FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Rozvoj webové aplikace Kytarový zpěvník
Student:	Jakub Jabůrek
Vedoucí:	Ing. Zdeněk Rybola, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Práce se zaměřuje na rozvoj webové aplikace Kytarový zpěvník vyvíjené průběžně studenty FIT. Cílem práce je provést analýzu, návrh a realizaci klíčových vylepšení aplikace zaměřené především na možnosti editace, zobrazování, export a tisk písní i zpěvníků.

Pokyny:
- ve spolupráci s vedoucím práce stanovte konkrétní požadavky na vylepšení aplikace a podrobně je analyzujte
- navrhněte způsob řešení těchto požadavků v souladu se stávající architekturou
- implementujte vylepšení dle provedeného a schváleného návrhu
- provedená vylepšení důkladně otestujte
- provedená vylepšení zdokumentujte v souladu s existující dokumentací

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 30. prosince 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Rozvoj webové aplikace Kytarový zpěvník

Jakub Jabůrek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Zdeněk Rybola, Ph.D.

11. května 2022

Poděkování

V první řadě bych chtěl poděkovat vedoucímu mé práce, Ing. Zdeňku Rybolovi, Ph.D., za odborné vedení, konzultace a čas, který mi v průběhu vypracování práce věnoval. Dále děkuji rodině a přátelům, kteří mě po celou dobu studia podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2022 . . . . . .

Odkaz na tuto práci

Jabůrek, Jakub. *Rozvoj webové aplikace Kytarový zpěvník*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Bakalářská práce se zabývá rozvojem webové aplikace Kytarový zpěvník, která již několik let vzniká v rámci výuky na fakultě. Klíčové vlastnosti aplikace jsou vyhledávání a přidávání textů písní s kytarovými akordy. V rámci práce proběhla analýza současných funkcí a implementace týkající se editace, zobrazování a tisku písní a zpěvníků. Řešené požadavky se týkaly vytvoření databáze grafických schémat akordů, zobrazování těchto schémat a vylepšení a odstranění chyb tiskového exportu písně. Během návrhu řešení byly vytvořeny nové formáty pro definici schématu akordu a pro ukládání textu písně. V průběhu realizace vznikla knihovna, která umí vytvořené formáty zpracovávat, a integrovala se do aplikace. Nakonec proběhlo testování a příprava k nasazení nové verze aplikace do produkčního prostředí.

Klíčová slova webová aplikace, PHP, zpěvník, píseň, akord, PDF, XML, knihovna

Abstract

The bachelor thesis covers the development of the web application Kytarový zpěvník, which has been in development for several years as part of courses at the faculty. The key features of the application are searching and contributing song lyrics with guitar chords. This thesis analyzed the current features and implementation related to editing, viewing, and printing songs and songbooks. The requirements addressed consisted of building a database of graphical chord diagrams, displaying these diagrams, and improving and eliminating issues with song export for print. Throughout the solution design process, new formats were created for defining chord diagrams and for storing song lyrics. During the implementation, a library that can process the created formats was developed and integrated into the application. Finally, testing and preparation for deployment of the new version of the application to the production environment were conducted.

Keywords web application, PHP, songbook, song, chord, PDF, XML, library

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Přístup k internetu je v rozvinutých společnostech natolik zásadní a rozšířenou věcí, že se otevřela diskuze o jeho zařazení mezi základní práva člověka [1]; v České republice k němu má přístup přes 80 % obyvatel [2]. Vzhledem k tomu, že pro běžného uživatele jsou webové stránky synonymem k internetu, není překvapivé, že právě web se stal pro značnou část populace zdrojem informací i zábavy.

Tématem mé závěrečné práce je rozvoj webové aplikace Kytarový zpěvník. Ta vzniká již několik let v rámci předmětů BI-SP1 a BI-SP2 (Softwarový týmový projekt 1 a 2) pod vedením Ing. Zdeňka Ryboly, Ph.D., a její rozšíření bylo náplní i několika závěrečných prací před mou. Na vývoji aplikace jsem se podílel již v rámci zmíněných předmětů, nyní budu pokračovat v implementaci dalších požadavků vedoucího práce.

Kytarový zpěvník slouží k vyhledávání a sdílení textů a akordů písní. Snaží se oproti klasickým zpěvníkům přinést výhody, které na papíře nejsou možné – rychlé vyhledávání, sdružování písní do vlastních zpěvníků, sociální aspekty (například žebříčky nejlépe hodnocených uživatelů nebo možnost přidávat si uživatele do přátel), transpozice akordů jedním kliknutím a další.

Nezapomíná se ani na primární účel zpěvníku – aby se podle něj zpívalo a hrálo. K tomu webová aplikace nabízí jednak režim automatického posouvání obrazovky s textem, ale i možnost tisku textu a akordů na papír. Právě funkcemi pro zobrazování, export, tisk a editaci písní a zpěvníků se budu ve své závěrečné práci zabývat.

V teoretické části práce provedu analýzu stávající funkčnosti aplikace, použitých technologií a společně s vedoucím specifikujeme nové požadavky, které budu ve své práci řešit. Podívám se také na existující webové stránky podobného zaměření a vyberu několik existujících implementací klíčových problémů, které se v Kytarovém zpěvníku řeší. Následně navrhnu způsob implementace nových požadavků, podle kterého budu postupovat v praktické části závěrečné práce. Po dokončení implementace bude následovat testování a v závěru příprava k nasazení provedených změn.

Cíle práce

Má závěrečná práce se soustředí na rozvoj existující aplikace podle pokynů vedoucího. Prvním cílem tedy je stanovení požadavků na změny ve spolupráci s vedoucím a jejich analýza. Specifikace požadavků poslouží jako vstup pro návrh řešení a rešerši existujících řešení a využitelných technologií.

Dalším cílem je v rámci praktické části práce implementovat řešení podle vypracovaného návrhu. Součástí práce je vytvoření automatických testů pro novou funkcionalitu a tvorba (respektive aktualizace) dokumentace v souladu s již existující dokumentací projektu.

Výsledkem práce bude dodání nové verze aplikace, která bude připravená k nasazení, a bude obsahovat implementované změny podle požadavků vedoucího.

Kapitola 1

Analýza

První částí softwarového procesu je bezesporu analýza problému, který se bude řešit. Vzhledem k tomu, že má závěrečná práce se zabývá rozšířením již existujícího softwarového produktu, je logické, že se nejprve zaměřím na zmapování jeho aktuální podoby, včetně relevantních implementačních podrobností.

Teprve po seznámení se s aplikací přejdu ke specifikaci nových požadavků od zadavatele, což je v mém případě vedoucí práce Ing. Zdeněk Rybola, Ph.D. Na závěr analýzy provedu rešerši existujících řešení zadaných požadavků a případných alternativ k existující implementaci v Kytarovém zpěvníku.

1.1 Stávající funkčnost aplikace

V této sekci čtenáři představím aplikaci Kytarový zpěvník a její hlavní části; v detailu popíši ty, které přímo souvisejí s realizací funkčních požadavků editace, zobrazování, exportu a tisku písní a zpěvníků, a tím pádem i s problematikou řešenou v této závěrečné práci.

1.1.1 Stručný popis aplikace

Kytarový zpěvník je webová aplikace pro vyhledávání, vkládání a sdílení textů a kytarových akordů písní. Nabízí funkce pro organizaci písní do zpěvníků, hodnocení a komentování, zadávání požadavků na vložení dosud nepřidaných písní, exportování písní do PDF formátu a další.

Aplikaci je možné využívat pasivně jako návštěvník, který může písně pouze vyhledávat a texty jen číst. Nebo se lze zaregistrovat a tím získat možnost písně do databáze i vkládat. Zároveň se zpřístupní další sada funkcí aplikace, jako je hodnocení písní, vytváření zpěvníků (tedy souborů – nejen vlastních – písní) nebo přidávání ostatních uživatelů mezi své přátele a sdílení svých písní s nimi.

1.1.2 Píseň a zpěvník

Píseň je základním kamenem celé aplikace. Uchovává se o ní řada informací, jako je název písně, jméno interpreta, název alba, rok vydání, případně i sada tagů pro snadnější organizaci. Klíčovou součástí písně je také text a akordy v textu.

Každý uživatel si může vytvořit libovolné množství zpěvníků a vkládat do nich libovolné (i ty, které sám nevytvořil) veřejné písně. Kromě seznamu písní má také každý zpěvník své jméno. Pořadí písní ve zpěvníku lze volně měnit.

K písním i zpěvníkům mohou přihlášení uživatelé přidávat textové komentáře a hodnotit je na škále 1–5 hvězdiček. Hodnocení písní a zpěvníků se mimo jiné používá k sestavení žebříčků nejlepších uživatelů aplikace.

Písně a zpěvníky nemusí být veřejné, vlastník (případně administrátor) může rozhodovat o tom, kteří konkrétní uživatelé budou mít k objektu přístup. V základu se viditelnost dělí do třech úrovní:

1. veřejná (i pro nepřihlášené uživatele),
2. pouze pro přátele vlastníka,
3. soukromá (jen vlastník).

Neveřejnou píseň nebo zpěvník může jejich vlastník nasdílet konkrétním uživatelům, kteří k nim získají přístup. Kromě vyjmenování jednotlivých uživatelů je možné píseň nebo zpěvník nasdílet celé skupině přátel.

Uživatel může cizí písně a zpěvníky tzv. převzít, čímž se odkaz na cizí píseň nebo zpěvník vloží do seznamu jeho vlastních písní, respektive zpěvníků. Tím ale nezíská možnost píseň či zpěvník upravovat. V takovém případě musí vytvořit prostou kopii vybrané písně nebo zpěvníku.

1.1.3 Vkládání a editace písní

Vkládání a úprava textu a akordů písně probíhá ve webovém rozhraní prostřednictvím obyčejného textového pole. Akordy se formátují uzavřením do hranatých závorek a vepsáním před písmeno, nad kterým se mají zobrazit. Vizte příklad 1.1.

```
[D]Na Okoř je cesta jako žádná ze sta,
[A7]vroubená je stromama[D].
Když jdu po ní v létě, samoten na světě,
[A7]sotva pletu nohama[D].
[G]Na konci té cesty [D]trnité,
[E]stojí krčma jako [A7]hrad.
```

Výpis kódu 1.1 Lidová píseň Okoř [3, s. 92]

Formát je daný historickým vývojem aplikace a zadáním vedoucího projektu – původně byly k dispozici alternativní možnosti zadávání textu a akordů, ty byly ale postupně odstraněny a výše uvedený zůstal jako nejpohodlnější z nich.

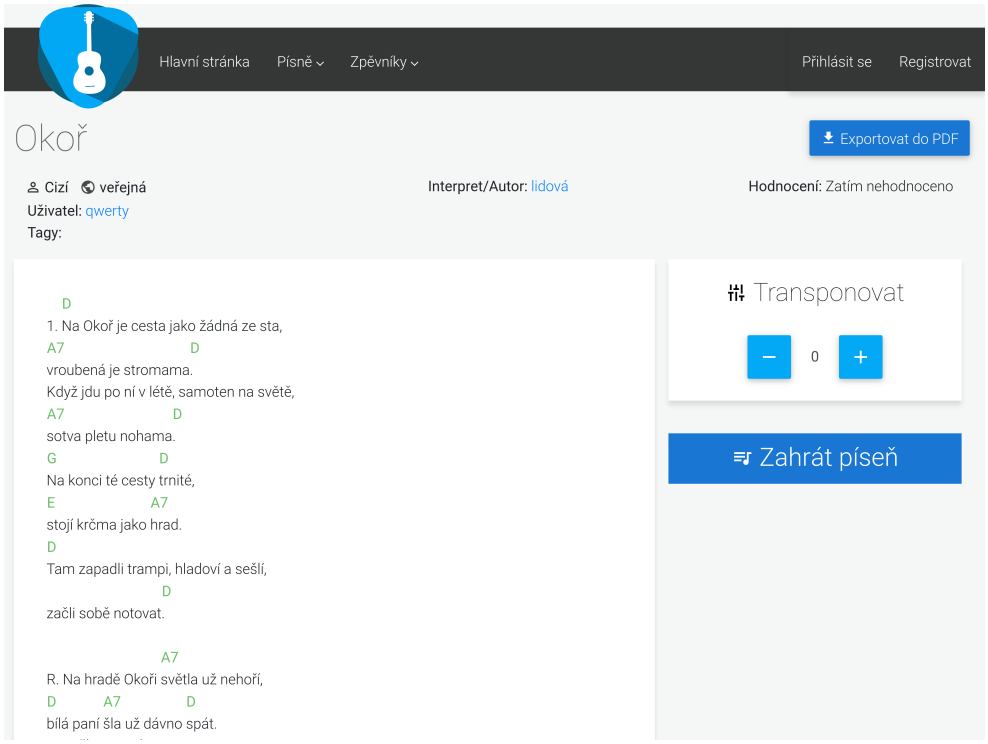
Ačkoliv se pro vkládání textu a akordů nepoužívá WYSIWYG editor, vedle textového pole je k dispozici okamžitý náhled, kde uživatel vidí výslednou podobu zadávaného textu.

1.1.4 Prohlížení písní

V aplikaci existuje několik možností, jak se mezi písněmi navigovat. Na hlavní stránce rotují karty s nejnovějšími a nejlépe hodnocenými písněmi, skrz které je možné se na detail písně prokliknout. Dále je možné využít vyhledávání, seznam všech písní nebo seznam písní, které přidal aktuálně přihlášený uživatel.

Uvedené možnosti vedou na stránku s detailem písně, kde jsou zobrazené všechny dostupné informace o písni, jako je její název, jméno interpreta, hodnocení, informace o viditelnosti a nastavených sdíleních a zejména text s akordy.

Zdrojová podoba textu písně (vizte výpis kódu 1.1) je převedená do HTML kódu, který je nastylovaný tak, aby se akordy původně uzavřené do hranatých závorek zobrazovaly nad textem písně na správné pozici a byly od textu barevně odlišené (podobně jako v exportu do PDF, vizte dále). Zobrazuje se pouze textový název akordu, nikoliv jeho vizuální podoba.



Obrázek 1.1 Webové zobrazení detailu písně

Webové rozhraní umožňuje akordy transponovat, to znamená změnit tóninu celé písně. Transpozice se používá v případech, kdy hráč není schopen některé akordy zahrát nebo zpěvák některé tóny zazpívat. Tlačítky lze okamžitě aktuální píseň převést buď do nižší nebo vyšší tóniny. Transpozici je dostupná pro všechny uživatele, pokud ji provádí vlastník písně, může originální verzi nahradit transponovanou.

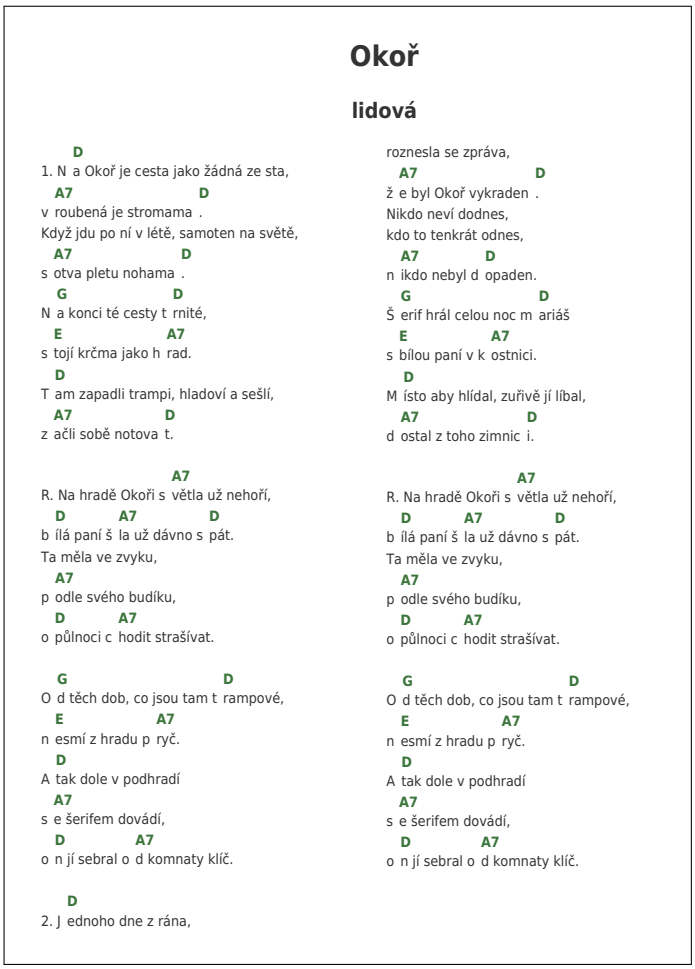
Pro pohodlnější hraní je možné tlačítkem *Zahrát píseň* aktivovat upravené zobrazení textu písně. Dojde ke skrytí všech ostatních prvků webu a text se zobrazí přes celou obrazovku velkým písmem. V tomto režimu také lze zapnout funkci autoscroll, která bude text automaticky posouvat (k dispozici je několik úrovní rychlosti posunu).

Na stránce s detailem písně je dále výpis komentářů a pole pro přidání nového komentáře. Přihlášení uživatelé mají skrze tlačítko *Možnosti písně* přístup k dalším funkcím (například převzetí), vlastník pak k editaci a další správě písně.

1.1.5 Export písní a zpěvníků do PDF

Na stránce s detailem písně či zpěvníku je k dispozici tlačítko *Exportovat do PDF*. Kliknutím se text a akordy písně vysází do dokumentu velikosti A4 a ten se nabídne ke stažení. Aby se text pokud možno vešel na jednu stránku, sází se do dvou sloupců. Tato funkce je vhodná zejména pro tisk písně jako v klasickém papírovém zpěvníku. Exportovaná píseň obsahuje kromě textu a akordů i název, jméno interpreta, album a rok vydání (pokud jsou k dispozici).

Před exportem zpěvníku je možné omezit písně, které budou do exportu zařazeny, a volitelně vložit titulní stranu s názvem zpěvníku, obsah a číslování stránek. Kromě exportu zpěvníku je možné vybrat i sadu jinak nesouvisejících písní a exportovat je do jednoho PDF dokumentu (taktéž s volitelným obsahem a číslováním stránek).



Okoř

lidová

D
1. N a Okoř je cesta jako žádná ze sta,
A7 D
v roubená je stromama .
Když jdu po ní v létě, samoten na světě,
A7 D
s otva pletu nohama .
G D
N a konci té cesty t rnité,
E A7
s tojí krčma jako h rad.
D
T am zapadli trampi, hladoví a sešlí,
A7 D
z ačli sobě notova t.

A7
R. Na hradě Okoři s větla už nehoří,
D A7 D
b ílá paní š la už dávno s pát.
Ta měla ve zvyku,
A7
p odle svého budíku,
D A7
o půlnoci c hodit strašívat.

G D
O d těch dob, co jsou tam t rampové,
E A7
n esmí z hradu p ryč.
D
A tak dole v podhradí
A7
s e šerifem dovádí,
D A7
o n jí sebral o d komnaty klíč.

