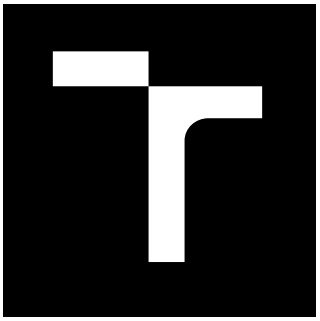
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2024 Jakub Neděla



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

FRAMEWORK PRO INTERAKTIVNÍ WEBOVÉ UČEBNICE HUDEBNÍ TEORIE

FRAMEWORK FOR INTERACTIVE WEB TEXTBOOKS OF MUSIC THEORY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jakub Neděla**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Lieskovan**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

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

Student: Jakub Neděla ***ID:*** 240186
Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Framework pro interaktivní webové učebnice hudební teorie

POKYNY PRO VYPRACOVÁNÍ:

Bakalářská práce má za cíl vývoj interaktivní učebnice hudební teorie. Ta bude vytvořena pomocí frameworku založeného na moderních webových technologiích. Vytvořená interaktivní učebnice bude studentům umožňovat lépe porozumět hudební teorii a zlepšit jejich hudební znalosti a dovednosti. Software bude navržen tak, aby bylo jednoduché jej do budoucna jednoduše implementovat do nových ale i stávajících webových frameworků a CMS. Součástí práce bude návod pro uživatele a vývojáře, kteří by pomocí frameworku chtěli vytvořit interaktivní učebnici. Cílem bakalářské práce je analýza současného stavu, nalezení vhodných nástrojů a poznatky přetransformovat do funkčního řešení, kdy výsledkem bude funkční kapitola interaktivní učebnice hudební teorie.

DOPORUČENÁ LITERATURA:

[1] FRISBIE, Matt. Professional JavaScript for Web Developers. John Wiley & Sons, 2019.
[2] ZENKL, Luděk. ABC hudební nauky. Editio Barenreiter, 2014.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 28.5.2024

Vedoucí práce: Ing. Tomáš Lieskovan

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

ABSTRAKT

Hudební notace hraje klíčovou roli ve zpřístupnění a interpretaci hudebního obsahu a zároveň slouží jako prostředek komunikace mezi hudebníky a interprety. Tato práce zkoumá existující frameworky pro zobrazování notové osnovy a přehrávání zvuků na webových stránkách. Cílem práce je navrhnout a implementovat řešení umožňující zobrazování a přehrávání notových záznamů. Pro dosažení tohoto cíle jsou z rozebrány možnosti různých knihoven. Pro realizaci programu pro webové stránek byly vybrány frameworky VexFlow a Tone.js. Výsledkem práce je vlastní softwarový nástroj, který umožňuje zobrazování a přehrávání notových záznamů na webových stránkách s cílem podpořit lepší porozumění hudební notace.

KLÍČOVÁ SLOVA

framework, hudební notace, přehrávání, tone.js, vexflow

ABSTRACT

Musical notation plays a crucial role in accessing and interpreting musical content while also serving as a means of communication between musicians and performers. This work explores existing frameworks for displaying musical scores and playing sounds on web pages. The aim of the project is to design and implement a solution enabling the display and playback of musical scores. To achieve this goal, various library options are examined. For the implementation of the program for web pages, the VexFlow and Tone.js frameworks have been selected. The result of the work is a proprietary software tool that allows the display and playback of musical scores on web pages with the aim of promoting a better understanding of musical notation.

KEYWORDS

framework, music notation, playback, tone.js, vexflow

NEDĚLA, Jakub. *Framework pro interaktivní webové učebnice hubební teorie*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023. Vedoucí práce: Ing. Tomáš Lieskovan

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Jakub Neděla
VUT ID autora:	240186
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Framework pro interaktivní webové učebnice hubební teorie

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

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské panu Ing. Tomáši Lieskovanovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Hudba a programování jsou na první pohled často vnímány jako dvě zcela odlišné oblasti. Hudba je tradičně spojována s uměleckým vyjádřením emocí, estetiky a tvořivosti, zatímco programování je obvykle spojováno s logikou, strukturou a technickými dovednostmi. S rozvojem technologií a digitalizace však dochází k zajímavému propojení těchto dvou oblastí, což otevírá nové možnosti pro tvorbu, prezentaci a analýzu hudby.

Cílem práce je prozkoumat možnosti různých frameworků nebo volně dostupných knihoven a následný vývoj vlastního softwarového nástroje, který umožní pomocí vhodné kombinace z dříve popsaných frameworků zobrazovat notové záznamy, zvýrazňovat je v čase a přehrávat je.

Bakalářská práce je členěna do pěti kapitol.

V první kapitole je popsán vývoj notace a dále je představena základní hudební teorie, která je nezbytná pro porozumění následujícím částem práce. Jsou popsány základy programovacích jazyků, jako jsou HTML a JavaScript. Jsou zde popsány také data formáty, ve kterých lze hudební notaci předávat.

V druhé kapitole jsou uvedeny komerční programy zabývající se zobrazováním a přehráváním hudební notace. Kapitola je dále zaměřena na různé knihovny, pomocí kterých lze zobrazovat notové osnovy na webových stránkách a na frameworky věnující se přehrávání nebo generování zvuků. Na konci kapitoly je popsán MIDI standard.

Kapitola realizace se věnuje samotnému vývoji aplikace. Nejdříve je zde popsán výběr vhodných frameworků včetně důvodů, proč byly vybrány. Dále je zde popsána struktura souborů projektu a obsah těchto jednotlivých souborů.

Ve čtvrté kapitole je uvedeno, jak je možné aplikaci implementovat do webové stránky a příklady zpracované z učebnice hudební teorie.

Poslední kapitola popisuje limitace spojené s aplikací a popisuje jaké problémy by mohl řešit další vývoj.

1 Úvod do problematiky

Kapitola úvod do problematiky je rozdělena na tři části, kde v první z nich je popsána hudební teorie, v druhé jsou popsány základy programovacích jazyků HTML a JavaScript a v poslední části jsou uvedeny příklady data formátu v kterých lze hudební notaci z programátorského hlediska předávat.

1.1 Hudební teorie

Hudební teorie zahrnuje studium struktur a principů, které jsou základem hudební kompozice a interpretace. Následující podkapitoly jsou zaměřeny na několik základních komponent hudební teorie, které jsou nezbytné pro pochopení hudební notace jako takové a správné provádění hudebních skladeb.

1.1.1 Notace

Při pohledu na první formy hudební notace se může zdát, že jednotlivé záznamy jsou velmi nejednoznačné a snadno by si je mohl každý z čtenářů vykládat po svém. Nedílnou a nepsanou součástí těchto původních forem byla i lidová tradice předávat si ústně dodatečné informace rozšiřující tyto notové zápisy. Stejně tak, jako se vyvíjela hudba samotná, tak i notace prošla průběhem času řadou inovací, které se přizpůsobovaly potřebám hudebníku během různých hudebních období [1].

Dělení notace

Notové záznamy se dělí do dvou hlavních skupin. Na fonetickou a diastematickou formu [1].

Fonetická forma se vyznačuje tím, že je udávána v podobě písmen, čísel nebo různých znaků. Tuto formu můžeme pozorovat na příkladu u starověké řecké hudby nebo u hudby čínské [1].

Diastematická forma, známá také jako forma intervalová, kterou můžeme pozorovat u západní hudby, udává jednotlivé zvuky graficky [1].

Mimo to existují i příklady dalších jedinečných forem, které byly vynalezeny různými národnostmi nebo kulturami. Příkladem je etiopská, arménská nebo ruská notace [1].

Neumy

Jako základ západní notace se považují neumy. Samotné slovo neuma pochází z řečtiny a dá se přeložit jako znak, což také definuje podobu této původní notace.

Prvotní zmínky o neumech se datují do 9. století. Neumy byly z počátku značky napsány nad příslušné slabiky, aby značili průběh melodie. Pomocí neumatických značek ještě nebyly zaznamenávány přesné výšky jednotlivých tónů, a proto musela být melodie stále předávána i ústní formou. I z tohoto důvodu neumatická notace nenahradila stávající ústní notaci, nýbrž ji doplnila a vylepšila [1].

Před rokem 1000 se začínají objevovat zapsané neumy v různých úrovních, které již začínají udávat první informace o výšce tónu a intervalech mezi nimi. Neumy byly napsány na různých úrovních i přes chybějící notové linky a také byli již jednotlivé neumy natahovány do různých délek. Přesné intervaly prozatím nebylo možné z takového zápisu určit, ale již pomohli přibližnějšímu určení intervalů v závislosti na porovnání relativní výšky jednotlivých neumů [1].

Koncem desátého století se již začaly objevovat první notové linky. Původně byly vyryté do kusu pergamenu. Následně se začaly psát inkoustem. Tyto čáry jsou předchůdci notové osnovy, tak jak ji známe nyní. Nejdříve byly pouze dvě linky. Červená linka pro F a žlutá pro C. Poté postupem času byly rozšířeny tyto dvě linky na čtyři, kde byly přidány A a E linky. Jakmile byly tyto čtyři linky časem ustanoveny, tak se odpustilo od dvou výrazně barevných linek a barva všech se sjednotila do černé nebo červené barvy. Pro označení F nebo C linky se začalo používat jejich písmeno, které se psalo nad jednu z těchto linek. Neumy se poté začaly psát mezi tyto linky, což je již výrazně podobné tomu, jak známe notaci nyní [1].

Guido z Arezza

O další vývoj notové osnovy se zasloužil Guido z Areza, díky kterému se poprvé v historii západní notace dal předávat hudební záznam i bez pomoci ústně předávaných informací. Někteří hudební vědci uvádějí, že to byl právě Guido, kdo vymyslel čtyřlinkový notový záznam, ale objevují se i názory, u kterých například Oliver Strunk tvrdí, že sám Guido s největší pravděpodobností nevymyslel tento systém, ale pouze výrazně přispěl k jeho rozvoji. Jeho přínos notaci se ale nedá upřít, i díky tomu, že v jeho díle *'Prologue to his Antiphoner'* byla popsána charakteristika čtyřlinkového systému a také objasněny jednotlivé barvy linek [1].

Rytmická notace – Leoninus a Perotinus

V případě předávání informací o výšce jednotlivých tónů byl již notový zápis poměrně dobře vyvinut. Další vývoj měl tedy za cíl podrobněji obsáhnout i rytmickou složku hudby.

V tomto ohledu se na vývoji podíleli dva skladatelé z Notredamské školy Leoninus a Perotinus. Dle amerického muzikologa Williama Waita byly změny imple-

mentované touto dvojící jedněmi z těch nejvíce směrodatných v hudební historii. Nejnámějším počinem Leoninuse bylo sepsání a posbírání díla *'Magnus liber organi'*. Perotinus zrevidoval a sepsal části z *'Magnus liber'*. Díky tomu nebylo poprvé v historii toto polyfonní dílo zaznamenáno pouze co do výšek tónů, ale přibyl i rytmický zápis. Rytmika byla zapsána pomocí modální notace, která byla z počátku definována pouze dvěma slovy: *krátce* a *dlouze*. Tyto dvě hodnoty byly nakonec uspořádány do šesti různých rytmických módů a považují se za předchůdce ligatury. Přesto, že modální notace nepřiřazovala rytmus přesně do taktů a délka jednotlivých tónů byla stále velmi proměnná, tak se nedá upřít fakt, že se jednalo o první vývoj v oblasti rytmického systému [1].

Franko Kolínský

Franko Kolínský dále posunul a rozvinul modální systém, tedy systém notace vícehlasé hudby. Jednotlivým notám byly přiřazeny přesně dané délky pomocí různých tvarů not. Nadefinoval nový systém hodnot délek tónů jako *longs*, *breves* a*semibrave*. Kombinací těchto délek vznikl systém, kde záleželo, zda *breve* předcházela nebo následovala *long*, a tím bylo určeno, zda se jedná o dokonalé spojení, které mělo hodnotu tří jednotek času nebo nedokonalé o hodnotě dvou délek. Toto na první pohled složité spojení by se dalo v dnešní moderní notaci přirovnat k systému tečky za notou, která prodlužuje notu o polovinu její délky [1].

Ars Nova

Během 14. stolení prošla hudební notace zásadním vývojem, který položil základy notace, které se používaly ještě následujících šest století. Došlo ke třem zásadním inovacím. Jedním z nich bylo vytvoření *minima*[1], které mělo ještě menší hodnotu než do tehdy nejmenší *semi-breve*[2]. Další prvek bylo rozšíření notace o nové termíny jako jsou*modus*, *tempus* a *prolatio*. Také se zde poprvé se objevují termíny *major*[3] a *minor*[4] [1].

1.1.2 Dynamika

Jako dynamika se označovuje sílá hry, respektive hlasitost přednesu hudby [10].

K označení hlasitosti slouží v notovém záznamu dynamická znaménka nebo české, případně italské slovní výrazy a jejich zkratky. Příklady těchto znamének jsou uvedeny v tabulce 1.1[10].

