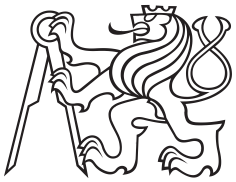
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

# Přehrávání MIDI pomocí Web Audio API

Bc. Marek Šach

Vedoucí: RNDr. Ondřej Žára
Obor: Otevřená Informatika
Studijní program: Softwarové Inženýrství
Květen 2021

## Poděkování

Děkuji vedoucímu RNDr. Ondřeji Žárovi za spolupráci a vedení při tvorbě této práce.


## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 19. května 2021

# Abstrakt

Cílem této práce je návrh a vytvoření klientské webové aplikace, která bude přehrávat MIDI soubory s využitím uměle syntetizovaných zvuků hudebních nástrojů pomocí Web Audio API.

Nejprve provedu seznámení s rozhraním MIDI, souborovým formátem SMF a koncepty digitální syntézy zvuku. Dále popíšu, jakým způsobem lze využít Web Audio API k syntéze zvuků hudebních nástrojů. Nakonec navrhnu a vytvořím aplikaci, která bude syntézu zvuků hudebních nástrojů demonstrovat na několika příkladech a tyto nástroje bude využívat k přehrávání MIDI souborů.



**Klíčová slova:** Web Audio API, MIDI, SMF, digitální syntéza zvuku

**Vedoucí:** RNDr. Ondřej Žára



# Abstract

This thesis aims to design and implement a client web application that will be able to play MIDI files with artificially synthesized sounds using the Web Audio API.

First of all, I will introduce the MIDI (*Music Instrument Digital Interface*), SMF (*Standard MIDI File*) and basic concepts of digital audio synthesis. After that, I will describe how to synthesize sounds of musical instruments using the Web Audio API. Finally, I will design and implement an application that will demonstrate the musical instruments' sounds synthesis with a few examples and it will be using those instruments for a MIDI playback.



**Keywords:** Web Audio API, MIDI, SMF, digital sound synthesis

**Title translation:** MIDI playback using Web Audio API Synth

# Obsah

# Obrázky



# Tabulky

# Kapitola 1

# Úvod

Tématem této práce je vytvoření klientské webové aplikace, která bude sloužit k přehrávání souborů ve formátu MIDI s využitím zvuků syntetizovaných pomocí Web Audio API.

Aplikací, které umožňují přehrávání MIDI samozřejmě existuje mnoho, prakticky každý běžný DAW (*digital audio workstation*) software tuto funkcionalitu nabízí. Každý takový přehrávač MIDI potřebuje pro přehrávání nějaký virtuální hudební nástroj, který bude schopen produkovat zvuky při přehrávání. Virtuální hudební nástroj je potřeba z toho důvodu, že soubor MIDI je pouze jakousi strojově čitelnou analogií k notovému zápisu, popisuje tedy tóny (popř. další hudební data), které mají být přehrány, ale nespecifikuje již to, jak mají znít, jakým zvukem mají být přehrány. Obecně tedy tóny zapsané v MIDI mohou být přehrány zvukem jakéhokoliv virtuálního nástroje, který je k přehrávači přiřazen.

Virtuální hudební nástroje existují dvojího typu - samplované, které využívají nahraných vzorků (angl. *samplů*) jiných hudebních nástrojů, a syntezátory, které zvuky generují uměle pomocí oscilátorů a dalších připojených efektů.

V této diplomové práci se zaměřím na druhý zmíněný přístup - mým cílem bude představit a prakticky realizovat vytvoření přehrávače MIDI souborů a k němu několik syntetizovaných virtuálních hudebních nástrojů. Tato aplikace bude realizována jako klientská webová aplikace běžící v běžných webových prohlížečích. Samotné přehrávání a syntéza zvuků hudebních nástrojů bude realizována pomocí Web Audio API, tedy základního a nízkoúrovňového rozhraní pro práci se zvukem ve webových prohlížečích.

V textu této práce nejprve čtenáře seznámím s koncepty a technologiemi potřebnými k implementaci této aplikace, konkrétně tedy provedu krátké seznámení se standardem MIDI, dále se základními principy syntézy zvuku a s Web Audio API. Následně popíšu, jak lze pomocí těchto konceptů a technologií implementovat kýženou aplikaci.

Dále též představím, jak by tato aplikace šla rozšířit pomocí Web MIDI API, které by umožnilo připojení externí MIDI klaviatury a „živé“ hraní na již implementované virtuální hudební nástroje.

# Kapitola 2

# Protokol MIDI a souborový formát SMF

MIDI je zkratka pro *Musical Instrument Digital Interface*. Jedná se o standard pro reprezentaci hudebních dat a pro digitální komunikaci mezi hudebními nástroji a počítači. SMF je zkratka pro *Standard MIDI File* a je to jeden ze standardů, který definuje způsob serializace hudebních událostí. Tento formát tedy lze vnímat jako jakýsi strojově čitelný notový zápis, který vychází ze standardu MIDI. [1]

Soubory ve formátu SMF (obvykle s příponou *.mid*) budou vstupem přehrávače, který budu v rámci této práce implementovat. V této kapitole tedy představím základní a pro nás důležité vlastnosti protokolu MIDI a formátu SMF.

## 2.1 MIDI

Jak bylo řečeno v úvodu této kapitoly, standard MIDI slouží ke dvěma základním účelům. Zaprvé nám definuje rozhraní pro komunikaci mezi zařízeními v reálném čase, a zadruhé definuje způsob serializace hudebních dat.

Základní jednotkou pro práci v prvním zmíněném případě (komunikace v reálném čase) jsou tzv. **MIDI zprávy**. MIDI zpráva sestává ze dvou částí - *status byte* a *data bytes*.

*Status byte* obecně řečeno představuje kód nějaké hudební události, a *data bytes* tuto událost nějak podrobněji popisují.

Pro správnou interpretaci *status byte* je potřeba uvést, že existují dva druhy zpráv - zprávy systémové („obecné“) a zprávy, které mohou být přiřazeny ke specifickému MIDI kanálu. MIDI kanály představují koncept, jak lze logicky dělit zasílané MIDI zprávy. Pomocí členění zpráv do více MIDI kanálů můžeme např. pomocí jedné připojené MIDI klaviatury či kontroleru ovládat více různých virtuálních hudebních nástrojů, každý poslouchající pouze zprávy z určitého kanálu.

| Typ zprávy | Formát zprávy | Význam |
|---|---|---|
| Systémové zprávy | `nnnnnnnn` | `n` - typ zprávy |
| Ostatní zprávy | `nnnncccc` | `n` - typ zprávy, `c` - MIDI kanál |

**Tabulka 2.1:** MIDI zprávy - formát status byte

Systémové zprávy jsou svým významem obecnější a nejsou přiřazeny k žádnému konkrétnímu MIDI kanálu (využívají se např. pro reprezentaci stavu obecných ovládacích prvků MIDI klaviatury/kontroleru, typicky různých tlačítek, faderů nebo pedálů). Zprávy se specifikací MIDI kanálu se naopak používají pro události představující přímo „hraní“ na hudební nástroj (např. stisk klávesy na klaviatuře, puštění klávesy,...).[2]

Podle druhu zprávy tedy mohou být bity ve *Status byte* interpretovány dvojím způsobem, jak je naznačeno v tabulce 2.1. Seznam konkrétních kódů a jim odpovídající typy zpráv lze dohledat v dokumentaci standardu MIDI.[2]

Struktura *data bytes* je různá v závislosti na typu zprávy, který je popsán ve *status bytes*. Jedna z nejčastějších akcí je např. stisk nebo puštění klávesy (kód *note on/note off*). *Data bytes* těchto akcí obsahují údaje *note number*[1] (výška tónu) a *velocity* (síla úhozu, jakou byla klávesa stisknuta). Kromě stisku a puštění klávesy budeme v případě naší aplikace potřebovat ještě informaci o stisku/puštění sustain pedálu (pro prodloužení tónu). Události týkající se stisku pedálu jsou přenášeny pomocí systémové zprávy (tzn. nejsou přiřazeny ke specifickému kanálu) a *data bytes* specifikují, zda se jedná o stisk nebo puštění pedálu.

Pro potřeby naší aplikace nebudeme potřebovat znalost dalších druhů MIDI zpráv, proto již další příklady uvádět nebudu. V případě zvídavosti čtenáře lze samozřejmě význam různých *status bytes* dohledat přímo ve specifikaci MIDI zpráv.[2]

## 2.2 SMF

SMF je zkratkou pro *Standard MIDI Files*. Jedná se o formát souborů, v nichž jsou serializována data reprezentující hudební události. Jak bylo řečeno v úvodu této kapitoly, jedná se o jakýsi strojově čitelný notový zápis.

V souboru typu SMF lze obecně ukládat poměrně hodně různých informací. Podobně, jako u „klasického“ notového zápisu, obsahuje obecné informace týkající se celého „partu“ (např. informace týkající se předznamenání, rytmu, atd.). Sekce s těmito obecnými se je označena jako hlavička (angl. *head*).

[1] *MIDI note* je celočíselný kód odpovídající konkrétnímu tónu. Standard MIDI přiřazuje ke každému půltónu celočíselný kód, půltóny jsou číslovány vzestupně od nižších tónů k vyšším. Např. tón A4 (440Hz) odpovídá *MIDI note* = 69.

Pro účely našeho přehrávače je však důležitější část následující, která obsahuje tzv. stopy (angl. *tracks*). Jednu takovou stopu můžeme vnímat jako analogii not pro jeden konkrétní hudební nástroj v rámci partitury pro celé hudební těleso. V jedné stopě mohou být opět definované jak některé obecné informace, týkající se celé stopy (např. na jaký nástroj má být stopa zahrána, jaký MIDI kanál je ke stopě přiřazen,...), tak výčet konkrétních hudebních událostí.

Pro účely našeho přehrávače se s výkladem obsahu stopy dále omezím na sekci *tones*, jež je výčtem tónů, které mají být přehrány. U každého tónu v sekci tones můžeme vyčíst tyto důležité informace:

- **MIDI note** - výška tónu
- **Start** - čas začátku tónu
- **Duration** - čas začátku tónu
- **Velocity** - intenzita (síla úhozu/hlasitost) s jakou má být tón přehrán

V rozboru formátu by se dalo nadále pokračovat, nicméně z hlediska toho, co potřebujeme znát pro tvorbu přehrávače SMF souborů jsou již popsané znalosti dostatečné. Při implementaci našeho přehrávače budeme potřebovat umět přečíst jednotlivé stopy, umět k nim přiřadit virtuální hudební nástroje, a nakonec je pomocí těchto nástrojů přehrát. Zvídavé čtenáře se zájmem o detailnější poznání souborového formátu SMF opět odkáži na jeho oficiální dokumentaci.[1]

# Kapitola 3

# Syntéza zvuku a Web Audio API

Přehrávač implementovaný v rámci této práce bude přehrávat vstupní SMF soubory pomocí zvuků napodobujících hudební nástroje. Běžný postup pro digitální imitaci zvuku skutečných hudebních nástrojů je tzv. samplování, což je technika, při které se využívají nahrávky skutečných hudebních nástrojů a přehrávání těchto nahrávek slouží k napodobení zvuku originálního nástroje.

Samplování však v našem řešení využívat nebudeme. Cílem této práce je implementovat přehrávač, který bude napodobovat zvuky uměle, bez nutnosti využívat nahrané samply skutečných nástrojů.

V případě aplikace popisované v této práci využijeme k imitaci zvuků hudebních nástrojů techniku zvanou *modulární syntéza*. Tato technika spočívá v generování signálu a jeho následné úpravě pomocí samostatných, mezi sebou propojených, modulů. V rámci této kapitoly čtenáři představím základní principy syntézy zvuku jako důležitou prerekvizitu pro implementace syntezátoru, a dále též Web Audio API, jakožto nástroj, který budeme k implementaci samotného syntezátoru využívat.

## 3.1 Principy syntézy zvuku

Syntézou zvuku rozumíme umělé vytvoření nějakého akustického signálu. V našem případě se budeme pokoušet vytvořit takový signál, který se bude co možná nejvíce podobat zvuku nějakého skutečného nástroje.

Než se však pustíme do vymýšlení konstrukce samotného syntezátoru, představím základní parametry, které zvuk charakterizují.

### 3.1.1 Výška a barva tónu - frekvenční charakteristika

Hudební zvuk o určité sluchem rozlišitelné výšce nazýváme tón. Fyzikálně tón můžeme definovat jako periodické akustické vlnění.[1]

Pro potřeby syntézy zvuku je velice důležitý poznatek, že každé periodické vlnění lze rozložit na nějakou množinu sinusových funkcí (o různých frekvencích, amplitudách a fázových posunech), které když spolu sečteme, poskládají (nebo alespoň blízce aproximují) zpět originální signál.[3]

Provedení takového rozkladu a zjištění zastoupení jednotlivých sinusových funkcí říkáme analýza ve frekvenční oblasti signálu. Provedením takové analýzy zjistíme tzv. frekvenční charakteristiku signálu (zvuku). Grafu, který vizualizuje zastoupení jednotlivých frekvencí v signálu pak říkáme spektrogram. Při provedení frekvenční analýzy akustického signálu je pro nás pak důležité především zastoupení těchto frekvencí:

- **Základní frekvence**

  Nejnižší přítomná (nezanedbatelná) frekvence. Ta z hudebního hlediska určuje výšku tónu, který slyšíme.

- **Vyšší harmonické frekvence**

  Vyšší harmonické frekvence jsou frekvence, jejichž hodnota je celočíselným násobkem základní frekvence. Jejich zastoupení určuje z hudebního hlediska barvu poslouchaného tónu.

  Budeme-li tedy porovnávat stejně vysoký tón zahraný na dva různé hudební nástroje (tedy dva tóny stejné výšky, ale různé barvy), budou se lišit právě v zastoupení vyšších harmonických frekvencí.

Budeme-li se snažit napodobit barvu zvuku nějakého hudebního nástroje, potřebujeme znát míru zastoupení základní a vyšších harmonických frekvencí. Frekvenční analýzu můžeme buď provést sami nebo si můžeme vypůjčit hodnoty z již existujících tabulek, které popisují frekvenční charakteristiku příkladů různých typů zvuků.[3]

K popsaným faktům je nutné dodat, že toto samotné poznání nepředurčuje snadnou imitaci skutečného hudebního nástroje. Napodobit zvuk akustického hudebního nástroje je totiž v praxi velmi složité, a to z několika důvodů.