D
2. J ednoho dne z rána,
roznesla se zpráva,
A7 D
ž e byl Okoř vykraden .
Nikdo neví dodnes,
kdo to tenkrát odnes,
A7 D
n ikdo nebyl d opaden.
G D
Š erif hrál celou noc m ariáš
E A7
s bílou paní v k ostnici.
D
M ísto aby hlídal, zuřivě jí líbal,
A7 D
d ostal z toho zimnic i.

A7
R. Na hradě Okoři s větla už nehoří,
D A7 D
b ílá paní š la už dávno s pát.
Ta měla ve zvyku,
A7
p odle svého budíku,
D A7
o půlnoci c hodit strašívat.

G D
O d těch dob, co jsou tam t rampové,
E A7
n esmí z hradu p ryč.
D
A tak dole v podhradí
A7
s e šerifem dovádí,
D A7
o n jí sebral o d komnaty klíč.

Obrázek 1.2 Tiskové zobrazení písně

Současné řešení vykazuje chyby při zalamování textu a vkládání akordů doprostřed slov. Některé verše jsou zalomeny zbytečně brzy, naopak někdy se zalomení neprovede a další verš pokračuje na stejném řádku. Akordy uprostřed slov pak vytvářejí v textu písně nevyžádané mezery (jsou vidět na obrázku 1.2, kde byly akordy pro ukázku posunuty dovnitř slov). Jedním z cílů mé práce je tyto chyby odstranit.

1.2 Architektura aplikace

Než začnu vysvětlovat konkrétní technologie, které se v Kytarovém zpěvníku používají, uvedu pohled na aplikaci z hlediska její architektury. Architekturou softwarového systému se rozumí jeho členění a struktura z hlediska celého systému, nikoliv konkrétních implementačních detailů. Jedná se o logické (to se týká Kytarového zpěvníku) nebo fyzické rozdělení systému na několik vrstev či částí a aplikování určitých architektonických vzorů a principů.

1.2.1 Vícevrstvá architektura

Základním principem, podle kterého je navržena většina webových aplikací, je vícevrstvá architektura. Ta rozděluje komponenty aplikace do několika vrstev podle jejich účelu.

Kytarový zpěvník konkrétně sleduje rozdělení do tří vrstev.

prezentační Vrstva, se kterou pracuje uživatel. Slouží k vykreslení uživatelského rozhraní, prezentaci dat a k předávání událostí interakce aplikační vrstvě. V kontextu webové aplikace si ji lze představit jako HTML a CSS kód, webový server, ale i webové API rozhraní nebo PHP kód starající se o přípravu šablon (tzv. *presentery*, vizte sekci 1.2.2).

aplikační Obsahuje implementaci aplikační logiky, tedy například zpracování událostí z prezentační vrstvy, volání metod z datové vrstvy a přípravu dat pro prezentační vrstvu. Zároveň by zde měla být umístěna i business logika, jako například validace dat. U webové aplikace by se mohlo jednat například o PHP skripty.

datová V datové vrstvě aplikace je umístěná logika, která poskytuje přístup k úložišti dat (to je například databázový server nebo souborový systém). V dobře navržené aplikaci jde o abstrakci, která implementační detaily přístupu k úložišti zapouzdřuje. [4, s. 2]

Pro úplnost je vhodné upřesnit použité termíny. Vrstva (anglicky *layer*) označuje logické oddělení v rámci aplikace. V anglické literatuře je možné se setkat i s pojmem *tier*, který se používá, pokud jsou jednotlivé vrstvy implementovány v oddělených aplikacích, případně běží na separátních prvcích infrastruktury [5, s. 55]. Pro kontext Kytarového zpěvníku je korektní první varianta, jelikož se jedná o jednu celistvou aplikaci.

1.2.2 Model–View–Presenter

Návrhový vzor Model–View–Presenter (MVP) podobně jako třívrstvá architektura rozděluje aplikaci na tři logické celky:

model Definuje datové entity, se kterými aplikace pracuje, a přidruženou business logiku. Dalším vrstvám poskytuje rozhraní, skrze které lze s modelem manipulovat, sám ale o ostatních vrstvách neví.

view Zobrazuje uživatelské rozhraní a data a předává příkazy od uživatele ke zpracování do presenteru.

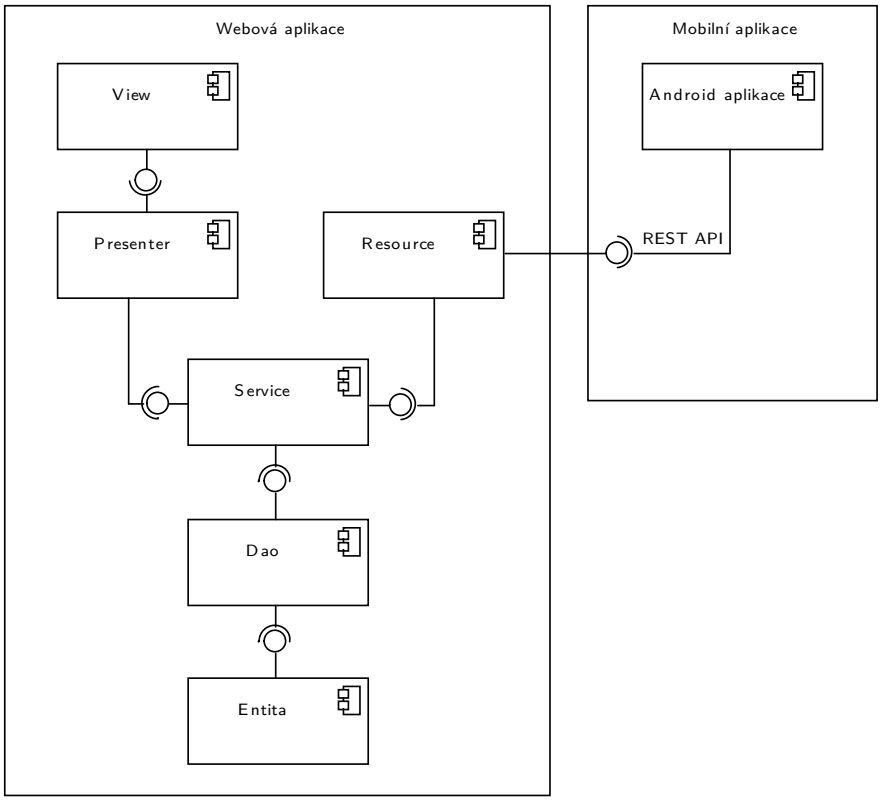
presenter Reaguje na vstup a akce od uživatele, provádí příslušné manipulace s modelem a poskytuje data pro view. [6, s. 235–236]

Zde se skutečně jedná o rozdělení logické, oproti vícevrstvé architektuře je také v některých implementacích povolena (ač nedoporučena) interakce nesousedních vrstev (například view může zavolat metodu modelu, ale prezentační tier nemůže přímo komunikovat s datovou tier).

Princip Model–View–Presenter byl pro Kytarový zpěvník zvolen především proto, že je to návrhový vzor Nette frameworku a jiný vzor (například Model–View–Controller, ze kterého MVP vychází) by se v kombinaci s Nette používal komplikovaně. (Podrobněji výběr použitých technologií odůvodňuji v úvodu sekce 1.3.)

1.2.3 Rozdělení do balíčků

V diagramu 1.3 lze vidět již popsané rozdělení aplikace na model, view a presenter. Model tvoří balíčky *Service* (obsahuje business logiku), *Dao* (Data Access Object; zapouzdřuje přístup k úložišti dat) a *Entita* (definuje datové struktury), do presenterů můžeme kromě balíčku *Presenter* zařadit i *Resource*.



Obrázek 1.3 Diagram balíčků

Balíček *Resource* značí implementaci REST API pro přístup k datům, které využívá mobilní aplikace. Ta je vyvíjená v rámci jiného softwarového projektu a v této práci se jí budu zabývat pouze v ohledu zachování zpětné kompatibility datového formátu API.

1.3 Použité technologie

V předchozí části jsem vysvětlil logickou strukturu implementace aplikace, zde přejdu k popisu použitých technologií. Webové aplikace se běžně rozdělují na dvě části: serverovou a klientskou. Vzhledem k tomu, že obě části v tomto případě používají odlišné programovací jazyky, rozdělím je i při popisu použitých technologií. Nakonec se zmíním i o nástrojích používaných při vývoji.

V analytické části své práce popisuji aktuální stav aplikace; zvolené programovací jazyky, knihovny a nástroje jsou tedy výsledkem práce mých předchůdců (a z části i mé, jakožto člena dvou týmů, které na aplikaci pracovaly v rámci výuky na fakultě). Tím nejdůležitějším byl výběr jazyka a frameworku pro serverovou část a návrh architektury, které provedl v roce 2018 v rámci své bakalářské práce Jan Holan [7].

1.3.1 Backend

Serverová část Kytarového zpěvníku je napsaná v jazyce PHP. To je v současné době jeden z nejpopulárnějších jazyků pro programování webových aplikací. Ve vývoji je od roku 1994, kdy vznikl jako způsob pro jednoduché publikování osobních webových stránek. Tomu odpovídá možnost vkládání HTML kódu přímo dovnitř PHP skriptů a syntaxe

inspirovaná jazykem Perl. Jedná se také o jeden z nejvíce podporovaných serverových jazyků na běžných sdílených webhostinzích.

V dnešní době je PHP pokročilý jazyk s velkou komunitou, množstvím existujících knihoven a balíčků. Kytarový zpěvník používá verzi PHP 7.1, která vyšla v roce 2016.

Webové PHP aplikace ke svému běhu potřebují i webový server, který bude předávat HTTP požadavky PHP kódu a odesílat zpět odpovědi. Kytarový zpěvník je odlazen a předkonfigurován pro server Apache.

1.3.1.1 Framework

Nette framework [8] je český PHP framework pro vývoj webových aplikací. V Česku a na Slovensku je velmi rozšířený. Poskytuje připravené implementace většiny funkcí, které každá webová aplikace potřebuje, jako je zpracování HTTP požadavků, routování, práce s formuláři, dependency injection, správu konfigurace, debugger a další. Kytarový zpěvník používá Nette framework ve verzi 2.4. Kromě toho také používá další dvě knihovny určené pro Nette – Kdyby/Doctrine 3.3 a Latte 2.4.

1.3.1.2 Šablony

Latte je šablonovací systém od tvůrců Nette frameworku a je doporučený pro Nette framework. Latte šablony obsahují text a HTML kód, které se odešlou v odpovědi klientovi, a speciální značky, kterými se vkládají data připravená v presenteru nebo vytvářejí řídící konstrukce, jako jsou například podmínky.

1.3.1.3 Databáze

Doctrine je knihovna poskytující abstrakci pro přístup k různým databázovým strojům, jako je například MySQL nebo PostgreSQL. Zároveň také poskytuje implementaci tzv. ORM, tedy mapování PHP objektů do relační databáze. Knihovna Kdyby/Doctrine pak přináší integraci Doctrine ORM do Nette frameworku.

V kódu aplikace se vytvářejí entitní třídy s atributy, které odpovídají tabulkám a sloupcům v databázi. Implementace rozhraní z *Dao* balíčku pak podle vzoru Data Mapper [9, s. 165] poskytují CRUD operace s entitami – vytvoření, načtení z databáze, editaci a mazání. Kromě těchto základních operací samozřejmě poskytují i specializované metody, například pro filtrování či agregaci dat.

Datová vrstva Kytarového zpěvníku očekává databázi MySQL, je kompatibilní i s odvozenou MariaDB. Případná změna databázového serveru by se díky abstrakci Doctrine obešla bez velkých změn zdrojového kódu, kromě úpravy konfigurace a několika ručně psaných SQL skriptů.

1.3.1.4 Generování PDF

Vzhledem k tomu, že jedním z cílů této práce je i vylepšení PDF exportu, je potřeba zmínit knihovnu mPDF [10], která se ve verzi 8.0 aktuálně pro tvorbu PDF dokumentů používá. Jejím vstupem je specializovaná HTML šablona, do které se vyplní text písně a mPDF ji převede do PDF formátu.

1.3.2 Frontend

Klientská část aplikace, tedy ta, se kterou pracuje uživatel, se označuje jako frontend. Vzhledem k tomu, že mluvíme o webové aplikaci, jsou základem frontendu technologie HTML, CSS a JavaScript.

1.3.2.1 HTML

HTML je značkovací jazyk, pomocí kterého se na webové stránce vytváří elementy a text. Webové prohlížeče aktuálně implementují HTML ve stavu tzv. *living standard*. To znamená, že jazyk se průběžně vyvíjí bez dělení do verzí a prohlížeče změny průběžně implementují. Jak jsem zmínil výše, samotný HTML kód se generuje z Latte šablon.

1.3.2.2 CSS

Pro grafické zpracování webové stránky se používají stylopisy v jazyce CSS. Jazyk CSS podobně jako HTML také opustil klasické verzovací schéma, Kytarový zpěvník používá ty části, které podporuje většina moderních prohlížečů.

Většina stylů pochází z CSS frameworku Bootstrap 4.3, který nastavuje design mnoha elementům a komponentám (například tlačítka, boxy, tabulky, ...) a poskytuje utility pro skládání vlastního layoutu stránky (například nastavení barvy, rámečku, odsazení a podobně). Framework se přiloží k webové stránce a vlastní stylopis jej jen doplňuje. Další důležitou knihovnou pro design je Material Design Icons, jenž obsahuje širokou škálu open-source vektorových ikon, které se v Kytarovém zpěvníku používají.

Psaní rozsáhlých stylopisů v běžném CSS je nepohodlné, proto vznikly tzv. preprocesory s rozšířenými funkcemi a syntaxí, ze které se výsledné CSS zkompiluje. V Kytarovém zpěvníku se používá preprocesor SASS, ten umožňuje například používání proměnných nebo zanořování selektorů.

1.3.2.3 JavaScript

JavaScript je programovací jazyk původně určený pro prostředí webového prohlížeče, ačkoliv se v poslední době stává velmi rozšířeným i na backendu (to se Kytarového zpěvníku ale netýká). Kytarový zpěvník jej využívá ve skriptech pro větší interaktivitu webových stránek, například pro zobrazování dialogových oken.

Pro zjednodušení a zrychlení vývoje je na webu použitá JavaScriptová knihovna jQuery 3.3. Ta poskytuje implementaci nejčastěji používaných funkcí ve webových aplikacích, které chybí ve standardním API webových prohlížečů.

1.3.3 Vývojové prostředí

Na závěr popisu použitých technologií se zmíním o nástrojích, které používají během vývoje samotní programátoři.

1.3.3.1 Docker

Docker je nástroj pro izolaci aplikací do kontejnerů a jejich jednoduchou správu. Kontejnery se definují pomocí textových konfiguračních souborů a je možné je publikovat do on-line repozitáře, ze kterého si je mohou další uživatelé stáhnout a používat.

Pokud vezmeme příklad PHP aplikace, tak klasický postup pro zprovoznění vývojového prostředí je instalace webového serveru, PHP interpreteru a jejich konfigurace dle požadavků konkrétní aplikace. To musí provést na svém počítači každý vývojář a opakuje se při nasazení na testovací, produkční a další prostředí. Docker umožňuje v repozitáři s aplikací jednou nadefinovat kontejnery, které webový server, PHP interpreter a požadovanou konfiguraci obsahují, a vývojáři je jedním příkazem spustí. Kontejnery je možné použít i na serveru, kde bude aplikace provozována, čímž se dosáhne homogenní konfigurace napříč všemi prostředími.

Během letního semestru 2020/2021 tým, jehož jsem byl součástí, v rámci předmětu BI-SP1 Docker kontejnery pro Kytarový zpěvník nastavil. Produkční i testovací prostředí

je však hostované na klasickém sdíleném webhostingu, který Docker nepodporuje, takže kontejnery se používají jen během vývoje.

1.3.3.2 Testování

Automatizované testování aplikace je řešeno primárně prostřednictvím jednotkových testů PHP kódu. Jednotkové testy mají za úkol izolovaně otestovat třídní metody a zkontrolovat, že vracejí očekávané výsledky. V aplikaci se aktuálně používají dva odlišné testovací frameworky: Nette Tester a PHPUnit.

V zimním semestru 2021/2022 se náš tým v rámci předmětu BI-SP2 zaměřil na vylepšení a doplnění existujících jednotkových testů. Zjistili jsme, že existující testy nesplňují požadavky na kvalitní jednotkové testy – zejména vyžadovaly funkční databázi naplněnou velmi konkrétními daty, databázi při běhu modifikovaly a nebylo možné je spouštět paralelně. Zároveň jsme narazili na limity Nette Testeru, který například nepodporoval vytváření tzv. mock objektů, tedy instancí určité třídy s upraveným chováním pro potřeby konkrétního testu.

Z těchto důvodů jsme se rozhodli přidat testovací framework PHPUnit, přepsat některé nevyhovující testy a dále využívat jen PHPUnit. Zbylé Nette Tester testy jsou funkční (z časových důvodů jsme nemohli přepsat všechny testy), ale počítá se s jejich postupnou konverzí do PHPUnitu a následným odstraněním Nette Testeru.

Nastavené jsou i nástroje pro statickou analýzu kódu (PHPStan) a kontrolu coding style (Easy Coding Standard). Spolu s jednotkovými testy se automaticky spouští v rámci *CI pipeline* (vizte sekci 3.2.3). Každý z nich lze spustit jedním příkazem i v lokálním terminálu během vývoje.

Nadto jsme na doporučení vedoucího projektu připravili projekt pro SonarQube, další (a ve srovnání s PHPStanem pokročilejší) nástroj pro statickou analýzu zdrojového kódu. Připravili jsme dokumentaci a Docker kontejner, ale analýzu musí vývojář spustit v lokálním prostředí a ručně. SonarQube je totiž rozdělený na server a skener, přičemž server zpřístupňuje výsledky skenování, takže jej nelze spustit v rámci pipeline a vygenerovat statický report, jako u PHPStanu.

1.3.3.3 Správa závislostí

Co se backendu týče, všechny použité PHP knihovny včetně Nette frameworku jsou instalovány prostřednictvím nástroje pro správu závislostí Composer. Seznam závislostí se deklaruje v souboru `composer.json`. Při sestavení aplikace se závislosti jedním příkazem stáhnou a pomocí tzv. *autoloaderu* se zpřístupní v PHP aplikaci.

Závislosti pro frontend pak spravuje nástroj Yarn, zpětně kompatibilní se známějším Npm. Ten stahuje závislosti, které se na frontendu přímo používají (například Bootstrap), a nástroje pro kompilaci SASS, kompresi JavaScriptu a podobně, které se používají během vývoje.

Výsledné stylopisy a soubory s JavaScriptem se pomocí nástroje Webpack komprimují a spojují do jednoho souboru, který je připojený k webové stránce.

1.4 Implementační detaily

Protože významná část praktické části mé závěrečné práce se bude zabývat zpracováním textu a akordů písně, představím nejdříve implementaci těchto procesů za aktuálního stavu aplikace.