[1] minima - dnes nota půlová
[2] semi-brave - dnes nota celá
[3] major - dur
[4] minor - moll

Dynamické pojmy		
Italský název	Zkratka	Interpretace
piano pianissimo	ppp	co nejslaběji
pianissimo	pp	velmi slabě
piano	p	slabě, tiše
mezzopiano	mp	středně slabě
mezzoforte	mf	středně silně
forte	f	silně
fortissimo	ff	velmi silně
forte fortissimo	fff	co nejsilněji

Tab. 1.1: Dynamické pojmy [10]

Pro změnu mezi jednotlivými úrovněmi hlasitostí existují dynamická znaménka *crescendo* a *decrescendo*. Tato znaménka se značí protáhnutými znaky menší/větší než. *Crescendo* je znaménko pro zesílení. Naopak *decrescendo* slouží k zeslabení hlasitosti hry v průběhu notového zápisu [10].

1.1.3 Klíče

Takzvaný klíč se používá pro označení umístění jedné noty v notové osnově, dle které se dále řídí i umístění ostatních not. Od této noty se dále nazavuje na základní tónovou řadu *c d e f g a h* [10].

Rozlišujeme různé klíče. Nejčastěji se používají následující čtyři klíče:

Houslový klíč

Houslový klíč, známý také jako G klíč, určuje postavení noty G1 na druhé lince [10].

Basový klíč

Basový klíč, známý také jako F klíč, který určuje postavení noty f na čtvrté lince [10].

C klíč altový

Altový C klíč určuje postavení noty c1 na třetí lince [10].

C klíč tenorový

Tenorový C klíč určuje postavení noty c1 na čtvrté lince [10].

1.1.4 Artikulační znaménka

Artikulační znaménka udávají informaci o tom, jak mají být dané noty interpretované. Některé nástroje mají svá specifická artikulační znaménka a naopak některá artikulační znaménka jsou všeobecná [10].

Legato

Legato, neboli vázaně, znamená že se tóny hrají těsně za sebou bez mezer. Znakem pro legato je oblouček spojující noty různé, ale i stejné výšky. V případě smyčcových nástrojů se legato provádí jedním tahem smyčce. U zpěvu nebo dechových nástrojů se provádí jedním nádechem [10].

Akcent

Akcent značí zdůraznění daného tónu nebo akordu oproti tónům ostatním. Toto zdůraznění je výrazné oproti klasicky zahraným tónům. Akcenty třídíme do různých úrovní dle velikosti zdůrazění, respektive síly provedení [10].

Tenuto

Tenuto znamená vydržení hodnoty noty s malým přízvukem. Tenuto značíme vodorovnou čárkou nad nebo pod notou. Taktéž jej můžeme označit slovní zkratkou *"ten"* [10].

Staccato

Staccatem se rozumí krátce, oddělovaně. Tóny v melodii se hrají krátce a mezi nimi vznikají pomlčky. Značí se tečkou nad nebo pod notou. Ostré staccato je možné v notovém zápisu označit klínkem. Naopak pro nepříliš ostré staccato se používá vodorovná čárka nad nebo pod staccatovou tečku [10].

Pizzicato

Značí styl hry, kdy hudebník drnká prsty o struny [10].

Glissando

Jako glissando je označené skouznutí z jednoho tónu na druhý. Během tohoto skouznutí se výška tónu mění plynule. Nejčastěji jej můžeme zaznamenat pro smyčcové nástroje, kde se dobře provádí díky skluzu prstem po struně. Značí se čárkou nebo vlnovkou spojující dva tóny různé výšky [10].

1.2 Programovací jazyky

Programovací jazyky jsou klíčové nástroje, které umožňují vývojářům vytvářet různé aplikace a webové stránky. Tato kapitola se zabývá dvěmi základními programovacími jazyky, které jsou široce používány v oblasti webového vývoje. První podkapitola je zaměřa na HTML, který je základním stavebním kamenem webových stránek, a druhá podkapitola na JavaScript, který umožňuje interaktivitu a dynamické chování webových aplikací.

1.2.1 HTML

HTML, neboli Hypertext Markup Language, je značkovací jazyk sloužící k uspořádání dokumentu a také ke specifikaci hypertextových odkazů. Pomocí tohoto jazyka můžeme také definovat umístění jednotlivých objektů jako jsou obrázky, různá média nebo text a dalších sekcí, které nejsou sami o sobě definovány a zobrazovány prohlížečem. Jak už samotný název jazyka napovídá, tak HTML slouží také k tvorbě hypertextových odkazů pro propojení dokumentů v rámci jedné webové stránky nebo i s ostatními webovými stránkami na internetu [2].

HTML je oblíbený pro svou jednoduchou čitelnost a použití. Zdrojový kód HTML je poměrně obsáhlý a při některých kódech může být až více než polovina kódu ve výsledku nezobrazována prohlížečem. Jedná se o vnořený jazyk, který funguje na principu vkládání pokynů nebo tagů do dokumentu. Kód použije informace uložené uvnitř tagů a pomocí nich určuje, jak zobrazovat jednotlivé části dokumentu [2].

Tyto tagy jsou vloženy mezi symboly menší než a větší než ('<>'). Tagy zároveň ohraničují text ze začátku a konce, kde první určuje, jak bude daný text vypadat a před druhým tagem je navíc lomítko ('/'), které ohraničuje konec této funkce. Celý HTML dokument ohraničuje tag '<html>' [2].

1.2.2 JavaScript

JavaScript je objektově orientovaný událostmi řízený programovací jazyk, jehož velkou výhodou je jeho velká rozšířenost a dostupnost. Používá se pro programovaní webových stránek, zvlášť kvůli jeho jednoduché implementaci interaktivních prvků. Samotný JavaScript je implementovaný do webových stránek pomocí HTML, který označuje začátek a konec skriptu v JavaScriptu. Takový příkaz pro implementaci do HTML je označovaný tagem '<script>' [3].

Díky vysoké flexibilitě záleží pouze na vývojářích, zda bude jejich kód jednoduchý nebo komplexně napsaný. Tento jazyk rovněž podporuje několik programovacích stylů zápisu mezi kterými lze vybírat, včetně funkčního nebo více komplexního, objektově orientovaného, stylu.[4].

JavaScript je volně napsaný jazyk, což znamená, že typ proměnných není deklarovaný. Rovněž se odlišuje tím, že na rozdíl od většiny nejrozšířenějších programovacích jazyků nerozlišuje mezi celočíselnými a desetinnými čísly [4].

1.3 Data formáty

Dataformáty jsou struktury, které slouží k ukládání a přenosu dat v různých aplikacích a systémech. Tato kapitola se věnuje několika běžně používaným formátům, které jsou klíčové pro výměnu dat mezi aplikacemi. První podkapitola je věnována XML. V druhé podkapitole je pojednáváno o formátu MusicXML, což je specifický formát určený pro notové zápisy a hudební data. Třetí podkapitola je věnována formátu JSON. V závěrečné podkapitole jsou porovnány XML a JSON.

1.3.1 XML

XML je značkovací jazyk pro textové dokumentace. Zároveň splňuje kritéria a je schváleným W3C standardem pro značkování dokumentu. XML syntaxe využívá jednoduchých tagů, které značí jednotlivá data a tyto tagy jsou zároveň dobře rozeznatelné a čitelné i pro uživatele. Data uvnitř těchto tagů jsou uloženy jako řetězce textu [5].

XML je také velmi oblíbený díky velké flexibilitě, kde dokáže být díky jednoduché upravitelnosti použit jednak při webových stránkách, elektronických zařízeních pro výměnu dat nebo u vektorové grafiky [5].

Velkou výhodou XML je, že v rámci jeho flexibility umožňuje uživatelům vytvoření vlastních základních tagů, dle jejich potřeb. V případě hudebníků se může jednat například o tagy, které budou popisovat různé noty, houslové klíče či jiné potřebné znaky pro hudební notaci. Tato jednoduchá rozšiřitelnost a také to, že se jazyk dokáže adaptovat k dalším jiným využitím, dala volnou cestu k hudebnímu rozšíření MusicXML (viz kap. 1.3.2)[5].

1.3.2 MusicXML

Prvotním cílem bylo vytvořit jazyk, v rámci kterého by šlo předávat data a informace o notovém zápise. Jediným takto rozšířeným nástrojem používaným i komerčními softwéry byl MIDI standard (viz kap. 2.5), ale ten poskytoval pouze omezené a velmi malé množství informací, které by reprezentovalo reálný notový zápis v plné míře. Jak už název napovídá, MusicXML vychází z XML jazyka (viz kap. 1.3.1), který poskytuje ideální možnosti a technologie pro obsažení komplexních dat o notovém zápisu, a také je dobře čitelný,jak pro počítače, tak i pro uživatele. MusicXML je

zdarma volně dostupný, jak pro komerční, tak nekomerční využití a stal se z něj standart pro výměnu informací o notaci. S tímto jazykem dokáží pracovat velmi často používané programy, jako jsou například Sibelius, Finale nebo Cubase, který jej dokáží jak přečíst, tak i v něm psát (viz kap. 2.1.1) [6].

Při návrhu nového jazyka pro výměnu dat mezi formáty měli vývojáři dva hlavní cíle. Základ tohoto jazyka se odvíjel od dvou již existujících akademických hudebních formátů pro hudební notaci MuseData a Humdrum. Tyto formáty byly solidním technickým základem pro vznik nového jazyka, mimo jiné i díky jejich rozsáhlým knihovnám. Druhým důležitým úkolem bylo, aby MusicXML byl schopný podporovat výměnu dat s nejpoužívanějším programem na trhu, kterým bylo Finale od společnosti Coda Music Technology. Toho docílili tím, že MusicXML byl schopen dvoucesté přeměny do MuseData. Dokázalo také přečíst soubory z NIFF a zapisovat je do standartního MIDI formátu [7].

1.3.3 JSON

JSON, neboli JavaScript Object Notation je data formát, jehož hlavním účelem je sloužit pro komunikaci aplikace po síti, nejčastěji přes RESTful API. Přes toto API fungují stránky jako jsou Twitter, Facebook, GitHub nebo DropBox. Tento formát je podporován všemi hlavními programovacími jazyky, jako je například JavaScript, C#, PHP nebo Python. JSON se skládá z objektů, polí a názvů respektive párů hodnot. V případě využití formátu pro výměnu primárních dat přes internet postupně nahrazuje XML díky jeho jednoduché struktuře a přímému spojení s JavaScriptem. V porovnáním s XML dosahuje při formátování menších velikostí, díky čemuž mohou být tato data rychleji přenášeny v síti [8].

1.3.4 Porovnání XML a JSON

Odhaduje se, že v případě internetových prohlížečů je JSON formát oproti XML schopný členit soubory stonásobněkrát rychleji. Naopak XML má mnohem větší základnu podporujících rozšíření a škálu identifikátorů pro jmenný prostor. [9].

2 Frameworky a technologie

Následující kapitola je věnována představení širokého spektra možností pro vývoj hudebních aplikací a softwaru. Zaměřuje se na již vzniklé komerční i open-source programy, nástroje a frameworky, které jsou klíčové pro práci s notovým zápisem, syntézou zvuku a MIDI komunikací.

2.1 Komerční možnosti

Dnešní trh nabízí spoustu možností notačních programů od různých výrobců. Tyto programy nabízí v základních verzí velmi podobné nástroje a liší se subjektivní uživatelskou přívětivostí nebo zaměřením na určitý typ práce s notovým záznamem. Příkladem nejznámějších a nejpoužívanějších notačních programů jsou Sibelius, Finale nebo Dorico. Liší se také typem zakoupení, kdy Sibelius je na bázi měsíčního respektive ročního předplatného, zatímco licence pro Finale nebo Dorice se dají zakoupit jednorázově. Dorico nabízí základní balíček pro počítače nebo ipad zdarma.

2.1.1 Srovnání jednotlivých softwarů dle možností zobrazování

Práce Rettinghausera, Querfurtha a Bogdahna porovnala notační programy dle možností exportu ve formátu MusicXML. Uvádějí, že v muzikologii se staly jednotlivé notační softwary nedílnou součástí pro vytváření digitálních záznamů. Porovnali čtyři nejběžnější notační programy: Dorico, Finale, MuseScore (viz kap. 2.2.1) a Sibelius. V tabulce 2.1 je porovnána schopnost jednotlivých programů zobrazovat následující symboly a objekty [11].

2.2 OpenSource možnosti

2.2.1 MuseScore

Musescore je open-source software určený k notovému záznamu. Řadí se do skupiny editorů s grafickým uživatelským rozhraním pracujícím na principu WYSIWYG, neboli anglicky what-you-see-is-what-you-get, což znamená, že jsou noty vkládány přímo do virtuálního notového záznamu zobrazovaného přímo na stránce. MuseScore umožňuje přehrávání notových záznamů, stejně tak jako importování nebo exportování MusicXML (kap. 1.3.2) nebo MIDI (kap. 2.5) souborů [12].