Prvním problémem je fakt, že i v případě jednoho typu nástroje mají tóny různé výšky a různé intenzity obvykle i odlišnou barvu. Neliší se tedy pouze svou frekvencí a amplitudou, ale též svou spektrální charakteristikou.

[1] Tón jsme označili jako periodické akustické vlnění. V praxi u akustických hudebních nástrojů však toto vlnění není téměř nikdy periodické, obsahuje i různé neperiodické složky (šumy, ruchy). Pro naše potřeby tuto definici můžeme zjednodušit tak, že tónem nazýváme takové vlnění, ve kterém periodické složky výrazně převažují nad těmi neperiodickými.

Druhým problémem je, že kromě samotného „čistého“ tónu nástroje jsou v signálu obsaženy další složky, které vznikají např. rezonancí některých částí hudebního nástroje, odrazy zvuku v těle nástroje nebo i prostou obsluhou nástroje. Zastoupení těchto složek je velmi složité zachytit a popsat, navíc se též liší u tónů různé výšky a intenzity.[2] Další komplikací je fakt, že tón akustického nástroje nezní po celou dobu stejně - jeho barva (tedy spektrální charakteristika) se v průběhu času mění.

Z těchto důvodů vyplývá, že zvuk skutečného akustického nástroje je v praxi velmi složitý a je téměř nemožné jej přesně matematicky popsat a aproximovat. Zároveň jsou všechny zmíněné „drobnosti“, které zvuk akustického nástroje utváří, pro jeho charakter a autenticitu často naprosto zásadní.

Při snaze napodobit zvuk nějakého hudebního nástroje pomocí syntezátoru je tedy prakticky nemožné postihnout a věrohodně napodobit všechny vlastnosti zvuku akustického nástroje.

V praxi se vždy budeme pokoušet izolovat a napodobit ty nejvýraznější a nejdůležitější aspekty zvuku konkrétního hudebního nástroje, tak se můžeme ke kýženému zvuku výrazně přiblížit. Pravděpodobně nikdy však tento výsledek nebude úplně autentický, k nerozpoznání od akustického nástroje.

### 3.1.2 Intenzita zvuku - amplituda, obálky a LFO

Dalším parametrem, který charakterizuje nějaký zvuk, je jeho intenzita (fyzikálně definována jako amplituda signálu) a především její změny v průběhu času. Většina hudebních nástrojů má tento průběh specifický a když se budeme snažit zvuk těchto nástrojů napodobit, budeme muset i jejich amplitudový průběh co nepřesněji napodobit.

Představme si jako příklad průběh amplitudy u tónu klavíru. Po stisku klávesy na klaviatuře amplituda signálu velmi rychle vystoupá na svou maximální hodnotu. Když budeme dále držet klávesu stisknutou (nebo budeme držet sešlápnutý pedál), tón bude dále znít, ovšem jeho intenzita bude pomalu klesat. Když nakonec klávesu na klaviatuře pustíme (popřípadě pustíme sešlápnutý pedál), zvuk se relativně rychle utlumí a jeho amplituda klesne na nulu.

Když se budeme pokoušet synteticky napodobit zvuk klavíru, popřípadě libovolného jiného hudebního nástroje, budeme potřebovat kromě barvy jeho zvuku též napodobit amplitudový průběh jeho intenzity. Obě kritéria jsou pro kvalitu napodobení hudebního nástroje velmi důležitá.

[2] Barvu tónu hraného na nástroj často významně ovlivňuje též technika hraní, např. u strunných nástrojů to, v jakém místě a jakým způsobem strunu rozechvějeme, na které struně tón hrajeme, a podobně.

Pro popis průběhu amplitudy signálu v čase využíváme tzv. *obálky* (anglicky *envelopes*). V případě většiny hudebních nástrojů nám poslouží tzv. *ADSR Envelope* (*attack, decay, sustain, release*). Ta sestává ze čtyř částí, které svou strukturou dobře odpovídají tomu, jak se v praxi hraje na hudební nástroje a jak tímto hraním určujeme průběh amplitudy tónu.

Jak vypadá ADSR envelope demonstruje obrázek 3.1. Jednotlivé fáze ADSR envelope mají následující význam: [4]

- **Attack**

  Část bezprostředně po stisku klávesy. Během tohoto úseku má obálka rostoucí průběh a amplituda obvykle rychle roste z nuly až do svého maxima.

- **Decay**

  Část následující za attackem, během které úroveň amplitudy obvykle rychle poklesne na úroveň, na které se ustálí pro následující fázi sustain. Můžeme se však setkat i s obálkami, kde decay klesne až na nulu (obálka touto fází tedy skončí), anebo s obálkami, kde má fáze decay nulovou délku (z fáze attack se rovnou přejde na sustain, decay se vynechá).

- **Sustain**

  Třetí část obálky, která má oproti ostatním fázím obálky speciální vlastnost, a sice nemá danou délku. První dvě fáze, attack a decay, mají danou délku. Fáze sustain vyplňuje čas od skončení fáze decay po puštění klávesy klaviatury. Během fáze sustain úroveň modulace obálkou typicky buď setrvává konstantní nebo postupně klesá.

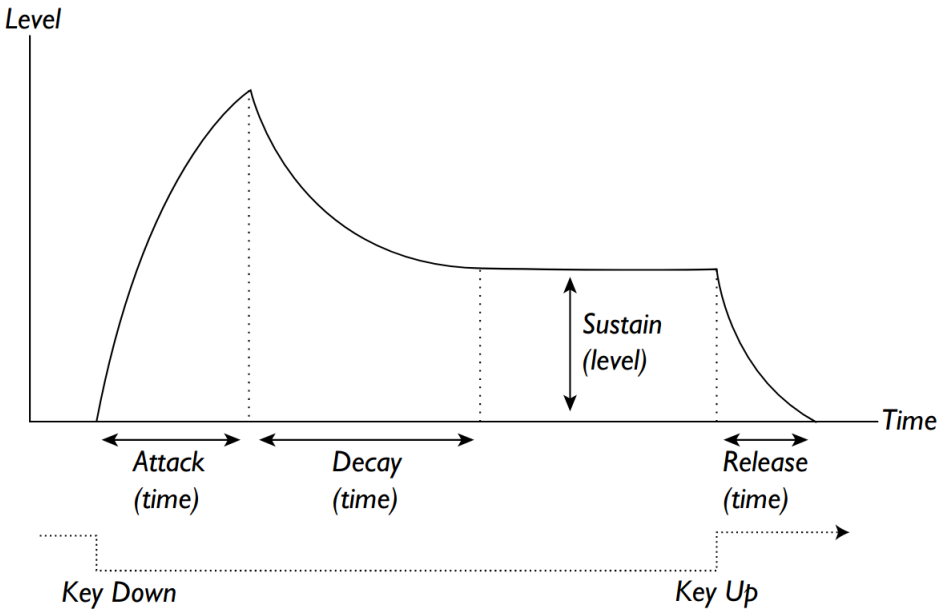
- **Release**

  Poslední fáze obálky, která je zahájena puštěním klávesy. Typicky slouží k rychlému poklesu (rychlejšímu, než u sustainu) úrovně hodnoty amplitudy na nulu, tedy ke konečnému utlumení tónu.

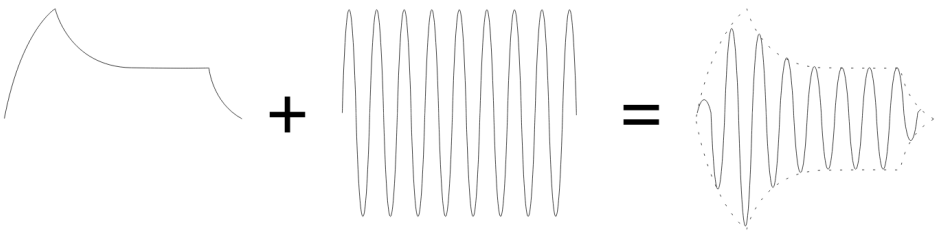
V praxi, když se budeme pokoušet napodobit zvuk nějakého hudebního nástroje, budeme v první řadě potřebovat signál se správnou barvou zvuku. Pomocí obálek pak budeme modulovat amplitudu tak, aby měla stejný průběh, jako by tomu bylo u originálního nástroje (obrázek 3.2).

Obálka však nemusí takto modulovat pouze amplitudu signálu, může modulovat obecně libovolný parametr v časové oblasti. Modulaci amplitudy budeme využívat pravděpodobně nejčastěji, ta je potřeba u každého tónu. Modulace jiných parametrů je však též nesmírně užitečná - modulací vhodných parametrů si můžeme pomoci např. k vytvoření proměnlivé barvy tónu v průběhu času.

Dalším prostředkem, pomocí kterého můžeme modulovat akustický signál je tzv. *LFO* (*Low Frequency Oscillator*). Technicky vzato funguje *LFO* velmi




**Obrázek 3.1:** ADSR Envelope (zdroj: [4])



**Obrázek 3.2:** Modulace signálu pomocí obálky (zdroj: [4])

podobně, jako obálka - máme nějakou funkci, pomocí které můžeme modulovat akustický signál v čase. V případě konceptu *LFO* je tato funkce periodická (většinou má sinusový průběh) a je generována oscilátorem o velmi nízké frekvenci (mimo slyšitelné spektrum). Navíc začátek této funkce, na rozdíl od obálky, není závislý na času začátku tónu, který je modulován.

Příkladem využití *LFO* může být modulace frekvence v průběhu času, která vytvoří efekt *vibrato* (tón se periodicky lehce zvyšuje a následně klesá) nebo třeba modulace amplitudy, která vytvoří efekt *tremolo* (tón se periodicky lehce zesiluje a následně tlumí). Jak jsem však již zmínil, stejně jako v případě obálek můžeme být v praxi *LFO* využit k modulaci libovolného parametru signálu.[4]

## 3.2 Schéma syntezátoru

Nyní, když máme základní znalosti o vlastnostech akustického signálu, můžeme se ve výkladu posunout k samotným syntezátorům, tedy zařízením, která syntetický zvuk vytvářejí.

Syntezátor běžně sestává z různých navzájem propojených modulů, ve kterých akustický signál vzniká a následně se upravuje tak, aby měl na výstupu kýžené vlastnosti. Obecně můžeme rozdělit tyto moduly na dva typy:

- **Zdrojové uzly (oscilátory)**

  Modul generující akustický signál - zde zvuk vzniká. Nejčastěji je zdrojem zvuku v syntezátoru oscilátor, ten generuje periodický signál. Ve speciálních případech však též můžeme využít jiné zdroje zvuku, které oproti oscilátoru umí generovat i neperiodický signál. V syntezátoru může být jeden nebo více různých zdrojů zvuku, signál z těchto zdrojů zvuku můžeme následně libovolně sčítat a vytvářet tak složitější a komplexnější signál.

- **Efekty**

  Efekty jsou moduly, které přijímají na vstupu již nějaký existující signál a tento signál nějakým způsobem upravují. Příkladem takových efektů mohou být např. filtry, které omezují frekvenční rozsah signálu propouštěný na výstup, gain upravující amplitudu signálu, případně nějaké další, komplexnější efekty.

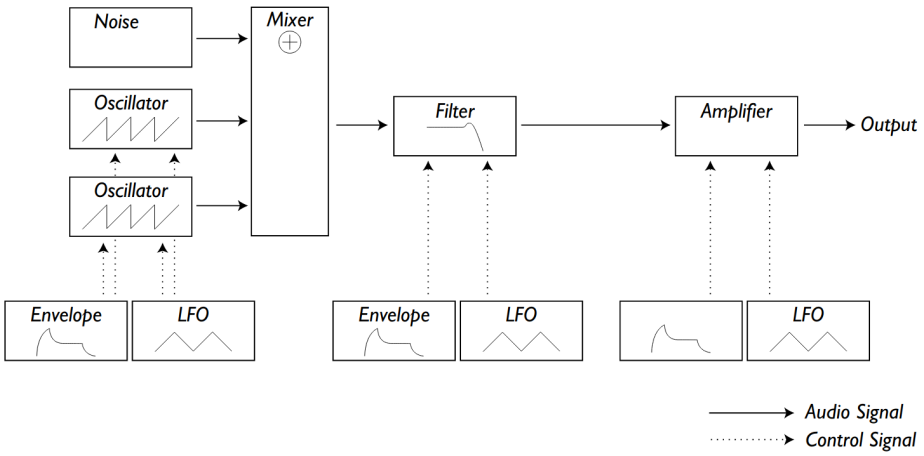
Funkce jednotlivých modulů je standardně určena hodnotami nějakých parametrů. Tyto parametry mohou být buď staticky nastaveny nebo se mohou v průběhu času měnit, průběh této změny většinou definujeme pomocí obálek nebo LFO.[4]

Příklad, jakým způsobem mohou být za sebou popsané moduly zapojeny, a jak může schema syntezátoru vypadat, zobrazuje obrázek 3.3.

## 3.3 Web Audio API

Web audio API je základní a nízkoúrovňový nástroj pro práci se zvukem ve webových prohlížečích a právě s jeho pomocí budeme implementovat přehrávač MIDI souborů a syntetizovat zvuky hudebních nástrojů.