1.4.1 Ukládání textu písně a akordů

Na začátku analytické části jsem uváděl výpis kódu 1.1 s textem písně a akordy. Zmiňoval jsem, že v uvedeném formátu se text zapisuje v administraci, nyní doplním, že pro uložení do databáze se používá jiný formát, který popíši v této části.

Po odeslání formuláře s akordy vloženými do textu písně proběhne jejich extrahování do samostatného slovníku, kde indexy značí původní pozici akordu v textu. Do jednoho sloupce v databázi se uloží text očištěný od akordů, do druhého slovník s akordy.

Z textu ve výpisu kódu 1.1 se tedy do databáze uloží data z výpisů 1.2 a 1.3.

```
Na Okoř je cesta jako žádná ze sta,
vroubená je stromama.
Když jdu po ní v létě, samoten na světě,
sotva pletu nohama.
```

Výpis kódu 1.2 Sloupec `lyrics`

```
{
  "1":"D", "37":"A7", "56":"D",
  "100":"A7", "117":"D"
}
```

Výpis kódu 1.3 Sloupec `chords`

1.4.2 Převod mezi formáty textu písně a akordů

Z dosud popsaných implementačních detailů je zřejmé, že aplikace musí být schopna převádět mezi různými formáty reprezentace textu písně a akordů:

1. z editačního formátu do databázové reprezentace,
2. z databázové reprezentace do editačního formátu,
3. z databázové reprezentace do HTML kódu,
4. z databázové reprezentace do PDF formátu.

V části 1.3.1 jsem vysvětlil, že generování PDF exportu probíhá ze speciální HTML šablony, bod 4. je tedy víceméně totožný se 3. a věnovat se mu nyní nebudu. Ve stručnosti nastíním, jakým způsobem funguje implementace konverze mezi uvedenými formáty.

formulář → databáze Převod probíhá po odeslání a úspěšné validaci formuláře. Implementace je provedena cyklem, který vždy najde následující pár levé a pravé hranaté závorky, smaže značku s akordem z textu, a uloží pár pozice v textu a akord do slovníku s akordy.

databáze → formulář Před vložením dat písně do polí editačního formuláře se načte slovník s akordy, iteruje se skrz jeho položky a pro každý akord se do textového pole vloží text písně od předcházejícího akordu do aktuálního akordu a za něj se připojí akord uzavřený do hranatých závorek.

databáze → HTML Text písně se rozdělí do pole, kde každý prvek je jeden verš písně. Následně se každý verš rozdělí na pole dvojic akord–text, kde text dvojice obsahuje text verše za akordem dvojice až do dalšího akordu nebo konce verše. Takto předzpracovaná data se předají šabloně, která z nich vytvoří finální HTML kód.

1.5 Texty a akordy ve zpěvníku

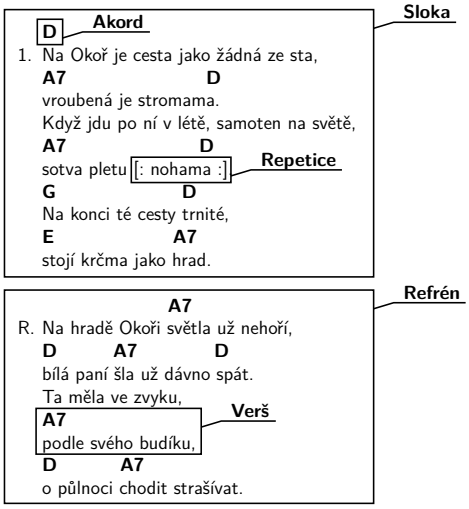
Existující zobrazení textu písně ve webové aplikaci odpovídá struktuře lidových a populárních písní v papírových kytarových zpěvnících. V rámci analýzy krátce popíši, jak takové rozložení vypadá a jak se znázorňují schémata akordů.

1.5.1 Struktura textu

Text písně můžeme rozdělit na sloky (sloku lze připodobnit kapitole v běžné próze) [11, s. 130]. Sloky jsou obvykle číslované a jedna z nich bývá označená jako refrén. Refrén se typicky opakuje po každé sloce a pro úsporu místa se většinou text refrénu vypisuje pouze jednou, při dalších opakováních se na něj odkazuje jeho zkratkou (obvykle *R* nebo *Ref*).

Sloku lze dělit na verše (označuje jednu rytmickou jednotku, jeden řádek) [11, s. 143]. Nad verš se zapisují akordy – vždy nad místo v textu, kde se má příslušný akord začít hrát. Sloky mají obvykle stejný rytmus (s výjimkou refrénu), proto se akordy většinou píší pouze k první sloce a zbytek písně se hraje podle ní.

V textu se mohou vyskytnout speciální značky pro tzv. repetici. Ta značí opakování vyznačené části textu (většinou dvojité, může být připojena poznámka s jiným počtem). Do repetice může být zahrnuta libovolná část verše, celý verš nebo i celá sloka.



■ **Obrázek 1.4** Struktura textu písně

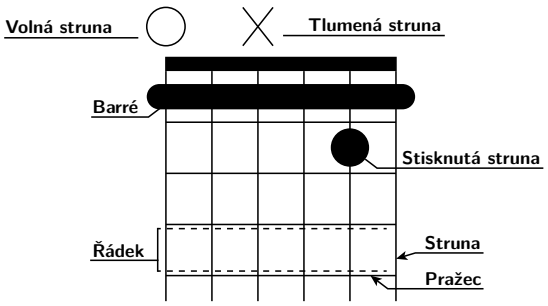
1.5.2 Schéma akordu

Schéma akordu je grafické znázornění, jak se příslušný akord hraje, tedy zejména které struny se mají stisknout.

Základní šablonou je prázdný tzv. hmatník (část kytary, nad kterou jsou natažené struny) s vyznačenými strunami a pražci. Začátek hmatníku je označen silnou čarou. Některé akordy se hrají níže na hmatníku, pak se schéma nekreslí od začátku. Místo silného ohraničení na začátku se naznačí, že akord pokračuje nahoru, a u prvního vyobrazeného pražce se římskou číslicí uvede jeho skutečné číslo.

Při hraní prsty tisknou strunu mezi dvěma pražci, na obrázku 1.5 označeno jako řádek. Stisknutá struna se ve schématu značí vybarveným kolečkem. Některé akordy mohou vyžadovat stisknutí více strun najednou jedním prstem (tzv. barré), ve schématu se znázorňuje pruhem přes stisknuté struny.

Nad každou strunou se může objevit nejvýše jedna značka, která udává, zda struna nemá v akordu zaznít (značí se křížkem), nebo má znít prázná (značí se prázdným kolečkem). [12]



Obrázek 1.5 Schéma akordu

1.6 Specifikace požadavků

Jedním z nezbytných úkolů v rámci procesu softwarového inženýrství – a tedy i mé závěrečné práce – je stanovení požadavků na vyvíjený systém. Specifikace požadavků slouží primárně k vyjasnění zadání se zákazníkem (v tomto případě s vedoucím práce) a pro plánování dalších činností softwarového inženýrství.

Každý požadavek má mít dostatečně přesný a detailní popis, který je důležitý nejen pro správnou implementaci, ale i pro ověření, zda byl požadavek splněn. Pro snadné odkazování má požadavek obvykle svou zkratku nebo číslo a k lepšímu plánování softwarového procesu pomáhá evidence dalších atributů, jako je složitost nebo priorita požadavku. [13, s. 50–56]

Specifikaci požadavků jsem provedl ve spolupráci s vedoucím práce v rámci konzultací, přičemž jsme vycházeli z nápadů na vylepšení a hlášení chyb sesbíraných v průběhu let, kdy byla aplikace vyvíjena, a (zejména v případě nefunkčních požadavků) z předcházejících znalostí aplikace, jejího vývoje a provozu.

Požadavky se kategorizují primárně na funkční a nefunkční, podle toho, zda se jedná o požadavek na funkcionalitu a chování systému, nebo na technické provedení. [14, s. 4]

1.6.1 Funkční požadavky

Požadavky v této kategorii definují očekávané chování aplikace za určených podmínek [13, s. 7]. Ve své práci je budu číslovat prefixem *F*. Zdrojem požadavků jsou primárně issues v nástroji GitLab (vizte sekci 3.2.2) a konzultace s vedoucím práce.

▶ F1. Netisknuté akordy

Editační rozhraní písně má umožnit označit část textu, ve které budou akordy při exportu pro tisk skryté. Ve zdrojovém textu zůstanou akordy zapsané a webové zobrazení písně je bude zobrazovat jako dosud, pouze export do PDF akordy uvnitř označené části bude ignorovat.

Pro označení je potřeba použít nějakou formu tagů, která nebude kolidovat s běžným textem, případně s jiným značením specifickým pro text písně ve zpěvníku.

Typ požadavku: Funkční požadavek
Druh změny: Nová funkcionalita
Priorita: Vysoká
Složitost: Střední

► F2. Schémata akordů

V administrační části webového rozhraní by měla být dostupná evidence kytarových akordů a jejich schémat. Správce webu je bude moci zobrazovat, přidávat, editovat a mazat. U každého akordu bude evidován název, libovolný počet alternativních názvů a schéma akordu.

Aplikace má po instalaci (respektive upgradu) obsahovat výchozí sadu základních akordů, kterou bude administrátor doplňovat. Součástí tohoto požadavku není žádné rozhraní pro procházení akordů nebo zobrazování schémat běžnými uživateli.

Typ požadavku: Funkční požadavek
Druh změny: Nová funkcionalita
Priorita: Střední
Složitost: Vysoká

► F3. Zobrazení schémat akordů

Navazuje na evidenci akordů implementovanou v požadavku F2 (zde se jedná o rozhraní pro běžné uživatele). Na webové stránce s textem písně bude uveden seznam použitých akordů a jejich schémata. Názvy akordů v textu písně budou aktivní a po kliknutí se zobrazí schéma daného akordu.

Párování akordu z textu písně na schéma z evidence bude probíhat na základě názvu zadaného do textu a názvu nebo některého alternativního názvu vyplněného v evidenci.

Typ požadavku: Funkční požadavek
Druh změny: Nová funkcionalita
Priorita: Střední
Složitost: Střední

► F4. PDF export písně

Návštěvníci webu mají mít možnost kliknutím na tlačítko stáhnout píseň ve formátu PDF se zobrazením optimalizovaným pro tisk na stránku A4. Tento požadavek již má existující implementaci, která ale obsahuje dvě zásadní chyby – špatné zalamování veršů a mezery uprostřed slov v místě výskytu akordu. Před akceptací požadavku je nutné chyby odstranit.

V tiskovém výstupu je text vysázený do dvou sloupců. Některé řádky mohou být příliš dlouhé, pak ale dochází k zalomení textu uprostřed slova a další verš pokračuje bez zalomení na stejné řádce. Správné chování je v případě nutnosti verš zalomit mezi slovy, a následující verš musí být opět na nové řádce.

Vložení akordu doprostřed slova způsobuje vytvoření mezery v místě, kde se akord nachází. Takové chování je špatně. Může se stát, že se blízko za sebou nachází více akordů, jejichž názvy vyžadují více místa, než je v textu k dispozici. V takovém případě je správné vytvořit v textu (i uprostřed slova) dodatečnou mezeru, aby se akordy vešly přesně na místo, kde jsou uvedené.

Typ požadavku: Funkční požadavek
Druh změny: Oprava chyb
Priorita: Vysoká
Složitost: Vysoká

► F5. Přizpůsobení exportu do PDF

Na stránce s detailem písně by měla být pod tlačítkem *Možnosti písně* k dispozici další položka, která otevře nastavení PDF exportu dané písně. Volba bude přístupná pro vlastníka písně a u převzaté písně pro uživatele, který ji převzal.

V rámci nastavení PDF exportu bude možné konfigurovat počet sloupců, do kterých se bude text písně sázet, velikost písma a bude možné povolit tisk akordů, které jsou označené jako netisknuté (vizte požadavek F1).

Typ požadavku: Funkční požadavek
Druh změny: Nová funcionalita
Priorita: Střední
Složitost: Střední

1.6.2 Nefunkční požadavky

Tento druh požadavků klade nároky na vlastnosti a charakteristiky aplikace [13, s. 7] (například použité technologie), případně na další části softwarového procesu (například dokumentaci nebo testování). V mé práci mají tyto požadavky prefix *N*. Vycházejí přímo z mých zkušeností z vývoje aplikace v předcházejících semestrech.

▶ N1. Programovací jazyk

Použitý programovací jazyk serverové části musí být PHP, kompatibilní s verzí 7.1 (stejně jako dosud). Interaktivní části frontendu budou napsané v jazyku JavaScript tak, aby byly kompatibilní s běžně používanými prohlížeči.

Typ požadavku: Nefunkční požadavek
Týká se: Implementace

▶ N2. Architektura implementace

Implementace musí být řádně integrovaná do existujícího kódu, dodržovat existující programovací konvence a neporušovat zavedenou architekturu aplikace.

Typ požadavku: Nefunkční požadavek
Týká se: Implementace

▶ N3. Kompatibilita se sdíleným webhostingem

Implementace backendu musí být provozuschopná na běžném sdíleném PHP webhostingu. To znamená zejména:

1. nevyžadovat specifickou konfiguraci PHP nebo webového serveru,
2. nevyžadovat nestandardní PHP rozšíření,
3. nespouštět žádné externí programy,
4. nevolat jiný než PHP kód.

Typ požadavku: Nefunkční požadavek
Týká se: Implementace

▶ N4. Automatické testy

Nově implementovaná funkcionalita musí být na vhodných místech pokrytá automatizovanými jednotkovými testy, které musejí být spouštěné spolu s již existujícími testy v CI prostředí. Při zásazích do existující implementace musí být upraveny i testy.

Typ požadavku: Nefunkční požadavek
Týká se: Testování

▶ N5. Dokumentace

Nová či změněná funkcionalita musí být zdokumentovaná v uživatelské příručce na webu Kytarového zpěvníku. Zdrojový kód musí obsahovat komentáře tříd, metod a podobně,

ze kterých lze automaticky vytvořit dokumentaci. Změny v existující implementaci musí být reflektovány v dosavadní dokumentaci v nástroji Enterprise Architect a popis nové funkcionality vhodnou formou do dokumentace doplněn.

Typ požadavku: Nefunkční požadavek
Týká se: Dokumentace

► **N6. Kompatibilita s Android aplikací**

REST API, které aplikace poskytuje, musí zachovat kompatibilitu s Android aplikací Kytarového zpěvníku. To znamená zachování existujícího rozhraní a datových struktur tak, aby současná verze Android aplikace byla plně funkční s novou verzí webové aplikace.

Omezení se netýká nové funkcionality, tu lze do API přidat po konzultaci s vedoucím práce.

Typ požadavku: Nefunkční požadavek
Týká se: Implementace

1.7 Existující podobná řešení

V této části provedu rešerši existujících implementací. Nejprve se zaměřím na celé webové aplikace podobné Kytarovému zpěvníku, následně (vzhledem k zaměření mé závěrečné práce) projdu strojové formáty pro zápis hudebních textů a knihovny pro generování PDF dokumentů.

Během rešerše se omezím na funkcionalitu a knihovny relevantní pro řešené požadavky (vizte sekci 1.6.1).

1.7.1 Webové aplikace s podobným zaměřením

Koncept kytarového zpěvníku byl již mnohokrát zpracovaný, proto existuje řada funkčních webů podobných aplikaci Kytarový zpěvník. Z rešerše vynechávám weby, které nejsou zaměřené na sdílení kytarových akordů (tedy například databáze jen textů písní). Z nejznámějších podobně zaměřených webů lze vyjmenovat následující:

- Supermusic [15]
- písničky-akordy.cz [16]
- spísničkou.cz [17]
- Brnkni.cz [18]

Všechny čtyři weby mají totožnou základní sadu funkcí – prohlížení textů a akordů písní, tisk, transpozici akordů a zobrazení schémat akordů. Hlavní odlišnosti jsou v grafickém zpracování a nadstavbových funkcích (například Brnkni.cz poskytuje automatické posouvání textu písně, podobně jako aplikace Kytarový zpěvník).

Aplikace Kytarový zpěvník oproti zmíněným alternativním webům nezobrazuje schémata akordů, což bude implementováno v rámci požadavku F3. Výše zmíněné weby zobrazují jak seznam použitých akordů a schémata vedle textu písně, tak konkrétní schéma po najetí myší na akord v textu. Z popisu požadavku vyplývá, že Kytarový zpěvník bude používat stejné řešení.

1.7.2 Formáty pro zápis hudebního textu

Vzhledem k tomu, že hlavní náplň mé závěrečné práce se týká práce s textem a akordy písní, bude nutné se zaobírat způsobem uložení těchto dat. Návrhem vhodného formátu se budu zabývat později a z této rešerše budu vycházet.

1.7.2.1 MusicXML

Nejrozšířenějším otevřeným formátem pro strojové zpracování partitur je MusicXML [19]. Již z popisu plyne, že MusicXML jde daleko nad rámec ukládání textu a akordů – jedná se o komplexní XML formát, který slouží primárně k zaznamenávání not. Základním kamenem MusicXML dokumentu je notová osnova, pod kterou může být zapsaný text písně. Formát umožňuje zachytit i akordy pro libovolný strunový nástroj.

```
<harmony>
  <root>
    <root-step>G</root-step>
  </root>
  <kind text="6">major-sixth</kind>
  <bass>
    <bass-step>D</bass-step>
  </bass>
  <frame>
    <frame-strings>6</frame-strings>
    <frame-frets>5</frame-frets>
    <frame-note>
      <string>5</string>
      <fret>5</fret>
    </frame-note>
    <frame-note>
      <string>4</string>
      <fret>5</fret>
    </frame-note>
    <frame-note>
      <string>3</string>
      <fret>4</fret>
    </frame-note>
    <frame-note>
      <string>2</string>
      <fret>3</fret>
    </frame-note>
  </frame>
</harmony>
```

Výpis kódu 1.4 Akord G6 v notaci MusicXML

Pro jazyk PHP existují open-source knihovny pro parsování MusicXML dokumentů, u všech ale jde pouze o osobní projekty jednotlivých autorů a všechny jsou starší než poslední verze MusicXML formátu z roku 2021. V podstatě tedy lze všechny vyloučit jako nepoužitelné.

1.7.2.2 LilyPond

Podobný účel jako MusicXML má formát (a stejnojmenný projekt) LilyPond [20]. Opět jde o formát, jehož primárním účelem je zaznamenání partitury. Oproti MusicXML je ale syntaxe LilyPond inspirovaná projektem TeX [21], tj. místo značkovacího jazyka XML se pro vytváření obsahu používají příkazy v textu dokumentu uvozené zpětným lomítkem.