Srovnání exportů do MusicXML různých notačních softwarů				
	Dorico 5.0	Finale v25.5	MuseScore 4.1	Sibelius 2022.12
Arpeggios	-	X	/	/
Articulation	/	X	/	/
Barlines	X	X	X	X
Clefs	X	X	X	X
Dynamics	X	X	X	/
Dynamics position	X	X	X	-
Fermatas	-	X	X	/
Glissandos	-	X	X	/
Hairpins	X	X	X	X
Hairpins line style	X	N/A	-	-
Hairpins niente	X	N/A	-	-
Keys (extended)	/	N/A	/	N/A
Keys (standard)	X	X	X	X
Notes	X	X	X	X
Ornaments	-	X	X	/
Repeats	X	X	X	-
Rest positions	-	X	X	-
Text	-	X	X	X
Time signatures	X	X	X	X
Trills	-	X	X	/
Legenda:	X podporováno / částečně podporováno		- nepodporováno N/A nedostupné	

Tab. 2.1: Srovnání exportů do MusicXML různých notačních softwarů [11].

2.2.2 LilyPond

LilyPond je programovatelný kompilátor pro vytváření hudební notace. Částečně využívá jazyku Scheme a obsahuje také interpretor GUILE Scheme, pomocí kterého rozšiřuje funkcionalitu samotného LilyPondu [13].

Vstupní hudební informace jsou předávány v textovém formátu. Jedná se o batch program, což znamená, že při vyvolání programu jsou přečteny soubory a následně jsou zpracovány již bez jakékoli interakce uživatele. Program při zpracovávání provádí následující čtyři kroky. Nejprve je analyzován vstup zadaný uživatelem a sestaven do stromu syntaxe. Druhý krok interpretace převádí hudební informace do podoby grafických objektů, které tvoří neformátovanou partituru. Dalším krokem

je formátovaní této partitury a následně je tato partitura zapsána do koncového souboru [13].

Může být volně kopírován, používán nebo upravován pod podmínkami GNU General Public License, díky čemuž jej lze řadit mezi open-source software [13].

2.2.3 MusiXTeX

Jedná se o software sloužící k sázení hudebních záznamů, který používá systém pro sazbu TeX. Je navržen a zaměřen převážně na záznam polyfonní, instrumentální nebo orchestrální hudby. MusicTeX dokáže obsáhnout 9 různých nástrojů a pro každý z těchto nástrojů až 4 notové osnovy. Nejedná se o ideální formát k přenosu obsáhlých dat o notovém zápisu nebo tvorbu a tisk celých not, ale jeho využití spočívá v implementaci úseků notových záznamů do textu. Rizikem při práci s MusicTeX jsou vzhledem k velkému množství dat překlepy, anebo problémy se zalamováním řádků v případě nepravidelného záznamu. V případě MusicTeX se nejedná o kompilátor, který by překládal standartní hudební notaci do formátu TeX nebo by sám komplexně rozhodoval o zapisování hudební údajů z estetického pohledu. Je schopen sazby základních objektů jako např. osnovy, noty, akordy nebo jednotlivé ozdoby, ale pro ideální zobrazení z estetického pohledu se doporučuje propojení s předkompilátorem, který by měl rozsáhlejší možnosti k sazbě notačních objektů [14].

2.3 Frameworky pro notovou osnovu

2.3.1 VewFlow

Jedná se o open-source API, které slouží k vykreslení hudební notace ve webovém prohlížeči. Jde o knihovnu, která je celá realizovaná v programovacím jazyce JavaScript. Tento framework taktéž podporuje implementaci v rámci HTML5 Canvas a SVG [15].

VewFlow obsahuje rozmanitou paletu různých hudebních prvků a značek potřebných pro obsáhlou hudební notaci. Díky tomu umožňuje zobrazování od jednoduchých hudebních prvků jako jsou houslové kliče, noty, pomlky až po rozsáhlé struktury notového zápisu, zápis tabulátorů a podobně. [16].

Toto API pracuje na principu generování a vytváření každého objektu zvlášť z nadefinovaných knihoven. Každý symbol musí být v kódu samostntně vytvořen a umístěn ručně do notové osnovy [17].

Celý kód a obsah VexFlow je dostupný pod MIT licencí na githubu a je volně dostupný k nekomerčnímu použití [15].

VexTab

VexTab je rozšíření, pomocí kterého je možné rozšířit standartní notový zápis pro tabulátory. Je navržen tak, aby byl snadno použitelný a na rozdíl od ASCII tabulek, jejichž hlavní doménou je snadná čitelnost, se soustředí na jednoduchý zápis. [18]

2.3.2 ABC.js

ABC Music Notation je formát, který slouží k zobrazování a vkládání hudební notace do webových stránek. Pracuje na principu, kdy k zobrazování notového záznamu stačí pouze řetězce znaků [19].

Přehrávání pomocí ABC.js

Abc.js vytváří audio buffer, který pracuje na stejném principu jako WAV soubor a tím pádem obsahuje celou skladbu, kterou má přehrávat. Využívá hudební databáze neboli banky, kde samotné zvuky pochází z této banky ve které je každá nota každého hudebního nástroje uložena samostatně jako separátní soubor a všechny tyto jednotlivé záznamy jsou poté poskládány do audio bufferu. K realizaci jednotlivých tónů nebo samotných nástrojů využívá MIDI specifikaci (kap. 2.5)[20].

Syntetizátor

Pro generování zvuku z notové soustavy je základním kódem a prvním potřebným krokem vytvoření syntetizátoru pomocí objektu *'CreateSynth()'*, který načítá potřebné noty. Pro načtení těchto dat do syntetizátoru je potřeba připojení k internetu, pokud nejsou načítány interní data [20].

Výpis 2.1: Kód pro vyvoření syntetizátoru v rámci ABC.js [20]

```
var synth = new ABCJS.synth.CreateSynth();
```

Příkaz *'prime()'* slouží k vytvoření samotného bufferu, kde již tato funkce může být použita v offline režimu, vzhledem k tomu, že všechna potřebná data jsou již načtena při vytvoření syntetizátoru. Tato funkce je posledním krokem nutným k následujícím funkcím, které přesněji upravují, jak se bude záznam přehrávat [20].

2.4 Frameworky pro syntetizátory

2.4.1 Tone.js

Tone.js je webový audio framework, který slouží pro vytváření interaktivních zvuků v prostředí webového prohlížeče. Framework je navržen tak, aby byl uživatelsky

přívětivý jak pro hudebníky, tak pro hudební inženýry nebo programátory, kteří vytvářejí hudební aplikace založené na webovém rozhraní. Tone.js nabízí možnosti pro přehrávání předpřipravených syntetizátorů nebo efektů, ale také nabízí funkcionalitu umožňující uživatelům vytváření vlastních syntetizátorů nebo efektů [21].

Samotný vývoj Tone.js, který měl za cíl vytvořit frameworku pro vytváření interaktivní hudby, se řídil třemi hlavními kritérii: hudebností, modularitou a synchronizací.

Tone.js je open-source a celý kód je dostupný pod MIT licencí na githubu [22].

Tone.Synth

Tone.Synt je základním syntetizátorem, který obsahuje jeden oscilátor a ADSR obálku. Existují dvě možnosti jak jej ovládat a předávat mu hlavní parametry.

Jedna z možností využívá dvou samostatných příkazů *'triggerAttack'* respektive *'triggerRealease'*, jehož použití lze pozorovat na výpisu 2.2. První jmenovaný určuje notu, která má být zahrána a také čas od kdy je tato nota hrána, tedy čas při kterém roste hodnota amplitudy. *'TriggerRealease'* naopak udává dobu, kdy se amplituda tónu vrací zpátky na hodnotu 0 [22].

Výpis 2.2: Příklad implementace 'triggerAttack' a 'triggerRelease' [22]

```
const synth = new Tone.Synth().toDestination();
const now = Tone.now()
synth.triggerAttack("C4", now)
synth.triggerRelease(now + 1)
```

Druhá metoda (viz. výpis 2.3) využívá příkazu *'triggerAttackRelease'*, který kombinuje oba příkazy zmíněné v první metodě, tedy *'triggerAttack'* a *'triggerRelease'*. Příkaz *'triggerAttackRelease'* obshuje tři parametry. První hodnota udává notu, kterou má syntetizátor zahrát. Tento parametr může být zadán buď hodnotou frekvence v hertzích nebo jako výška tónu a jeho příslušná oktáva. Hodnota délky noty je udávána jako druhý paramter v pořadí. Tato hodnota je matematickým ekvivalentem, který odpovídá délce not. Například nota čtvrťová je vyjádřená jako *'4n'* nebo notě osminové přísluší hodnota *'8n'*. Poslední volitelný argument určuje počátek, kdy by se měla daná nota začít přehrávat [22].

Výpis 2.3: Příklad implementace 'triggerAttackRelease' [22]

```
const synth = new Tone.Synth().toDestination();
const now = Tone.now()
synth.triggerAttackRelease("C4", "8n", now)
synth.triggerAttackRelease("E4", "8n", now + 0.5)
synth.triggerAttackRelease("G4", "8n", now + 1)
```

Monofonické syntetizátory

Monofonickými syntetizátory jsou ty, které generují a dokáží hrát pouze jeden tón. V případě knihovny Tone.js se jedná na příklad o *'Tone.FMSynth'*, *'Tone.AMSynth'* nebo *'Tone.NoiseSynth'* [22].

Polyfonické syntetizátory

Syntetiázor *'Tone.PolySynth'* je schopný hrát, respektive udržet více tónů zároveň. Parametry, s kterými pracuje tento syntetizátor, jsou stejné jako u *'Tone.Synth'* nebo u jiných monofonních syntetizátorů, tedy *'triggerAttack'* a *'triggerRelease'*. Rozdíl je akorát v logice příkazů, kdy při následujícím příkazu *'triggerAttack'* se tón přidává k předchozímu a v rámci příkazu *'triggerRelease'* musí uživatel zadat kterou notu nebo pole not má syntetizátor uvolnit. Příklad použití polyfonického syntitátoru je uveden ve výpisu 2.4 [22].

Výpis 2.4: Příklad realizace polyfonického syntetizátoru [22]

```
const synth = new Tone.PolySynth(Tone.Synth).toDestination();
const now = Tone.now()
synth.triggerAttack("D4", now);
synth.triggerAttack("F4", now + 0.5);
synth.triggerAttack("A4", now + 1);
synth.triggerRelease(["D4", "F4", "A4"], now + 2);
```

Sampler

Tone.js dokáže také načítat a přehrávat externě vytvořené nahrávky nebo samply. Také je možné využít sampleru, který dokáže přehrávat několik vzorků zároveň z kterých může být jejich kombinací vytvořen hudební nástroj. Pokud jsou jednotlivé vzorky uložené a seřazené podle not, tak sampler také dokáže upravit výšky těchto tónů tak, aby vyplnil mezery mezi jednotlivými tóny (viz. výpis 2.5). *'Tone.Sampler'* pracuje na technologii polyfonního syntetizátoru [22].

Funkce rampTo

Tone.js je navržen tak, aby mohl ovládat téměř jakékoliv jednotlivé parametry. Díky tomu je také schopen pomocí funkce *'rampTo'* plynule měnit parametr frekvence z jednoho tónu na druhý a docílit tak plynulého přechodu z jednoho tónu na druhý [22]. Použití této funkce je uvedeno na výpisu 2.6.

Výpis 2.5: Příklad realizace sampleru [22]

```
const sampler = new Tone.Sampler({
  urls: {
    "C4": "C4.mp3",
    "D#4": "Ds4.mp3",
    "F#4": "Fs4.mp3",
  },
  baseUrl: "https://tonejs.github.io/audio/salamander/",
}).toDestination();
Tone.loaded().then(() => {
  sampler.triggerAttackRelease(["Eb4", "G4", "Bb4"], 0.5);
})
```

Výpis 2.6: Funkce 'rampTo' [22]

```
const osc = new Tone.Oscillator().toDestination();
osc.frequency.value = "C4";
osc.frequency.rampTo("C5", 2)
```

Tone.js a MIDI

Tone.js dokáže přečíst soubory pouze ve formátu JSON (viz kap. 1.3.3). Samotné MIDI soubory (viz kap. 2.5) nedokáže přečíst, takže tyto soubory musí být nejdříve převedeny na formát JSON [22].

2.4.2 Note-Parser

API note-parser je napsáno čistě v JavaScriptu a při předání dané noty vrací příslušné MIDI hodnoty nebo hodnotu frekvence příslušící pro daný tón. Základním příkazem je *'parser.parse'*, který vrací všechny dříve zmíněné parametry (viz. výpis 2.7) [24].

Výpis 2.7: Příkaz 'parser.parse' a jeho výsledné hodnoty [24]

```
parser.parse('c#4') // => { letter: 'C', acc: '#',
midi: 61, freq: 277.1826309768721 }
```

Pro zjištění pouze jednoho parametru slouží separátní příkaz *'parser.midi'*, jehož návratové hodnoty lze pozorovat na příkladech výpisů 2.8, respektive 2.9 pro funkce, které vrací hodnotu v MIDI, respektive pro frekvenci [24].