Web audio API vytváří a poskytuje přístup k audio kontextu, tzv. *Audio context*, ve kterém provádí všechny operace spojené s prací se zvukem. Konkrétní operace nad audio signálem v Audio contextu jsou definovány a prováděny pomocí uzlů, tzv. *Audio nodes*. Audio nodes mohou být navzájem




**Obrázek 3.3:** Příklad schématu syntezátoru (zdroj: [4])

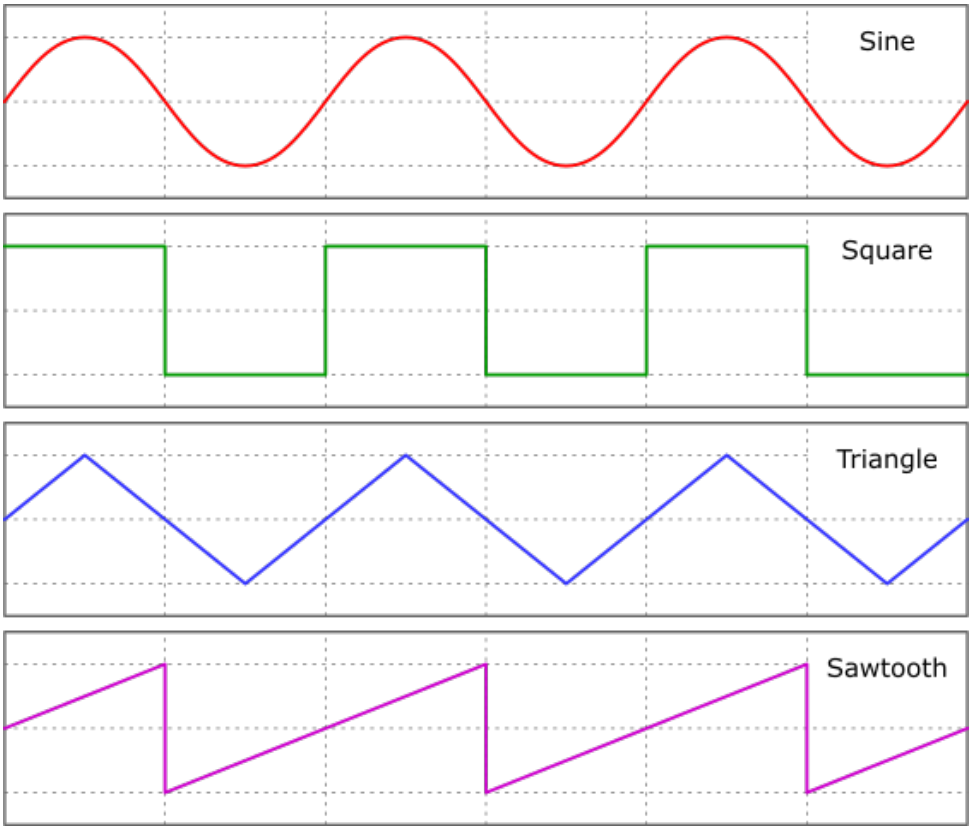

**Obrázek 3.4:** Schéma Audio contextu ve Web Audio API (zdroj: [5])

propojovány a tím vytvářet orientovaný graf, kterým audio signál prochází a postupně se modifikuje.[5] V tomto smyslu je práce a zvukotvorba pomocí Web Audio API stejná, jako při návrhu obecného syntezátoru, popsaného v předchozí sekci této práce.

Stejně jako u běžných syntezátorů můžeme uzly ve Web Audio API rozdělit na uzly, které zvuk vytvářejí (*source nodes*) a na uzly, které signál pouze upravují (*effects*). Na konci grafu tvořeného propojenými uzly v Audio contextu je speciální uzel *destination*. Uzel destination má pouze vstup a nemá výstup. Signál, který je doveden do uzlu *destination* se webový prohlížeč pokusí přehrát na výstupním zařízení. Popsané obecné schéma práce s web audio API zobrazuje obrázek 3.4.

### 3.3.1 Oscillator node

Pro vytváření akustického signálu ve Web Audio API slouží zdrojové uzly (*source nodes*). Těch existuje více druhů, obecně je můžeme rozdělit na uzly, které dostanou jako vstup nějakou nahranou audio stopu a tu přehrávají




**Obrázek 3.5:** Základní funkce generované OscillatorNode (zdroj: [6])

(např. `AudioBufferSourceNode`, `MediaElementAudioSourceNode`), a na uzly, které zvuk tvoří čistě synteticky (`OscillatorNode`).[5]

Zdrojové uzly přehrávající nahrané stopy by byly užitečné pro implementaci samplovaného nástroje. Naším cílem je však vytvoření syntezátoru, který bude tvořit zvuk bez využití nahraných samplů. Pro generování audio signálu budeme tedy kromě několika výjimek využívat `OscillatorNode`.

`OscillatorNode` umožňuje generování různých tvarů křivek. Jednak umožňuje generování křivek odpovídající základním (a pro syntezátory běžným) matematickým funkcím (`sine`, `square`, `sawtooth` a `triangle` [5], tyto funkce zobrazuje obrázek 3.5).

Kromě základních funkcí však `OscillatorNode` nabízí ještě jinou zajímavou cestu, jak lze generovat signál o libovolném průběhu, respektive zvuk libovolné barvy. Umožňuje totiž generování speciální vlastní, tzv. `custom` funkce. U oscilátoru typu `custom` definujeme průběh funkce, kterou bude oscilátor generovat, pomocí objektu `PeriodicWave`. Ten v sobě nese data, která popisují zastoupení jednotlivých harmonických frekvencí ve spektru požadovaného výsledného signálu. Přesněji řečeno v sobě nese dvě pole čísel (`real` a `imag`), která představují koeficienty Fourierovy řady, která zastoupení a vlastnosti harmonických frekvencí v signálu popisuje.[7]

Při vytváření zvuku o kýžených vlastnostech a barvě tedy můžeme využít jeden nebo více libovolných oscilátorů a jejich signál dle potřeby sčítat. V mnohých případech lze navíc podobný signál vytvořit různými přístupy. Např. signál některého ze základních typů oscilátorů a za ním připojený efekt (např. filtr) můžeme nahradit pouze oscilátorem typu `custom`, kde definujeme přímo spektrum výsledného signálu (a příslušný efekt tudíž nevyužijeme).

Při návrhu syntezátoru je tedy vhodné uvážit různé přístupy a jejich možné výhody. Například je užitečné rozmyslet, které vlastnosti zvuku může být užitečné ovládat samostatně a ty se pokusit izolovat do samostatných uzlů Můžeme například chtít měnit parametr nějakého efektu v čase, v takovém případě je vhodné mít k dispozici samostatný uzel realizující tento efekt, a nikoliv rovnou „hotový“ signál s `custom` oscilátoru, u kterého již obdobnou úpravu v čase nemůžeme snadno provést. Obecně však neexistuje univerzální návod na „optimální“ návrh syntezátoru. Různé přístupy mohou disponovat různými výhodami a nevýhodami a pro posouzení kvality různých přístupů nakonec nejlépe poslouží jejich obliba a míra využití v hudební praxi. Objektivní a měřitelná kritéria kvality různých přístupů se hledají jen velmi těžko.

### 3.3.2 Gain node

`GainNode` je uzel, který ve Web Audio API slouží k řízení amplitudy procházejícího signálu. Má tedy vstup, kudy je do něj přiveden signál, ten může `GainNode` zesílit nebo zeslabit a poslat do dalšího uzlu.

Ovládání amplitudy signálu je ve Web Audio API relativně kritická záležitost, proto se u ní na chvíli zastavím. V předchozí části této kapitoly jsem nastínil myšlenku, že v rámci syntézy jednoho tónu můžeme využívat jeden nebo více oscilátorů. Uvažme též, že jeden hudební nástroj může hrát více tónů současně. Zároveň přehrávaný MIDI soubor může obsahovat více stop, tudíž v přehrávači může hrát mnoho hudebních nástrojů současně.

Signál ze všech zdrojů (postupně ze všech oscilátorů, virtuálních hudebních nástrojů a stop) se postupně sčítá. Sčítání signálu provedeme jednoduše tak, že výstup z více uzlů připojíme na vstup pouze jednoho uzlu, zde se signál sečte. A při sčítání signálu si musíme dát pozor na důležitou věc - amplituda signálu z oscilátoru při výchozím nastavení nabývá hodnoty „jedna“[3]. Budeme-li sčítat signál z více takových zdrojů, výsledná amplituda může být nakonec vyšší než jedna. A zde nastává problém - Web Audio API negarantuje bezchybné přehrání se signálu o amplitudě vyšší než jedna, při překročení této hranice může dojít ke zkreslení výsledného zvuku.

Provedl jsem experiment, abych chování Web Audio API při překročení této hranice otestoval. Vytvořil jsem jednoduchou stránku, kde jsem k jednomu

[3]Hodnota amplitudy signálu ve Web Audio API je bezrozměrná, nemá dané jednotky.

výstupu (`destination`) postupně připojoval a spouštěl více a více stejných sinusových oscilátorů. Součet jimi generovaných signálů tak ve svém maximu v uzlu `destination` překročil hodnotu jedna. Experiment jsem provedl v prohlížečích *Mozilla Firefox*[4] a *Google Chrome*[5]. Zajímavé zjištění bylo, že v případě Firefoxu překročení hodnoty 1 nečinilo žádné potíže a zkreslení nevznikalo - výsledný zvuk se pouze zesiloval, zůstával však čistý. V případě Chromu však překročení této hranice při přehrávání začalo způsobovat zkreslení signálu a výsledný zvuk byl poškozený.

V případě implementace přehrávače každopádně potřebujeme předejít nežádoucímu zkreslení. K tomu nám může pomoci citlivá aplikace uzlu `GainNode`, kterým příliš silný signál ztlumíme tak, aby jeho amplituda nepřekračovala kritickou hranici.

Další nesmírně důležitou vlastností `GainNode` je možnost plánovat události budoucí. Parametry definující chování uzlů ve Web Audio API jsou totiž standardně reprezentovány objektem třídy `AudioParam`. Nejinak tomu je i v případě hodnoty gainu. Objekt typu `AudioParam` umožňuje jednak nastavení hodnoty parametru okamžitě, navíc však poskytuje též metody pro naplánování změny v budoucnosti a také pro plánování postupné změny v čase (lineární nebo exponenciální) směrem k cílové hodnotě (např. funkce `setValueAtTime()`, `setTargetAtTime()`,...).[8]

Plánování změny hodnoty gainu v průběhu času budeme využívat při vytváření ADSR obálek a simulování přirozeného útlumu nástroje. Jinými slovy bude v našem virtuálním hudebním nástroji za zdrojem zvuku prakticky vždy připojen gain, který bude zvuk tlumit, případně jinak upravovat, tak, jak by tomu bylo u skutečného hudebního nástroje, který se snažíme napodobit.

K obálkám bych ještě poznamenal, že jejich aplikace je mnohdy užitečná i v případě velmi rychlých změn, u kterých by se zdálo, že je změnu možné provést skokově. Skoková změna však může výsledný zvuk poškodit a způsobit v něm jakési nežádoucí „lupnutí“. Chceme-li takovému „lupnutí“ předejít, je vhodné i takový rychlý přechod ošetřit obálkou a provést plynule.

Uvažme jako příklad hudební nástroje mající velmi krátký (zdánlivě skokový) attack a release. U takových nástrojů by se zdálo, že stačí nastavit gain na konstantní hodnotu a zahájení, respektive konec tónu provést pouze zapnutím a následným vypnutím oscilátoru. Takový skokový začátek a konec tónu by v přehrávači mohl způsobit zmíněné „lupnutí“. Proto je vhodnější takový tón implementovat tak, že oscilátor nejprve zapneme a následně plynule zvýšíme hodnotu zesílení gainu na cílovou hodnotu. Analogicky tón ukončíme - nejprve snížíme hodnotu gainu plynule na nulu a teprve poté vypneme zdrojový oscilátor. Plynulé zesílení i zeslabení můžeme provést velmi

[4] testováno v prohlížeči Mozilla Firefox verze 88.0.1 (64-bit) na OS Microsoft Windows 10 Pro, verze 10.0.19041

[5] testováno v prohlížeči Google Chrome verze 90.0.4430.93 (Official Build) (64-bit) na OS Microsoft Windows 10 Pro, verze 10.0.19041

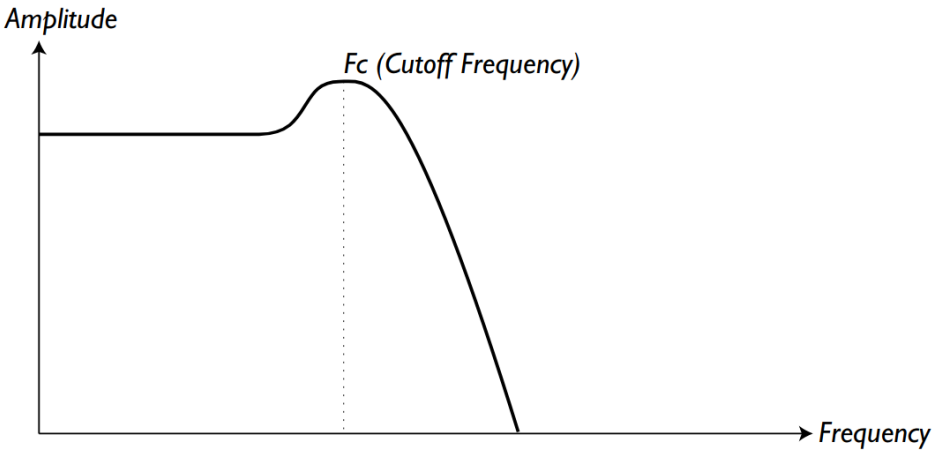
rychle (např. během několika milisekund). Výsledek pak bude znít téměř jako skoková změna, avšak díky plynulosti těchto změn nebude hrozit vznik zmíněného „lupnutí“.

### 3.3.3 Biquad filter node

V kontextu syntézy zvuku je velmi důležitým a v praxi často využívaným efektem tzv. filtr. Ten nám umožňuje filtrovat, respektive utlumit některé části frekvenčního spektra signálu. Ve Web Audio API nám funkcionalitu filtru poskytne uzel zvaný `BiquadFilterNode`.[9]

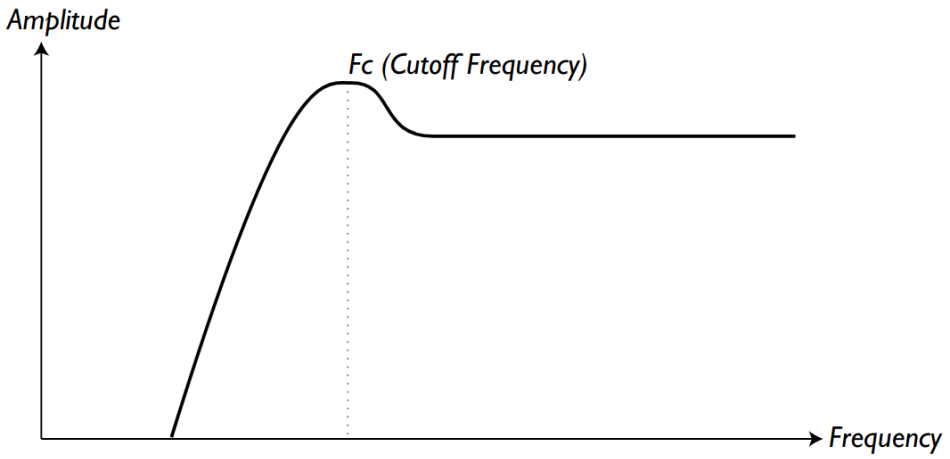
Existují různé druhy filtrů podle toho, jakou část frekvenčního spektra signálu filtrují. `BiquadFilterNode` disponuje funkcionalitou široké škály filtrů. Druh filtru, tedy jakým způsobem bude `BiquadFilterNode` signál filtrovat, můžeme nastavit pomocí parametru `BiquadFilterNode.type`. Příklady základních filtrů, které jsou v hudební praxi hojně využívány, jsou následující.

