &&
event.enabled) {
          spectoda.emitPercentageEvent("shoot", (event.velocity / 127) * 20,
event.channel);
        }
      });
    },
  },
  // …
];
```

**Ukázka kódu 16 - Ukázka implementace Spectoda Preview v projektu - vlastní zpracování**

Implementovaný seznam tnglList, lze dále rozšiřovat o další módy a na jeho základě se vygenerují v seznamu módů tlačítka. Kompletní implementaci náhledu můžeme vidět na obrázků níže.




**Obrázek 16 – grafické rozhraní přepínání vizualizace – vlastní zpracování**

Vytvořený algoritmus a nástroje představují základ pro tvorbu poutavých audiovizuálních vystoupení, kde se propojuje hudba se světelnými efekty v reálném čase. Díky modulární struktuře a možnosti rozšiřování lze systém dále přizpůsobovat specifickým potřebám a experimentovat s novými vizuálními koncepty.

## 4.4 Back-endová část

V této části bude popsána implementace serverové logiky, komunikace s databází, přihlášení a následně popsán způsob hostování aplikace.

### 4.4.1 Ukládání a načítání MIDI do databáze

Ukládání MIDI skladeb lze provádět několika způsoby, ať už využitím externích úložišť typu S3, tak způsoby ukládání binárních dat do databáze.

```
export const midis = createTable("midis", {
  id: serial("id").primaryKey().notNull(),
  name: text("name").notNull(),
  description: text("description"),
  midi: text("midi"),
  text: text("text"),
  userId: varchar("userId", { length: 255 }).references(() => users.id, {
    onDelete: "cascade",
  }),
});
```

**Ukázka kódu 17 – MIDI skladby – databázové schéma v DrizzleORM – vlastní zpracovaní**

Toto schéma uchovává informace o MIDI uloženém v databázi, popsané skrze DrizzleORM. Vzhledem k tomu že průměrný MIDI soubor má velikost v řádu desítek kilobajtů, použijeme zakódování do textu v Base64 kódování.

### 4.4.2 Tvorba fulltext vyhledávací funkce v PostgreSQL

Pro vyhledávání si vytvoříme vlastní PostgreSQL funkci emulující fulltext fuzzy search (tzv. vyhledávaní s podporou drobných překlepů). K tomu je třeba nainstalovat několik rozšíření.

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE EXTENSION IF NOT EXISTS fuzzystrmatch;
CREATE EXTENSION IF NOT EXISTS unaccent;
```

**Ukázka kódu 18 – Aktivace PostgreSQL rozšíření fuzzy search a odstraňování diakritiky – vlastní zpracování**

Funkce využívá několik rozšíření PostgreSQL, která je potřeba před použitím aktivovat, každé z těchto rozšíření má v naší funkci svůj účel:

- **pg_trgm:** poskytuje podporu pro trigram matching, který se používá pro přibližné řetězcové porovnávání.

- **fuzzystrmatch:** přidává funkce pro fuzzy řetězcové porovnávání, jako je například metoda similarity [28]
- **unaccent:** Toto rozšíření odstraňuje diakritiku z textových řetězců.

Díky tomu může uživatel vyhledat skladbu i když zdá pouze například větu ze skladby (políčka name, description a text).

```
CREATE OR REPLACE FUNCTION fuzzy_search_midi(search_string TEXT)
RETURNS TABLE (
    id INT,
    name TEXT,
    description TEXT,
    text TEXT
    userId INT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT m.id, m.name, m.description, m.text, m."userId"
    FROM "midis" AS m
    WHERE (
        unaccent(m.name) ILIKE unaccent('%' || search_string || '%')
        OR similarity(unaccent(m.name), unaccent(search_string)) > 0.3
        OR unaccent(m.description) ILIKE unaccent('%' || search_string || '%')
        OR similarity(unaccent(m.description), unaccent(search_string)) > 0.3
        OR unaccent(m.text) ILIKE unaccent('%' || search_string || '%')
    )
    ORDER BY
        similarity(unaccent(m.name), unaccent(search_string)) DESC,
        similarity(unaccent(m.description), unaccent(search_string)) DESC,
        similarity(unaccent(m.text), unaccent(search_string)) DESC;
END;
$$ LANGUAGE plpgsql;
```