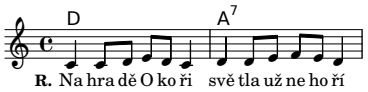
Knihovny v PHP pro práci s formátem LilyPond neexistují.

```
\chords {
    \set chordChanges = ##t
    d2 d2 a2:7 a2:7
}

\new Staff \relative c' {
    c4 c8 d e8 d c4
    d4 d8 e f8 e d4
}

\addlyrics {
    \set stanza = #"R."
    Na hra dě O ko ři svě tla už ne ho ří
}
```

Výpis kódu 1.5 Verš písně Okoř [3, s. 92] v notaci LilyPond



Obrázek 1.6 Partitura vygenerovaná z výpisu kódu 1.5

1.7.2.3 ChordPro

Jako poslední formát zmíním ChordPro [22]. Jedná se o jednoduchý textový formát určený pro zaznamenání textu a akordů písní. Podmnožina jeho syntaxe se již nyní používá v Kytarovém zpěvníku v editačním formuláři a je podporovaná pro import písně ze souboru `.chordpro`.

```
{start_of_chorus}
[D]Na hradě Okoři [A7]světla už nehoří,
[D]bílá paní [A7]šla už dávno [D]spát.
Ta měla ve zvyku [A7]podle svého budíku
[D]o půlnoci [A7]chodit straší[D]vat[D7].
[G]Od těch dob, co jsou tam [D]trampové,
[E]nesmí z hradu [A7]pryč.
[D]A tak dole v podhradí [A7]se šerifem dovádí,
[D]on ji sebral [A7]od komnaty [D]klíč.
{end_of_chorus}
```

Výpis kódu 1.6 Refrén písně Okoř [3, s. 92] v notaci ChordPro

Texty ve formátu ChordPro se díky jeho jednoduchosti snadno vytvářejí ručně, ale formát má poměrně omezené vyjadřovací schopnosti – umožňuje označit akord, sloku s refrénem a zadat metadata jako je například název písně.

Vzhledem k tomu, že nepoužívá žádný standardizovaný značkovací jazyk (jako například výše zmíněné XML), nelze pro jeho načtení použít příslušné parsery, a je nutné formát zpracovávat například pomocí regulárních výrazů nebo stavovým automatem znak po znaku.

Pro jazyk PHP je k dispozici knihovna [23], která formát umí načíst a převést do HTML, JSON nebo prostého textu.

1.7.3 Knihovny pro generování PDF

Protože v požadavku F4 se budu zabývat exportem do PDF, rozhodl jsem se do rešerše zařadit i PHP knihovny pro generování PDF dokumentů. Knihovny jsem vybral na základě jejich popularity v repozitáři balíčkovacího nástroje Composer [24].

1.7.3.1 FPDF

Jako první zmíním knihovnu FPDF [25], která umožňuje vytvářet obsah PDF dokumentu postupným voláním metod knihovny. Poskytované API je poměrně strohé – k dispozici jsou pouze metody pro vložení textu, základních geometrických útvarů a omezené stylování. Složitější struktury (například tabulky) musí konzument API vytvořit ručně.

Knihovna nevyžaduje žádné PHP rozšíření a podporuje PHP 5.1 a vyšší, PHP 7 a PHP 8. K dispozici je dokumentace API a hotové příklady nejčastějších využití. Podle oficiální webové stránky je knihovna licencovaná pod permisivní licencí (tj. není nijak omezené její použití [26, s. 63]), ačkoliv nepoužívá žádnou známou open-source licenci.

1.7.3.2 TCPDF

Další knihovnou, která generuje PDF imperativním způsobem (jako zmíněná FPDF), je TCPDF [27]. Její API je mnohem rozsáhlejší – například umí generovat čárové kódy, poskytuje více možností stylování.

TCPDF nevyžaduje žádné PHP rozšíření a podporuje PHP 5.3 až PHP 7.2, vyšší verze PHP generují varování o používání zastaralých funkcí v knihovním kódu. Podle oficiální webové stránky se dále knihovna vyvíjet nebude a pro nové projekty je doporučena knihovna tc-lib-pdf [28] od stejného autora. U této knihovny je ale upozornění, že není připravena pro produkční nasazení a velká část funkcionality není implementována.

Obě knihovny jsou licencované pod open-source licencí GNU LGPL 3, tj. lze je používat bez omezení za podmínky, že modifikace kódu knihovny musí být zveřejněné pod stejnou licencí. Zveřejnění kódu celé aplikace, která knihovnu používá, není vyžadováno [29].

1.7.3.3 mPDF

Takto knihovna se používá v aktuální implementaci Kytarového zpěvníku. PDF dokument generuje převodem z HTML kódu. Podpora HTML a CSS standardů je ale omezená, například nelze vložit blokový element do tabulky a ne všechny vlastnosti CSS jsou implementovány. Proto je běžnou praxí, že pro mPDF se vytváří speciální HTML dokumenty a nekonvertují se dokumenty, které se posílají webovým prohlížečům.

mPDF nevyžaduje žádné PHP rozšíření a podporuje PHP 5.6 a vyšší, PHP 7 a PHP 8. K dispozici je jak dokumentace API, tak rozsáhlý manuál včetně ukázek kódu. Knihovna je licencovaná pod GNU GPL 2, tedy ji lze používat za podmínky, že zbytek aplikace bude mít stejnou licenci [30]. Autoři knihovnu považují za *feature complete* – plánují ji udržovat, ale ne přidávat nové funkce.

1.7.3.4 dompdf

Metodu generování PDF souboru z HTML kódu používá i knihovna dompdf [31]. Podporuje téměř všechny vlastnosti CSS 2.1 a HTML 4, v dnešní době se však jedná o starší standardy, proto se i pro dompdf vytváří speciální HTML dokumenty tak, aby je knihovna byla schopná zpracovat.

Knihovna nevyžaduje žádné PHP rozšíření a podporuje PHP 7.1 a vyšší a PHP 8. API dokumentace není k dispozici, součástí projektu ale je manuál s příklady. Licence knihovny je GNU LGPL 2.1 [32].

Kapitola 2

Návrh

Tato kapitola navazuje na provedenou analýzu a obsahuje návrh řešení specifikovaných požadavků, tj. budu popisovat změny, které v Kytarovém zpěvníku plánuji provádět, a novou funkcionalitu, kterou do něj budu přidávat. Zatímco kapitola Analýza se v částech 1.1–1.4 zabývala stávajícím stavem aplikace, nyní se jedná o výstupy mé vlastní práce. Vytvořený návrh řešení bude sloužit jako detailní podklad k následné realizaci.

2.1 Přehled řešených problémů

Dle specifikace požadavků a analýzy současné implementace jsem identifikoval a v seznamu níže popsal několik důležitých kroků, které bude nutné provést a které budou mít zásadní vliv na zvolenou architekturu a způsob řešení.

- **Návrh formátů pro uložení schématu akordu a textu písně**

Požadavek F2 (schémata akordů) je zcela nový, aplikace dosud schémata nepodporuje. Proto je nutné vybrat formát, ve kterém se budou schémata definovat.

Požadavky F1, F3 a F4 pak vyžadují rozšíření možností stávající reprezentace textu písně a akordů (vizte sekci 1.4.1) a komplexnější práci s textem písně. Stávající způsob ukládání textu písně do databáze je k dalšímu rozšiřování nevhodný, proto je potřeba jej nahradit takovým, který bude mít více vyjadřovacích schopností a bude do budoucna lépe rozšiřitelný.

- **Doplnění možnosti označit akord jako netisknutý**

Po konzultaci s vedoucím práce jsme stanovili, že ačkoliv formát textu písně v databázi se změní, formát, ve kterém se píseň vkládá do formuláře ve webovém rozhraní, zůstane stejný. Pro implementaci požadavku F1 tedy bude nutné doplnit možnost označit akordy v bloku textu jako netisknuté.

- **Generování obrázku se schématem akordu**

Formát pro definici schématu akordu bude popisovat elementy schématu (vizte sekci 1.5.2). Obrázek, který bude webový prohlížeč schopný zobrazit, z něj bude nutné následně vygenerovat.

- **Převod textu písně mezi různými formáty**

Vzhledem k tomu, že formát pro zadávání textu písně se měnit nebude, bude nutné implementovat obousměrný převod mezi formátem pro uložení do databáze a formátem pro editaci.

Protože se změní formát, ve kterém je text písně uložený, bude nutné přepsat i logiku generující HTML kód pro stránku s detailem písně a generování PDF exportu.

Z provedené specifikace požadavků vyplývají i další nutné úkoly (například vytvoření administrace schémat akordů), ty však pro návrh řešení nejsou tolik podstatné. Budu se jimi zabývat v kapitole pojednávající o realizaci.

2.2 Architektura

Vzhledem k tomu, že většina funkčních požadavků se týká textu písní a akordů, zaměřím se při návrhu zejména na tento aspekt Kytarového zpěvníku. Z popisu specifikovaných požadavků a z provedené analýzy současné implementace vyplývá, že pro splnění požadavků bude potřeba provést zásadní změny a rozšíření v implementaci uložení a zpracování textu písně, zobrazení písní a v PDF exportu.

Zároveň se ale implementace týká jen specifických komponent aplikace, lze ji snadno ohraničit a specifikovat rozhraní, které bude zbytek aplikace používat. Z těchto důvodů se nabízí existující implementaci úplně nahradit a novou vytvořit jako knihovnu, která se do Kytarového zpěvníku zaintegruje. Součástí knihovny bude specifikace formátů pro definici schématu akordu a textu písně a PHP implementace pro práci s těmito formáty.

Ačkoliv typickým znakem softwarové knihovny je, že implementaci v ní obsaženou používá více různých aplikací, a navrhovanou knihovnu bude používat pouze Kytarový zpěvník (alespoň v době psaní této práce), tento návrh dokáže přinést určité výhody. Zejména se jedná o vypracování podrobně zdokumentovaných formátů pro ukládání akordů a textů, které usnadní budoucí vývoj aplikace, a vynucení vytvoření kvalitního rozhraní a dodržení principů *low coupling* a *high cohesion*. Tyto principy tvrdí, že kód jednoho modulu softwarového systému by měl být co nejméně provázaný se zbytkem systému a měl by obsahovat co nejméně kódu nesouvisejícího se zaměřením modulu [33, s. 76 a 95].

Protože v současné implementaci není parsování textu písně při vkládání a jeho formátování při výpisu vyčleněno do samostatného modulu, ale kód je umístěn v několika kopiích v presenterech (vize sekci 1.2.2), v rámci své práce bych provedl refaktoring tak, aby byla implementace čistší. V tom kontextu tedy vyčlenění této logiky do samostatné knihovny a její nahrazení v Kytarovém zpěvníku voláním knihovních metod nepředstavuje zásadní zvýšení náročnosti oproti implementaci požadavků přímo v kódu aplikace.

2.3 Způsob řešení

Na základě výše popsaných argumentů jsem se rozhodl pro variantu implementace části funkcionality odděleně od kódu aplikace v rámci knihovny. Knihovna bude součástí výstupu mé práce a provedu její integraci do Kytarového zpěvníku tak, abych splnil specifikované požadavky. Stejně jako zbytek aplikace, i knihovna bude splňovat nefunkční požadavky.

2.3.1 Funkcionalita knihovny

Knihovna bude zodpovědná za následující části funkcionality:

- **Specifikace formátů pro ukládání schémat akordů a textů písní**

Na straně knihovny budou textově (čitelně pro lidi) i formálně definovány formáty, v jakých se bude text písně a definice akordu ukládat. Zároveň bude knihovna poskytovat rozhraní a implementaci pro čtení a zápis těchto formátů.

- **Objektová reprezentace textu písně a akordu**

V programovém kódu knihovny budou definovány třídy, které budou při běhu programu reprezentovat text písně a definici akordu. Tyto třídy budou součástí rozhraní knihovny, čtení z uloženého formátu tedy vytvoří jejich instance a metody pro zápis budou na vstupu očekávat tuto objektovou reprezentaci.

- **Vykreslování schémat akordů do obrázku**

Vzhledem k tomu, že hlavním využitím schématu akordu je v jeho grafickém zobrazení, knihovna bude poskytovat v rámci svého rozhraní implementaci vykreslující obrázek se schématem akordu z objektové reprezentace.

- **Převod textu písně do HTML a PDF podoby**

Podobně jako u bodu s vykreslováním schématu akordu lze i pro text písně předpokládat, že uživatelé knihovny budou chtít text písně lidsky čitelným způsobem prezentovat. Proto knihovna ve svém rozhraní bude mít i implementaci převodu objektové reprezentace písně do HTML kódu a PDF dokumentu.

2.3.2 Úpravy aplikace

Zbylou funkcionalitu nutnou pro splnění specifikovaných požadavků lze považovat za specifickou pro Kytarový zpěvník, nemá tedy smysl ji implementovat v rámci knihovny. Jedná se například o administrační rozhraní, úpravy uživatelského rozhraní, ale i textový formát pro editaci textu písně prostřednictvím formuláře ve webovém rozhraní a formát dat v REST API.

Tyto záležitosti proto budu implementovat v kódu aplikace (respektive upravím existující implementace). Z hlediska návrhu řešení nejsou příliš zajímavé a z toho důvodu se jim budu věnovat až v kapitole 3 v rámci realizace.

2.4 Formáty pro uložení schémat akordů a textů písní

V sekcích 1.5.1 a 1.5.2 jsem vysvětlil, jaké informace a datové struktury musí formáty uchovávat. Z rešerše v sekci 1.7.2 vyplývá, že žádný existující formát není vhodný (ani pro definici schématu akordu, ani pro text písně). Proto si nyní stanovím kritéria a navrhnu nové formáty (jeden pro definici schématu akordu a druhý pro uložení textu písně s akordy), které při realizaci použiji.

Protože navrhované formáty budou použity pouze při strojovém zpracovávání, není nutné, aby je mohl člověk snadno editovat. Naopak by měl existovat jednoduchý způsob, jak je v běžně používaných programovacích jazycích[1] načíst a zpracovat.

Nový způsob ukládání textu písně navrhuji zejména proto, že ten existující nelze rozumně rozšířit tak, aby s ním bylo možné implementovat specifikované požadavky. Lze očekávat, že Kytarový zpěvník se v rámci předmětů BI-SP1 a BI-SP2 bude dále vyvíjet, proto je dalším kritériem pro návrh formátů jejich budoucí snadná rozšiřitelnost.

Dvěma nejčastěji používanými formáty pro ukládání a přenášení strukturovaných dat jsou v dnešní době JSON a XML. V rámci syntaxe těchto formátů je možné vytvářet dokumenty s libovolnou strukturou a daty podle řešené domény.

[1] Nejen PHP, v budoucnu může formáty načítat například i mobilní aplikace.

2.4.1 JSON

JSON je jednoduchý formát pro výměnu dat založený na dvojicích klíč a hodnota a na seznamech hodnot [34]. Knihovny (případně nativní metody) pro parsování a serializaci (generování JSON textu z dat v paměti programu) jsou k dispozici v téměř každém programovacím jazyce.

```
{
    "klíč": "hodnota",
    [
        "seznam",
        3,
        "hodnot"
    ]
}
```

Výpis kódu 2.1 Ukázka formátu JSON

2.4.2 XML

XML je značkovací jazyk, struktura dokumentu se skládá z elementů, jejich atributů a textu. Počáteční značka elementu (tzv. tag) je tvořená názvem elementu a případnými atributy uzavřenými mezi znaky *menší než* a *větší než*. Po počátečním tagu následuje obsah elementu ukončený uzavíracím tagem, který tvoří znaky menší než, lomítko, název elementu a větší než. Elementy bez obsahu se zkracují na počáteční tag, který má před koncovým větším lomítko. Atributy se deklarují názvem atributu, rovnítkem a hodnotou atributu. Elementy nelze křížit, to znamená, že element `A`, který začal uvnitř `B`, musí skončit před koncem `B`.

```
<tag atribut="hodnota">te<x>x</x>t</tag>
<seznam>
    <prvek>hodnota</prvek>
    <prvek>seznamu</prvek>
</seznam>
```

Výpis kódu 2.2 Ukázka formátu XML

Téměř všechny programovací jazyky umějí XML dokumenty parsovat a vytvářet, případně jsou k dispozici knihovny. Kolem XML existuje celá řada podpůrných nástrojů a technologií. Například se jedná o XSD, což je doporučení konsorcia W3C, jak formálně popsat schéma XML dokumentu. Na základě takového formálního popisu je možné ověřit, že XML dokument je validní, nebo dokonce takový dokument (například s náhodnými daty) vygenerovat.

2.4.3 Zvolený formát

Oba dva formáty splňují definovaná kritéria. Výhodou XML je existence širšího ekosystému navázaných technologií, ale zejména to, že se jedná o značkovací jazyk, tedy je pro uložení textu písně vhodnější.

Na základě této rešerše jsem se rozhodl, že schémata akordů a texty písní budu ukládat a zpracovávat jako XML dokumenty. Dále v této práci vytvořím specifikaci obou typů dokumentů.

2.5 Objektová reprezentace akordů a textů písní

Program (v tomto případě prostřednictvím knihovny) musí nějakým způsobem za běhu reprezentovat ve své paměti načtené schéma akordu nebo text písně. Jazyk PHP podporuje objektově orientovaný styl programování, naprosto intuitivně se tedy nabízí vytvořit třídy, jejichž instance budou akord a píseň reprezentovat.

V následujících dvou sekcích uvedu třídní model objektové reprezentace. Již nyní ale poukáži na vlastnost, kterou budou mít všechny třídy z modelu společnou. Rozhodl jsem se pro takový návrh, kde se data do atributů předají při vytvoření instance, atributy budou nepřístupné zvenčí (označené jako privátní) a třídy budou implementovat pouze metody pro čtení atributů, nikoliv pro změnu. To znamená, že jednou vytvořená instance je neměnná.

Tento princip se nazývá *immutable objects* [35, s. 131–132] a často se používá právě u doménových objektů. Takové objekty se typicky používají napříč celou aplikací a jejich neměnnost zajistí, že nebude možné nechtěně změnit stav objektu a tím ovlivnit všechny části aplikace, které mají na objekt referenci.

2.5.1 Akord

Jak vypadá schéma akordu jsem popsal již v sekci 1.5.2. Z popisu lze přímočaře odvodit, že schéma je možné v objektovém systému reprezentovat následujícím způsobem:

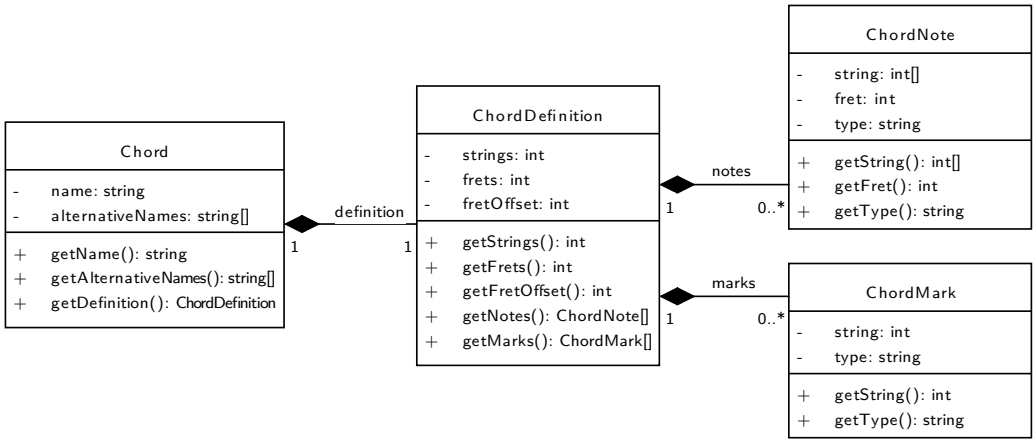
1. Kontejnerem pro celý akord bude třída `Chord`. Atribut `name` bude obsahovat jméno akordu, atribut `alternativeNames` pak seznam případných alternativních názvů. Schéma akordu bude definované v atributu `definition` instancí třídy `ChordDefinition`.
2. Třída `ChordDefinition` bude mít atribut `strings` pro určení počtu strun nástroje a `frets` pro počet pražců. Zmínil jsem, že některé akordy se hrají níže na hmatníku, který se pak nevyobrazuje celý. Začátek zobrazené části udá atribut `fretOffset` (například pokud je počet pražců 7 a hodnota atributu 3, ve schématu budou vidět pražce iv–vii).