- **Lowpass filter** propouští nízké frekvence a tlumí vysoké frekvence.



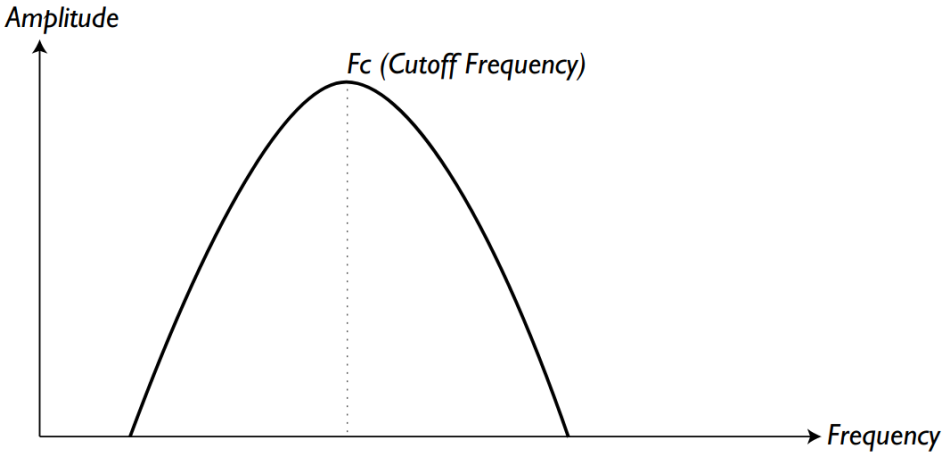

**Obrázek 3.6:** Lowpass filter (zdroj: [4])

- **Highpass filter** propouští vysoké frekvence a tlumí nízké frekvence.



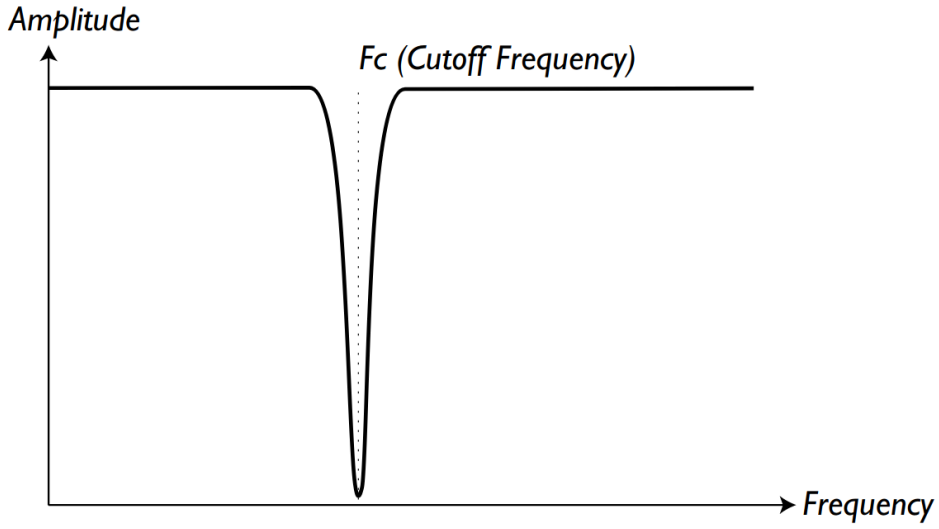

**Obrázek 3.7:** Highpass filter (zdroj: [4])

- **Bandpass filter** propouští frekvence z určitého intervalu, tlumí frekvence nižší a vyšší.




**Obrázek 3.8:** Bandpass filter (zdroj: [4])

- **Notch filter** utlumí frekvence z určitého intervalu, propouští frekvence nižší a vyšší.




**Obrázek 3.9:** Notch filter (zdroj: [4])

Druhů filtrů, které umí `BiquadFilterNode` zastat, existuje více, jako příklad a pro účely této práce však uvedený výčet bohatě postačí.

U každého druhu filtru lze upřesnit další parametry, které jeho funkci definují, jako např. frekvence (hranice intervalu, kde je filtr aktivní), gain (míra útlumu filtrované frekvenční oblasti). Více parametrů již popisovat nebudu, pro naší aplikaci vystačíme se znalostí těch, které jsem již zmínil. V případě zájmu o poznání více parametrů mohu zvídavého čtenáře odkázat na oficiální dokumentaci `BiquadFilterNode`.[9]

# Kapitola 4

# Implementace přehrávače a syntézy hudebních nástrojů

Z předchozích kapitol již máme dostatečné znalosti o standardu MIDI, základních konceptech syntézy zvuku i Web Audio API. Nyní je tedy možné postoupit dále a nabyté poznatky využít k vytvoření jednoduché klientské aplikace, která bude umožňovat přehrávání MIDI souborů pomocí syntetizovaných zvuků hudebních nástrojů.

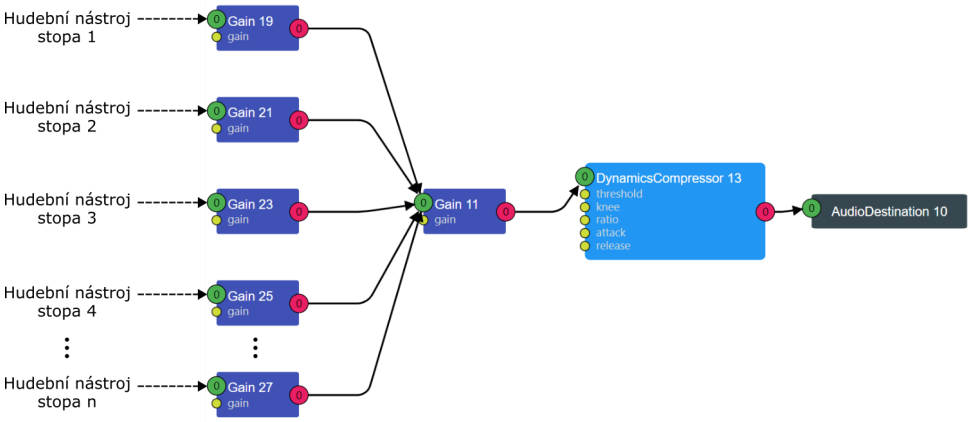
## 4.1 Návrh aplikace

Aplikace, kterou budeme vytvářet, bude jednoduchá webová single-page aplikace, veškerá funkcionalita bude tedy uživateli k dispozici na jedné stránce. Uživateli umožní upload MIDI souboru a jeho následné přehrání.

Po nahrání MIDI souboru bude mít uživatel možnost nastavit, jakým způsobem bude soubor přehrán. Na stránce se k tomuto účelu vypíše tabulka se seznamem přítomných MIDI stop. U každé stopy bude uveden její název[1], možnost volby hudebního nástroje a též možnost úpravy hlasitosti přehrávání stopy.

Jednotlivé stopy k sobě tedy budou mít přiřazené virtuální hudební nástroje. Za každým nástrojem bude připojen `GainNode` sloužící k regulaci hlasitosti přehrávání příslušné stopy. Signál z jednotlivých stop bude následně přiveden na jeden hlavní „master“ `GainNode`, jež bude regulovat hlasitost celého přehrávače.

[1] U stopy v MIDI souboru sice může být uveden její název, často se jako název stopy používá název hudebního nástroje, kterým by stopa měla být přehrána. Standard MIDI však nedefinuje, jak přesně by měl konkrétní hudební nástroj znít, z toho důvodu tento atribut můžeme vnímat pouze jako informativní.




**Obrázek 4.1:** Schéma MIDI přehrávače ve Web Audio API

Master gain bychom již teoreticky mohli připojit k uzlu `destination`, kde by se sloučený signál z celého přehrávače přehrál. Při tomto přístupu však stále hrozí nebezpečí, že do uzlu `destination` přivedeme příliš silný signál, což by mohlo způsobit zkreslení a degradaci výsledného zvuku. Tomu můžeme předejít citlivým nastavením `GainNode`s, které úroveň signálu cestou přehrávačem upravují.

Problém s přehráváním potenciálně příliš silného signálu můžeme zmírnit pomocí `DynamicCompressorNode`. Jedná se o efektový uzel, který slouží ke tlumení nejsilnějších složek signálu v případě, že je jeho amplituda příliš velká a mohla by způsobit degradaci výsledného zvuku.[10]

Při ideálním nastavení (které teoreticky musí s citem připravit uživatel), tedy kompresor není potřeba vůbec, respektive jím procházející signál není nijak upraven. V případě, že by z přehrávače ovšem vycházel příliš silný signál, kompresor utlumí jeho nejsilnější složky tak, aby amplituda výsledného signálu nepřekračovala kritickou hranici pro riziko degradace zvuku při přehrávání. Tento návrh přehrávače by tedy měl umožnit relativně komfortní a bezproblémové užívání přehrávače. Schéma popsaného přehrávače je naznačeno na obrázku 4.1.

### 4.1.1 Objektový návrh

Základní jednotkou, s níž budeme při přehrávání v naší aplikaci pracovat, je jeden tón. Tento přístup odpovídá i struktuře dat v MIDI souboru (jednotlivé stopy obsahují seznam tónů). Pro připomenutí zopakuji, že jeden tón v MIDI souboru s sebou nese čtyři důležité informace: midi notu (výšku tónu), intenzitu tónu, čas začátku a délku trvání.

Naše aplikace bude tón vytvářet pomocí prostředků dostupných ve Web Audio API, tedy pomocí zdrojových uzlů (většinou oscilátorů) a dalších za nimi připojených efektových uzlů. Schémata tónů jednotlivých hudebních nástrojů

mohou být různá, implementace vytvoření tónu pro jednotlivé nástroje bude specifická. Samotné schéma a nízkoúrovňovou práci s Web Audio API ovšem zapouzdříme do objektu tónu a následně již bude možné tóny všech hudebních nástrojů ovládat stejným způsobem, přes stejné rozhraní. V naší aplikaci jsem toto rozhraní definoval třídou `BaseTone`, od které mohou být tóny jednotlivých hudebních nástrojů odděděny.

Vytváření speciální třídy `<InstrumentName>Tone` pro každý hudební nástroj však není nutné. Nástroje, které vystačí se základní funkcionalitou tónu (tj. především jeho plynulé utlumení jednoduchou obálkou), mohou své tóny reprezentovat přímo instancemi obecné třídy `BaseTone`. Implementace speciální třídy pro tóny nových nástrojů je tedy nutná pouze v případě potřeby nějaké specifické funkcionality, která není implementována ve třídě `BaseTone`.

Samotný hudební nástroj, tedy objekt, který slouží k vytváření jednotlivých tónů (kterých je obvykle při přehrávání jedné stopy mnoho), jsem implementoval zvlášť, pro každý hudební nástroj v samostatném objektu - vzdáleně tento přístup může připomínat factory pattern, kde hudební nástroj je „továrna na vytváření tónů“. V mém případě se o factory pattern v pravém slova smyslu nejedná, jelikož každý hudební nástroj má svou vlastní třídu, produkující svůj specificky znějící tón.

Zvolené schéma poskytne jednotné rozhraní pro vytváření, spouštění a zastavování tónů všech nástrojů. Aplikace bude tedy snadno rozšiřitelná. Pro přidání nového hudebního nástroje stačí implementovat příslušnou třídu `ToneFactory` (případně i specifický `Tone`), která umožní tvorbu a řízení tónů nového nástroje.

Další dvě důležité třídy budou `MidiTrackPlayer` a `MidiPlayer`. Jak název napovídá, tyto třídy budou zajišťovat přehrávání MIDI souboru. Přesněji, k objektu `MidiTrackPlayer` je přiřazena jedna MIDI stopa a jedna instance `ToneFactory`, pomocí které umí `MidiTrackPlayer` tuto stopu přehrát, respektive vytvořit odpovídající tóny a naplánovat jejich přehrání.

`MidiPlayer` je již pouze objekt, který v sobě sdružuje více instancí třídy `MidiTrackPlayer`. Tento model je analogický k tomu, že píseň zapsaná v MIDI souboru může sestávat z více stop. `MidiPlayer` pak umožňuje obsluhovat tyto stopy najednou (např. spustit nebo zastavit všechny stopy).

Poslední třídou, kterou zde představím, je `MidiController`. K jeho rozboru se dostaneme podrobněji až v další kapitole této práce. Naši aplikaci pomocí této třídy rozšíříme o podporu externě připojené MIDI klaviatury - kromě přehrávání MIDI souborů tedy bude umožněno hrát na virtuální nástroje „živě“, pomocí připojené MIDI klaviatury nebo kontroleru.

Celé schéma, které jsem právě popsal, je pro přehlednost znázorněno na obrázku 4.2.

### 4.1.2 Načtení vstupu a vytvoření přehrávače

Jak již bylo řečeno dříve, vstupem pro přehrávač je soubor typu SMF (*standard MIDI file*), který sestává z hlavičky s obecnými informacemi a následného seznamu stop s výčtem tónů.[2] Jelikož se jedná o běžný souborový formát, není třeba implementovat vlastní parser, k přečtení SMF souboru jsem využil parser z knihovny *Tone.js*.[3]

Po načtení vstupního souboru dojde k vytvoření přehrávače. Hlavní částí přehrávače je seznam přítomných stop. Ten je implementován pomocí tabulky, kde každý řádek odpovídá jedné stopě. Každý takový řádek obsahuje několik ovládacích prvků, pomocí nichž může uživatel nastavovat parametry přehrávání příslušné stopy. Tento řádek je implementován jako *custom HTML Element*, který je oděděn od elementu `<tr>`.