**Ukázka kódu 19 - Tvorba PostgreSQL funkce pro fulltextové vyhledávání MIDI – vlastní zpracování**

Při zavolání funkce **fuzzy_search_midi**, se děje několik kroků:

1. Hledání řetězců, které obsahují hledaný výraz (ILIKE '%hledany_vyraz%') v názvu, popisu nebo textu skladby po odstranění diakritiky pomocí unaccent().
2. Výpočet podobnosti (similarity) mezi hledaným řetězcem a názvem, popisem nebo textem skladby po odstranění diakritiky. Pokud je podobnost větší než 0.3, je záznam považován za shodu.
3. Výsledky jsou seřazeny sestupně podle podobnosti hledaného řetězce s názvem, popisem a textem skladby.

Funkce tedy umožňuje najít skladby, i když uživatel nezadá přesný název nebo popis, ale pouze část textu nebo přibližný výraz.

Tento přístup poskytuje uživatelsky přívětivé fulltextové vyhledávání, které toleruje drobné chyby a nepřesnosti ve vyhledávaném výrazu. Zároveň dává přednost přesnějším shodám tím, že výsledky řadí podle míry podobnosti.

### 4.4.3 Vyhledávání MIDI

V minulé části se připravila PG funkce pro fulltextové vyhledávání skladeb. V této části je použita a popíše se zde logika vypisování seznamu MIDI skladeb.

K načítaní MIDI vzniknul tRPC endpoint *trpc.midi.list*:

```
import { and, asc, eq, sql } from "drizzle-orm";
import { prepareMidiFromCloud } from "~/pages/play/[id]";

const midiRouter = createTRPCRouter({
  list: protectedProcedure
    .input(
      z.object({
        query: z.string().max(100).optional(),
      }),
    )
    .query(async ({ ctx, input }) => {
      if (!input.query) {
        return await ctx.db
          .select({
            id: midis.id,
            name: midis.name,
          })
          .from(midis)
          .where(eq(midis.userId, ctx.session.user.id))
          .orderBy(asc(midis.name));
      }

      const MIDI_QUERY_SQL = sql`SELECT id, name FROM
fuzzy_search_midi(${input.query}) fsm WHERE userId = ${ctx.session.user.id}
ORDER BY name ASC`;

      const midiList = (await ctx.db.execute(MIDI_QUERY_SQL)) as Midi[];
      return midiList;
    }),
   // …
});
```

**Ukázka kódu 20 - endpoint vyhledávání MIDI – vlastní zpracování**

Tento TRPC endpoint sloužící pro vyhledávání funguje ve dvou režimech. Pokud **query** není zadán, procedura provede dotaz do databáze pomocí DrizzleORM a vrátí seznam všech MIDI skladeb patřících přihlášenému uživateli.

Dotaz vybírá sloupce **id** a **name** z tabulky **midis**, filtruje záznamy podle **userId** přihlášeného uživatele a řadí výsledky vzestupně podle názvu skladby. V případě že parametr **query** je vyplněn podmínka prázdné hodnoty se nesplní provede vyhledávácí dotaz pomocí **fuzzy_search_midi** funkce, kterou jsme si vydefinovali výše. Vzhledem k tomu že DrizzleORM přímé volání PG funkcí nepodporuje je v implementaci použit čistý SQL dotaz.