Vstupním parametrem pro *'parser.freq'* může být přímo tón se specifikací v které oktávě se nachází, anebo také parametr MIDI [24].

Výpis 2.8: Příkaz 'parser.midi' a jeho výsledné hodnoty [24]

```
parser.midi('A4') // => 69
parser.midi('blah') // => null
parser.midi(60) // => 60
parser.midi('60') // => 60
```

Výpis 2.9: Příkaz 'parser.freq' a jeho výsledné hodnoty [24]

```
parser.freq('A4') // => 440
parser.freq('A3', 444) // => 222
parser.freq(69) // => 440
```

2.5 MIDI

MIDI neboli „Musical Instrument Digital Interface“ je komunikační protokol, který slouží k předávání digitálních dat. Jádro tohoto jazyku nepředává informace o hudbě, respektive data z notového zápisu [25].

Samotné MIDI je nejvíce cílené na elektronickou hudbu zvláště díky tomu, že MIDI je schopné zaznamenávat jednotlivá data potřebné ke generování zvuku jako je například délka tónu a její počátek nebo síla úderu. Pro porovnání mezi MIDI standartem a notovým zápisem můžeme říct, že notový zápis udává obsáhlé a komplexní informace o tom, jak by měla být daná skladba hudebníkem zahrána na rozdíl od MIDI protokolu, který zaznamenává komplexní informace o vystoupení daného hudebníka na elektronický nástroj, který je schopný tyto údaje zaznamenávat. Pomocí MIDI může být takto zaznamenané vstoupení také znovu vytvořeno [25].

2.5.1 MIDI zprávy

Samotná komunikace jednotlivých zařízení mezi sebou je založena na numerických kódech. MIDI nepopisuje samotný zvuk, ale pouze číselné údaje, které udávají informace pro jednotlivé tóny. Tyto kódy komunikují jednotlivé akce vyvolané uživatelem jako jsou stisky kláves nebo aktivace různých tlačítek. Samotná MIDI specifikace přiřazuje zprávám čísla, která specifikují o jaký typ zprávy se jedná [25].

Každý byte zprávy začíná počátečním bitem, kterým je logická nula. Za ní následuje 8 bitů, kdy první z nich je nejméně významným a celých těchto osm bytů udává hodnotu nebo status zařízení. Celou zprávu zakončuje stop bit, kterým je logická jednička. Specifikaci o jakou zprávu se jedná, respektive o to jak moc dalších bytů zpráva obsahuje, aby byla kompletní udává prvním byte zprávy a vždy se tedy jedná o tzv. statusový byte. Data byty mají možný rozsah hodnot mezi 0 – 127 [25].

2.5.2 Typy MIDI zprávy

Jednotlivé zprávy se dají rozdělit do dvou hlavních skupin. Kanálové zprávy a systémové zprávy [25].

Kanálové zprávy přenáší převážně jednu ze čtyř základních zpráv. Nota zapnuta, nota vypnuta, změna kontroleru, změna programu. Tyto zprávy jsou uloženy ve 3 bitech, které následují za nejvýznamějším bytem. Čtyři nejméně vyznamné byty naopak určují jeden ze šestnácti MIDI kanálů, kde prvnímu kanálu náleží hodnota 0000 a šestnáctému kanálu hodnota 1111[25].

Systémové zprávy můžeme rozdělit do tří hlavních skupin: běžné systémové zprávy, zprávy přenášené v reálném čase a exkluzivní systémové zprávy.

MIDI specifikace obsahuje sedm běžných systémových zpráv, kdy pět z nich je přesně definovaných. Mezi ty řadíme zprávy o ladění, výběru písně, výběru pozice v rámci písně, zprávu přenášející časový MIDI kód a zprávu zakončující konec zprávy. [25].

Systémové real-time zprávy mají za úkol synchronizovat čas více MIDI zařízení, aby běželi zároveň. Při živých vystoupeních, případně u playbacku je zapotřebí, aby jednotlivým nástrojům pracujícím na principu MIDI jako jsou sekvencery, bicí samplery nebo syntetizátory, odpovídalo stejné tempo. To zajišťují systémové real-time zprávy, které jsou odesílány v přesně daných pravidelných intervalech. Systémové real-time zprávy ovlivňují celý běh MIDI zařízení a z tohoto důvodu neobsahují žádná data, která by specifikovala o jaké jednotlivé kanály se jedná. I z tohoto důvodu obsahují tyto zprávy pouze jediný statusový byte bez jakýchkoliv následujících data bytů [25].

2.5.3 MIDI kabely

MIDI kabely přenášejí úrovně napětí odpovídající hodnotám jedniček a nul v binární soustavě. Tyto kabely mají také své příslušné porty na jednotlivých zařízeních u kterých by nemělo být zaměněno připojení audio a midy kabelů, protože by mohlo dojít k poničení jednotlivých přístrojů [25].

2.5.4 WEBMIDI.js

Toto API umožňuje interakci mezi MIDI ovladačem, respektive hudebním nástrojem umožňujícím předávání MIDI informací a webovými stránkami [26].

3 Realizace

Tato část je věnována praktické realizaci projektu, detailnímu popisu struktury souborů a jejich vzájemné integraci. Každý soubor má svou specifickou úlohu a přispívá k funkčnosti celé aplikace. Cílem je poskytnout pohled na organizaci projektu a jeho funkce.

3.1 Použité frameworky a důvody výběru

V projektu jsou použity knihovny:

- VexFlow (kap. 2.3.1)
- Tone.js (kap. 2.4.1)

Knihovna VexFlow byla zvolena k zobrazování not v notové osnově. Nabízí k dispozici obsáhlou dokumentaci, které usnadňuje práci s touto knihovnou. Je zde na rozdíl od ostatních knihoven popsaných v teoretické části použit více programátorský přístup ke vkládání not, které nejsou zadávány formou jednoho obsáhlého textového pole. Stěžejní pro výběr byl také fakt, že je VexFlow napsaný v programovacím jazyce JavaScript stejně tak jako následující vybraná knihovna.

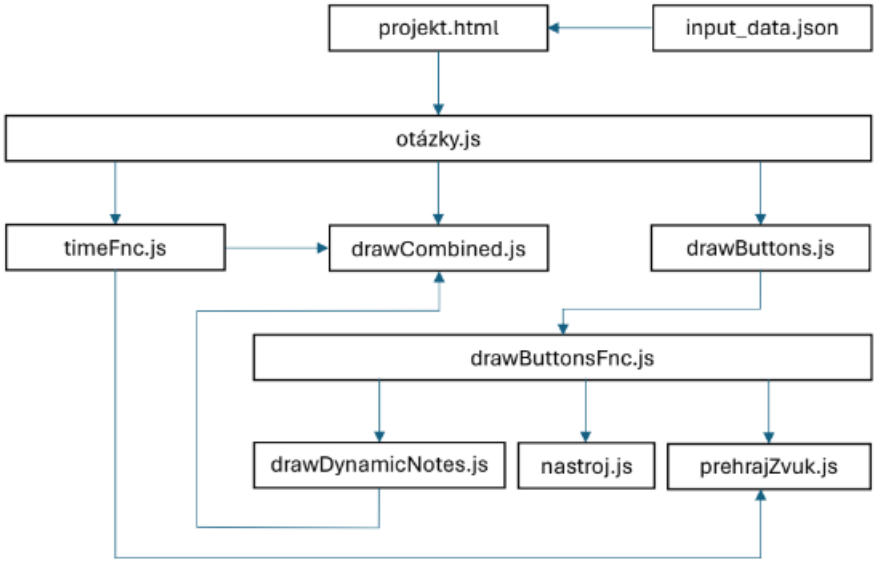
Tone.js byl vybrán pro přehrávání not. Tato knihovna nabízí obsáhlé řešení, jak přehrávat zvuky od jednoduchých monofonických až po polyfonické. Dá se využít jeho variability, kdy je sám o sobě schopný generovat signály, což se dá využít například u plynulé změny z jednoho tónu na druhý, ale stejně tak dokáže přehrávat jednotlivé samply. Hlavní výhodou pro případné budoucí rozšíření projektu je sapmler, který nabízí možnost vytvoření vlastního hudebního nástroje a dle dokumentace nepotřebuje vložit všechny tóny jednotlivě, ale dokáže si je sám z pár vzorků upravit a vyplnit celý svůj rozsah. Výhodou knihovny je taktéž obsáhlá dokumentace.

3.2 Struktura souborů a integrace

Celý projekt obsahuje následující soubory:

- projekt.html (kap. 3.2.1)
- input_data.json (kap. 3.2.2)
- otazky.js (kap. 3.2.3)
- drawButtons.js (kap. 3.2.4)
- drawButtonsFnc.js (kap. 3.2.5)
- timeFnc.js (kap. 3.2.6)
- convertor.js (kap. 3.2.8)
- drawCombined.js (kap. 3.2.7)
- nastroj.js (kap. 3.2.9)

- prehrajZvuk.js (kap. 3.2.10)



Obr. 3.1: Diagram posloupnosti souborů

Na obrázku 3.1 je poslounost, jak jsou volané jednotlivé soubory a funkce napsané v těchto souborech.

3.2.1 projekt.html

V této části je vytvořena základní struktura HTML souboru. Dochází zde k načítání knihoven VexFlow (kap. 2.3.1) a Tone.js (kap. 2.4.1) a také jsou zde načítány jednotlivé JavaScript soubory. Kromě definice jednotlivých kontejnerů jsou zde nadefinované i základní akce, které se spouští při kliknutí na jednotlivá tlačítka. V tomto souboru jsou načítána i vstupní data ze souboru *input_data.json*, kde dochází k definici dat uživatelem, které popisují jednotlivé notové osnovy a jejich obsah.

3.2.2 input_data.json

Definování všech dat souvisejících s notovu osnovou probíhá v tomto souboru. Uživateli, respektive zadaveteli daných osnov, stačí pracovat pro úspěšné vykreslení pouze s tímto souborem. Přesná syntaxe zadávání a definování osnov je popsána v souboru *Manuál pro zadávání* přiloženém v příloze práce.

Pozn. Práce se nadále nebude věnovat vstupnímu zadávání hodnot a to i při jednotlivých souborech aplikace a funkcích souvisejících s definicí not a bude vycházet z tohoto manuálu.

3.2.3 otazky.js

V souboru otazky.js se nachází logika, která přebírá vstupní data uživatele v potřebné struktuře uložené v souboru *input_data.json* (kap. 3.2.2) a zpracovává je tak, aby následující soubory s nimi již mohli na přímo pracovat a nedocházelo k dalšímu načítání z json souboru. Z tohoto souboru se také volají všechny ostatní navazující funkce, jak pro vykreslení osnovy, tak pro její přehrávání a zvýraznění.

Díky načtení vstupních dat v souboru *projekt.html* a předání informací pouze pro aktuální notový záznam pracuje tento a všechny následující soubory s daty právě pouze pro tento jeden záznam. Všechny potřebné výstupní informace, které zpracovává tento soubor jsou uloženy do objektu *"parametry"*, v kterém jsou předávány všechny dříve nadefinované proměnné. Zpracování vstupních dat probíhá voláním funkce *"displayCurrentQuestion"*.

Funkce *"displayCurrentQuestion"* obsahuje funkce nebo logiku pro zpracování následujících dat:

- Přepočítání tempa (kap. 3.2.3)
- Zpracování základních informací pro notovou osnouv (takt, klic, tempo, název) (kap. 3.2.3)
- Transpozice pro celou osnovu (kap. 3.2.3)
- Definice pole not a příslušných specifických znaků (kap. 3.2.3)
- Artikulace a anotace (kap. 3.2.3)
- Dynamika (kap. 3.2.3)
- Legato, Glissando, Trioly (kap. 3.2.3)
- Transpozice (kap. 3.2.3)
- Zpracování dat o poloze, jednotlivých svorkách spojující osnovy a názvech osnov

Celý kód pracuje tak, že pro daný notový záznam ukládá informace postupně pro každou osnovu zvlášť.

Přepočítání tempa

Kód umožňuje zadání tempa dle délky trvání noty celé, půlové nebo čtvrťové. Interně ale pracuje s nejběžnějším údajem tedy délkou noty čtvrťové. V tomto souboru volá funkci *"convertTempoToQuarterNote"* ze soubrou *timeFnc.js*, která slouží právě k tomuto přepoču.

Zpracování základních informací

Ve funkci *"displayCurrentQuestion"* jsou dále pro jednotlivé osnovy nejdříve zpracovány a uloženy informace pro jednotlivé osnovy o klíči, předznamenání, taktu,

tempo, názvu dané osnovy nebo typu ohraničení dané osnovy. Příklad takovéhoto zpracování je udeveden na výpisu 3.1.

Kód načítá informace pro daný index v pořadí v jakém jsou uloženy. Tyto informace ukládá postupně do pole tak, že výsledem pro jednotlivé promněné je pole dat, kde každé osnově přísluší jeden prvek tohoto pole.