   V atributu `notes` bude seznam instancí `ChordNote`, který bude reprezentovat stisknuté struny. Atribut `marks` pak bude obsahovat instance `ChordMark`, které budou definovat značky nad strunami.
3. Pro určení pozice stisknuté struny bude mít třída `ChordNote` atributy `string` a `fret`. Stisk může být nepovinný, typ stisku bude uveden v atributu `type`. Možné typy budou uvedené v konstantách[2] začínajících na `TYPE_` v třídě `ChordNote`.

   Atribut `string` bude obsahovat pole o jednom až dvou prvcích. Pokud v něm budou dva prvky, daný `ChordNote` bude označovat barré mezi strunami uvedenými v poli.
4. Třída `ChordMark` reprezentující značku nad strunou bude mít atribut `string` udávající ovlivněnou strunu. Atribut `type` určí druh značky a bude nabývat hodnot konstant začínajících na `TYPE_` v třídě `ChordMark`.

Pro větší názornost v obrázku 2.1 uvádím diagram tříd v notaci UML. V diagramu lze vidět privátní atributy tříd (uvozenené znakem mínus) a veřejné metody (uvozené znakem plus). Propojení s plným diamantem značí kompozici – například nedílnou součástí instance třídy `Chord` je `ChordDefinition`. [36, s. 35–39 a 67]

[2] Jazyk PHP podporuje výčtové typy až od verze 8.1, pro kód kompatibilní s nižšími verzemi je běžnou praxí výčet nahradit konstantami v třídě.



Obrázek 2.1 Diagram tříd pro schémata akordů

2.5.2 Píseň

Objektová reprezentace písně taktéž bude odpovídat popisu v sekci 1.5.1, pro každý druh objektu v zápisu písně (například akord, repetici nebo sloku) vytvořím příslušnou třídu. Všechny druhy objektů budou implementovat rozhraní `Node`, objekty, které obsahují další objekty (například sloka obsahuje verše), navíc implementují rozhraní `NodeContainer`.

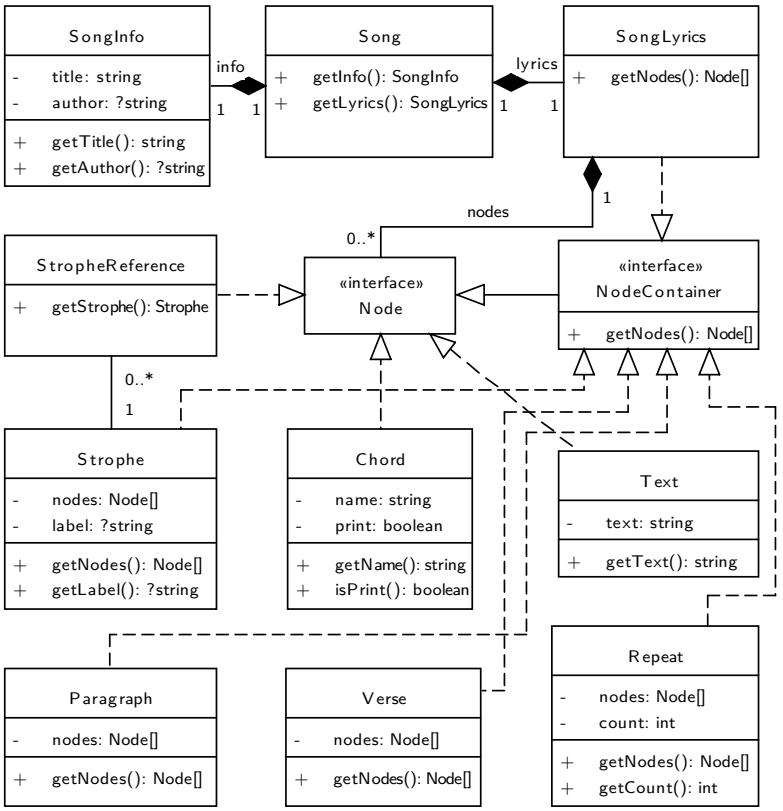
1. Třída `Song` bude zapouzdřovat celou píseň.
2. `SongInfo` bude obsahovat základní metadata písně, konkrétně její název a nepovinně interpreta.
3. Samotný text písně bude obsažen v `SongLyrics`, která již implementuje rozhraní `NodeContainer`. Přímým obsahem `SongLyrics` může být buď repetice, sloka nebo odkaz na ni.
4. Sloku reprezentuje třída `Strophe`, která implementuje `NodeContainer` a obsahuje nepovinný atribut `label`. Tento atribut nese její označení, může to být například *1* pro první sloku nebo *R* pro refrén, ze strany knihovny ale nejsou dána žádná omezení. Ve sloce se mohou vyskytovat jen repetice nebo odstavce.
5. Sloka se může dělit na více odstavců, každý z nich bude reprezentován instancí třídy `Paragraph` (implementuje `NodeContainer`). Povoleným obsahem odstavce jsou pouze repetice a verše.
6. Jednotlivé řádky odstavce jsou verše, odpovídá jim třída `Verse` (taktéž implementuje `NodeContainer`). Ve verších může být uložen text, akordy nebo repetice.
7. Čistě textové části budou vyjádřeny jako instance třídy `Text`.
8. Akordy budou reprezentované instancemi třídy `Chord`. Tato třída nijak nesouvisí s třídou `Chord` z předcházející sekce, která obsahuje definici schématu akordu; obě třídy budou v odlišném jmenném prostoru. Tato třída bude jméno akordu a pravdivostní atribut `print`, který bude udávat, zda se má daný akord objevit v tiskovém výstupu.

   Knihovna neposkytuje žádné úložiště textů písní a definic akordů, to bude implementováno až v kódu aplikace Kytarového zpěvníku, proto i párování názvu akordu z textu písně na definici jeho schématu bude zajišťovat aplikace.
9. Již několikrát zmíněná repetice odpovídá třídě `Repeat`. V repetici může být obsaženo více objektů, proto implementuje rozhraní `NodeContainer`. Atribut `count` udává počet opakování obsahu repetice a musí nabývat hodnoty nejméně 2.

10. Populární písně obvykle obsahují speciální sloku, refrén, která se v průběhu písně opakuje typicky po každé sloce. Většinou se v textu písně text refrénu neopakuje, jen se na příslušné místo vloží jeho značka. To bude umožňovat třída `StropheReference`, která odkazuje na jinou sloku, která by se na jejím místě měla hrát. Implementace povoluje odkazování na libovolnou sloku, nejen na refrén.

Implementace `NodeContainer` svůj obsah vyjadřují jako seznam dalších objektů. Pro verše bude platit, že akord patří k textu za ním následujícímu, to znamená, že v grafickém výpisu sekvence `Text(ces)Chord(D)Text(ta)` se akord *D* zobrazí nad písmenem *t*.

Podobně jako u akordu, i zde pro přehlednost uvedu diagram tříd. Čárkovaná čára s trojúhelníkem na konci označuje tzv. realizaci – to znamená, že třída implementuje rozhraní, na které trojúhelník ukazuje. Plná čára značí dědění. [36, s. 45 a 69–70]



Obrázek 2.2 Diagram tříd pro text písně

2.6 Schéma XML dokumentů

Obecný XML dokument může obsahovat libovolné elementy, atributy a jejich hodnoty (pochopitelně s respektem k syntaktickým pravidlům XML). Elementy se mohou libovolně zanořovat, opakovat nebo chybět. XML schéma podobu dokumentu upřesňuje tím, že formálně definuje povolenou sadu elementů, atributů, hodnot a strukturu dokumentu.

Standardem pro strojově zpracovatelnou definici schémat se stalo XSD, což je sám o sobě XML dokument, který popisuje schéma jiného dokumentu [37], jak jsem uvedl v předcházejícím odstavci. XML dokument lze validovat oproti XSD – to znamená automaticky ověřit, že danému schématu odpovídá. Tím se mohou autoři dokumentů ujistit, že je aplikace očekávající data s daným schématem budou schopné zpracovat.

Kompletní příklady XML dokumentů podle navrhovaných formátů jsou k nahlédnutí v příloze A, v této kapitole budu jejich schéma popisovat slovně a graficky.

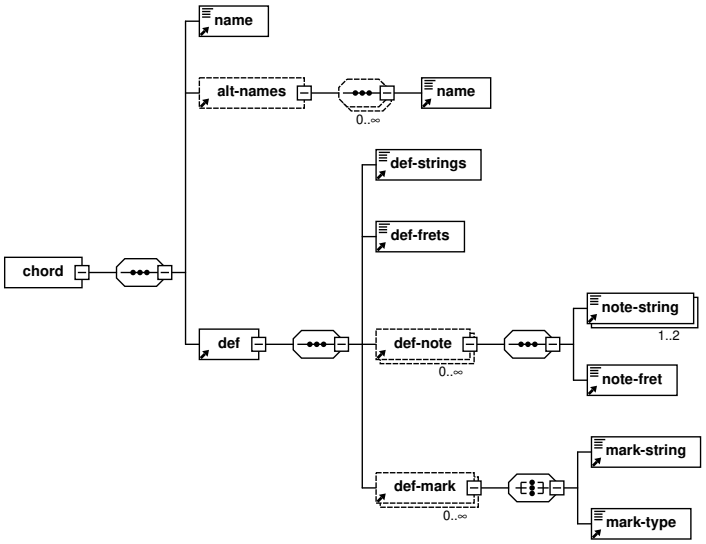
2.6.1 Akord

Navržené schéma XML dokumentu, který definuje akord, je inspirované definicí akordů ve formátu MusicXML (vize sekci 1.7.2.1) a kopíruje objektovou reprezentaci ze sekce 2.5.1. Odsazení v následujícím výpisu odpovídá tomu, jak do sebe budou elementy zanořené.

`<chord>`: kořenový element dokumentu

- `<name>`: jméno akordu
- `<alt-names>`: alternativní jména akordu, nepovinné
  - `<name>`: alternativní jméno akordu
- `<def>`: definice schématu akordu
  - `<def-strings>`: počet strun nástroje, hodnotou je přirozené číslo
  - `<def-frets>`: počet pražců ve schématu, hodnotou je přirozené číslo. Nepovinný atribut `offset`, hodnotou je nezáporné celé číslo, omezuje zobrazovanou oblast pražců.
  - `<def-note>`: stisknutá struna
    - `<note-string>`: číslo struny (zleva doprava, začíná jedničkou). U barré akordu se tento element vyskytne dvakrát, jednou pro strunu, kterou barré začíná, podruhé pro strunu, kterou končí.
    - `<note-fret>`: číslo pražce (ze shora dolů, začíná jedničkou)
  - `<def-mark>`: značka nad strunou
    - `<mark-string>`: číslo struny (zleva doprava, začíná jedničkou)
    - `<mark-type>`: druh značky, povolené hodnoty jsou `muted` nebo `open`

Pro přehlednost uvádím XML schéma i v grafické podobě (obrázek 2.3), kde jednotlivé elementy XML dokumentu jsou znázorněny obdélníky a zanoření je podle spojujících čar zleva doprava.



Obrázek 2.3 Schéma XML dokumentu s definicí akordu

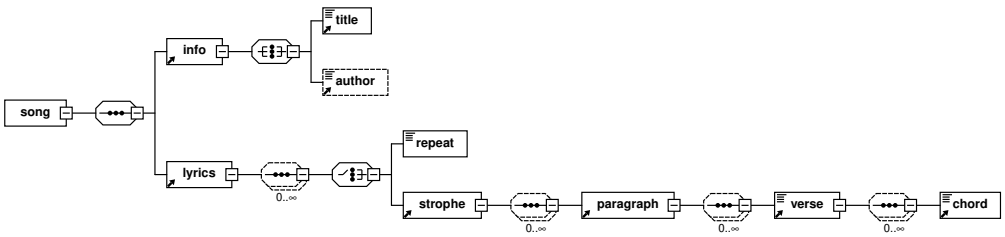
2.6.1.1 Píseň

Schéma XML dokumentu s textem písně odpovídá objektové reprezentaci uvedené v sekci 2.5.2. Odsazení ve výpise níže znázorňuje, jak jsou do sebe elementy zanořené.

`<song>`: kořenový element dokumentu
- `<info>`: základní[3] metadata o písni
  - `<title>`: název písně
  - `<author>`: jméno autora písně, nepovinné
- `<lyrics>`: text písně
  - `<strophe>`: sloka. Nepovinný atribut `label` obsahuje označení sloky (například číslo nebo označení refrénu), hodnotou je libovolný text.
    - `<paragraph>`: odstavec ve sloce
      - `<verse>`: verš (řádek odstavce)
        - *čistý text*: text verše
        - `<chord>`: akord. Nepovinný pravdivostní atribut `print` udává, zda se má akord objevit v tiskovém výstupu; výchozí hodnota je pravda.

V kterékoliv úrovni zanoření pod `<lyrics>` se může vyskytovat element `<repeat>`, který značí, že obsah v něm uzavřený je v repetici. Počet opakování označuje nepovinný atribut `count` s výchozí hodnotou 2.

Pro přehlednost opět uvádím i grafické znázornění schématu dokumentu. Funkce elementu `<repeat>` je v obrázku 2.4 zjednodušena – jak jsem vysvětlil, element se smí nacházet na libovolné úrovni pod elementem `<lyrics>`.



Obrázek 2.4 Schéma XML dokumentu s textem písně

2.7 Interakce aplikace a knihovny

Z představeného způsobu řešení vyplývá, že největší změnou v aplikaci bude přepracování způsobu uložení textu písně a klíčovým bodem interakce knihovny a aplikace bude čtení datových formátů a generování výstupů (obrázek, webová stránka nebo PDF dokument). V této závěrečné sekci návrhu řešení vysvětlím, jak budou tyto procesy probíhat.

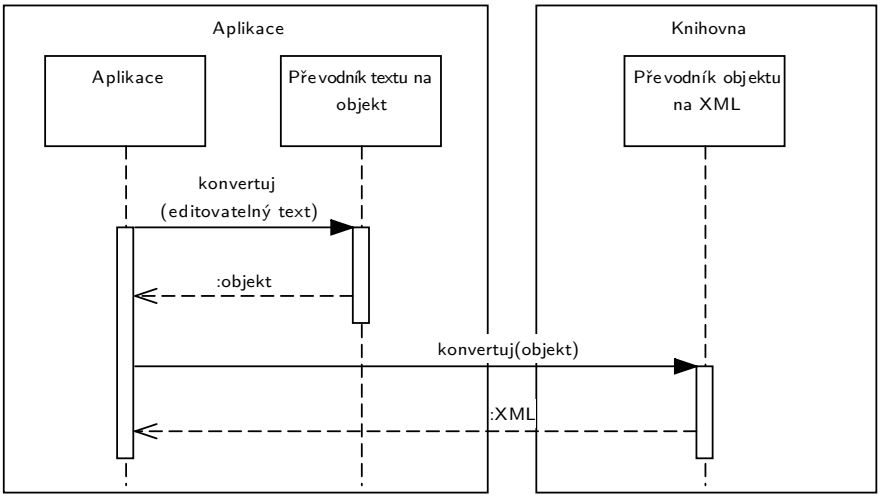
Pro názornost použiji jednoduché sekvenční diagramy v UML notaci, které zachycují posloupnost interakce mezi objekty. V horní části každého diagramu jsou v obdélnících uvedené objekty zapojené do procesu. Horizontální čáry s šipkami znázorňují interakci objektů, plná čára znamená volání, přerušovaná odpověď. [36, s. 53–55]

[3] Kytarový zpěvník k písním ukládá i další metadata, ale protože navržení formát má být nezávislý na konkrétní aplikaci, rozhodl jsem se do XML zahrnout jen název a autora. Ostatní metadata si aplikace uloží mimo (například do patřičného sloupce v databázi).

▶ Editovatelný text → XML

Tento scénář se použije při vkládání textu písně uživatelem a jeho ukládání do databáze. Nejprve je potřeba text převést na objektovou reprezentaci písně a následně uložit v XML formátu.

Jak jsem zmínil v sekci 2.3.2, formát textu editovatelného skrze webové rozhraní Kytarového zpěvníku je specifický pro aplikaci, proto je tento převod umístěn do kódu aplikace, nikoliv knihovny.

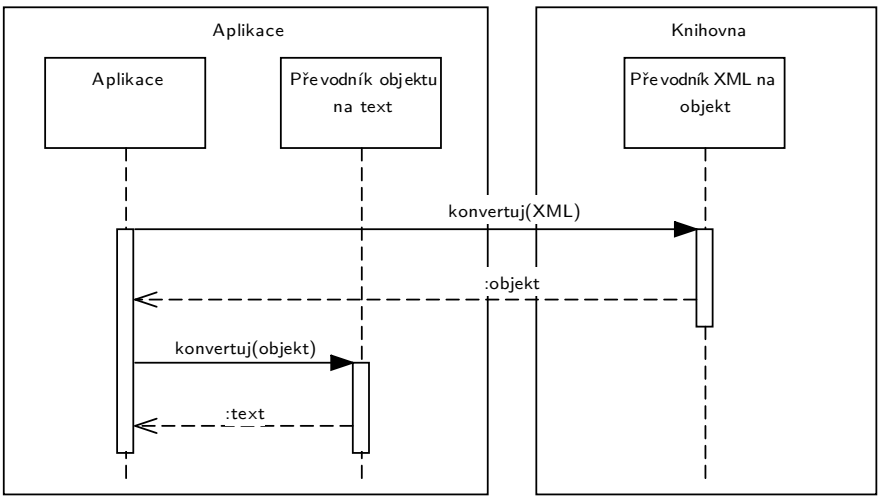


Obrázek 2.5 Sekvenční diagram převodu editovatelného textu na XML

Analogicky bude fungovat konverze z formátu dat, které používá REST API, na XML. Místo převodníku z textu se použije převodník z datové struktury API.