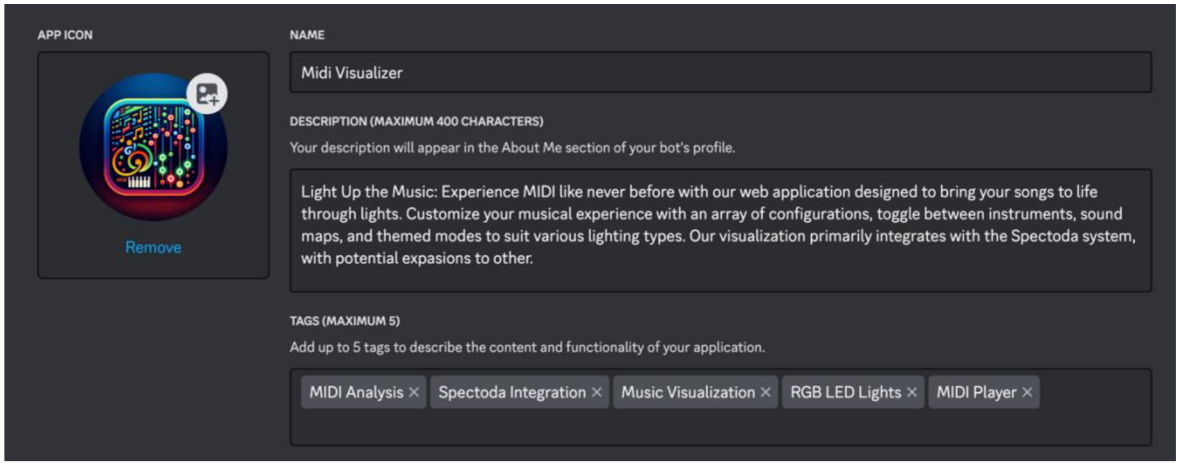
Vstupy jsou ochráněny skrze Zod, který hlídá že vstupní hodnota je objekt, s volitelným vyplněním jeho atribut **query**, který může být pouze text, a to s maximální délkou 100 znaků. Zároveň je dotaz chráněn i vůči potenciálnímu SQL Injection útoku pomocí **sql``** template literal funkce, která vstupní data sanitizuje.


**Obrázek 17 – ukázka rozhraní, výsledek vyhledávání nahraných skladeb obsahující spojeni "mel jsem" - vlastní zpracování**

V seznamu můžeme vidět I přesto že není text “mel jsem” obsažen přímo v názvu skladby, přesto byl vylistován, a to díky obsazení v titulkách, a to i s diakritikou (přesto že ve vyhledávacím dotazu je bez ní).

### 4.4.4 Implementace přihlášení

Pro přihlášení projekt používá knihovnu NextAuth.js, která řeší z velké části přihlášení za nás. Pro přihlašování budeme používat externího OAuth providera. Zde jsem si vybral Discord.




**Obrázek 18 – Vyplnění App developer formuláře pro umožnění přihlašování skrze Discord – vlastní zpracování**

Po implementaci NextAuth.js potřebujeme vytvořit tabulky, které budou ukládat data o uživatelích do databáze. Používáme zde koncept uživatele a účtu.

```
export const users = createTable("user", {
  id: varchar("id", { length: 255 }).notNull().primaryKey(),
  name: varchar("name", { length: 255 }),
  email: varchar("email", { length: 255 }).notNull(),
  emailVerified: timestamp("emailVerified", {
    mode: "date",
  }).default(sql`CURRENT_TIMESTAMP`),
  image: varchar("image", { length: 255 }),
});
```

**Ukázka kódu 21 – Uživatel, databázové schéma v DrizzleORM – vlastní zpracování**

Účet (account) je zástupná jednotka přihlášení uživatele z různých zařízení, případně i jiných OAuth provideru, jako je Google OAuth, Github atd. Ty se odkazují na konkrétního uživatele. Jeho unikátnost máme ověřenou emailovou adresou, kterou nám OAuth poskytuje.