Výpis 3.1: Příklad zpracování základních informací - klíč

```
const klic = currentQuestion[`clef${i}`];
klicInfo.push(klic);
```

Transpozice pro celou osnovu

Pomocí parametru *"transpoze_sound"* je možné transponovat přehrávání všech tónů v jedné osnově o libovolný počet půl tónů. Při zadání pouze tohoto parametru zůstává nadále zachované zobrazení not v notové osnově jako bez tohoto parametru. Tento parametr se dá tedy využít v případech, kdy chceme, aby byl daný záznam zaspán do notové osnovy v jedné tónině, ale byl přehráván v tónině jiné. Příklad zadání v json souboru je uveden na výpisu 3.2.

Obdobně lze použít parametr *"transpoze_note"*, který pracuje na stejném principu jako *"transpoze_sound"* akorát pro vykreslování not. Například pro hodnotu *"-2"* parametru *"transpoze_note"* se budou všechny noty vykreslovat do notové osnovy o dva půl tónu níže, ale přehrávání zůstane zachováno na původních tónech.

Lze také použít kombinaci obou parametrů, jejímž výsledkem bude posun not, jak pro vykreslování, tak pro přehrávání.

Výpis 3.2: Příklad pro zadání transpozice pro osnovu

```
"transpoze_sound1": 2,
"transpoze_note1": 2,
```

Definice pole not a příslušných specifických znaků

Je zde definováno pole not *"vexFlowNotesArray"*, do kterého je postupně uložena každá ze vstupně nadefinovaných not. Kódu je také předávána informace o stupni aktuální transpozice, tak aby bylo možné osnovu libovolně transponovat a kód s nasledující informací také náležitě pracuje.

Pro každou notu je zde vytvořen objekt *"StaveNote"* pomocí knihovny VexFlow. Strukturu objektu lze pozorovat na výpisu 3.3. Tomuto objektu jsou předány vstupní informace popisující výšku a délku tónu a klíč náležící dané notě. Pro první vykreslení je všem notám přidělena černá barva.

Výpis 3.3: Objekt 'StaveNote'

```
const staveNote = new Vex.Flow.StaveNote({
    keys: note.keys,
    duration: note.duration,
    clef: klic,
}).setStyle({ fillStyle: "black", strokeStyle: "black"});
```

Na výpisu 3.4 je uveden kód, pomocí kterého se k jednotlivým notám přídávají modifikátory hudebních symbolům jako je křížek, béčko nebo odrážka.

Výpis 3.4: Obsah objektu 'parametry'

```
const keysLength = note.keys.length;
for (let i = 0; i < keysLength; i++) {
    const accidental = `accidental${i + 1}`;
    if (note.hasOwnProperty(accidental) && note[accidental].
       trim() !== "") {
        staveNote.addModifier(new Vex.Flow.Accidental(note[
           accidental]), i);
    }
}
```

Artikulace a anotace

Program také umožnuje zadání různých hudebně specifických artikulačních znaků, tedy například toho, jak mají být jednotlivé noty zahrány (viz. tab. 3.1). Při zadávání znaků k notě je také možné zvolit, zda se má znak vykreslovat nad nebo pod notou. Tyto znaky jsou zadávány ve výchozím souboru přímo k dané notě.

K jednotlivým tónům je také možné přidat anotační znaky například název akordu. Tato anotace bude vypsána nad notu.

Dynamika

V případě dynamiky kód rozlišuje mezi dvěmi různými vstupy.

- dynamicSymbols
- hairpin

DynamicSymbols obsahuje zadávání jednotlivých textových symbolů určujících dynamiku například pro piano nebo forte, zatímco hairpin obsahuje informace o grafických znaménkách crescendo a decrescendo.

Vstupní zadání dynamiky pracuje na obdobném principu jako pro noty. Pro jednotlivé takty jsou vloženy dynamická znaménka s jejich příslušnou délkou. V souboru *"otazky.js"* jsou zpracovány tato vstupní data a uložena do pole hodnot s kterým

Přehled akrtikulačních symbolů a zadávacího formátu					
artikulace	formát	obrázek	artikulace	formát	obrázek
Staccato	'a.'		Staccatissimo	'av'	
Akcent	'a>'		Tenuto	'a-'	
L. H. Pizzicato	'a+'		Snap Pizzicato	'ao'	
Zdvih nahoru	'a|'		Zdvih dolů	'am'	
Fermata nad notou	'a@a'		Fermata pod notou	'a@u'	
Marcato	'a^ '				

Tab. 3.1: Přehled artikulačních symbolů

bude program následně pracovat. Pro každý prvek dynamiky je vytvořen objekt z knihovny VexFlow *"TextDynamics"*, který pracuje s potřebnými vstupními daty, tedy znakem a délkou trvání tohoto znaku. Tato data jsou zároveň ukládána do pole *"allDynamicStartTimeTakt"* a následně proměnná *"allDynamicStartTime"* je doplněna také o informaci *"currentTimeSeconds"*, která obsahuje reálný čas v sekundách od kdy je dané znaménko aktivní. Tento čas je již díky funkci *"TimeToSeconds"* přepočítán i se zohledněním tempa dané osnovy.

Informace o crescendu/decrescendu jsou uloženy do proměnné *"allHairpin"* a *"allHairpinStartTime"*. Do obou jsou uložena stejná vstupní data. Tato data, zvláště pak pro *"allHairpinStartTime"* jsou nadále ještě upravována a přepisována.

Legato, glissando, trioly

V případě ukládání nadefinovaných informací jako je legato, glissando nebo trioly pracuje kód se zadáváním informací v podobě čísel k notám, které určují, které dvě noty mají být daným stylem spojeny. Výsledkem všech těchto případů v souboru *otazky.js* je pole obashující několik dalších pod polí, rozřazených do připadajících osnov a taktů, kde v poslendím poli, které připadá taktům jsou informace k jednotlivým notám. V případě, že v souboru vstupních dat je pro danou notu informace zadána, je předáno dané čislo do tohoto ukazatele. Pokud informace není předávaná, je uložena hodnota *"null"*.

Transpozice

Funkce *"transpose"* umožňuje transpozici not o určitý počet půltónů nahoru nebo dolů. Pro každou notu ve vstupním poli se určí směr a velikost transpozice. Nota se rozdělí na základní tón a oktávu a podle toho, zda obsahuje křížek nebo béčko, se najde její index v odpovídajícím poli not. Tento index se upraví podle směru a velikosti transpozice.

Pokud transpozice přesáhne rozsah dostupných not, index se upraví a oktáva se podle potřeby zvýší nebo sníží. Nově vypočtená nota se sestaví z transponovaného názvu noty a upravené oktávy. Transponované noty se uloží do pole, které funkce na závěr vrátí.

3.2.4 drawButtons.js

V souboru *drawButtons.js* probíhá definice tlačítek sloužících k různým typům přehrávání. Kód umožňuje přehrávat následující varianty:

- Každá osnova samostatně
- Přehrání osnov spojených svorkou - např. pro piano
- Přehrání všech osnov

Funkce je volána na konci funkce *"displayCurrentQuestion"* napsané v souboru *otázky.js*. Funkce hledá jednotlivé nadefinované elementy v souboru *projekt.html*. Jednotlivé moduly jsou také nejdříve vyčistěny vždy při volání této funkci, díky čemuž jsou po změně stránky s notovými osnovami vykresleny znovu tlačítka v počtu odpovídajícím vstupnímu zadání. Funkce také vypisuje názvy jednotlivých sekcí tlačítek. Pro jednotlivé sekce jsou volány funkce uložené v souboru *drawButtonsFnc.js*, které obsahují další logiku potřebnou pro vykreslování tlačítek, volbu nástroje nebo spouštějí navazující další funkce.

3.2.5 drawButtonsFnc.js

Tento soubor obsahuje funkce navazující na soubor *drawButtons.js*, který přebírá vyčištěné kontejnery a vykresluje do nich tlačítka dle potřebné varianty. K již zmíněným variantám (viz. kap. 3.2.4) připadají tyto funkce:

- Každá osnova samostatně - *"buttonsForEach"*
- Přehrání osnov spojených svorkou - *"buttonsForBrace"*
- Přehrání všech osnov - *"playAll"*

Tyto funkce obsahují stejnou logiku pro přehrávání a zvýraznění přehrávaných not. Kód nadále volá funkci *"prehrajZvuk"* podrobněji rozebranou v kapitole 3.2.10 a funkci *"drawDynamicNotes"*.

funkce "drawDynamicNotes"

Funkce *"scheduleFunction"* je navržena tak, aby prováděla akce v určitých časových intervalech podle předem definovaného harmonogramu. Tento harmonogram je uložen v poli *"vysledneRealTime"*, kde každá hodnota představuje zpoždění v sekundách. Funkce postupně provádí úkoly, jako je aktualizace hudební notace a dynamiky, a poté se sama volá rekurzivně.

Funkce začíná kontrolou, zda aktuální indexu (*"currentIndex"*) je menší než délka pole *"vysledneRealTime"*. Pokud ano, pokračuje dále. Poté se vypočítá zpoždění *"delay"* pro aktuální časový úsek, který se získá z pole *"vysledneRealTime"*. Tento časový úsek se vynásobí 1000, aby se převedl na milisekundy, protože *"setTimeout"* pracuje s milisekundami. Proměnná *"aktualniCas"* se aktualizuje přičtením zpoždění k předchozímu času. Následně se zjišťuje, které noty jsou aktuálně aktivní, pomocí funkce *"zjistiHodnotu"*, která bere v úvahu aktuální čas a pole *"secteneRealTime"*.

Časovač *"setTimeout"*, zavolá funkci po uplynutí vypočítaného zpoždění. Uvnitř časovače se provádí několik akcí. Pomocí funkce *"updateAllOsnovy"* jsou aktualizovány jednotlivé osnovy, přesněji tedy noty, které jsou zpracovány stejně jako v souboru *otazky.js* s rozdílem, že pro noty, které jsou v časovači aktivní dochází ke změně barvy.

V časovači dochází také ke změně dynamiky, pokud je dynamika nadefinována. Nejprve se kontroluje, zda existují definované začátky dynamiky v poli *"allDynamicStartTime"*. Pokud ano, určuje se, zda je aktuální čas v rozsahu nějakého dynamického znaménka (crescenda nebo decrescenda). Pokud je aktuální čas v rozsahu takového znaménka, vypočítá se aktuální hlasitost pomocí lineární interpolace mezi počáteční a koncovou hodnotou hlasitosti pro dané znaménko. Jinak se hlasitost nastaví na základě aktuální dynamiky.

Na závěr časovače je volána funkce *"drawCombined"* k opětovnému vykreslení notace s barevně upravnenými notami na základě aktuálních parametrů přehrávání.

Poté se aktualizuje pořadí not pro všechny aktivní noty. Na samotný závěr časovače dochází k navýšenní aktuální indexu a funkce *"scheduleFunction"* se volá znovu rekurzivně s aktuálními parametry, což způsobí další iteraci procesu.

3.2.6 timeFnc.js

V souboru *timeFnc.js* jak už název napodívá, jsou uloženy funkce související s úpravou časových parametrů. Funkce nejdříve upravuje časy začátků a délek not. Následně pracuje se zpracováním časových údajů, pro dynamické znaménka crescendo/ decrescendo. Zpracování všech těchto údajů probíhá ve funkci *"updateTime"*, která je volána v soubouru *otazky.js*. Výsledkem funkce *"updateTime"* jsou tyto proměnné.

- allStartTimes (kap. 3.2.6)
- vysledeneRealTime (kap. 3.2.6)
- secteneRealTime (kap. 3.2.6)
- hairpinRealTimes (kap. 3.2.6)

allStartTimes

Vstupní proměnnou tohoto kódu je proměnná *"allStartTimes"*. Tato proměnná je pole rozdělené do polí, dle jednotlivých osnov. Pokud osnova obsahuje více hlasů, tak i toto pole pro osnovu je rozděleno na další podpole.

Výpis 3.5: Příklad obsahu proměnné 'allStartTimes'

```
[                                                                    1
    [                                                                2
        [0, 0.5, 1, 0.5],   // 1. osnova - 1. hlas                   3
        [0, 0.25, 1, 0.25, 0.25, 0.5, 0.25] // 1. osnova - 2.        4
            hlas
    ],                                                               5
    [0, 2]  // 2. osnova                                             6
]                                                                    7
```

Kód na výpisu 3.5 ukazuje, jak vypadá proměnná *"allStartTimes"*. První hodnota každého pole je 0, což označuje časový počátek notové osnovy. Všechny ostatní hodnoty obsahují již přepočtené délky, dle závislosti na tempu jednotlivých not.

vysledeneRealTime

Výsledem a tedy obsahem pole *"vysledneRealTime"* po volání příslušných funkcí je pole, které obsahuje rozdíly mezi notami. Pokud se jedná o osnovu pouze s jedním hlasem, tak je toto pole totožné. Pakliže se nachází v osnově více hlasů, tak jsou

hodnoty přepočteny tak, aby hodnoty udávali dobu, za jak dlouho zazní jiná nota nehledě na to ve kterém hlase se nachází. Takto přepočtené hodnoty z výpisu 3.5 jsou uvedeny ve výpisu 3.6.