Nově vytvořený element v sobě sdružuje ovládací prvky nastavující přehrávání stopy, konkrétně seznam virtuálních hudebních nástrojů, z nichž může uživatel jeden zvolit pro přehrávání, a také uživatelem nastavitelný `GainNode`, k němuž je zvolený virtuální nástroj připojen a řídí tak hlasitost přehrávání stopy.

`GainNode`, který je nastavitelný uživatelem, je též implementován jako *custom HTML Element*, v tomto případě oděděný od `<input type="range">` a zapouzdřuje v sobě zmíněný `GainNode`, jehož hodnota zesílení odpovídá vstupní hodnotě nastavené uživatelem. Stejným způsobem je implementován i gain, do něhož je přiveden signál ze všech stop, a kterým se řídí hlasitost celého přehrávače.

Nakonec disponuje přehrávač ještě tlačítkem pro spuštění přehrávání. Kliknutí na toto tlačítko iniciuje vytvoření a naplánování přehrání všech tónů, tedy prakticky vzato zahájí přehrávání. Stejným tlačítkem je následně možné přehrávač i zastavit.

Pro úplnost doplním slovní popis návrhu uživatelského rozhraní screenshotem výsledné aplikace, ten je k vidění na obrázku 4.3.

[2] Soubory typu SMF obvykle poznáme podle přípony `.mid`. Občas však touto příponou bývají chybně označeny i soubory typu RMID (*RIFF-based MIDI File format*), které mohou zapouzdřovat soubor SMF spolu s dalšími dodatečnými daty. Tyto soubory by měly být označeny i jinou příponou, doporučuje se `.rmi`.[11] Náš přehrávač však očekává na vstupu soubor typu SMF, úspěšné přečtení RMID v tuto chvíli není podporováno.

[3] *Tone.js* je open source audio framework, více informací o něm je k nalezení na jeho webové stránce: `https://tonejs.github.io/`

## 4.2 Příklady syntézy hudebních nástrojů

Vycházejíc z popsaného schématu aplikace v minulé sekci, implementace zvuků hudebních nástrojů pomocí Web Audio API je zapouzdřena ve třídách `<InstrumentName>ToneFactory` a případně `<InstrumentName>Tone`.

Implementace vytvoření a zastavení tónu bude pro každý hudební nástroj specifická. Vždy však bude sestávat z vytvoření zdrojových a efektových uzlů, jejich propojení, nastavení potřebných parametrů a spuštění, respektive zastavení ve správný čas.

Než však přejdu k popisu syntézy konkrétních nástrojů, ve stručnosti se pokusím shrnout to, co budou mít všechny hudební nástroje společné.

- **Zdroj zvuku**

  Všechny tóny potřebují nějaký zdroj zvuku, respektive jeden nebo více zdrojových uzlů, které budou generovat akustický signál. Ve většině případů bude tuto roli zastávat `OscillatorNode`, ve speciálních případech však můžeme využít i jiné zdrojové uzly (např. `AudioBufferSourceNode`), o tom ale více až později.

  Nastavení parametru `OscillatorNode.frequency` bude určovat výšku přehrávaného tónu.

- **Modulace, obálky, LFO**

  Většina typů uzlů ve Web Audio API má nastavitelné parametry, které definují chování těchto uzlů. Tyto parametry jsou standardně objekty odděděné od třídy `AudioParam`, které kromě okamžitého nastavení své hodnoty umožňují i plánování nastavení své hodnoty v budoucnosti, případně i plánování postupné změny hodnoty v průběhu času.

  Pomocí této funkcionality budeme moci implementovat analogii k obálkám, případně LFO, které definují proměnlivou modulaci signálu v průběhu času.

  Pro implementaci analogie k obálkám využijeme plánování lineárních nebo exponenciálních změn hodnot parametrů v čase pomocí metod třídy `AudioParam`, konkrétně využijeme metody `setTargetAtTime()` nebo `setValueCurveAtTime()`[4].

  Velmi podobným způsobem jako obálky můžeme implementovat i analogii k LFO - hodnotu parametru v objektu `AudioParam` je totiž možné upravovat i pomocí oscilátoru.

[4] Pro plánování změn hodnoty v čase u třídy `AudioParam` bychom teoreticky mohli též použít metody `linearRampToValueAtTime()` nebo `exponentialRampToValueAtTime()`. Tyto metody však zatím nejsou plně podporovány v prohlížeči Mozilla Firefox. Proto se jejich užití raději vyhneme a budeme používat metody `setTargetAtTime()` nebo `setValueCurveAtTime()`, které pro naše účely poslouží stejně dobře. [12]

Toto lze provést tak, že vytvoříme `OscillatorNode`, jehož výstup však nepřipojíme jako vstup pro jiný `AudioNode`, ale jako vstup pro objekt `AudioParam`. V takové situaci je hodnota parametru v objektu `AudioParam` řízena hodnotou připojeného oscilátoru. Zde bych ještě připomněl fakt, že amplituda signálu generovaného oscilátorem je ve výchozím stavu bezrozměrná „jedna“. Pokud tedy oscilátor připojíme přímo k modulovanému parametru, bude LFO hodnotu tohoto parametru zvyšovat a snižovat právě o jedna. Chceme-li pomocí LFO modulovat o jinou hodnotu, můžeme to provést vložením pomocného `GainNode` mezi oscilátor a modulovaný parametr. Jím můžeme amplitudu řídícího signálu z oscilátoru zvýšit nebo snížit a docílit tak modulace požadované úrovně.

- **Jednotné rozhraní**

  Tóny všech hudebních nástrojů se budou vytvářet skrze jednotné rozhraní, které je definováno v metodě `AbstractToneFactory.createTone()`. Tato metoda přijímá čtyři parametry, které vytvářený tón definují.

  - `midiNote`
    Číslo označující výšku tónu[5].
  - `velocity`
    Intenzita, s níž má být tón zahrán.
  - `startTime`
    Čas začátku tónu. Tento čas je brán vzhledem k času audio kontextu, v němž je tón vytvářen. Pokud je tento čas menší nebo roven hodnotě `AudioContext.currentTime`, tón se začne přehrávat okamžitě, v opačném případě dojde k naplánování budoucího přehrání.
  - `endTime`
    Čas, kdy se má zahájit ukončení (utlumení) tónu. Přesněji řečeno jde čas, kdy má začít fáze *release*, tedy analogie k útlumu tónu při ukončení jeho hraní (např. po puštění klávesy klaviatury). Ukončení tónu může být naplánováno na konkrétní čas vzhledem k času audio kontextu. Tón však může být vytvořen i bez tohoto parametru - v takovém případě tón začne hrát a hraje do té doby, dokud přímo na jeho instanci není zavolána metoda `BaseTone.stop()` nebo dokud se samovolně neutlumí.

  Při vytvoření tónu tedy dojde k vytvoření potřebných uzlů, nastavení parametrů a jeho zahájení v kýžený čas.

  Ukončení tónu je posléze možné, jak jsem již naznačil, dvěma způsoby. První možností je specifikace času ukončení tónu již během jeho vytváření. V druhém případě, kdy čas ukončení tónu předem neznáme, nemusíme jej předem specifikovat a tón ukončit až později zavoláním metody `BaseTone.stop()`.

[5] jednotlivé půltóny mají ve standardu MIDI přiřazený číselný kód

Samotné zastavení pak tedy bude znamenat provedení fáze *release*, tedy typicky utlumení hlasitosti (např. plynulou změnou hodnoty gain), zastavení zdrojových uzlů, případně odstranění referencí na použité uzly, které již nebudou potřeba. [6]

Tímto bych uzavřel výčet obecných konceptů, které jsou společné pro implementaci hudebních nástrojů. V další části práce již přejdeme ke konkrétním příkladům hudebních nástrojů.

### 4.2.1 Snare Drum

Prvním hudebním nástrojem, jehož implementaci představím, je snare drum. Tento bubínek má dva hlavní zdroje zvuku. Prvním zdrojem je blána, která se po úderu paličkou rozechvěje a produkuje tón bubnu. Druhým zdrojem zvuku jsou dráty připevněné na spodní straně bubínku, které při úderu paličkou „chrastí“ a dotváří tak charakteristický zvuk snare bubínku.

První zmíněnou složku zvuku bubínku, zvuk chvějící se blány, jsem záměrně označil slovem tón. Toto označení z hudebního hlediska znamená, že u něj lze rozpoznat výšku a rámci syntézy jej budeme napodobovat pomocí periodického signálu z oscilátoru.

Jako základní signál napodobující zvuk blány bubnu poslouží dobře dobře sinusový oscilátor. Výška tónu snare bubínku se obvykle pohybuje zhruba v intervalu mezi 120Hz a 250Hz[13], frekvenci oscilátoru tedy nastavíme odpovídajícím způsobem, ideálně na hodnotu z tohoto intervalu.

Zajímavým efektem, který se u syntézy bicích nástrojů používá, je pokles výšky tónu. Na začátku je tedy frekvence oscilátoru trochu vyšší a po zahájení tónu frekvence relativně rychle exponenciálně klesá k cílové hodnotě.[14] Frekvence oscilátoru, stejně jako všechny další parametry audio uzlů ve Web Audio API dědí od třídy `AudioParam`, tudíž můžeme tento pokles hravě naplánovat.

Dále je potřeba vytvořit vytvořit obálku pro průběh hlasitosti tónu bubnu. Bubínek má velmi krátký attack, tedy velmi rychle (prakticky okamžitě po svém začátku) dosáhne své maximální hlasitosti. Po nabytí maxima je tón naopak relativně rychle tlumen. Úplný útlum navíc provedeme již během fáze *decay*, bez následujících fází následný *sustain* a *release*, tzn. útlum bubínku nebude závislý na času konce MIDI tónu (resp. na okamžiku puštění klávesy MIDI klaviatury). Útlum je opět exponenciální, což ostatně odpovídá fyzikálnímu popisu tlumení obecného oscilátoru.

Popsanou úpravu amplitudy provedeme pomocí uzlu `GainNode`, který připojíme za oscilátor. Úvodní hodnota gainu nastavíme přímo úměrně k parametru

[6] Smazáním reference na nepoužívané objekty umožníme, aby je mohl výhledově garbage collector odstranit z paměti.

`velocity`, tedy intenzitě tohoto tónu. Ihned od začátku pak na gainu nastavíme exponenciální pokles až k úplnému utlumení bubnu. Když je buben utlumen, můžeme zastavit i gainu předřazený oscilátor.

Druhou složkou zvuku bubnu, kterou budeme chtít napodobit, je již zmíněný zvuk drátů. Ten již z hudebního hlediska není tónem, jde o jakési chrastění u něhož nelze určit výšku.

Abych tento ruch napodobil, vytvořím tzv. *white noise*. To je zvuk, jehož akustický signál je tvořen funkcí s náhodným průběhem. Takový signál zní jako šum bez sluchem určitelné výšky a v našem případě dobře poslouží pro napodobení zvuku drátů snare bubínku.

Jelikož *white noise* není periodický, pro jeho generování nepoužiji oscilátor, ale `AudioBufferSourceNode`. Ten slouží ke generování audio signálu na základě dat uložených v paměti. V našem případě, když chceme generovat *white noise*, budou tato data pole náhodných reálných čísel nabývajících hodnoty z intervalu $\langle -1, 1 \rangle$.

White noise následně bude potřeba utlumit podobným způsobem, jako tomu bylo u tónu bubínku. Nastavení maximální hlasitosti vzhledem k `velocity` a následný útlum zvuku drátů provedeme opět pomocí `GainNode`, a to stejným způsobem, jako tomu bylo u tónu bubínku.

Oba tyto gainy budou nakonec připojeny ke své destinaci, tedy teoreticky buď k `AudioContext.destinationNode` nebo v případě naší aplikace k dalšímu uzlu k dalšímu zpracování zvuku nástroje.

Schéma syntézy tónu snare bubínku přikládám k náhledu na obrázku 4.4.

### 4.2.2 Kick drum

Druhým nástrojem, jehož implementaci popíšu, bude kick drum, česky též označovaný jako *basový buben* nebo *kopák*.

Jelikož se stále jedná o buben, jsou i jeho vlastnosti (a z toho plynoucí implementace) podobné snare bubínku popsanému v předchozí části práce.

Podobně jako v případě snare bubínku tvoří základ zvuku tón blány bubnu. Ta se bude chovat velmi podobně jako tomu bylo v případě snare bubínku, pouze s tím rozdílem, že tón basového bubnu je o poznání nižší, což je způsobeno větším rozměrem bubnu a jeho blány. Obvyklá výška tónu kópáku se uvádí přibližně mezi $\langle 80, 150 \rangle$Hz [13].

Ostatní vlastnosti tónu kopáku budeme uvažovat stejné, jako tomu bylo u snare bubínku, tzn. budeme implementovat pokles výšky tónu (=pokles frekvence signálu) v čase a též budeme gainem připojeným za oscilátor řídit útlum tónu bubnu. Implementace obou těchto modulací signálu bude totožná

s postupem v případě tónu snare bubínku, proto ji již nebudu opakovat a podrobněji popisovat.

Na rozdíl od snare bubínku nemá kopák na své zadní straně připevněny žádné dráty, ani jiná podobná zařízení, která by ke zvuku bubnu přidávaly nějakou další charakteristickou složku. Přesto však můžeme sluchem zhodnotit, že zvuk bubnu obsahuje výraznou složku ruchů. Aby zvuk kopáku zněl věrohodněji, vyhodnotil jsem pokusem jako užitečné tento zvuk též lehce „zašpinit“, tedy dopřidat k periodickému signálu též nějaký ruch.

Jako zdroj ruchu jsem stejně jako u snare bubínku použil white noise (opět generovaný pomoci `AudioBufferSourceNode`, jež jako zdroj čte buffer obsahující náhodné hodnoty). Obdobně, jako v případě snare bubínku, za zdroj ruchu připojím `GainNode`, který bude řídit útlum bubnu.

Ruch u kopáku je však potřeba odlišit od zvuku drátů snare bubínku, který je hodně výrazný, pronikavý a ve svém spektru obsahuje hodně vysokých frekvencí (resp. v jeho spektru jsou všechny frekvence zastoupeny přibližně rovnoměrně, jelikož průběh tohoto signálu je náhodný).

Ruch, který je při syntéze zvuku kopáku užitečný, by měl být méně pronikavý a měl by obsahovat spíše nízké frekvence. K této úpravě využijeme `BiquadFilterNode`, který připojíme za gain modulující hlasitost šumu. `BiquadFilterNode` nám pak poslouží jako low-pass filtr, tedy propustí pouze nízké frekvence a vyšší utlumí.