Stejným způsobem jako Discord jsou implementovány i další způsoby přihlášení jako je Google, či speciálně Email, který funguje je bázi zasílání magic link (přihlašovacího odkazu do emailové schránky).

```
export const authOptions: NextAuthOptions = {
  /// ...
  adapter: DrizzleAdapter(db, createTable) as Adapter,
  providers: [
    DiscordProvider({
      clientId: env.DISCORD_CLIENT_ID,
      clientSecret: env.DISCORD_CLIENT_SECRET,
    }),
    GoogleProvider({
      clientId: env.GOOGLE_CLIENT_ID,
      clientSecret: env.GOOGLE_CLIENT_SECRET,
    }),
    EmailProvider({
      server: {
        host: env.EMAIL_SERVER_HOST,
        port: env.EMAIL_SERVER_PORT,
        auth: {
          user: env.EMAIL_SERVER_USER,
          pass: env.EMAIL_SERVER_PASSWORD,
        },
      },
      from: env.EMAIL_FROM,
    }),
  ],
};
```

**Ukázka kódu 22 – Přihlašovací poskytovatelé OAuth – vlastní zpracování**

V ukázce můžeme vidět implementaci, citlivé přístupové údaje nejsou obsaženy přímo ve zdrojovém kódu, ale máme je uložené v tajném **.env** souboru.

V souboru .env jsou tajné klíče, v ukázce níže jsou pouze ukázkové údaje, nikoli reálné.

```
POSTGRES_USER=bakalarka_uhk
POSTGRES_PASSWORD=Heslo1234 # ukazkové heslo

NEXTAUTH_SECRET=
…
DISCORD_CLIENT_ID=
```

**Ukázka kódu 23 – Ukázkový .env soubor – vlastní zpracování**

I vyplněnost tohoto souboru máme také kontrolovanou a to knihovnou **@t3-oss/env-nextjs**, která tyto údaje dále hlídá a v případě chybějících polí aplikaci při startu ukončí s chybou upozorňující potenciální problém.

```
export const usersRelations = relations(users, ({ many }) => ({
  accounts: many(accounts),
}));
export const accounts = createTable(
  "account",
  {
    userId: varchar("userId", { length: 255 })
      .notNull()
      .references(() => users.id),
    type: varchar("type", { length: 255 })
      .$type<AdapterAccount["type"]>()
      .notNull(),
    provider: varchar("provider", { length: 255 }).notNull(),
    providerAccountId: varchar("providerAccountId", { length: 255 }).notNull(),
    refresh_token: text("refresh_token"),
    access_token: text("access_token"),
    expires_at: integer("expires_at"),
    token_type: varchar("token_type", { length: 255 }),
    scope: varchar("scope", { length: 255 }),
    id_token: text("id_token"),
    session_state: varchar("session_state", { length: 255 }),
  },
  (account) => ({
    compoundKey: primaryKey({
      columns: [account.provider, account.providerAccountId],
    }),
    userIdIdx: index("account_userId_idx").on(account.userId),
  }),
);
```

**Ukázka kódu 24 – Účet a jeho vazba k uživateli tabulka – převzato, upraveno [29]**

Toto schéma tabulek následně řeší samotné provázání uživatelského účtu s poskytovatelem přihlášení.

### 4.4.5 Hosting

Pro hosting databáze byl zvolen vlastní hosting na Linuxové VPS v cloudu Oracle v prostředí Docker kontejneru.

```
version: '3'
services:
  postgres:
    image: postgres:15.3
    restart: always
    ports:
      - "5432:5432"
    env_file:
      - ./.env
```

**Ukázka kódu 25 – docker-compose.yml pro nasazení databáze aplikace na VPS – vlastní zpracování**

Použití vlastního hostingu na VPS umožňuje plnou kontrolu nad prostředím a konfiguračními možnostmi databáze. Databáze běží izolovaně v Docker kontejneru, což zajišťuje snadnou přenositelnost.

Samotná aplikace napsaná ve frameworku Next.js je hostována na platformě Vercel.

**Vercel** je cloudová platforma zaměřená na hostování moderních webových aplikací, zejména těch postavených na frameworcích jako Next.js, React, Angular nebo Vue. Poskytuje automatické nasazování CD (Continuous Deployment) přímo z Git repozitáře, což značně zjednodušuje a zrychluje proces nasazení aplikace.