▶ XML → editovatelný text

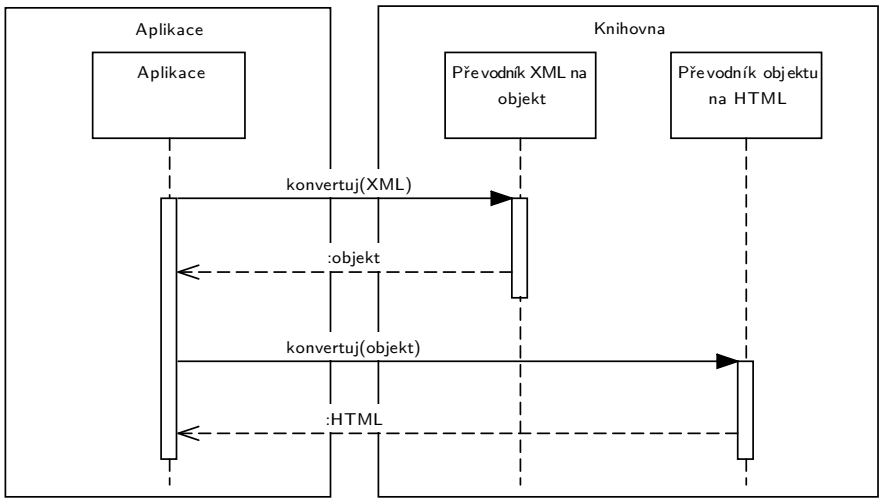
Zde se jedná o opačný směr než v předchozím případě, tedy při editaci textu existující písně. Myšlenka tohoto procesu je ale stejná, včetně aplikování na data pro REST API.



Obrázek 2.6 Sekvenční diagram převodu XML na editovatelný text

▶ XML → HTML pro webové zobrazení

Tento proces proběhne při zobrazení stránky s detailem písně, kde je potřeba převést XML strukturu uloženou v databázi na zobrazitelný HTML kód. Implementaci tohoto převodu jsem se rozhodl umístit do knihovny, jak jsem vysvětlil v sekci 2.3.1.



■ **Obrázek 2.7** Sekvenční diagram převodu XML na HTML

Knihovna poskytuje implementaci i pro převod do PDF dokumentu, takový proces bude vypadat stejně, jen místo převodníku do HTML použije převodník do PDF.

Kapitola 3

Realizace

Další fází mé závěrečné práce je realizace požadavků a kroků, které byly v předcházejících dvou kapitolách analyzovány a navrženy. Než začnu s popisem samotné implementace, představím nástroje a postupy pro dokumentaci a řízení softwarového projektu, které jsou v projektu zavedené. Pak se zastavím u XML dokumentů navržených v předcházející kapitole, a vytvořím pro ně skutečně formální definici.

Následně budu pokračovat v popisu řešení podle toho, jak na sebe jednotlivé fáze navazují. Nejprve uvedu, jak jsem implementoval knihovnu na základě vytvořeného návrhu, jakým způsobem jsem organizoval její kód, jaké návrhové vzory jsem použil a jaký jsem zvolil algoritmus pro sázení textu písně do PDF.

Realizace bude pokračovat integrací knihovny do aplikace, kde vysvětlím, jakým způsobem jsem knihovnu a Kytarový zpěvník propojil, jaké úpravy jsem prováděl v backendu aplikace a jak vypadá rozšířený editační formát textu písně.

Na závěr této kapitoly se zaměřím na úpravy Kytarového zpěvníku z hlediska frontendu a uvedu, jak se změnilo jeho uživatelské rozhraní, aby byly zpřístupněny nové funkce.

Paralelně s realizací mé práce pracoval v rámci výuky v předmětu BI-SP1 studentský tým na mobilní aplikaci Kytarového zpěvníku, která využívá REST API poskytované webovou aplikací. Během jejich práce objevili několik chyb, které jsem po domluvě s vedoucím nad rámec původního zadání opravil. Jednalo se o jednotky problémů a podrobně se jim zde věnovat nebudu.

3.1 Dokumentace

V této sekci popíši, jakým způsobem se v projektu tvoří a organizuje dokumentace, a to jak programátorská, tak analytická, administrátorská i uživatelská. Všechny zmíněné formy dokumentace již existují, já jsem je během realizace upravil a doplnil tak, aby reflektovaly změny v aplikaci a obsahovaly potřebné informace o nově implementované funkcionalitě.

3.1.1 Enterprise Architect

Asi nejdůležitějším dokumentačním nástrojem je v tomto případě Enterprise Architect. Řadí se mezi tzv. CASE nástroje, které mají za úkol podporovat činnosti softwarového inženýrství. Umožňují vytváření komplexní analytické dokumentace, modelování (například objektové nebo E/R modely), sběr požadavků a další (záleží také na konkrétním CASE nástroji) [38, s. 4–6].

Projekt Kytarového zpěvníku v Enterprise Architectu obsahuje modely obchodních procesů, funkční i nefunkční požadavky, stavové diagramy, modely případů užití, doménový a E/R model, sekvenční diagramy a diagram architektury s instalační příručkou.

Aktualizoval jsem doménové a E/R modely tak, aby odpovídaly změnám provedeným během realizace. Zároveň jsem do projektu doplnil diagramy vytvořené v souvislosti s návrhem a realizací knihovny (diagramy balíčků, tříd a sekvenční diagramy).

3.1.2 Instalační příručka

Administrátor aplikace (v tomto případě vedoucí práce) je zodpovědný za nasazování nových verzí dodaných studenty na svůj webhosting. K tomu potřebuje konkrétní instrukce, jaké soubory aplikace na web musí nahrát, jak aplikaci nakonfigurovat a jak provést aktualizaci ze starší verze.

Tyto údaje (včetně například požadavků na konfiguraci serveru) jsou obsažené v instalační příručce. Jedná se o dokument, který je uložený v Enterprise Architectu a zároveň exportovaný do PDF a přibalený v každém balíčku s aplikací, který administrátor obdrží.

Z důvodu změny formátu, ve kterém je uložen text písně, bude nutné při aktualizaci aplikace převést existující záznamy do nového formátu (více vizte dále v sekci 4.5). Instalační příručku jsem proto rozšířil o instrukce pro provedení aktualizace.

3.1.3 Příručka pro vývojáře

Protože Kytarový zpěvník se vyvíjí v rámci předmětů Softwarový týmový projekt 1 a 2 a závěrečných prací, střídání vývojářů je velmi časté. Zároveň na aplikaci pracují i studenti, kteří nemají takové zkušenosti s vývojem webových aplikací.

Aby noví vývojáři mohli se zdrojovým kódem začít rychle pracovat a aplikaci si v lokálním prostředí snadno spustit (vizte sekci 1.3.3.1 o Dockeru), je součástí repozitáře i příručka pro vývojáře. Ta stručně popisuje použité technologie a obsahuje návody, jak nastavit vývojové prostředí, spustit jednotkové testy a vysvětluje další úkony, které je potřeba během vývoje provádět.

Do projektu jsem během realizace nepřidával žádnou novou technologii, ani se nijak zásadně nezměnila struktura aplikace. Příručka pro vývojáře tedy zůstala aktuální.

3.1.4 Dokumentace zdrojového kódu

Během vývoje je důležitá dokumentace přímo ve zdrojovém PHP kódu, ta se tvoří pomocí dokumentačních komentářů u prvku (třídy, atributu, metody a podobně), který se dokumentuje. Tyto komentáře dokáže využít nástroj phpDocumentor pro vygenerování HTML dokumentace zdrojového kódu, kde jsou uvedené všechny třídy, rozhraní, jejich atributy, metody, parametry, datové typy[4], návratové hodnoty a textové popisy.

```
/**
 * Převede objektovou reprezentaci písně do HTML podoby.
 * @param Song $song píseň k převedení
 * @return string HTML kód
 */
public function toHtml($song);
```

Výpis kódu 3.1 Ukázka dokumentačního komentáře metody

[4]PHP je slabě a dynamicky typový jazyk: proměnné nemusejí mít předem deklarovaný typ (interpreter jej přiřadí až při běhu) a během vykonávání kódu se může typ jedné proměnné měnit. Proto se datové typy indikují v komentářích.

Přímo při programování tyto komentáře také zpracovávají IDE – vývojová prostředí, ve kterých programátoři píší zdrojový kód. Na základě informací v nich uvedených umí programátorovi napovídat, jaké metody je možné na objektu zavolat, jaký je jejich návratový typ, co metoda dělá a podobně.

V průběhu realizace jsem nově vytvořené třídy, rozhraní, metody a podobné opatřil dokumentačními komentáři a ve změněných třídách je patřičně aktualizoval.

3.1.5 Uživatelská příručka

Posledním druhem dokumentace, který zde zmíním, je uživatelská příručka. Ta je k dispozici přímo na webu s aplikací a popisuje veškerou dostupnou funkcionalitu. Je rozdělená do sekcí podle tématu, kterého se týkají (například *Písně*, *Zpěvníky* nebo *Hodnocení*). V každé sekci jsou uvedené otázky a problémy, na které mohou uživatelé narazit, a pomocí detailně popsaných postupů jsou zodpovězeny.

Obsah příručky závisí na tom, zda je uživatel přihlášený – nepřihlášeným uživatelům se například nezobrazí sekce o nastavení uživatelského profilu, protože to nemá smysl. Uživatelům s rolí administrátora se navíc zobrazuje administrátorská sekce, kde jsou popsané funkce, které jsou přístupné pouze jim.

Do uživatelské příručky jsem doplnil instrukce k přizpůsobení PDF exportu a aktualizoval návod na formátování textu písně. Do administrátorské části jsem pak přidal návod, jak vytvořit XML schéma akordu.

3.2 Řízení softwarového projektu

Vývoj Kytarového zpěvníku je povětšinou práce týmová, proto je potřeba projekt kvalitně řídit. Řízením softwarového projektu se myslí používané podpůrné nástroje a domluvené postupy a procesy, aby bylo možné vývoj plánovat, sledovat a práce programátorů byla efektivní. Nastavení projektového řízení Kytarového zpěvníku je určené vedoucím práce. V této sekci uvedu nástroje a pracovní postupy, které jsme používali během práce v týmu, já budu v jejich používání pokračovat i v rámci své bakalářské práce.

3.2.1 Verzování zdrojového kódu

Zdrojový kód aplikace ukládáme, sdílíme a verzujeme prostřednictvím systému Git [39]. To je distribuovaný verzovací systém, tedy každý vývojář má na svém stroji lokální kopii zdrojového kódu, ve které provádí změny. Po dokončení své práce vytvoří tzv. *commit*, což je soubor změn napříč jedním nebo více soubory doprovázený textovým popisem provedených změn. Git se vyznačuje podporou nelineární historie projektu, tedy umožňuje historii větvit. Toho využíváme pro oddělení různých fází vývoje – existuje větev s produkčním kódem, vývojovým kódem a více větví s rozpracovanými úkoly.

Sdílení Git repozitáře probíhá prostřednictvím fakultní instance nástroje GitLab [40], což je komplexní software pro hostování Git repozitářů a související činnosti (vizte dále).

3.2.2 Sledování úkolů

Součástí řízení softwarového projektu je evidence úkolů, plánování a sledování jejich průběhu. K tomu využíváme webové rozhraní GitLabu, který úkoly umožňuje sledovat ve formě tzv. *issues*. Úkolem se rozumí ať už požadavek na změnu, nebo hlášení o chybě. Každý úkol má své číslo, pomocí kterého se na něj lze snadno odkázat, nadpis a případně i detailnější popis. Pro odlišení různých typů úkolů, jejich stavu a priority používáme

štítky. Ke sledování průběhu vypracování úkolů GitLab umožňuje vyplnit časový odhad a dosud strávený čas.

Pokud se úkol týká změny ve zdrojovém kódu, vytvoříme po jeho dokončení v GitLabu merge request, což je požadavek na zařazení změn do původní Git větve. Určíme osobu zodpovědnou za zpracování merge requestu; ta má za úkol změny projít, schválit a provést sloučení větví. U vývojových větví je zodpovědná osoba některý z vývojářů, pokud odevzdáváme splněné zadání vedoucímu projektu, je jí právě vedoucí.

3.2.3 Continuous Integration

Při vývoji praktikujeme princip *Continuous Integration*, který doporučuje vývojářům několikrát denně svou práci nahrávat (*integrovat*) do sdíleného repozitáře. Po každé integraci by mělo proběhnout automatické sestavení aplikace a spuštění automatizovaných testů, aby bylo možné rychle odhalit nově zanesené chyby. [41]

V rámci GitLabu je možné nastavit tzv. *pipelines*, což je definice příkazů, které se mají provést po nahrání změn do repozitáře. Repozitář Kytarového zpěvníku je nastavený tak, aby vždy spustil jednotkové testy, statickou analýzu kódu, vygeneroval specifikaci REST API a sestavil nasaditelný balíček s aplikací. Vývojáři a vedoucí projektu tak mají okamžitou (a hlavně automatickou) zpětnou vazbu (například výsledky testů nebo výpisy chyb ze statické analýzy).

3.3 Formální specifikace XML dokumentů

V sekci 2.6 jsem navrhl XML formáty pro definici akordu a zápis textu písně. Dále jsem zmínil, že existuje standard XSD, který slouží pro formální definici schématu XML dokumentu. Je běžnou praxí, že systémy (aplikace, webové služby a podobně), které používají XML pro výměnu dat, publikují XSD schémata, aby si jejich uživatelé mohli jednoznačně ověřit validitu generovaných dokumentů. Proto jsem se rozhodl XSD schémata také sestavit.

XSD schéma je samo o sobě XML dokument s definovanou sadou elementů a atributů, které používá pro popis struktury jiných dokumentů. Při vytváření specifikace jsem využil následující konstrukce:

`sequence` pro definovaní seznamu elementů, které se v dokumentu musí objevit. Elementy se v dokumentu musí objevit ve stejném pořadí, v jakém jsou zapsané v sekvenci. U každého elementu je možné specifikovat, kolikrát nejméně a kolikrát nejvýše se musí v dokumentu objevit.

`choice` pro vytvoření disjunkce několika elementů, to znamená, že na místě `choice` se musí objevit jeden z nich.

`all` slouží k definování seznamu elementů, které se musí v dokumentu objevit, a to v libovolném pořadí. Na rozdíl od `sequence` není možné, aby se nějaký element z `all` seznamu objevil v dokumentu více než jednou.

Jednotlivé konstrukce se do sebe zanořují a tím vytváří celkovou strukturu dokumentu (vizte grafické znázornění v obrázcích 2.3 a 2.4). Pokud tedy v seznamu výše mluvím o tom, že element se může v dokumentu objevit v omezeném počtu, a v konkrétním případě je konstrukce zanořená do jiného elementu, platí omezení jen v kontextu daného zanoření, nikoliv celého dokumentu.

```
<xs:element name="song">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="info" />
      <xs:element ref="lyrics" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="lyrics">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="repeat" />
        <xs:element ref="strophe" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Výpis kódu 3.2 XSD schéma se sekvencí a disjunkcí

Schéma ve výpisu kódu 3.2 definuje, že v elementu `song` se musí za sebou objevit elementy `info` a `lyrics`. Pro element `lyrics` pak platí, že jeho obsahem musí být sekvence disjunkcí elementů `repeat` a `strophe`. To ve výsledku znamená, že v elementu se může vyskytovat libovolný počet `repeat` a libovolný počet `strophe` a to v libovolném pořadí.

```
<xs:element name="chord" type="xs:string" />

<xs:element name="verse">
  <xs:complexType mixed="true">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="repeat" />
        <xs:element ref="chord" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:simpleType name="markType">
   <xs:restriction base="xs:string">
    <xs:enumeration value="open" />
    <xs:enumeration value="muted" />
  </xs:restriction>
</xs:simpleType>
```

Výpis kódu 3.3 XSD schéma s textovým obsahem elementů

Do schématu lze zahrnout i elementy, které mají mít čistě textový obsah, ve výpise kódu 3.3 tomu odpovídá element `chord` s datovým typem `xs:string`. Například element `verse` ale obsahuje jak čistý text, tak další elementy. Povolené elementy jsou definované uvnitř již vysvětlené `sequence` a atribut `mixed="true"` u elementu `xs:complexType` po-

volí mezi ně vkládat i čistý text. Na textový obsah lze klást dodatečná omezení, jak ukazuje definice datového typu `markType`, která jako obsah elementu s tímto typem povoluje pouze hodnoty z výčtu `open` a `muted`.

```
<xs:element name="def-frets">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:positiveInteger">
        <xs:attribute name="offset" type="xs:nonNegativeInteger" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Výpis kódu 3.4 XSD schéma s atributem

Kromě elementů je samozřejmě nutné definovat i povolené atributy. Ve výpise kódu 3.4 je definován element `def-frets`, jehož datovým typem je `xs:positiveInteger` rozšířený o volitelný atribut `offset` s datovým typem `xs:nonNegativeInteger`. To znamená, že textovým obsahem elementu `def-frets` musí být jakékoliv celé číslo větší než nula a element může mít atribut `offset`, jehož hodnota musí být jakékoliv celé číslo větší nebo rovné nule. Pokud chceme deklarovat, že atribut je povinný, k elementu `xs:attribute` přidáme atribut `use="required"`.

Výsledkem formální definice jsou soubory `chord-1.xsd` a `song-1.xsd`, které obsahují specifikaci dokumentu pro definici akordu, respektive textu písně. Oba soubory jsou součástí dokumentace knihovny.

3.4 Implementace knihovny

V popisech implementačních detailů řešení začnu knihovnou. Třídami reprezentujícími objektový model akordu a písně se zde zabývat nebudu, ty jsem vytvořil již v rámci návrhu řešení v předcházející kapitole.

Knihovnu jsem začal implementovat v repozitáři odděleném od aplikace. Do repozitáře s knihovnou jsem umístil formální (`.xsd` soubory) a textový popis formátů pro definici akordu a zápis textu písně a samotnou implementaci funkcionality navržené v sekci 2.3.1. Součástí je také soubor `composer.json`, který knihovnu označuje jako balíček nástroje pro správu závislostí Composer (vizte sekci 1.3.3.3). Repozitář má nakonfigurovanou CI pipeline, která automaticky spustí testy knihovny a sestaví balíček s kódem.