S těmito dvěma složkami již syntézu ukončím, s tímto schématem již získáme relativně použitelný zvuk pro kick drum. Schéma tónu tohoto bubnu přikládám k náhledu na obrázku 4.5.

### 4.2.3 Drum kit

Speciálním nástrojem, který jsem do aplikace přidal, je též bicí souprava (angl. drum kit). Tento nástroj je specifický tím, že neimplementuje žádný nový zvuk, ale sdružuje zvuky ostatních bicích a perkusních nástrojů.

Doposud jsem popsal, jak lze implementovat kick drum a snare drum jako samostatné nástroje. Pro přehrávání MIDI souborů, popřípadě pro živé hraní pomocí MIDI klaviatury nebo controlleru tento přístup však není moc praktický.

Pokud bychom při přehrávání MIDI souboru přiřadili k jedné stopě jeden tento nástroj, každý tón v této stopě by byl přehrán pouze jako úder na tentýž buben. V případě hraní na MIDI klaviaturu/controller bychom pak při volbě tohoto nástroje stiskem libovolné klávesy přehráli tentýž zvuk.[1]

Ve standardu MIDI proto existuje speciální postup pro implementaci celé bicí sady, resp. sady perkusních nástrojů. Každý tón v MIDI souboru (popř.

událost *note on* při živém hraní) s sebou nese atribut `MidiNote`. Ta v případě melodických nástrojů určuje výšku tónu, jež má být přehrán. V případě perkusních nástrojů však neurčuje výšku tónu, ale odpovídá právě jednomu konkrétnímu bubnu, popř. jinému perkusnímu nástroji.

Díky tomuto přístupu můžeme v MIDI souboru zapsat celý part bicích nástojů do jedné stopy (výška tónu určí druh perkuse), popřípadě můžeme pomocí jedné MIDI klaviatury/controlleru simulovat hru na bicí soupravu (každá klávesa odpovídá jinému druhu perkuse).

Tento koncept jsem tedy implementoval i ve své aplikaci. *Drum kit* je nástroj sdružující v sobě ostatní implementované bicí nástroje. Při volání metody `DrumKitToneFactory.createTone()` se podle argumentu `MIDI note` zavolá vytvoření tónu konkrétního bicího/perkusního nástroje. Při vytváření tónu konkrétního perkusního nástroje je již argument `MidiNote` ignorován, výška tónu jednotlivých bubínků je při každém přehrání stejná.

S ohledem na fakt, že tohoto času mám ve své aplikaci implementovány pouze dva druhy bubnů (snare a kick drum), není tato bicí souprava užitečná více, než pro pouhou technickou demonstraci. Pokud však budou v budoucnu implementovány další druhy perkusních nástrojů, může být tato třída jednoduše rozšířena přidáním těchto perkusí.

### 4.2.4 Brass patch

Jako příklad melodického nástroje jsem zvolil syntetické žestě, tedy nátrubkové nástroje. Jedná se o zvukový rejstřík, který začal být s příchodem syntezátorů hojně využíván v hudební praxi, jelikož kapelám bez vlastní žesťové sekce umožňoval pomocí syntezátoru tento zvuk jistým způsobem zastoupit a nahradit.

Záměrně tento rejstřík označuji obecně jako žestě, ne jako jeden konkrétní žesťový nástroj. Žesťové nástroje jsou si totiž barvou zvuku relativně podobné a principy, které při tvorbě tohoto rejstříku popíšu, lze obecně vztáhnout na všechny nátrubkové nástroje. Druhým, a možná ještě důležitějším důvodem k tomuto obecnému označení je fakt, že je tento rejstřík primárně určen k polyfonnímu hraní a v praxi se využívá nejčastěji k zastoupení celé žesťové sekce. Pod názvem brass je navíc tento zvuk známý a zažitý i mezi hudebníky, nebudu se proto od této terminologie odchylovat.

Syntetické žestě jsem jako příklad zvolil ze dvou hlavních důvodů. Prvním důvodem je jejich relativně velká obliba a využívanost hudebníky v praxi, z toho důvodu jej považuji za reprezentativní příklad syntetizovaného hudebního nástroje.[7] Druhým důvodem volby tohoto zvukového rejstříku je přiměřená složitost jeho syntézy. Na jednu stranu je totiž tento rejstřík již

[7] Jako typický příklad využití syntetizovaných žestů v hudební praxi mohu uvést píseň *Final Countdown* od skupiny *Europe*.

dostatečně komplexní, abych mohl demonstrovat několik zajímavých technik, které jsou též užitečné i při syntéze jiných nástrojů, na druhou stranu je ale stále dostatečně jednoduchý, aby bylo relativně snadné jej reprodukovat a demonstrace zůstala přehledná a názorná.

Nyní tedy přejděme k samotné implementaci. Jako první ze všeho budeme potřebovat oscilátor generující signál barvou odpovídající znějícímu tónu žesťového nástroje (např. trumpety). Teoreticky bychom mohli zkoušet tvořit nějakou vlastní speciální křivku, v případě syntézy žesťových nástrojů to však není potřeba. Frekvenční spektrum žesťových nástrojů je totiž relativně podobné spektru obyčejného pilového oscilátoru (angl. *sawtooth oscillator*).

Prohlédněme si spektrogramy na obrázcích 4.6 a 4.7, kde je zobrazen spektrogram tónu trumpety, a téhož tónu hraného pilovým oscilátorem[8]. Mezi těmito spektrogramy můžeme vidět zjevnou podobnost. V obou případech vidíme relativně rovnoměrný a postupný útlum směrem od nižších k vyšším harmonickým frekvencím. V případě trumpety vidíme, že vyšší harmonické frekvence jsou oproti pilovému oscilátoru více utlumeny. Pokud však za oscilátor připojíme lowpass filtr, kterým vyšší harmonické frekvence omezíme, vytvoříme již zvuk s podobnou spektrální charakteristikou, jako je originální zvuk trumpety. Tento přístup tedy využijeme pro generování základního tónu - začneme s pilovým oscilátorem a za ním připojeným lowpass filtrem.

Dále se musíme zaměřit na imitaci „nasazení“ tónu, tedy krátkého úseku na začátku tónu, který je velmi průrazný, až perkusivní, a pro nátrubkové nástroje je charakteristický.

V případě syntézy žesťů pro imitaci nasazení můžeme využít dvě zajímavé techniky. První z nich je propouštění více vyšších frekvencí přes lowpass filtr v úvodu tónu. To znamená, že u filtru nebudeme mít fixně nastavenou *cutoff frequency* (tj. hranice od které filtr tlumí), ale její hodnotu budeme upravovat v průběhu času obálkou. Konkrétně během fáze attack necháme hodnotu filtru rychle vystoupat a následně během fáze decay opět klesnout na konečnou hodnotu, na které se hodnota filtru ustálí. Tímto krátkým propouštění vysokých frekvencí vytvoříme část tónu, která je zvukově výraznější a průraznější, než zbytek tónu. Celý tento proces nasazení (tedy fáze attack a decay) bude relativně krátký (desítky, max. stovky milisekund) - naším cílem není vytvořit „dlouhý“ přechod, chceme docílit pouze jakéhosi krátkého „zabzučení“ na začátku tónu.

Druhý trik, který při imitaci nasazení můžeme využít, je přidání úmělé nedokonalosti, konkrétně jakéhosi kratičkého rozladění. Pro jeho vytvoření využijeme druhý pilový oscilátor. Tento druhý oscilátor připojíme k filtru stejným způsobem, jako je již připojen první oscilátor. Tento druhý oscilátor však v úvodu tónu lehce rozladíme. Konkrétně nastavíme druhý oscilátor

[8]Data zobrazená ve spektrogramech jsem získal vlastní analýzou nahraného tónu trumpety a pilového oscilátoru v programu Audacity.

tak, aby tón zahájil nepatrně výše, a následně rychle poklesl na „správnou“ hodnotu.

Rozladění druhého oscilátoru by mělo být v praxi velmi malé a pokles na cílovou hodnotu velmi rychlý. Opět totiž nechceme, aby tento efekt byl skutečně slyšet jako rozladění a postupná změna výšky tónu. Při citlivém nastavení tohoto efektu však umíme přidat začátku tónu zvukovou průraznost a jakýsi perkusní efekt, který je pro žesťové nástroje typický.

Nyní již máme téměř připraven použitelný tón žesťů - máme hotové charakteristické nasazení tónu, stejně jako použitelný následný průběh tónu. Posledním vylepšením, které můžeme aplikovat, a které zde popíšu, je přidání nějakých nedokonalostí, či zpestřujících změn do znějícího tónu.

Uvažme, že tón žesťových nástrojů může znít dlouho. A v takovém případě může znít velmi nepřirozeně to, že je signál produkovaný oscilátory perfektně periodický a zní tudíž stále stejně. Takto perfektně periodický tón při hře na akustické nástroje v praxi nikdy nepotkáme. Jednak jej samotní hráči mohou cíleně modulovat pomocí technik *vibrato*[9] nebo *tremolo*[10], dále se v akustickém zvuku vyskytují nedokonalosti, které narušují periodicitu zvuku a činí jej „živější“.

Máme v zásadě dvě techniky, jak můžeme snadno takovýto syntetický signál „oživit“. První možnost je využití více oscilátorů, které jsou mezi sebou nepatrně rozladěny. Druhou možností je využít obálky nebo LFO pro modulaci některých nastavených parametrů v čase a vytvořit tak kýženou změnu zvuku v čase a z toho plynoucí „oživení“ výsledného zvuku. Pomocí LFO můžeme například modulací frekvence oscilátoru simulovat efekt *vibrato*, případně modulací amplitudy simulovat efekt *tremolo*. Tyto postupy můžeme též libovolně kombinovat a záleží již na vkusu a potřebě pro konkrétní aplikaci, jaké zpestřující efekty při syntéze využijeme.

Já ve svém příkladu využiji faktu, že již mám zapojené ve schematu dva oscilátory, první z nich nechám generovat signál o konstantní frekvenci a ke druhému připojím LFO, jež bude modulovat jeho frekvenci v čase a simulovat tak efekt *vibrato*.

Nyní jsem již popsal všechny techniky, které jsem využil k demonstraci syntézy žesťů. Zbytek schématu již bude stejný, jako tomu bylo v případě předchozích nástrojů. Za lowpass filtrem bude připojen gain, který řídí hlasitost tohoto zvuku na základě obdrženého parametru `velocity`. Pro úplnost opět přikládám schema tohoto syntezátoru, k náhledu je na obrázku 4.8

[9] periodická úprava výšky tónu v čase
[10] periodická úprava intenzity tónu v čase

## 4.3 Implementační poznámky

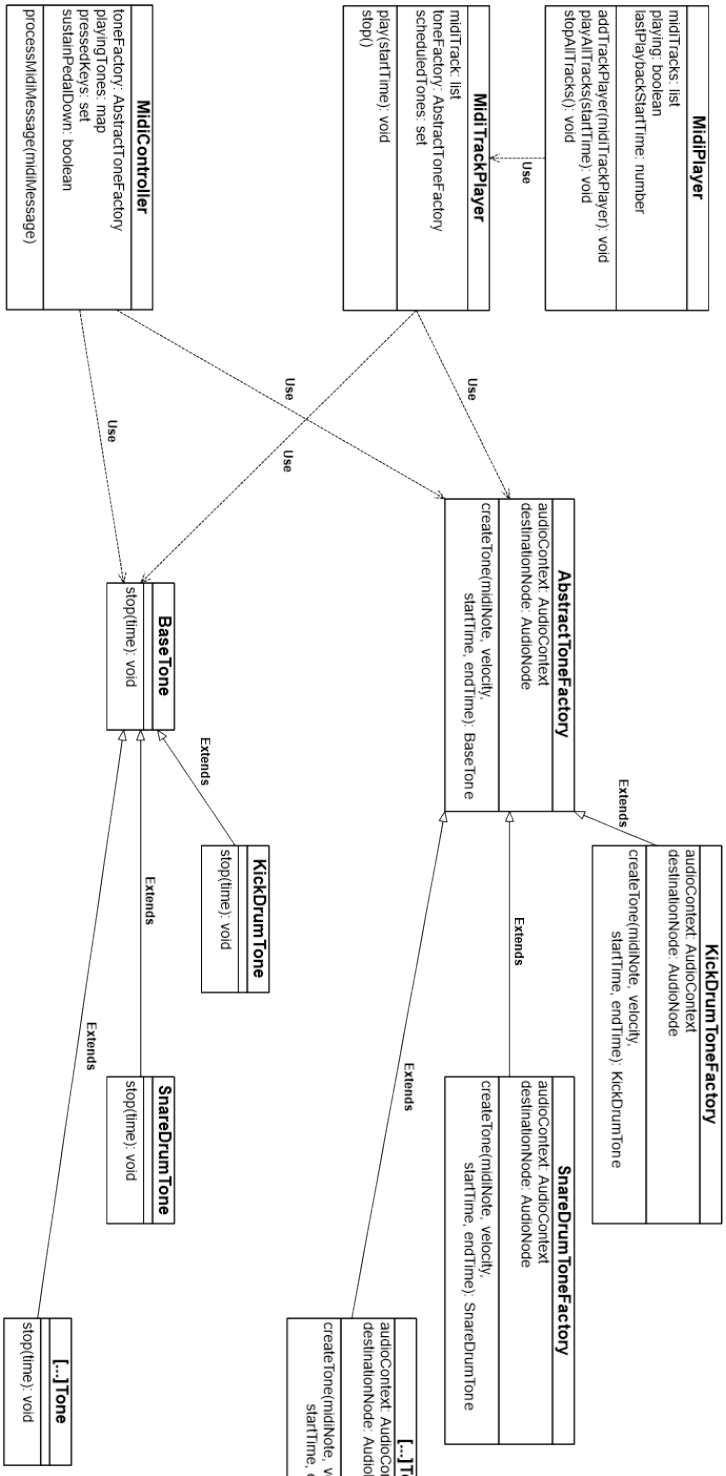
Během popisu příkladů implementovaných nástrojů jste si mohli povšimnout, že ve většině případů neuvádím přesné nastavení a hodnoty konkrétních parametrů použité při syntéze.