# 5 Závěr

Výsledkem práce je funkční a veřejně dostupný webový MIDI přehrávač, který umožňuje uživatelům jednoduché přehrávání skladeb přímo ve webovém prohlížeči bez nutnosti instalace dalších softwarových komponentů. Tato aplikace slouží nejen k poslechu, ale i jako zjednodušená alternativa k tradičním karaoke programům, jako je např. KaraFun, nebo Vanbasco Karaoke Player.

Jako další krok byla do projektu integrována možnost připojení světelných instalací Spectoda, což přineslo nové možnosti pro vizualizaci MIDI dat. Definice pro tyto zařízení a komunikace s BLE API Spectoda umožňují uživatelům v Spectoda Studio přizpůsobit světelné efekty dle vlastních preferencí. Pro fyzickou vizualizaci lze použít produkty jako jsou SpectaSticks, NARA Strip či instalace postavené na technologii Spectoda.

Programování tohoto projektu bylo technicky náročné a vyžadovalo kombinaci mnoha moderních technologií. Z důvodu technických omezení, se nepodařilo dosáhnout plánované úrovně vizualizací, což vedlo k tomu, že efekty nejsou tak vizuálně atraktivní, jak bych si představoval.

Výzvou zůstává také plně integrovat systém s alternativními řešeními a platformami, jako je například Philips Hue. To bylo omezeno schopnostmi webových prohlížečů a bylo by třeba implementovat pomocí lokálního Node.js procesu sloužícího jako most.

# 7 Seznam použité literatury


[1] MIDI.ORG. MIDI 1.0 Detailed Specifications. *midi.org* [online]. 1996 [vid. 2024-02-12]. Dostupné z: https://www.midi.org/specifications-old/item/the-midi-1-0-specification

[2] *USB MIDI převodník pro PC | Příslušenství | SOH.cz* [online]. [vid. 2024-03-20]. Dostupné z: http://eshop.soh.cz/prislusenstvi/i159-usb-midi-prevodnik-pro-pc

[3] BECK, David. Standard MIDI file format, updated. *Standard MIDI-File Format Spec. 1.1, updated* [online]. 1999 [vid. 2024-01-23]. Dostupné z: https://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html

[4] BREVE, Bernardo, Stefano CIRILLO, Mariano CUOFANO a Domenico DESIATO. Perceiving space through sound: mapping human movements into MIDI. In: [online]. 2020, s. 49–56. Dostupné z: doi:10.18293/DMSVIVA20-011

[5] BEVACQUA, Nicolas. *Practical Modern JavaScript: Dive Into ES6 and the Future of JavaScript*. B.m.: O'Reilly Media, Inc., 2017. ISBN 978-1-4919-4350-2.

[6] K, jayaprabhakar. Rethinking Atwood's Law. *Medium* [online]. 3. leden 2018 [vid. 2024-02-06]. Dostupné z: https://jayaprabhakar.medium.com/rethinking-atwoods-law-64a894b54aa4

[7] HARTINGER, David Čápka. *Lekce 1 - Úvod do JavaScriptu* [online]. 2023 [vid. 2024-01-29]. Dostupné z: https://www.itnetwork.cz/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk

[8] HICKEI, James. *Refactoring TypeScript: Keeping your code healthy*. B.m.: Packt Publishing Ltd, 2019. ISBN 978-1-83921-841-5.