Výpis 3.6: Příklad obsahu proměnné 'vysledneRealTime'

```
[                                                                    1
    [0, 0.25, 0.25, 0.75, 0.25, 0.25, 0.25, 0.25, 0.25],            2
       // 1. osnova
    [0, 2]  // 2. osnova                                            3
]                                                                    4
```

secteneRealTime

Obsahem tohoto pole jsou sečtené hodnoty jednotlivých hlasů proměné *"allStartTimes"*. Na výpisu 3.7 lze pozorovat takto upravené hodnoty oproti výpisu 3.5.

Výpis 3.7: Příklad obsahu proměnné 'secteneRealTime'

```
[                                                                    1
    [                                                                2
        [0, 0.5, 1.5, 2],        // 1. osnova - 1. hlas             3
        [0, 0.25, 1.25, 1.5, 1.75, 2.25, 2.5]   // 1. osnova        4
           - 2. hlas
    ],                                                               5
    [0, 2]  // 2. osnova                                             6
]                                                                    7
```

hairpinRealTimes

Kód zpracovává vstupní hodnoty *"allHairpinStartTime"* a upravuje hodnoty tak, aby jejich výsledkem bylo opět pole rozdělené do polí dle osnov. Funkce upravující vstupní hodnoty pracuje s proměnnou *"secteneRealTime"*, tempem a počtem dob v taktu.

Na výpisu 3.8 lze vidět struktura odsahu proměnné *"hairpinRealTime"*. Výsledkem pro každý znak je pole označující, zda se jedná o crescendo (*"C"*) nebo decrescendo (*"D"*). Dále obsahuje časové údaje, připadající notám v prvním hlase, kdy daný znak začíná být aktivní, a zároveň kdy účinnost daného znaku končí.

Výpis 3.8: Příklad obsahu proměnné 'hairpinRealTime'

```
['C',0.5, 3]                                                         1
```

Další funkce

Funkce *"convertDurationToBeatCount"* převádí délku noty na počet úderů v souladu se zadanou vstupní hodnotou taktu pro danou osnovu.

Výpis 3.9: Funkce 'convertDurationToBeatCount'

```
function convertDurationToBeatCount(duration, BeatNumber) {
    const durationMap = {
        'w': BeatNumber,
        'h': BeatNumber/2,
        'q': BeatNumber/4,
        '8': BeatNumber/8,
        '16': BeatNumber/16,
        '32': BeatNumber/32
    };
    return durationMap[duration];}
```

Pomocí funkce *"updateCurrentTime"* dochází k aktualizaci času na základě délky noty, aktuálního času a možných dodatečných parametrů not ovlivňujících délku dané noty. Například pro notu s tečkou se přidá k aktuálnímu času kromě její samotné délky navíc i polovina její délky, což je zapříčiněno právě tečkou za notou.

Poslední funkcí v tomto souboru je *"convertTempoToQuarterNote"*, která převádí tempo vztažené k jiné než čtvrťové notě na tempo, které odpovídají právě notě o této délce.

3.2.7 drawCombined.js

Tento soubor obsahuje kód, který slouží k vykreslení notové osnovy, not a všech dodatečných grafických znaků. Základem pro vykreslování je knihovna VexFlow (viz. kap. 2.3.1). Kód převážně pracuje již s upravenými proměnnými do potřebných formátů.

Na začátku je vždy vyčištěn html contejner poskytující prostor pro vykreslení notových záznamů. Při vytváření kontextu v rámci knihovny VexFlow, jsou využity parametry zadané uživatelem, sloužící pro definici velikosti rozhraní do kterého se bude notová osnova vytvářet. To umožňuje uživateli přizpůsobit si pole pro vytváření osnov dle vlastní potřeby.

Vykreslení prázných osnov

Limitací knihovny VexFlow je vykreslování více taktů za sebou. Knihovna je navržena tak, že sama o sobě dokáže vykreslit pouze jeden takt. Kód pro vykreslení

notových osnov, repsektive pro vykreslení více taktů pracuje tak, že vykresluje moduly knihovny VexFlow označené jako *"Stave"* postupně za sebe. Pro srozumitelnější porozumění je potřeba tedy modul *"Stave"* brát jako modul pro vykreslení jednoho taktu a ne osnovy.

Celé vykreslení notových osnov a not je uvnitř cyklu, který pracuje s celkovým počtem notových osnov pod sebou. Kód tedy pracuje na principu, kdy vytvoří kompletní první osnovu od notových linek, různých symbolů až po noty samotné a až poté pokračuje k další osnově.

Uživatel má při zadávání vstupních informací možnost zvolit si následující parametry z hlediska prostorového umístění osnov v rámci nadefinovaného rozhraní pro zobrazování notových osnov:

- "gapBetweenBorder"
  - velikost mezery mezi levým okrajem rozhraní a začátkem prvních taktů osnov
- "gapBetweenStaves"
  - velikost vertikální mezery mezi jednotlivými osnovami
- "firstBarLength"
  - délka prvního taktu každé osnovy
- "barLength"
  - délka všech ostatních taktů

Prvnímu taktu každé osnovy je přidáno označní klíče, taktu a předznamenání pro danou osnovu. Pokud je nadefinovaný název dané osnovy, většinou tedy označení, který nástroj připadá danému řádku, tak je zobrazen vlevo před prvním taktem. K prvnímu taktu první osnovy je navíc přiřazen identifikátor tempa. Tyto vytvořené osnovy jsou uloženy do pole *"dlouheOsnovy"*.

Vykreslení not

Kód je napsán v cyklech tak, aby postupně procházel nižší a nižší vrstvu a dostal se tedy od jednotlivých osnov přes její hlasy až po takty těchto hlasů, které jsou nejnižší úrovní, protože není potřeba přislupovat již k samotným notám vrámci taktů kvůli zadávání a vykreslování not knihovny VexFlow právě po taktech.

Stěžejní pro vykreslování not do taktů je logika, která přiřazuje noty rozdělené v taktech již vykresleným prazdným notovým osnovám.

Kód na výpisu 3.10 je napsát tak, aby reflektoval počet jednotlivých osnov. Dochází zde zejména k definici, respektive úpravě proměnné *"staveBarIndex"*. Tato proměnná je navržena tak, aby obsahovala hodnotu do jakého taktu osnovy mají být noty v taktu vykresleny. Pro první osnovu je tento index shodný s vykreslovanými takty not. Pro následující osnovy, jsou zde započteny právě délky předcházejících

osnov tak, aby takty odpovídaly osnovám. Zjednodušeně můžeme tedy pozorovat, že rozdíl je v ukládání prázdných osnov a uložení not v taktech. Zatímco prázdné takty jsou ukládány za sebe do jednoho pole nehledě na příslušné osnovy, noty v taktech jsou polem v poli hlasů, které je v poli notových osnov a kvůli tomu je tedy nutné přistupovat k těchto dvou proměným odlišnými indexy.

Výpis 3.10: Vykreslení osnov a hlasů

```
if(staveIndex == 1) {                                                    1
    staveBarIndex = barIndex;                                            2
} else {                                                                 3
                                                                         4
    let pocetOsnovX = 0;                                                 5
                                                                         6
    for (let r = 2; r <= staveIndex; r++){                               7
        pocetOsnovX += pocetJednotlivychOsnov[r-2];                      8
    }                                                                    9
    staveBarIndex = barIndex + pocetOsnovX;                              10
}                                                                        11
                                                                         12
Vex.Flow.Formatter.FormatAndDraw(ctx, dlouheOsnovy[                      13
   staveBarIndex], voice[barIndex]);
                                                                         14
if (dynamics != null){                                                   15
    Vex.Flow.Formatter.FormatAndDraw(ctx, dlouheOsnovy[                  16
       staveBarIndex], dynamics[barIndex]);
}                                                                        17
```

Z řádků 13 a 16 výpisu 3.10 je zřejmé, že vykreslení dynamických textových znamének funguje na stejném principu jako vykreslování not v taktech.

Vykreslení ligatur a glissand

Pro vykreslení ligatur a glisand se používají přislušné objekty knihovny VexFlow. Využívá se již zpracovaných vstupnních dat v potřebném formátu a pouze se implementují tyto proměnné do příslušných objektů.

Vykreslení dynamických znamének crescendo/decrescendo

Pokud jsou tato znaménka zadána, tak kód načte z dat v připraveném formátu takt a příslušnou notu pro kterou je toto dynamické znaménko definované, jak pro počáteční notu, tedy notu od které je znaménko aktivní, tak notu koncovou. Na

základě předaného typu znaménka, tedy hodnoty *"C"* nebo *"D"* se použije příslušný typ objektu z knihvony VexFlow *"StaveHairpin"*.

Vykreslení vertikálních svorek pro osnovy

Jednotlivé notové osnovy lze spojit různými linkami. Projekt umožňuje spojit jednotlivé osnovy jednoduchou čárou nebo dojitou. Také zde může být implementována hranatá nebo kulatá svorka. Kód funguje na základně vstupních dat uživatele, které udávají které dvě osnovy mají být takto spojeny. Pro každé takovéto spojení je možné také vykreslit příslušící text.

3.2.8 convertor.js

Soubor convertor.js obsahuje několik funkcí, které převádí vstupní data do formátu vhodného pro přehrávání pomocí knihovny Tone.js.

Funkce *"Convertor"* převádí všechny noty včetně jejich atributů na formát použitelný pro Tone.js. Nejprve vyrovná pole not do jednoho rozměru a poté mapuje jednotlivé noty do nového formátu. Převede názvy not na formát vhodný pro Tone.js, délky not na odpovídající formát a počáteční časy na sekundy. Zpracovává také pauzy.

Funkce *"NameToTone"* převádí notový zápis ve formátu "nota/oktáva"(například "c/4") na formát použitelný pro Tone.js jako je například "C4".

Funkce *"NameToDrum"* převádí formát vstupních not pro bicí soupravu na noty použitelné pro Tone.js. Pomocí switch konstrukce vrací odpovídající notu pro daný nástroj bicí soupravy. Pokud název není rozpoznán, vrací výchozí hodnotu "E3"připadající malému bubnu.

Funkce *"TimeToSeconds"* převádí čas začátku noty na sekundy i se zohledněním tempa skladby. Výsledkem je čas v sekundách.

$$T = \frac{60}{tempo} \cdot nota \tag{3.1}$$

T je čas v sekundách. Tempo, v angličtině označované jako BPM[1], je počet dob za minutu. Hodnota noty je udávaná v taktech.

Funkce *"DurationToTone"* převádí délku noty na formát použitelný pro Tone.js. Používá mapování, kde zkratky délek not jsou převedeny na odpovídající formáty pro Tone.js. Tabulka 3.2 popisuje odpovídající hodnoty mezi různými knihovnami.

[1]BPM - Beats per minute

Srovnání formátu zobrazování not pro knihovny VexFlow a Tone.js		
Nota	Formát VexFlow	Formát Tone.js
Celá	'w'	'1n'
Půlová	'h'	'2n'
Čtvrťová	'q'	'4n'
Osminová	'8'	'8n'
Šestnáctinová	'16'	'16n'
Třicetidvoutinová	'32'	'32n'

Tab. 3.2: Srovnání formátu zobrazování not pro knihovny VexFlow a Tone.js

3.2.9 nastroj.js

Soubor *nastroj.js* je určen k inicializaci a výběru různých hudebních nástrojů pomocí knihovny Tone.js. Obsahuje definice samplerů pro následující nástroje:

- Piano
- Syntetizátor
- Kytara
- Housle
- Violloncello
- Kontrabas
- Safoxon
- Xylofon
- Varhany
- Harfa
- Basa
- Bicí

Každý sampler je inicializován se sadou samplů odpovídajících tónům jednotlivých nástrojů a využívá knihovny Tone.js, kde je pomocí těchto vzorků možné vytvořit sampler.

Po načtení dokumentu je k prvku *"instrumentSelect"* přidán event listener, který umožňuje uživateli vybrat nástroj z nabídky.

Nachází se zde také funkce *"updateSynths"*, která přijímá vybraný nástroj a vrací příslušný sampler, čímž umožňuje přepínání mezi různými nástroji a jejich zvuky v aplikaci.

3.2.10 prehrajZvuk.js

Soubor *prehrajZvuk.js* obsahuje stejnojmennou funkci, která se stará o přehrávání not pomocí knihovny Tone.js. Tato funkce přijímá parametry *"allNotes"* a *"synth"*.

Výpis 3.11: Funkce 'prehrajZvuk'

```
function prehrajZvuk(allNotes, synth) {
    let now = Tone.now();
    allNotes.forEach((NOTE, index) => {
        const { note, noteDuration, startTimeInSeconds, rest
           } = NOTE;
        if (!rest) {
            const noteStartTime = startTimeInSeconds + now;
            synth.triggerAttackRelease(note, noteDuration,
               noteStartTime + index * 0.001);
        }
    });
}
```

Tato funkce je uvedena ve výpisu 3.11.