Rozhodl jsem se psát takto obecně z důvodu, že „ideální“ hodnoty nastavení těchto parametrů je ve většině případů velmi obtížné vědecky určit – v rámci této práce jsem finální hodnotu většiny parametrů určil na základě zkoušení různých možností a subjektivního posouzení, jaké hodnoty zní nejlépe a vedou k nejlepšímu výslednému zvuku.[11]

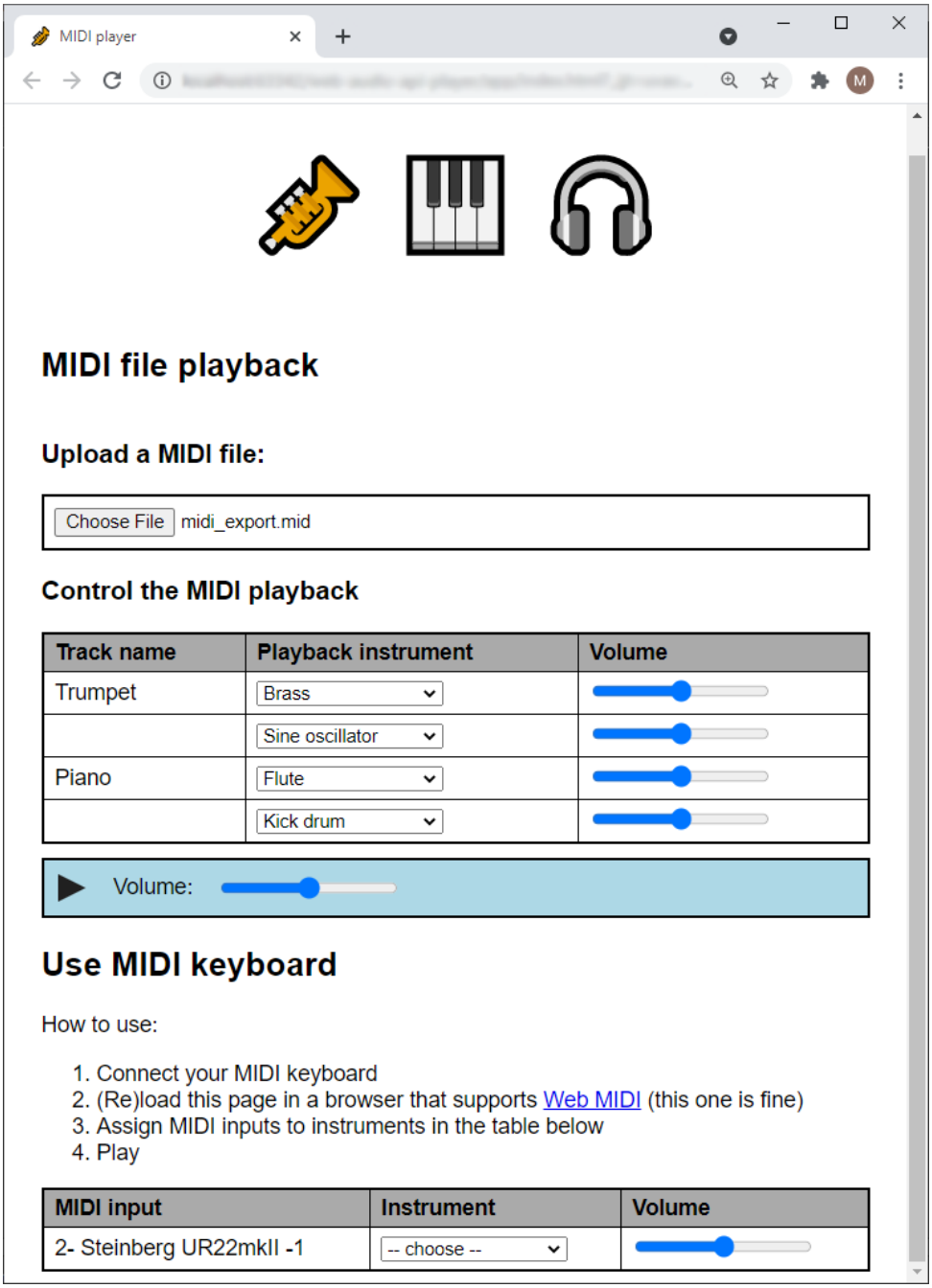
Zároveň ani není cílem této práce detailně rozebírat a analyzovat principy a výsledky samotné syntézy zvuku v závislosti na konfiguraci syntezátoru.

Smysl této práce vidím naopak v představení jakéhosi obecného návodu, jak lze běžné postupy používané při syntéze zvuku implementovat pomocí Web Audio API. Popis implementace je tedy veden právě v tomto duchu – nejedná se o představení jediného nebo ideálního řešení. Mou snahou je však poskytnout čtenáři dostatečný nadhled nad problematikou, aby byl nejen schopen mnou uvedené příklady sám reprodukovat a přizpůsobit svým potřebám, ale též s využitím popsaných technik vytvořit sám prakticky libovolný jiný hudební nástroj, respektive napodobit schéma a konfiguraci libovolného existujícího syntezátoru.

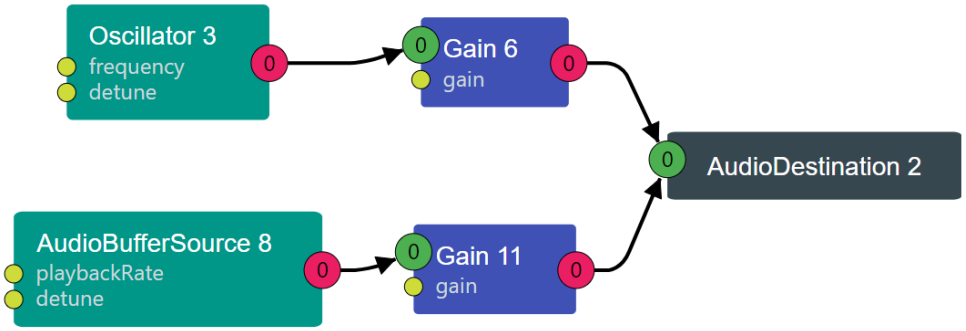
[11] Přesná konfigurace použitá k syntéze popisovaných příkladů hudebních nástrojů je k nalezení ve zdrojových kódech, které jsou přílohou této práce.



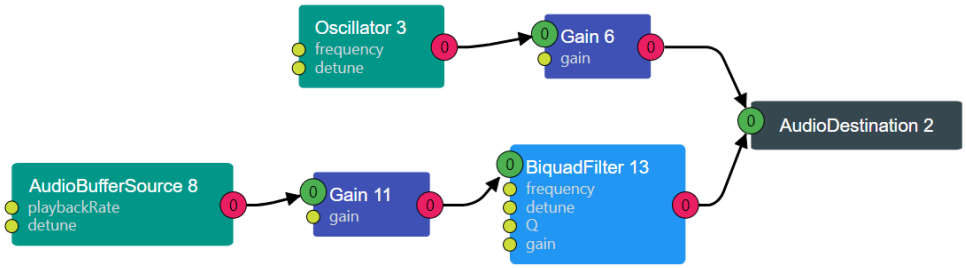

**Obrázek 4.2:** Objektový návrh aplikace



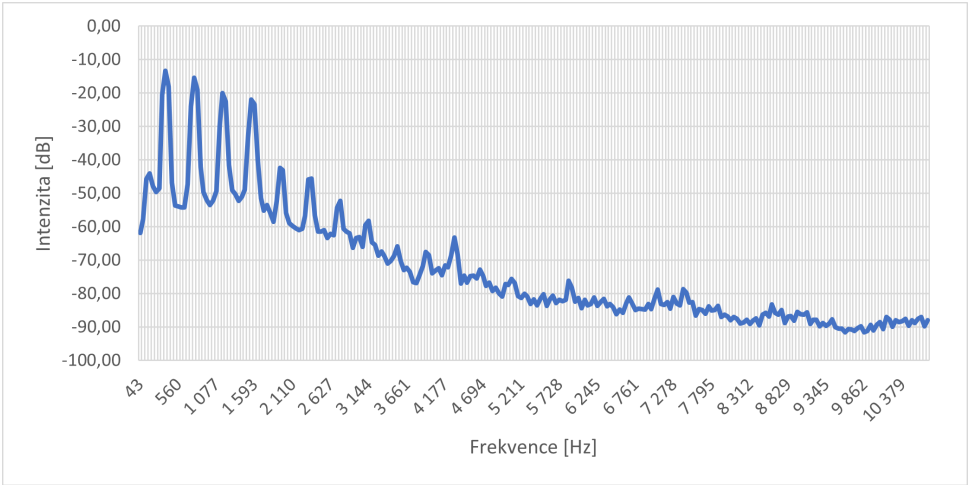

**Obrázek 4.3:** Screenshot aplikace



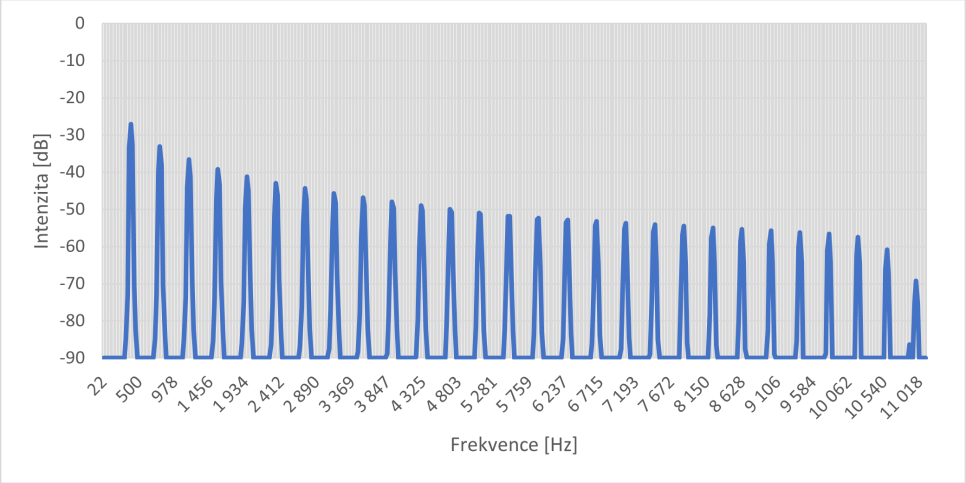

**Obrázek 4.4:** Schéma implementace - snare drum



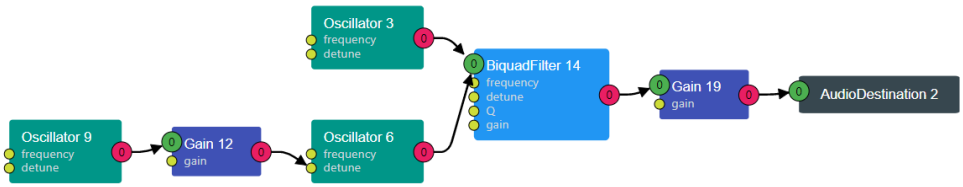

**Obrázek 4.5:** Schéma implementace - kick drum




**Obrázek 4.6:** Spektrogram tónu trumpety (tón G3)




**Obrázek 4.7:** Spektrogram tónu pilového oscilátoru (tón G3)




**Obrázek 4.8:** Schéma implementace - brass patch

# Kapitola 5

# Rozšíření aplikace o Web MIDI API

Další funkcionalita, kterou bude aplikace disponovat, je možnost připojení MIDI klaviatury a „živého“ hraní uživatele. Tuto funkcionalitu je možné implementovat pomocí Web MIDI API, tedy rozhraní sloužícího právě k připojení MIDI zařízení k počítači a zpracování z nich zasílaných MIDI zpráv ve webovém prohlížeči.

Ve své aplikaci tedy vytvořím sekci, která vypíše všechna MIDI zařízení, která jsou připojena k počítači.[1] Uživatel následně k těmto zařízením může přiřazovat hudební nástroje a upravovat jejich hlasitost, obdobně jako v případě stop z MIDI souboru. Aplikace pak na základě zpráv z MIDI klaviatury bude přehrávat tóny příslušného nástroje a uživatel bude moct takto „živě“ hrát na zvolený hudební nástroj.

## 5.1 Zpracování MIDI zpráv

MIDI klaviatury, popřípadě kontrolery, fungují na principu zasílání MIDI zpráv. Každý úkon, který na klaviatuře/kontroleru uživatel (hudebník) provede, způsobí vytvoření a vyslání MIDI zprávy, která tento úkon popisuje. MIDI zpráva je následně zpracována (např. v naší aplikaci) a provede odpovídající úkon s hudebním nástrojem (např. přehraje tón, upraví nějaký nastavující parametr,...).

Jedna MIDI zpráva sestává ze dvou částí: [2]

- **Status byte**

  Určuje typ události (např. stisk klávesy), případně kanál (angl. MIDI channel - některé zprávy mohou být logicky rozděleny do více kanálů, takto lze např. odlišit zprávy pro ovládání různých hudebních nástrojů).

[1] Nalezení a výpis se provede při načtení stránky. Pokud uživatel připojí MIDI zařízení k počítači později, je potřeba stránku načíst znovu.

- **Data bytes**

  Tato část zprávy v sobě nese upřesňující parametry k dané události. Její význam je specifický pro každý typ události. (Např. ve zprávě typu *note on* (stisknutí klávesy klaviatury), pak v datové části zprávy je uvedena výška tónu a intenzita stisku klávesy.

V případě naší aplikace se o zpracování MIDI zpráv bude starat instance třídy `MidiController`. Každému připojenému MIDI zařízení bude náležet jedna instance této třídy, která bude zpracovávat jím vysílané zprávy. `MidiController` v sobě zároveň bude mít specifikovaný hudební nástroj, respektive instanci třídy `ToneFactory`, jež bude na základě příchozích MIDI zpráv produkovat a přehrávat odpovídající tóny.

Při svém návrhu jsem se rozhodl neřešit dělení zpráv dle MIDI kanálu - v naší aplikaci budeme mít totiž vždy jeden MIDI controller pro jedno vstupní MIDI zařízení, a ten bude zpracovávat zprávy z tohoto zařízení bez ohledu na kanál. V případě potřeby je možné aplikaci upravit a rozšířit tak, aby se při zpracování MIDI zprávy kanál zohlednil, nicméně pro naše potřeby jsem k tomu neviděl důvod. Nyní tedy uživatel může ke každému připojenému MIDI zařízení zvolit jeden hudební nástroj, který bude zprávami z tohoto zařízení ovládat.