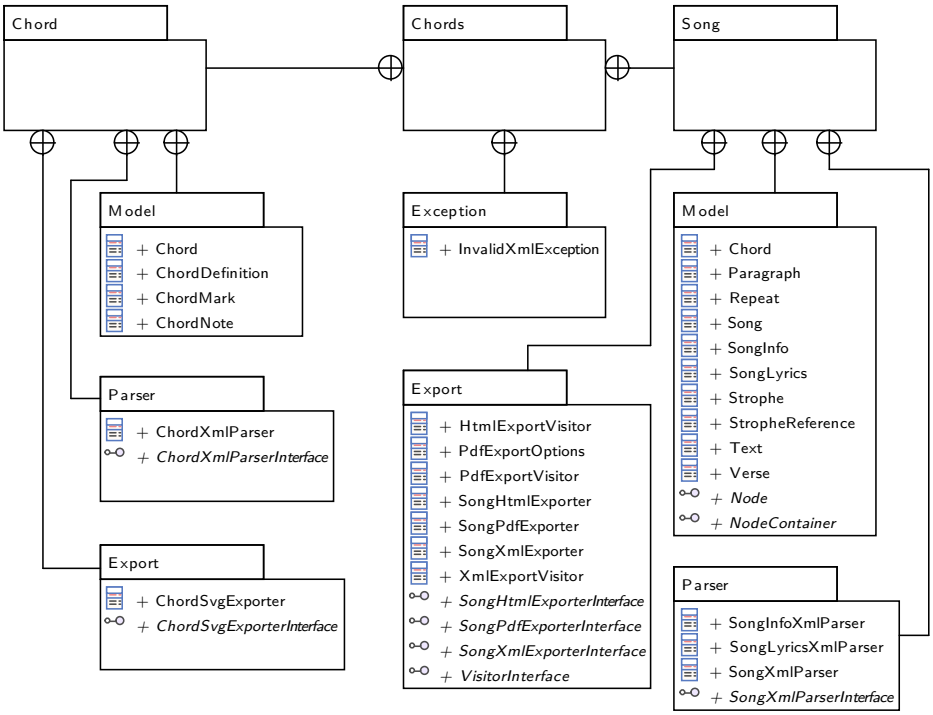
Z kapitoly 2 připomenu, že část implementace zodpovědnou za uložení definice akordu a textu písně do XML formátu, jejich objektovou reprezentaci, vykreslení schématu akordu a převodu textu písně do HTML a PDF podoby jsem se rozhodl implementovat odděleně od kódu Kytarového zpěvníku. Z uvedeného návrhu také plyne, že knihovna bude založená okolo dvou hlavních objektů: akordu a písně.

3.4.1 Rozdělení do balíčků

Balíčky (případně jmenné prostory) představují formu hierarchické organizace tříd a rozhraní. Otázku rozdělení do balíčků jsem řešil hned na začátku implementace a na základě myšlenky dvou hlavních objektů popsané v úvodu této sekce jsem se rozhodl jako hlavní rozdělení použít právě tyto dva objekty.

Na obrázku 3.1 jsem pomocí UML diagramu balíčků popsal rozdělení knihovny, včetně všech tříd a rozhraní. Balíček je reprezentován siluetou složky, jeho název je uveden na

jejím vrchu. Propojení s kolečkem a křížem na konci značí zanoření do balíčku, u kterého se symbol nachází (tzv. *nesting*).



■ **Obrázek 3.1** Diagram balíčků knihovny

Je vidět, že struktury uvnitř balíčků `Chord` a `Song` jsou shodné a nabízí se otázka, zda nebylo vhodnější provést rozdělení jiným způsobem. Ačkoliv názvy zanořených balíčků jsou stejné, implementace uvnitř se týká zcela jiných objektů a proto mi nepřijde rozumné je míchat dohromady. Stejně tak jsem přemýšlel, zda nevytvořit například jedno rozhraní pro třídu, která bude načítat XML soubory, nicméně vzhledem k tomu, že výstupem je jednou definice akordu a podruhé text písně, ani takové rozhraní nedává logicky smysl.

3.4.2 Parser XML

Třídy `ChordXmlParser` a `SongXmlParser` slouží k načtení formátů specifikovaných v kapitole 2.6 do objektové reprezentace příslušných objektů. Implementace v obou třídách k tomu samozřejmě využívá XML parser dostupný skrze rozšíření jazyka PHP, neimplementuje vlastní.

Struktura XML dokumentu pro akord je jednodušší než dokument s písní, proto se liší rozhraní, skrze které k parseru jednotlivé třídy přistupují. Třída `ChordXmlParser` pro načtení a zpracování XML dokumentu používá rozhraní poskytnuté rozšířením SimpleXML, zatímco `SongXmlParser` používá navíc rozšíření DOM.

Práce s rozhraním SimpleXML je jednodušší a v porovnání s rozšířením DOM stejný výsledek vyžaduje menší množství kódu. Během implementace jsem ale zjistil, že SimpleXML neumožňuje získat zvlášť čistý text a zvlášť elementy, což znemožnilo zpracovat verše písně.

```
<verse><chord>A7</chord>vroubená je stromama<chord>D</chord>.</verse>
```

■ **Výpis kódu 3.5** Verš písně v XML dokumentu

Ve verši se kombinují elementy značící akordy a čistý text, který reprezentuje text písně. Ty je nezbytně nutné oddělit, a proto jsem pro implementaci musel využít rozšíření DOM, které to umožňuje. Abych zachoval jednoduchost kódu, toto rozšíření se stará pouze o zpracování elementů `<verse>`, zbytek dokumentu s písní zpracovává SimpleXML.

Používání dvou různých rozšíření pro ten samý účel v tomto případě nepřináší zbytečné zdvojování vyžadovaných závislostí, jelikož SimpleXML i DOM jsou ve výchozím nastavení součástí PHP [42] [43]. Tím je také dodržen nefunkční požadavek N3.

3.4.3 Export objektové reprezentace

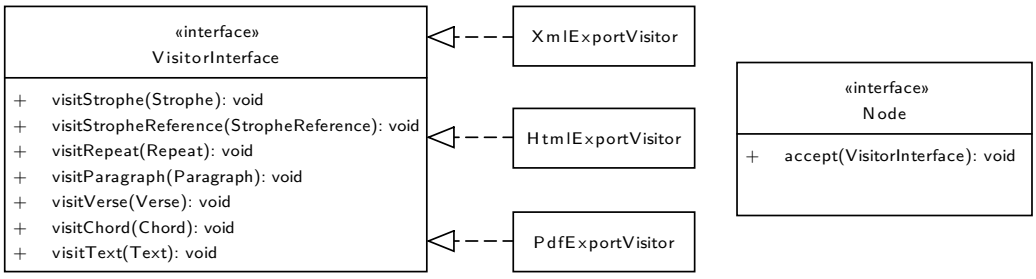
V této sekci vysvětlím, jakým způsobem jsem implementoval převod z objektové reprezentace písně do XML, HTML a PDF. Vzhledem k tomu, že již v tomto bodě implementace potřebuji vytvořit převodníky do tří různých formátů, jsem se rozhodl se správným návrhem zabývat podrobněji.

Jako první jsem se zabýval implementací exportu písně do XML. Vzhledem k tomu, že objektová reprezentace odpovídá struktuře XML dokumentu, rozhodl jsem se do třídy implementující export zařadit za každý typ objektu vlastní metodu (například `exportStrophe()`, `exportVerse()` a podobně), která z něj vytvoří XML element.

Připomenu-li, že objektovou strukturu bude potřeba exportovat i do dalších formátů (navíc nejen v rámci knihovny, i aplikace bude implementovat vlastní export do editovatelného textového formátu), nabídne se pro implementaci použít *visitor pattern*. Ten je vhodný právě pro provádění různých druhů operací (v tomto případě exportu do různých formátů) nad datovou strukturou bez nutnosti pro každou operaci přidávat nové metody do datové struktury a bez nutnosti dynamického větvení na základě toho, který druh objektu aktuálně zpracováváme (v PHP by to byla konstrukce `if` s operátorem `instanceof`).

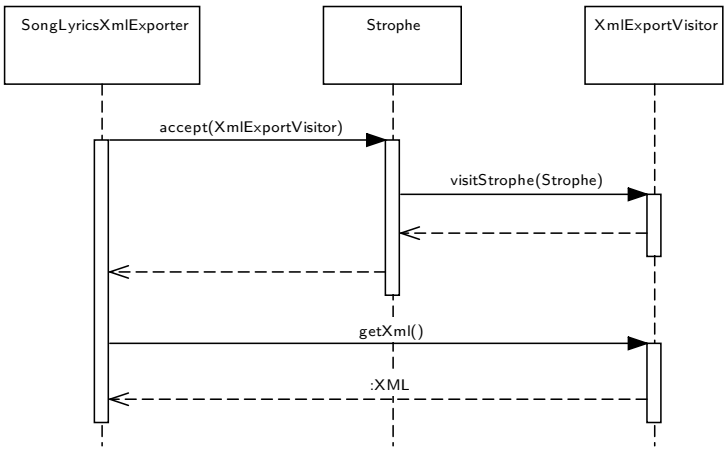
Implementace tohoto návrhového vzoru spočívá ve vytvoření rozhraní visitoru, které pro každý typ objektu v datové struktuře obsahuje metodu, která objekt zpracuje (tak, jak jsem původně navrhoval metody `export…`, ale budou pojmenované[5] ve formátu `visit…`). Každý objekt datové struktury pak implementuje metodu `accept`, která přijme ve svém argumentu objekt implementující rozhraní visitoru. Objekt datové struktury sám o sobě ví, jaký druh je, a jediným účelem jeho metody `accept` bude zavolat správnou metodu `visit…` z visitoru podle druhu objektu (tzv. *double dispatch*). Konkrétní implementace visitoru pak vykonávají příslušnou operaci (například jeden visitor pro export do XML, druhý visitor pro export do HTML). [44, s. 331–339]

Vzhledem k tomu, že každý objekt ve struktuře textu písně implementuje rozhraní `Node`, metodu `accept` jsem přidal právě do něj. Zcela konkrétně je visitor pattern znázorněný pro datovou strukturu písně pomocí UML diagramu tříd v obrázku 3.2 a posloupnost volání metod pro export sloky do XML lze vidět v sekvenčním diagramu na obrázku 3.3.



Obrázek 3.2 Diagram tříd visitor patternu

[5] Při pojmenovávání metod visitor patternu se držím názvosloví z literatury.



Obrázek 3.3 Sekvenční diagram použití visitoru

Kód mimo knihovnu nepoužívá přímo visitory, ale třídy `SongXmlExporter`, `SongHtmlExporter` a `SongPdfExporter`, které visitory používají uvnitř své implementace.

Argument nutnosti převodu do několika různých formátů neplatí pro objekt s definicí akordu (ten se pouze vykresluje jako vektorový obrázek), proto jsem pro něj visitor pattern nepoužil. Export jsem implementoval v rámci třídy `ChordSvgExporter`.

3.4.4 Export písně do PDF

Vzhledem k tomu, že export bude nově probíhat z objektové reprezentace textu písně a proto, že během analýzy byly objeveny zásadní chyby v aktuální implementaci generování PDF, rozhodl jsem se existující implementaci nepoužít a vytvořit zcela novou.

Nejprve bylo nutné zvážit, jakou knihovnu z provedené rešerše v sekci 1.7.3 pro implementaci použiji. Rozhodl jsem se přiklonit k přístupu generování PDF dokumentu z HTML kódu oproti imperativnímu vytváření PDF voláním metod knihovny. Rozhraní imperativně orientovaných knihoven poskytuje jen základní možnosti formátování, a vytvořit implementaci, která by dokázala správě sázet text a akordy včetně zalamování a dalších nutných záležitostí, by přesáhlo rámec této závěrečné práce.

K vyzkoušení mi zbyly knihovny mPDF (ta se používala v původní implementaci) a dompdf. Obě dvě podporují jen omezenou podmnožinu HTML a CSS a je nutné na jejich možnosti brát ohledy při vytváření rozložení HTML dokumentu. Během implementace jsem zjistil, že mnou zamýšlené řešení není kompatibilní s mPDF, proto jsem se rozhodl použít dompdf.

Pro vysázení textu písně a akordů nad textem jsem zvolil algoritmus, který každý verš rozdělí do menších bloků. První blok začne na začátku verše a další blok začne vždy, když se v textu objeví akord nebo mezera. V prvním řádku každého bloku je umístěný akord, ve druhém pak text písně. Tím dosáhnu toho, že akord je vždy nad znakem, který po něm následuje. Jednotlivé bloky se pak vyskládají na řádku za sebe. Pokud řádek není dostatečně dlouhý, dojde k zalomení na rozhraní dvou bloků (proto se bloky dělí i na mezerách).

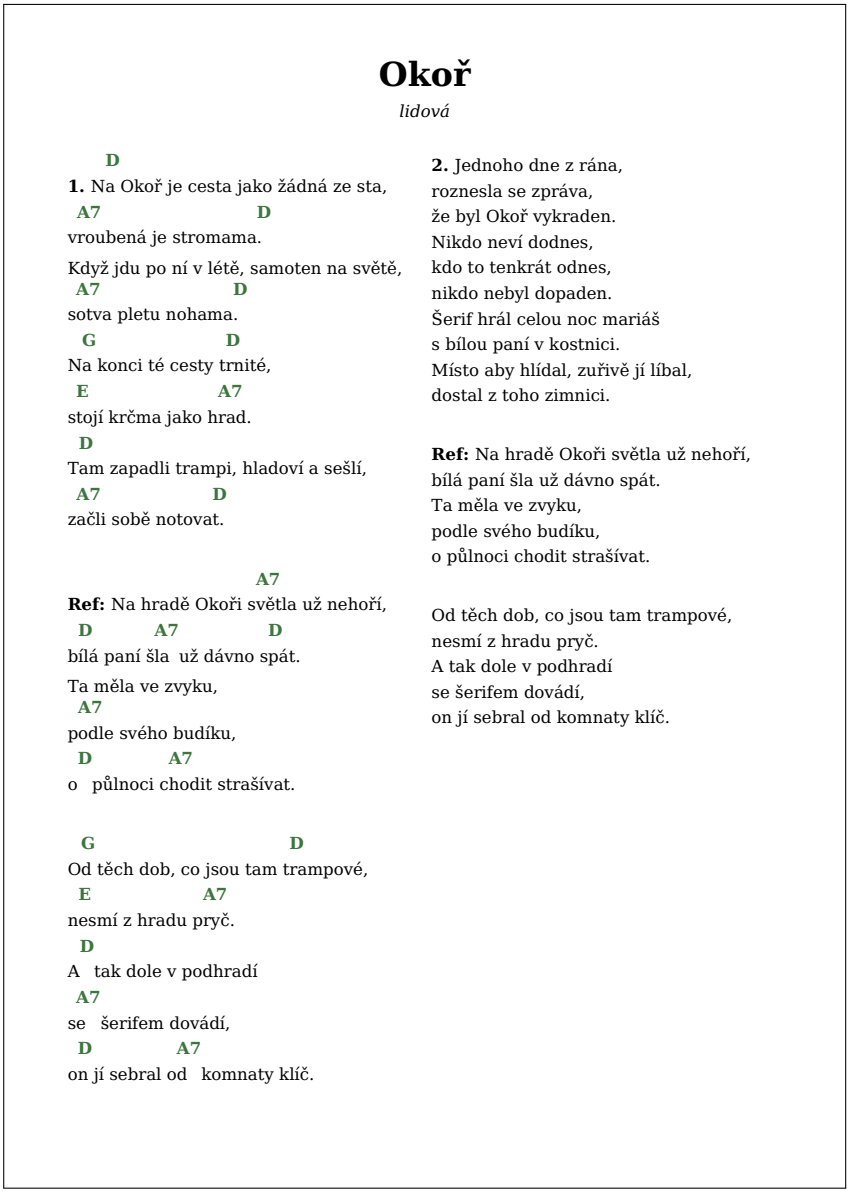
	D		**A7**				**D**
b	ílá	paní	šla	už	dávno	spát	.

Obrázek 3.4 Rozdělení verše do bloků

V případě, že verš neobsahuje žádné akordy (případně jsou všechny označené jako netisknuté), první řádek bloku se vynechá, aby nevytvářel v textu písně prázdné místo.

Text písně se pro úsporu místa sází do více sloupců (řádky bývají krátké), to ale knihovna dompdf neumí. To jsem ve své implementaci vyřešil tak, že se podle velikosti stránky a počtu sloupců vypočítá šířka jednoho sloupce a ta se knihovně dompdf předá namísto skutečné šířky papíru. Dompdf pak vygeneruje více stránek, které se vloží do nového PDF dokumentu vedle sebe. K tomu jsem musel využít knihovnu FPDF a její rozšíření FPDI [45], které umí PDF dokumenty spojovat.

Obrázek 3.5 obsahuje PDF export stejné písně jako obrázek 1.2, ale vygenerovaný novou implementací. Při porovnání lze vidět, že se podařilo odstranit problémy popsané v požadavku F4.



Okoř

lidová

D
1. Na Okoř je cesta jako žádná ze sta,
A7 D
vroubená je stromama.
Když jdu po ní v létě, samoten na světě,
A7 D
sotva pletu nohama.
G D
Na konci té cesty trnité,
E A7
stojí krčma jako hrad.
D
Tam zapadli trampi, hladoví a sešlí,
A7 D
začli sobě notovat.

A7
Ref: Na hradě Okoři světla už nehoří,
D A7 D
bílá paní šla už dávno spát.
Ta měla ve zvyku,
A7
podle svého budíku,
D A7
o půlnoci chodit strašívat.

G D
Od těch dob, co jsou tam trampové,
E A7
nesmí z hradu pryč.
D
A tak dole v podhradí
A7
se šerifem dovádí,
D A7
on jí sebral od komnaty klíč.

2. Jednoho dne z rána,
roznesla se zpráva,
že byl Okoř vykraden.
Nikdo neví dodnes,
kdo to tenkrát odnes,
nikdo nebyl dopaden.
Šerif hrál celou noc mariáš
s bílou paní v kostnici.
Místo aby hlídal, zuřivě jí líbal,
dostal z toho zimnici.

Ref: Na hradě Okoři světla už nehoří,
bílá paní šla už dávno spát.
Ta měla ve zvyku,
podle svého budíku,
o půlnoci chodit strašívat.