Na začátku funkce je získán aktuální čas pomocí *"Tone.now"*, který vrací časový údaj v sekundách od startu aplikace. Poté funkce prochází každou notu v poli *"allNotes"* pomocí metody *forEach*. Pro každou notu jsou z objektu dekonstruovány následující vlastnosti: *"note", "noteDuration", "startTimeInSeconds"* a *"rest"*.

Funkce dále kontroluje, zda hodnota *rest* je *"false"*, což znamená, že se nejedná o pauzu. Pokud je tato podmínka splněna, tak se pro každou jednotlivou notu vypočítá časový okamžik, kdy má nota začít hrát.

Nakonec je použita metoda *"synth.triggerAttackRelease"* z knihovny Tone.js (viz. kap. 2.4.1). Tato metoda spustí přehrání noty s danou výškou a délkou trvání v určený čas. K času *"noteStartTime"* se navíc přičte index noty násobený malým zpožděním 0.001 sekundy, což zajišťuje, že jednotlivé noty nezačnou hrát přesně ve stejný okamžik obdobně, jako to funguje například u standartu MIDI.

4 Způsob implementace a příklad zpracování učebnice

4.0.1 Způsob implementace do webové stránky

Pro vložení celého funkčního bloku pro přehrávání a zobrazování osnov je možné využít tagu *"iframe"*, pomocí kterého lze přidat tento blok na svou HTML stránku (viz. výpis 4.1). V parametru *"src"* je napsána cesta k souboru *"projekt.html"*, který je uložen mezi zdrojovými kódu frameworku, který byl navržen a popsán v této práci. Pro přiřazení souboru ze kterého se načítají vstupní data se používá *"json="* a název json souboru. Pokud by zadaný soubor nebyl definován nebo by celý identifikátor *"json="* chyběl, tak kód bude hledat a používat soubor *"input_data.json"*

Výpis 4.1: Implementace frameworku

```
<iframe src="projekt.html?json=input_data.json"></iframe>
```

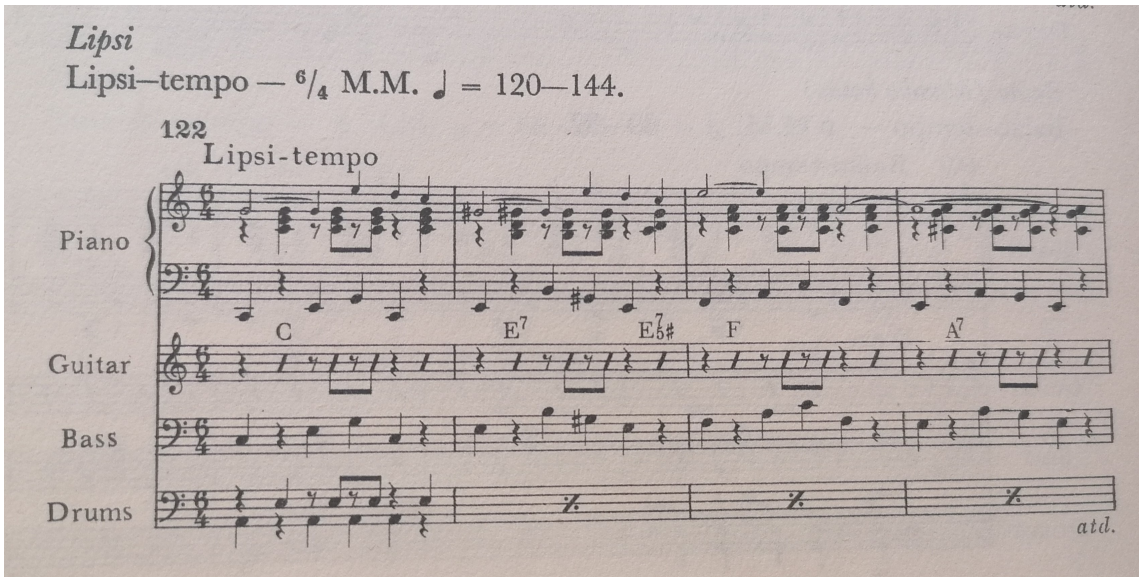
4.0.2 Zpracování záznamů z učebnice

Tato práce zpracovává tři příklady různých notových osnov. Pro porovnání je v následující kapitole uveden vždy příklad fotografie z knižní učebnice a snímek obrazovky, jak by takováto učebice mohla být zpracována pomocí navržené aplikace. Výhodou interaktivní webové verze je možnost přehrání jedné nebo všech notových osnov osnov, a zároveň s tím spojené zobrazení aktuálně přehrávaných not. Toto je nádstavba oproti knižní učebnici, kterou vzhledem k edukativním účelům měla tato práce poskytnout. Tyto příklady jsou definovány v příloze *input_data.json*.

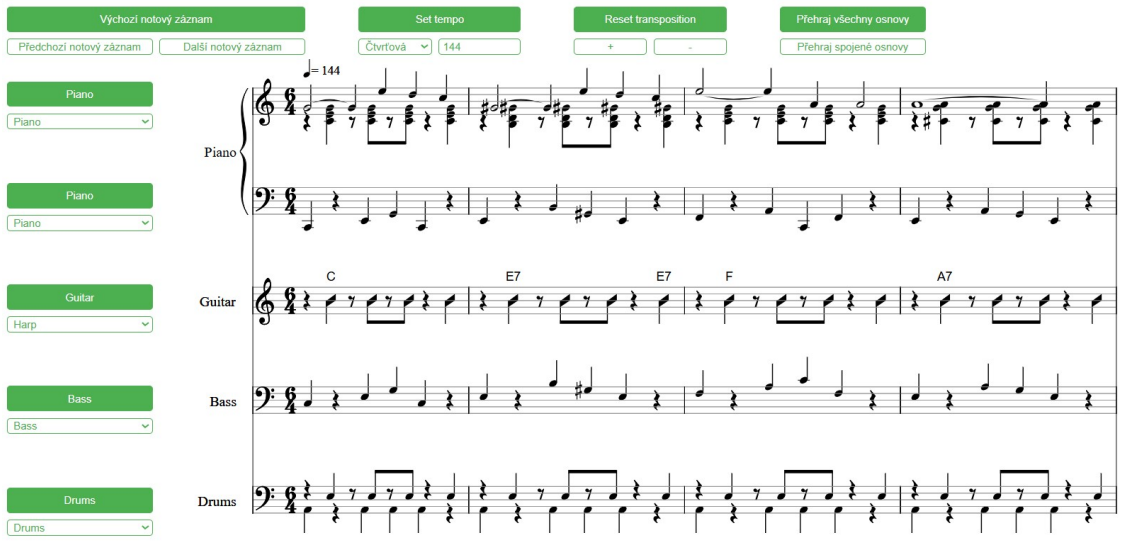
První dva příklady kopírují záznamy z knihy Zdeňka Krotila *Aranžování pro moderní taneční orechstr* [27]. Na obrázcích 4.1 a 4.3 jsou fotografie záznamů z učebnice. Na obrázcích 4.2 a 4.4 můžeme vidět tyto části učebnice zpracovány do aplikace.

Poslední příklad je zpracován z knihy *Základy aranžování moderní populární hudby*, kterou napsal Vlastimil Hála [27]. Porovnání zpracování těchto příkladů je uvedeno na fotografiích 4.5 a 4.6.

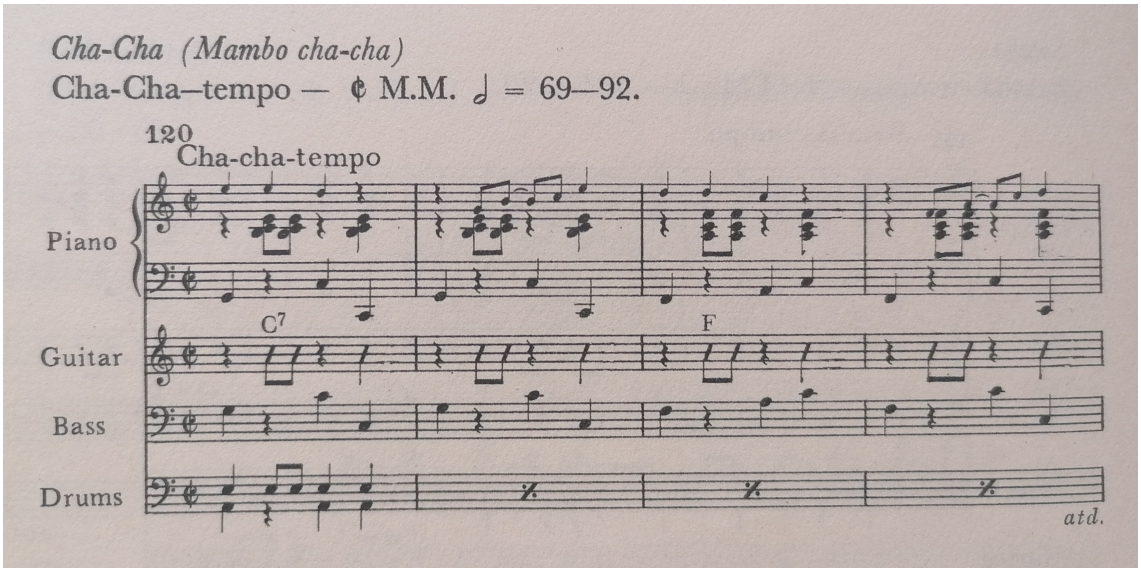
Kapitola pojednává o vedení tří hlasů na příkladadu u saxofonů. Výhodou webové aplikace je, že v případě saxofonů a jím podobným nástrojům je možné využít také transpozice pro osnovu tak, aby byl zohledněn fakt, že saxofony zní jinak něž se píšou. Zároveň pokud uživatel bude chtít, tak může využít více osnov pod sebou například pro demonstraci, jak se saxofon píše do notového záznamu X jaké noty notového záznamu hraje (viz. kap. 3.2.3).



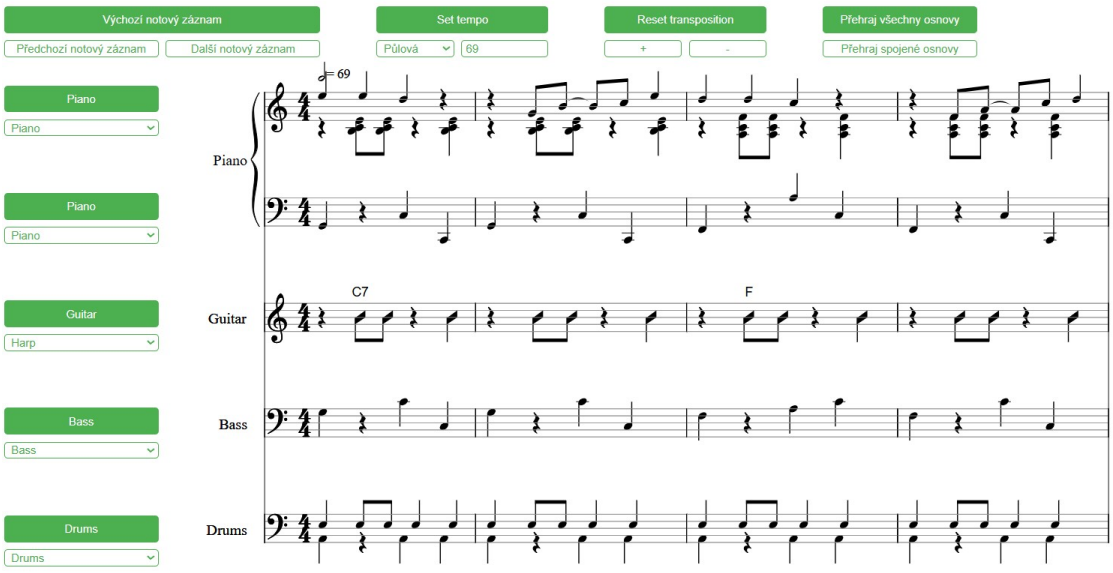
Obr. 4.1: Příklad záznamu Lipsi z učebnice [27]



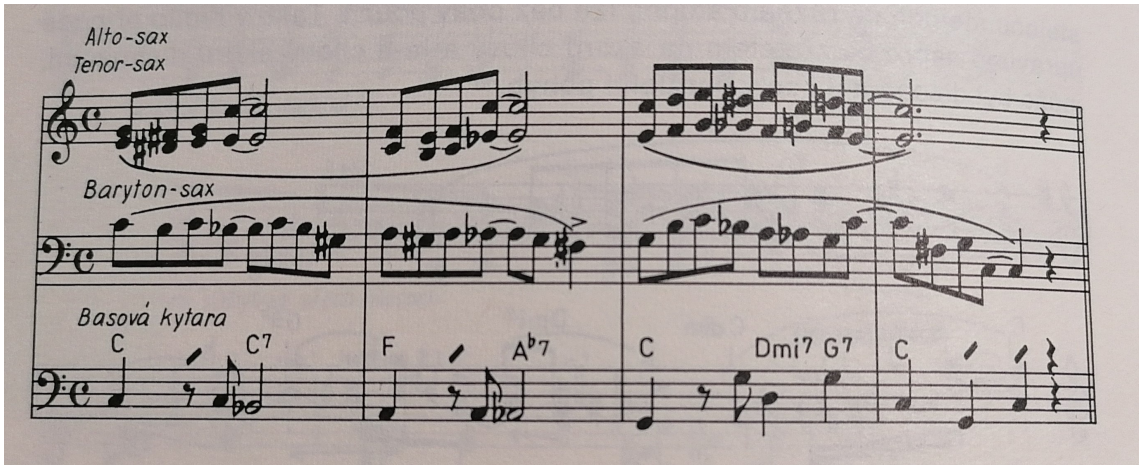
Obr. 4.2: Příklad záznamu Lipsi z webové stránky