Pro samotné ovládání hudebního nástroje budeme potřebovat zpracovávat a realizovat tři typy událostí:

- `Note On`

  Událost představující stisk klávesy na klaviatuře. V datové části zprávy je specifikována výška tónu a jeho intenzita. Při zpracování této události vytvořit a začít přehrávat příslušný tón. Referenci na objekt tohoto tónu si musíme následně uložit, abychom jej mohli později zastavit. Čas ukončení tónu totiž není předem znám.

- `Note Off`

  Událost pro puštění klávesy na klaviatuře iniciuje zastavení hrajícího tónu.[2] Výjimkou je však případ, kdy je uživatelem sešlápnutý sustain pedál, v takové situaci příslušný tón nezastavujeme, necháme jej znít dále a zastavíme jej až v okamžiku puštění pedálu.

- `Control Change` - `Damper Pedal on/off`

  Control Change je typ zprávy, který se využívá při manipulaci s různými doplňkovými ovládacími prvky MIDI klaviatury/kontroleru. Těmito prvky mohou být např. různé pedály, fadery či tlačítka. V datové oblasti

[2] Dle standardu MIDI je možné puštění klávesy klaviatury též reprezentovat pomocí zprávy typu `Note On` s hodnotou intenzity rovnou nule. Využití tohoto přístupu není v praxi moc časté, nicméně může být využito, proto v naší aplikaci musíme tento speciální případ ošetřit a interpretovat správně[2].

je pak specifikováno kódem, jaký ovládací prvek byl použit, a na jakou hodnotu byl nastaven. Pro naše potřeby je v tomto směru důležitý sustain pedál, který může být sešlápnut nebo puštěn.

S příchodem zprávy informující o změně stavu pedálu tedy jednak aktuální stav uložíme, abychom věděli, zda máme při zprávě typu `Note Off` tóny tlumit, či nikoliv. Dále, pokud příchozí zpráva představuje puštění sustain pedálu, musíme zastavit všechny znějící tóny, které již nemají stisknutou odpovídající klávesu klaviatury.

S těmito třemi typy událostí jsme schopni implementovat a realizovat plnohodnotné hraní na virtuální hudební nástroj pomocí MIDI klaviatury. Pro realizaci správného spouštění a především zastavování tónu je nezbytné si v paměti ukládat informace o tom, které klávesy jsou stisknuty a též reference na znějící tóny. Bez těchto dat by nebylo možné zastavovat správné tóny při puštění kláves nebo sustain pedálu. Kromě výše zmíněných základních operací je pak ještě potřeba ošetřit některé speciální případy, jako například pokud tón hraje při sešlápnutém pedálu a přijde pro něj nová událost `Note On`, musím starý zastavit, aby nehrály dva stejné tóny současně, apod.

Takto jednoduše je tedy možné implementovat možnost připojení MIDI klaviatury či kontroleru a využít je k „živému“ hraní na libovolný zvolený virtuální hudební nástroj.

# Kapitola 6

# Hodnocení popsaného řešení

Nyní jsme již seznámeni s návrhem a možnou implementací celé aplikace sloužící pro přehrávání MIDI pomocí zvuků hudebních nástrojů syntetizovaných pomocí Web Audio API.

Nyní samotný technický popis doplním o hodnocení tohoto řešení, zhodnocení dosažených výsledků a porovnání s jinými možnými přístupy k řešení tohoto problému.

## 6.1 Samplování vs. syntéza

Již v úvodních částech této práce jsem zmínil, že existují dva základní principy využívané k tvorbě zvuků virtuálních hudebních nástrojů, samplování a syntéza. Samplování je založeno na principu přehrávání nahraných samplů (vzorků) skutečných hudebních nástrojů, zatímco syntéza je založena na generování zvuků ryze umělých.

Pro napodobování zvuků akustických (či jiných existujících nástrojů) se postupně využívaly oba tyto přístupy. Syntéza měla historicky velkou výhodu ve své implementační jednoduchosti. Syntezátor, vágně řečeno, není nic víc než množina několika funkčních bloků, které slouží ke generování, popř. upravování signálu. Technicky vzato jsou tyto bloky relativně jednoduché. Ještě předtím, než se v hudbě začaly používat digitální technologie, bylo možné tyto bloky implementovat pomocí samostatných elektronických obvodů. Později, s příchodem počítačů, nebyl problém tento koncept simulovat i v digitální podobě.

Hudebníkům příchod syntezátorů přinesl velkou volnost a obrovskou škálu možností, jak jednotlivé funkční bloky propojit, nastavit, a jaký výsledný zvuk takto produkovat. Mimo vytváření zvuků kompletně nových se tedy syntezátory začaly využívat i pro napodobování nástrojů existujících. Vel-

kou výhodou syntezátoru tedy bylo, že umožňuje při dostatečné kreativitě napodobit více méně jakýkoliv zvuk, tedy i jakýkoliv hudební nástroj.

Na druhou stranu víme, že zvuky akustických hudebních nástrojů jsou v praxi velmi složité a precizně je matematicky popsat a následně modelovat je téměř nemožné.

V tomto směru přináší samplování obrovskou výhodu, a sice že nemusíme chování napodobovaného nástroje matematicky popisovat a aproximovat, jelikož pouze přehráváme jeho nahrávku, která je ze své podstaty autentická. U samplovaného nástroje určuje jeho kvalitu především množství a kvalita použitých samplů.

Zkusím tedy toto srovnání stručně shrnout. Hlavními výhodami syntézy jsou zaprvé svoboda v množství druhů, které můžeme vytvářet, a zadruhé její paměťová nenáročnost. Samplování je oproti tomu omezené pouze na existující nástroje, od kterých potřebujeme mít nahrané samply. Ty však samplování umožňuje napodobit typicky věrohodněji, protože matematické modelování jejich zvuku je příliš složité. Z toho důvodu se též k napodobování akustických nástrojů v hudební praxi častěji využívá právě samplování. Druhou nevýhodou samplování, i když v současné době již ne tak relevantní, je jeho větší paměťová náročnost.

## 6.2 Použitelnost a náročnost našeho řešení

Jednou z hlavních výhod klientských aplikací, které běží ve webovém prohlížeči, je jejich platformová nezávislost. Jinými slovy řečeno, v případě vývoje nativních aplikací je potřeba přizpůsobit (nebo dokonce vyvíjet zvlášť) různé varianty aplikace určené pro různé platformy. V případě klientských webových aplikací tento problém prakticky neřešíme, jelikož jej za nás řeší webový prohlížeč, který naši (jednou napsanou) aplikaci interpretuje a vykonává vždy stejně, nezávisle na platformě, na které běží.

Při vývoji webové aplikace musíme pouze dávat pozor na to, zda webové prohlížeče, na kterých může být naše aplikace používána, podporují všechny technologie, které jsme při vývoji aplikace využili.

Konkrétně v případě našeho přehrávače a syntézy hudebních nástrojů není s podporou prohlížečů zásadní problém. Pro přehrávání a syntézu zvuků využíváme Web Audio API, které je prakticky na všech moderních prohlížečích podporováno. Z běžně používaných prohlížečů, které jsou uvedeny na webu *caniuse.com*, není Web Audio API podporováno pouze v prohlížečích Internet Explorer a Opera Mini.[1]

[1] Informace byla získána 29. 4. z webu *caniuse.com*. Kompletní a aktualizovaný přehled lze dohledat na url `https://caniuse.com/?search=web%20audio%20api`

Přes všeobecnou dobrou podporu Web Audio API najdeme však i některé funkce, které nemají podporu u některého z významných prohlížečů. Naštěstí jsem však nebyl nucen žádnou takovou problematickou funkci využít, přehrávač i syntetizované zvuky by tedy měly fungovat ve všech významných moderních prohlížečích.

Problém s podporou nastal pouze v případě rozšíření aplikace o Web MIDI, které umožňuje připojení MIDI klaviatury pro „živé“ hraní. Web MIDI API totiž bohužel není podporováno tak široce, jako Web Audio API. Z významných prohlížečů je podporováno např. v Google Chrome, Edge, či Opera, podpora naopak chybí v případě Firefoxu či Safari.[2]

Vedle platformové nezávislosti je další výhodou aplikace, jak jsem již dříve zmínil, její malá datová náročnost. Jelikož zvuky hudebních nástrojů syntetizuji, není potřeba pro běh aplikace stahovat zvukové samply, což činí aplikaci datově velmi úspornou.

[2]Informace byla získána 29. 4. z webu *caniuse.com*. Kompletní a aktualizovaný přehled lze dohledat na url `https://caniuse.com/?search=web%20midi%20api`

# Kapitola 7

# Závěr

Cílem této práce bylo navrhnout a implementovat klientskou webovou aplikaci pro přehrávání MIDI s využitím zvuků virtuálních hudebních nástrojů syntetizovaných pomocí Web Audio API. Navíc jsem měl uvážit a případně rozšířit tuto aplikaci o možnost připojení externí MIDI klaviatury a „živého“ hraní na tyto virtuální hudební nástroje.

Těchto cílů se podařilo dosáhnout a zadání práce tak splnit. Podařilo se implementovat přehrávač, několik příkladů virtuálních hudebních nástrojů, i rozšíření aplikace o možnost připojení MIDI klaviatury pro „živé“ hraní. Aplikace disponuje kompletní požadovanou funkcionalitou a zároveň je navržena tak, aby bylo možné ji snadno rozšířit o další virtuální hudební nástroje.

Příklady virtuálních hudebních nástrojů, které jsem implementoval, tvoří zvuk vzhledem k zadání této práce synteticky. V tomto smyslu se podařilo napodobit běžné principy a schémata využívané u syntezátorů běžných v hudební praxi. V případě rozšiřování aplikace o další virtuální hudební nástroje však není nutné držet se pouze syntézy, samozřejmě je možné implementovat i nástroje samplované, které při napodobování zvuku existujících nástrojů mohou dosahovat lepších výsledků.

Přínos této práce vidím především v demonstraci možností Web Audio API jako zajímavého nástroje pro práci se zvukem a tvorbu platformově nezávislých multimediálních aplikací. Jako takové umožňuje napodobit schéma prakticky libovolného digitálního hudebního nástroje a realizovat jej ve webovém prohlížeči.

# Příloha A

# Literatura


[1] "Standard MIDI Files (SMF) Specification." https://www.midi.org/specifications-old/item/standard-midi-files-smf. Získáno: 27. 2. 2021.

[2] "Summary of midi 1.0 messages." https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message. Získáno: 27. 2. 2021.

[3] "31UCZ 3. cvičení: Aditivní syntéza, časové obálky." http://amber.feld.cvut.cz/vyu/zzs/zzs3/zzs3.htm. Získáno: 28. 2. 2021.

[4] Clavia DMI AB, 1996, *Nord Lead virtual analog - owners manual.*

[5] "MDN web docs - Web Audio API." https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API. Získáno: 28. 2. 2021.

[6] "Waveform." https://en.wikipedia.org/wiki/Waveform#/media/File:Waveforms.svg. Wikipedia, Wikimedia Foundation. Získáno: 3. 2. 2021.

[7] "MDN web docs - Oscillator Node." https://developer.mozilla.org/en-US/docs/Web/API/OscillatorNode. Získáno: 6. 3. 2021.

[8] "MDN web docs - Gain Node." https://developer.mozilla.org/en-US/docs/Web/API/GainNode. Získáno: 6. 3. 2021.

[9] "MDN web docs - Biquad Filter Node." https://developer.mozilla.org/en-US/docs/Web/API/BiquadFilterNode. Získáno: 4. 4. 2021.

[10] "MDN web docs - Dynamic Compressor Node." https://developer.mozilla.org/en-US/docs/Web/API/DynamicsCompressorNode. Získáno: 10. 5. 2021.

[11] "Sustainability of Digital Formats: Planning for Library of Congress Collections - RIFF-based MIDI File Format." https://www.loc.gov/

preservation/digital/formats/fdd/fdd000120.shtml. Získáno: 15. 5. 2021.

[12] "MDN web docs - Audio Param." https://developer.mozilla.org/en-US/docs/Web/API/AudioParam. Získáno: 22. 4. 2021.

[13] F. Koulakos, "Audio Frequencies - Percussion Frequencies." https://www.musical-u.com/learn/percussion-frequencies-part-1-drums/. Získáno: 23. 4. 2021.

[14] D. Sokolovskiy, "3 ways to make a kick drum." https://dsokolovskiy.com/blog/all/kick-synthesis/. Získáno: 22. 4. 2021.

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šach** Jméno: **Marek** Osobní číslo: **457109**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Přehrávání MIDI pomocí Web Audio API**

Název diplomové práce anglicky:

**MIDI playback using Web Audio API Synth**

Pokyny pro vypracování:

Seznamte se s prokolem MIDI a souborovým formátem SMF/MID. Dále nastudujte koncepty digitální syntézy zvuku v kontextu Web Audio
API (OscillatorNode, GainNode, BiquadFilterNode).
S pomocí klientských webových rozhraní naimplementujte několik základních 'hudebních nástrojů' (alespoň tři - melodický, kick drum, snare drum). Sestavte dále webovou aplikaci, která bude načítat SMF soubor a v reálném čase jej přehrávat pomocí syntézy zvuku díky výše uvedeným nástrojům. Vytvořte UI pro mapování jednotlivých MIDI kanálů na dostupné nástroje.
Porovnejte vzniklý přehrávač s tradičními řešeními, postavenými na samplování hudebních nástrojů. Diskutujte hardwarovou náročnost obou řešení a kompatibilitu napříč webovými prohlížeči. Uvažte možnosti rozšíření této aplikace o Web MIDI.

Seznam doporučené literatury:

https://www.midi.org/specifications-old/item/standard-midi-files-smf
https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
Pejrolo, A. and Metcalfe, Scott B.: Creating Sounds from Scratch, Oxford Univeristy Press 2017, ISBN 978-0199921874

Jméno a pracoviště vedoucí(ho) diplomové práce:

**RNDr. Ondřej Žára, Katedra počítačové grafiky a interakce**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **15.02.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **19.02.2023**

| RNDr. Ondřej Žára | | prof. Mgr. Petr Páta, Ph.D. |
|---|---|---|
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | podpis děkana(ky) |

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ Datum převzetí zadání

_____ Podpis studenta


© ČVUT v Praze, Design: ČVUT v Praze, VIC