[9] HILBOLLING, Susan, Hans BERENDS, Fleur DEKEN a Philipp TUERTSCHER. Sustaining Complement Quality for Digital Product Platforms: A Case Study of the Philips Hue Ecosystem. *Journal of Product Innovation Management* [online]. 2021, **38**(1), 21–48 [vid. 2024-02-04]. ISSN 0737-6782, 1540-5885. Dostupné z: doi:10.1111/jpim.12555

[10] GITHUB HUEJAY. *huejay/README.md at master · sqmk/huejay* [online]. 2018 [vid. 2024-02-06]. Dostupné z: https://github.com/sqmk/huejay/blob/master/README.md

[11] SPECTODA WEB. *F&Q - Spectoda* [online]. 2024 [vid. 2024-02-06]. Dostupné z: https://spectoda.com/caste-dotazy/

[12] SPECTODA WEB. Spectoda. *Spectoda* [online]. 2024 [vid. 2024-02-06]. Dostupné z: https://spectoda.com/

[13] GITHUB SPECTODA. Spectoda/spectoda-js. *Github* [online]. 2024 [vid. 2024-02-06]. Dostupné z: https://github.com/Spectoda/spectoda-js

[14] RUBAN. Mesh Topology – Advantages And Disadvantages of a Mesh Topology.

*OFBIT* [online]. 9. květen 2022 [vid. 2024-02-06]. Dostupné z: https://ofbit.in/mesh-topology-advantages-and-disadvantages-of-mesh-topology/

[15] Popular MIDIs. *BitMidi* [online]. [vid. 2024-04-23]. Dostupné z: https://bitmidi.com/

[16] LOOPAZON.COM. Top 13 nejoblíbenějších digitálních audio pracovních stanic (2023). *loopazon.com* [online]. 31. leden 2023 [vid. 2024-02-12]. Dostupné z: https://www.loopazon.com/blog/top-13-most-popular-digital-audio-workstations?sl=cs

[17] MUSESCORE.ORG. *Program na skládání hudby a zapisování not | MuseScore* [online]. 2024 [vid. 2024-02-12]. Dostupné z: https://musescore.org/cs

[18] *signal* [online]. [vid. 2024-02-12]. Dostupné z: https://signal.vercel.app/

[19] Bun — A fast all-in-one JavaScript runtime. *Bun* [online]. [vid. 2024-03-11]. Dostupné z: https://bun.sh

[20] *React* [online]. [vid. 2024-03-11]. Dostupné z: https://react.dev/

[21] Tailwind CSS: další evoluční krok pro CSS frameworky. *Vzhůru dolů* [online]. [vid. 2024-03-11]. Dostupné z: https://www.vzhurudolu.cz/prirucka/tailwind-css

[22] WÉBR, Daniel. *Webová aplikace pro stimulaci akustickými podněty k terapeutickému využití* [online]. B.m., 2023 [vid. 2024-03-24]. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Dostupné z: https://theses.cz/id/n4yc6h/

[23] *blitz-js/superjson* [online]. TypeScript. B.m.: ⚡Blitz. 31. březen 2024 [vid. 2024-04-01]. Dostupné z: https://github.com/blitz-js/superjson

[24] Research. *Magenta* [online]. [vid. 2024-04-01]. Dostupné z: https://magenta.tensorflow.org/research/

[25] RYOHEY. *ryohey/signal* [online]. TypeScript. 9. únor 2024 [vid. 2024-02-12]. Dostupné z: https://github.com/ryohey/signal

[26] PostgreSQL. *PostgreSQL* [online]. 11. prosinec 2023 [vid. 2024-03-11]. Dostupné z: http://postgres.cz/wiki/PostgreSQL

[27] *Drizzle ORM - Overview* [online]. [vid. 2024-04-16]. Dostupné z: https://orm.drizzle.team/docs/overview

[28] SANTHIKUMAR, Varun. Fuzzy text matching with PostgreSQL pg_trgm, fuzzystrmatch extension. *Medium* [online]. 24. září 2023 [vid. 2024-04-16]. Dostupné z: https://medium.com/@varun.santhikumar94/fuzzy-text-matching-with-postgresql-pg-trgm-fuzzystrmatch-extension-3cb25c2216b1

[29] *NextAuth.js* [online]. [vid. 2024-03-20]. Dostupné z: https://next-auth.js.org