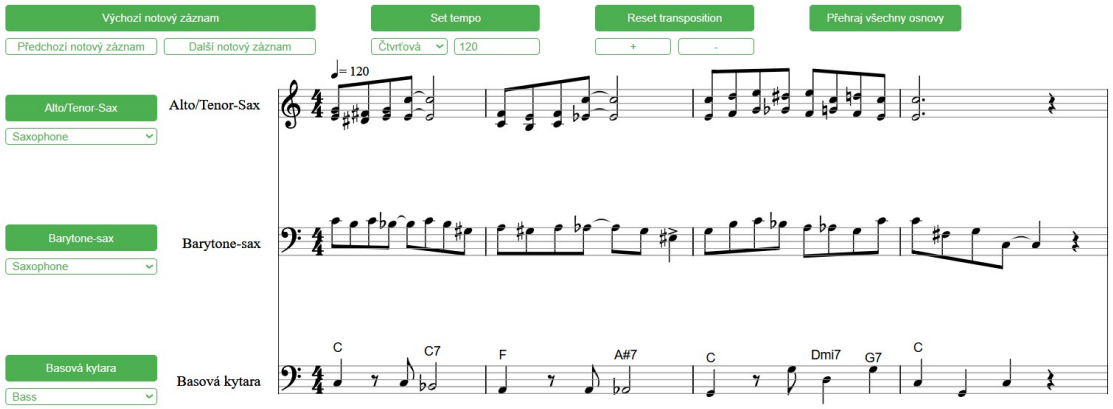
Obr. 4.3: Příklad záznamu Cha-Cha z učebnice [27]



Obr. 4.4: Příklad záznamu Cha-Cha z webové stránky



Obr. 4.5: Příklad záznamu pro saxofony z učebnice [28]



Obr. 4.6: Příklad záznamu pro saxofony z webové stránky

5 Limitace

Největší limitací je zobrazení ligatur. Program zpracovává pouze možnost zadání ligatur vrámci jednoho taktu a neobsahuje funkcionalitu pro vykreslení ligatur mezi různými takty.

Program je také omezen možnostmi, které nabízí pro zobrazování různých dodatečných znaků, čar a symbolů. Program poskytuje základní možnosti, ale do budoucna pro komplexnější využití by mohl být kód rozšířen o další symboly.

Pro ideální vykreslování by se vývoj do budoucna mohl zaměřit také na zobrazení více hlasů tak, aby v pomyslných sloupcích nad sebou byly odpovídající noty.

Jednou z limitací zobrazování je, že přehrávané noty se mění s každou změněnou notou. Přesněji to znamená, že pro vícehlasé záznamy se může stát, že délka noty v jednom hlase bude co se týče délky barevného zobrazení ovlivněna notami v druhém hlase. Pokud tedy například v jednom hlase bude nota celá a v druhém hlase dvě noty půlové, tak nota celá bude ovlivněna druhým hlasem a bude barevně zvýrazněna pouze o době noty půlové.

V případě dynamiky byl zvolen postup, který je dostačující pro přehrání v případě jedné osnovy. Hlasitost je měněna pro celý zvukový výstup a né pro jednotlivé noty. Proto jsou hlasitosti pro příslušná dynamická znaménka realizována pouze v případě přehrávání jedné osnovy a při přehrávání více osnov jsou tata dynamická znaménka opomíjena.

Při zvukovém přehrávání také chybí funkcionalita, která by respektovala artikulační znaménka nebo ligatury.

Program zároveň spolehá na správně zadané vstupní data a není ošetřen pro případy špatné formy těchto vstupních dat. Tento problém by při navazujícím rozšíření programu mohl být také ošetřen.

Závěr

Cílem této práce bylo prozkoumat možnosti existujících frameworků v oblasti zobrazování notové osnovy a přehrávání zvuků a následně navrhnout a implementovat řešení pro zobrazování a přehrávání notových záznamů na webových stránkách.

Práce byla rozdělena do pěti kapitol. První kapitola této práce se důkladně věnovala základům hudební teorie a historii notace. Popisovala vývoj notace od jejích prvních forem, které nebyly ještě zcela strukturované, až po formu, kterou známe dnes jako pětilinkový systém. Dále se kapitola zabývala dynamickými a artikulačními znaménky a druhy klíčů. Kromě toho se v první kapitole podrobněji rozebíraly programovací jazyky, které byly využity při implementaci aplikace HTML a JavaScript. Dále byly představeny formáty pro reprezentaci hudebních dat MusicXML a JSON.

Druhá kapitola byla zaměřena na komerční programy zabývající se zobrazováním a přehráváním hudební notace. Byly zde porovnány možnosti zobrazování programů Dorico, Finale, MuseScore a Sibelius. Kromě toho byly rozebrány open-source alternativy jako MuseScore, LilyPond a MusiXTeX. Hlavní pozornost byla věnována frameworkům VexFlow a ABC.js, které nabízejí možnosti zobrazování notové osnovy přímo na webových stránkách a frameworku Tone.js, který slouží k přehrávání a generování zvukových signálů. Byly rozebrány jejich funkcionality, vlastnosti a vhodnost pro konkrétní účely této práce. Závěr kapitoly se věnoval MIDI standardu.

Třetí kapitola byla soustředěna na samotnou realizaci projektu. Zde byly představeny vybrané frameworky pro implementaci zobrazování a přehrávání notových záznamů na webových stránkách, konkrétně VexFlow a Tone.js. V kapitole byl obsažen detailní popis procesu implementace funkcí a struktury projektových souborů. Byl zde uveden diagram posloupnosti souborů, který ilustruje celkovou architekturu aplikace od zobrazování not po jejich přehrávání a zvýrazňování v čase. Dále byly popsány technické detaily implementace, včetně práce s notovými daty a jejich transformací pro potřeby zobrazování a přehrávání.

Čtvrtá kapitola se soustředila na integraci vytvořené aplikace do webových stránek a prezentovala příklady reálného použití v praxi.

Poslední kapitola byla věnována limitacích aplikace a nastínila možné směry budoucího vývoje.

Výsledkem této práce je vlastní softwarový nástroj umožňující zobrazování a přehrávání notových záznamů na webových stránkách. Tento nástroj byl navržen pro edukativní účely a má za cíl podpořit lepší porozumění notových zápisů.

Literatura

[1] STRAYER, H.R., *From neumes to notes: The evolution of music notation. Musical Offerings* 4(1), p.1., 2013

[2] MUSCIANO, C. a KENNEDY, B. *HTML & XHTML: The Definitive Guide: The Definitive Guide.* . "O'Reilly Media, Inc.", 2002.

[3] WILTON, P. *Beginning JavaScript* "John Wiley & Sons", 2004.

[4] DIAZ, D. a HARMES, R. *Pro JavaScript design patterns* "Apress", 2008.

[5] HAROLD, E. R. a MEANS, W. S. *XML in a nutshell: a desktop quick reference.*. "O'Reilly Media, Inc.", 2004.

[6] GOOD, M. D. *Using MusicXML 2.0 for Music Editorial Applications* [Online]. 2009. Dostupné z URL: `https://michaelgood.info/publications/music/using-musicxml-2-0-for-music-editorial-applications/`. [cit. 2023-11-22].

[7] GOOD, M. D. *MusicXML: An internet-friendly format for sheet music* [Online]. 2001. Dostupné z URL: `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5617972667ff794da79a4cbb6b985e85f8487ddd`. [cit. 2023-11-22].

[8] MARRS, T. *JSON at work: practical data integration for the web* "O'Reilly Media, Inc.", 2017.

[9] NURSEITOV, N., PAULSON, M., REYNOLDS, R., & IZURIETA, C. *Comparison of JSON and XML data interchange formats: a case study.* "Caine, 9, 157-162.", 2009.

[10] ZENKL, Luděk. *ABC hudební nauky*, 8. vydání (v Bärenreiter Praha vydání druhé - dotisk). Praha: Bärenreiter, 2014. ISBN 978-80-86385-21-1.

[11] RETTINGHAUS, K. *Comparison of MusicXML export capabilities of different scorewriters.* 10.17613/5qq5-jn04. 2023.

[12] PENA DOCAMPO, D. *MIDI-based music score editor*, 2020.

[13] NIENHUYS, H.W., NIEUWENHUIZEN, J., *LilyPond, a system for automated music engraving.* In Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003) (Vol. 1, pp. 167-171). Firenza: Tempo Reale. 2003, May.

[14] TAUPIN, D., MITCHELL, R., & EGLER, A. *MusiXTEX. Using TEX to write polyphonic or instrumental music.* TUGboat, 14(3), 212-220. 1993

[15] CHEPPUDIRA, M. M. *VexFlow - HTML5 Music Engraving* [Online]. 2010. Dostupné z URL: `https://www.vexflow.com/`. [cit. 2023-11-21].

[16] *VexFlow2 - JavaScript Music Notation and Guitar Tab* [Online]. Dostupné z URL: `https://www.vexflow.com/tests/`. [cit. 2023-11-22].

[17] *OpenSheetMusicDisplay*[Online]. Dostupné z URL: `https://github.com/opensheetmusicdisplay/opensheetmusicdisplay`. [cit. 2023-11-22].

[18] CHEPPUDIRA, M. M. *VexFlow - Javascript Guitar Tablature* [Online]. 2012. Dostupné z URL: `https://vexflow.com/vextab/`. [cit. 2023-11-22].

[19] ROSEN, P. *Javascript library for instŕting music in the browser* [Online]. 2023. Dostupné z URL: `https://paulrosen.github.io/abcjs/`. [cit. 2023-11-22].

[20] ROSEN, P. a BAUMGARTNER, C. *Javascript library for instŕting music in the browser - Synthesized Sound* [Online]. 2023. Dostupné z URL: `https://paulrosen.github.io/abcjs/audio/synthesized-sound.html`. [cit. 2023-11-22].

[21] *Tone.js* [Online]. Dostupné z URL: `https://tonejs.github.io/`. [cit. 2023-11-22].

[22] *Tone.js - GitHub* [Online]. 2020. Dostupné z URL: `https://github.com/Tonejs/Tone.js`. [cit. 2023-11-22].

[23] *Tone.js - Documentation* [Online]. Dostupné z URL: `https://tonejs.github.io/docs/14.7.77/index.html`. [cit. 2023-11-22].

[24] DANIGB *Note-Parser* [Online]. 2017. Dostupné z URL: `https://github.com/danigb/note-parser`. [cit. 2023-11-22].

[25] ROTHSTEIN, J. *MIDI: A comprehensive introduction* 7. vydání. "AR Editions, Inc."1995.

[26] *WEBMIDI.js* [Online]. Dostupné z URL: `https://webmidijs.org/`. [cit. 2023-12-05].

[27] KROTIL, Zdeněk *Aranžování pro moderní taneční orchestr.* 1. vyd.. Praha : SNKLHU, 1960.

[28] HÁLA, Vlastimil. *Základy aranžování moderní populární hudby.* 2., opr. vyd. Praha: Panton, 1986. ISBN (Brož.):.

Seznam symbolů a zkratek

ADSR	Attack Decay Sustain Release
ASCII	American Standard Code for Information Interchange
BPM	Beats Per Minute
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
MIDI	Musical Instrument Digital Interface
MIT	Massachusetts Institute of Technology
NIFF	Notation Interchange File Format
PHP	Hypertext Preprocessor
SVG	Scalable Vector Graphics
W3C	World Wide Web Consortium
WAW	Waveform Audio File Format
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

Obsah elektronické přílohy

- `/` ..... kořenový adresář přiloženého archivu
  - `zdrojove_kody` ..... soubory se zdrojovými kódy
    - `convertor.js`
    - `drawButtons.js`
    - `drawButtonsFnc.js`
    - `drawCombined.js`
    - `examples.json`
    - `input data.json`
    - `nastroj.js`
    - `otazky.js`
    - `prehrajZvuk.js`
    - `projekt.html`
    - `styles.css`
    - `timeFnc.js`
  - `Manuál pro zadávání` ..... Návod pro zadání notových záznamů
  - `Apache24` ..... Localhost pro spuštění aplikace
  - `Návod ke spuštění` ..... Návod pro spuštění projektu pomocí localhostu