# 8 Přílohy

**Odkaz na webovou aplikaci:** https://bp-midi.lukaskovar.com

**GitHub repositář:** https://github.com/sirluky/bp-midi-visualizer

# 9 Zadání práce z IS (eVŠKP)

Univerzita Hradec Králové
Fakulta informatiky a managementu

## Zadání bakalářské práce

| | |
|---|---|
| **Autor:** | **Lukáš Kovář** |
| Studium: | I2100226 |
| Studijní program: | B1802 Aplikovaná informatika |
| Studijní obor: | Aplikovaná informatika |
| **Název bakalářské práce:** | **Systém pro vizuální reprezentaci MIDI souborů s interaktivním osvětlením** |
| Název bakalářské práce AJ: | System for visual representation of MIDI files with interactive lighting |

**Cíl, metody, literatura, předpoklady:**

Cílem je vytvořit webovou aplikaci, která bude umět vizualizovat přehrávané MIDI soubory na světlech. Aplikace bude obsahovat různé konfigurátory pro přepínání mezi hudebními nástroji, zvukovými mapami a tematickými módy vhodná pro různé typy světel, speciálně RGB led pásky. Vizualizace bude primárně implementována na Spectoda systém, s případným rozšířením na další systémy (např Philips Hue).

Osnova:

- Úvod
- Teoretická východiska
- Praktcká část
- Shrnutí výsledků
- Závěry a doporučení
- Seznam použité literatury

Anon., [b.r.]. Bluetooth Technology Overview. Bluetooth® Technology Website [online] [vid. 2023-10-19]. Dostupné z: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

BUFFA, M., S. REN, O. CAMPBELL, T. BURNS, S. YI, J. KLEIMOLA a O. LARKIN, 2022. Web Audio Modules 2.0: An Open Web Audio Plugin Standard. In: WWW 2022 - Companion Proceedings of the Web Conference 2022 [online]. s. 364–369. ISBN 978-1-4503-9130-6. Dostupné z: doi:10.1145/3487553.3524225

CÔTÉ, J.-P., 2022. User-Friendly MIDI in the Web Browser. In: Proceedings of the International Conference on New Interfaces for Musical Expression [online]. ISSN 2220-4792. Dostupné z: doi:10.21428/92fbeb44.388e4764

FERREIRA, F. a M.T. VALENTE, 2023. Detecting code smells in React-based Web apps. Information and Software Technology [online]. 155. ISSN 0950-5849. Dostupné z: doi:10.1016/j.infsof.2022.107111

GOTOY, M. a Y. MURAOKA, 1998. An Audio-based Real-time Beat Tracking System and Its Applications. In: International Computer Music Conference, ICMC Proceedings. ISSN 2223-3881.

HERNANDEZ-OLIVAN, Carlos a Jose R. BELTRAN, 2023. Musicaiz: A python library for symbolic music generation, analysis and visualization. SoftwareX [online]. 22, 101365 [vid. 2023-10-19]. ISSN 2352-7110. Dostupné z: doi:10.1016/j.softx.2023.101365

HSIAO, S.-W., S.-K. CHEN a C.-H. LEE, 2017. Methodology for stage lighting control based on music emotions. Information Sciences [online]. 412–413, 14–35. ISSN 0020-0255. Dostupné z: doi:10.1016/j.ins.2017.05.026

SAPUTRA, R.E. a A.S. PRIHATMANTO, 2012. Design and implementation of BeatME as a Networked Music Performance (NMP) system. In: Proceedings of the 2012 International Conference on System Engineering and Technology, ICSET 2012 [online]. ISBN 978-1-4673-2376-5. Dostupné z: doi:10.1109/ICSEngT.2012.6339349

STASIOWSKI, Dominik, [b.r.]. Rozšíření volejbalové pomůcky Training lights do dalších sportovních odvětví. https://dspace.cvut.cz/handle/10467/108997

Anon., 2015. Web MIDI API [online] [vid. 2023-10-19]. Dostupné z: https://www.w3.org/TR/webmidi/

Zadávající pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: Ing. Jaroslav Langer

Datum zadání závěrečné práce: 15.10.2021