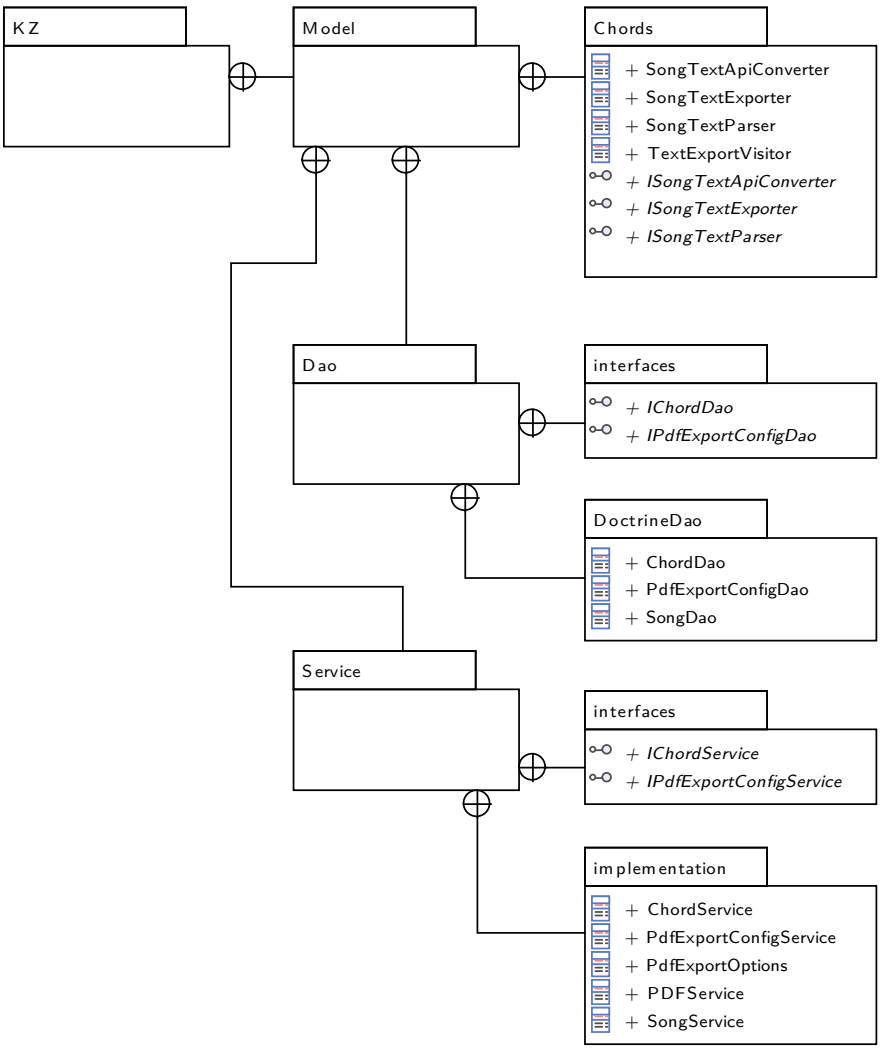
Od těch dob, co jsou tam trampové,
nesmí z hradu pryč.
A tak dole v podhradí
se šerifem dovádí,
on jí sebral od komnaty klíč.

Obrázek 3.5 Výsledek nově implementovaného exportu písně do PDF

3.5 Integrace knihovny do Kytarového zpěvníku

Další fází realizace bylo zaintegrování knihovny do kódu aplikace, tedy využití rozhraní, které knihovna poskytuje a naprogramování některých funkcí specifických pro Kytarový zpěvník. V této sekci se budu zabývat především implementací na backendu, tedy z hlediska databázových entit, servisních tříd a business logiky.

Pro implementaci nových tříd souvisejících s integrací knihovny (například převodník z, resp. do editačního formátu), které se nehodí jinam, jsem vytvořil nový balíček `KZ\Model\Chords`. Společně s ostatními nově vytvořenými a existujícími upravenými třídami je balíček vidět v diagramu na obrázku 3.6.



Obrázek 3.6 Diagram balíčků s nově přidanými a změněnými třídami business vrstvy

Dále zopakuji, že aplikace v Nette frameworku používají princip *dependency injection*, to znamená, že framework se stará o předání požadovaných závislostí do třídy [46] a třída si je nevytváří sama. Tohoto principu jsem se samozřejmě v průběhu realizace držel, a i když se v textu odkazuji na konkrétní implementace tříd, kód v aplikaci je závislý jen na rozhraních, které dané třídy implementují, nikoliv na konkrétní implementaci.

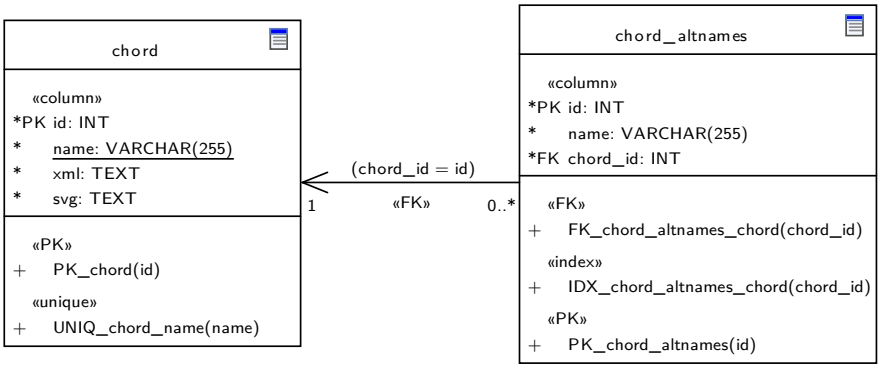
3.5.1 Vytvoření entitní třídy a service pro akordy

Funkcionalita pro vytváření schémat kytarových akordů je zcela nová, proto bylo třeba na straně aplikace vytvořit servisní třídu, která měla práci s akordy zapouzdřovat a entitní třídu, která bude definici akordu ukládat.

Přidal jsem entitu `Chord` s atributy `name`, `alternativeNames`, `xml` a `svg`. Ačkoliv jméno a alternativní názvy akordu jsou obsažené v XML definici, vyčlenil jsem je do samostatných sloupců v databázi, protože právě podle jmen bude aplikace akordy z textu písně párovat na jejich schéma. Procházet pokaždé úplně všechny XML dokumenty a jména hledat až v nich by bylo značně neefektivní.

Atribut `svg` obsahuje grafickou reprezentaci schématu akordu ve vektorovém formátu SVG. Grafika se taktéž generuje z XML, ale vzhledem k tomu, že zobrazování akordů je velmi časté, rozhodl jsem se atribut `svg` použít jako cache a negenerovat obrázek při každém požadavku znovu. Místo toho `ChordService` při vytvoření nebo editaci akordu obrázek pomocí knihovní třídy `ChordSvgExporter` vygeneruje a uloží do atributu `svg` entity `Chord`.

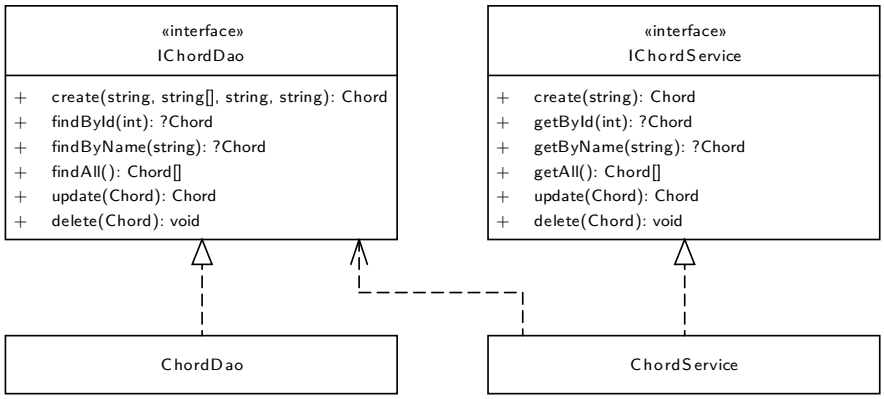
Abych dosáhl optimální struktury databáze (tzv. normální formy), rozhodl jsem se, že vytvořím další entitu `ChordAlternativeName`, která bude obsahovat vždy jedno alternativní jméno akordu a bude asociovaná právě s jedním akordem. Atribut `alternativeNames` entity `Chord` je pak kolekcí entit `ChordAlternativeName`. (Nevhodnou alternativou by bylo například umístění alternativních názvů oddělených čárkou do jednoho sloupce.)



Obrázek 3.7 Databázový model pro evidenci akordů

Ze sekce 1.2.3 připomenu, že přístup k datovému úložišti (v aktuální implementaci je to databáze) zapouzdřují DAO třídy. V souladu s tím jsem vytvořil nové rozhraní `IChordDao` a jeho implementaci `ChordDao`, které poskytuje metody pro vytvoření nového akordu, aktualizaci existujícího, odstranění akordu a vyhledání podle ID či názvu (včetně alternativních).

K DAO třídám kód aplikace nepřistupuje přímo, místo toho interaguje se servisní třídou. Proto jsem přidal rozhraní `IChordService` a jeho implementaci `ChordService`, které kopírují rozhraní `IChordDao`. Rozdílem je, že `ChordService` přijímá pouze XML definici akordu a extrakci názvu, alternativních názvů a vygenerování cachovaného SVG obrázku provede sama za pomoci knihovny. Do `ChordDao` pak předá již takto předpřipravená data.



Obrázek 3.8 Diagram tříd `ChordDao` a `ChordService`

3.5.2 Úprava entity a service pro písně

Protože se změnil formát uložení textu písně a akordů v databázi, bylo nutné upravit i entitní třídy, DAO třídy a servisní třídu. Jednalo se o odstranění atributů `lyrics` a `chords` z entity `Song` a přidání atributu `xml`. Rozhraní a implementace třídy `SongDao` jsem upravil tak, aby místo původních atributů přijímalo nový `xml`. Třída `SongService`, která zapouzdřuje přístup k `SongDao`, v původní implementaci také přijímala argumenty `chords` a `lyrics`, což vedlo k tomu, že v aplikaci byl na několika místech zduplikovaný kód, který převáděl editační formát na tyto dva atributy. Rozhodl jsem se změnit rozhraní `SongService` tak, aby přijímalo přímo editační formát a v rámci své implementace třída převedla text na XML, které předá do DAO.

Dále jsem do entit `Song` a `SongTaking` přidal atribut `pdfConfig`, který slouží k provázání s nově vytvořenou entitou `PdfExportConfig` (vizte předcházející sekci). Ve třídě `SongService` jsem vytvořil novou metodu `updatePdfConfig`, která umí entitu s konfigurací PDF exportu k písni nebo převzetí přiřadit.

Nakonec jsem v `SongService` vytvořil sadu metod `getSongModel` (získá objektovou reprezentaci textu písně z entity `Song`), `toHtml` (převede entitu na HTML), `toText` (převede entitu na editační formát) a `textToHtml` (převede editační formát na HTML), které se volají ze zbytku aplikace. Implementace těchto metod je jednoduchá, protože jen volá převodníky implementované v knihovně nebo v kódu aplikace. Dosáhl jsem tím ale odstínění ostatního kódu od přímých závislostí na převodnících a zbytek aplikace tak volá jen metody `SongService`.

3.5.3 Rozšířený editační formát textu písně

Na začátek této sekce připomenu, že uživatelé do formuláře ve webovém rozhraní nevkládají text písně v XML formátu, ale v lidsky dobře čitelném textovém zápisu (vizte příklad ve výpisu kódu 1.1). V rámci realizace jsem musel rozšířit existující editační formát o nové formátovací možnosti a naprogramovat parser, který jej převede do objektové reprezentace textu písně.

Všechny formátovací možnosti rozšířeného editačního formátu uvádím v následujícím přehledu.

► Označení akordu

Označování akordů zůstává stejné jako v dosavadním editačním formátu, název akordu se uzavře do hranatých závorek a umístí se před znak, nad kterým se má zobrazit.

```
[D]Na Okoř je cesta jako žádná ze sta
v[A7]roubená je stromama[D].
```

Výpis kódu 3.6 Ukázka textu písně s akordy

▶ Tag pro značku sloky

Sloky mohou být nově označené jejich číslem, případně libovolným jiným označením (například písmenem *R* pro refrén) – v původní implementaci se značka nijak nelišila od zbytku textu sloky.

Značku sloky je třeba umístit na začátek prvního řádku sloky a uzavřít do složených závorek, vizte příklad 3.7.

```
{1} [D]Na Okoř je cesta jako žádná ze sta
v[A7]roubená je stromama[D].
...

{R} Na hradě Okoři s[A7]větla už nehoří,
b[D]ílá paní š[A7]la už dávno s[D]pát.
```

Výpis kódu 3.7 Ukázka textu písně s označenými slokami

▶ Tagy pro repetici

Nový formát ukládání a reprezentace textu písně umožňuje explicitně označit repetici v textu. Pro editační rozhraní jsem zvolil standardní značení repetice, tedy uzavření mezi `[:` a `:]`. Za uzavírací značkou navíc může následovat indikátor počtu opakování ve formátu *n*`x`, kde *n* je počet opakování.

```
v[A7]roubená je [: stromama[D] :] 3x.
```

Výpis kódu 3.8 Ukázka textu písně s repeticí

Do repetice je možné uzavřít celý verš, ale aktuálně nelze mít repetici přes více řádek. Je ale možné do repetice uzavřít celou sloku, a to přidáním počtu opakování do závorek za její značku, vizte příklad 3.9.

```
{R} (2x) Na hradě Okoři s[A7]větla už nehoří
```

Výpis kódu 3.9 Ukázka textu písně s repeticí celé sloky

▶ Tagy pro zapnutí nebo vypnutí tisku akordů

Pro označení akordů jako netisknutých byl zaveden tag `{tisk:off}`. Ten je nutné umístit na samostatný řádek a všechny akordy, které za ním následují, se v PDF exportu budou ignorovat. Samotný řadek s tagem se ve výstupu neobjeví.

```
{tisk:off}
{2} [D]Jednoho dne z rána,
roznesla se zpráva,
```

Výpis kódu 3.10 Ukázka označení akordů jako netisknuté

Tisk akordů je možné obnovit tagem `{tisk:on}`, pro který platí stejná pravidla a po jehož použití se akordy za ním následující budou opět tisknout. Oba dva tagy jsou nepárové, to znamená, že není potřeba je nějakým způsobem ukončovat.

3.5.4 Převod objektu písně do editačního formátu

Do databáze se text písně nově ukládá pouze v XML formátu, to znamená, že při zobrazení editačního formuláře je nutné jej převést do editačního formátu popsaného v předcházející sekci.

V této fázi jsem mohl elegantně využít zavedeného visitor patternu a jednoduše naimplementovat nový druh visitoru, který objektovou reprezentaci textu písně transformuje do editačního formátu. Načtení XML formátu již mám zajištěné parserem, který jsem vytvořil v rámci programování knihovny.

Podobně jako u knihovních tříd `Song…Exporter` jsem i zde celou logiku zapouzdřil do třídy `SongTextExporter`.

3.5.5 Převod z/do API formátu

Pro splnění nefunkčního požadavku N6 je nutné zachovat existující REST API rozhraní, to v tomto případě znamená přijímat text písně v původním formátu a vracet jej v původním formátu. API používá stejný formát dat, v jakém se původně text písně a akordy ukládaly do databáze – zvlášť prostý text bez akordů a zvlášť pole s akordy a jejich umístěním, vizte sekci 1.4.1.

Pro implementaci převodníku z/do tohoto formátu se nabízí navázat na parser editačního formátu a převodník do editačního formátu, které už jsem implementoval. Vytvořil jsem tedy třídu `SongTextApiConverter`, která při převodu z API formátu vloží oddělené akordy do textu písně, ten vrátí a zbytek aplikace s ním pracuje jako s editačním formátem. Naopak při převodu do API formátu nejprve použije převodník do editačního formátu a z výsledného textu pak extrahuje akordy.

3.5.6 Generování PDF exportu

Jak jsem popsal v rámci analýzy v sekci 1.1.5, Kytarový zpěvník umožňuje kromě jednotlivých písní exportovat i celé zpěvníky nebo sadu několika vybraných písní. Vzhledem k tomu, že knihovna koncept zpěvníků ani sady písní nezná, implementaci této funkcionality je nutné provést na straně aplikace.

V kódu aplikace se o vytváření PDF výstupů starala třída `PDFService`, která se volala z příslušných presenterů. Tento přístup jsem zachoval a pouze částečně upravil její rozhraní tak, aby akceptovalo argument s parametry pro přizpůsobení exportu a místo Latte šablony s HTML zdrojem dokumentu vracelo binární text s již vygenerovaným PDF. Ostatní aplikační kód je tedy od knihovní implementace generování PDF odstíněný třídou `PDFService`.

Přizpůsobení exportu může pocházet z následujících dvou zdrojů:

1. Nově vytvořená databázová entita `PdfExportConfig`, která je asociovaná s entitami reprezentující píseň a převzetí písně, obsahuje nastavení exportu konkrétní písně, například velikost textu nebo počet sloupců.

   Tím, že konfiguraci lze připojit i k entitě označující převzetí písně, je splněná část funkčního požadavku F5, která požaduje nezávislé nastavení vlastníkem písně a uživatelem, který píseň převzal.

2. Dále jsem vytvořil třídu `PdfExportOptions` která zapouzdřuje volby exportu zpěvníku nebo sady písní, tedy zda se má tisknout titulní strana, obsah a číslovat stránky.

   Instance této třídy se vytváří jednorázově v presenteru podle toho, jaké možnosti uživatel pro konkrétní proces exportu zvolil.

Pokud uživatel exportuje samotnou píseň, presenter volá metodu třídy `PDFService` a ta dále volá `SongPdfExporter` z knihovny pro získání PDF dokumentu.

V případě, že se jedná o export zpěvníku nebo sady písní, třída `PDFService` nejprve pomocí knihovny vytvoří PDF exporty jednotlivých písní. Následně vytvoří nový prázdný dokument a jednotlivé písně do něj pomocí knihovny FPDI přidá. V případě, že objekt `PdfExportOptions` požaduje tisk titulní strany nebo obsahu, tyto stránky vytvoří a do dokumentu vloží před vkládáním písní. Knihovna FPDI při načtení PDF exportu každé písně vrátí informaci o počtu stran, která se využije pro vytvoření tabulky s obsahem. Pokud uživatel požaduje číslování stránek, do patičky každé vložené stránky s písní se přidá inkrementující se číslo stránky. Titulní stránka a obsah se do číslování stránek nepočítají (stejně jako v předchozí implementaci).

3.5.7 Přiložení kódu knihovny k aplikaci

Během vývoje jsem implementaci knihovny připravoval v repozitáři mimo Kytarový zpěvník a knihovnu jsem měl deklarovanou jako závislost prostřednictvím nástroje Composer. Na konci integračního procesu vyvstala otázka, zda toto řešení ponechat, nebo zda zdrojový kód knihovny začlenit do repozitáře s aplikací.

Standardní řešení by bylo ponechat separátní repozitář a knihovnu stáhnout skrze Composer, stejně jako u ostatních knihoven, které se v Kytarovém zpěvníku používají. Problém ale je, že kód knihovny je ve stejném režimu, jako kód aplikace – umístěný v neveřejném repozitáři hostovaném na fakultní instanci GitLabu. Vzhledem k tomu, že vývojářská příručka doporučuje provádět instalaci knihoven zevnitř Docker kontejneru s aplikací, přístupové údaje vývojářů ke GitLabu (například SSH klíč) z jejich systému by nebyly dostupné a instalace knihovny by tak byla značně komplikovaná. Další problém by nastal v CI pipeline, která by se také musela autorizovat k repozitáři s knihovnou.

Proto jsem po konzultaci s vedoucím zvolil přístup, který již přede mnou zvolil Tomáš Markacz (v rámci předmětů BI-SP1 a BI-SP2 na Kytarovém zpěvníku také pracoval). Ten vytvořil knihovnu pro implementaci REST API, a její zdrojový kód nakopíroval do repozitáře Kytarového zpěvníku. Kód své knihovny jsem tedy zkopíroval do podsložky `vendor/others` (do složky `vendor` jinak Composer umisťuje stažené závislosti), kde jej umí detekovat autoloader Composeru. Tím pádem není nutné řešit autorizaci k separátnímu repozitáři a zároveň je veškerý kód potřebný k fungování aplikace, který se vyvíjí v rámci výuky, na jednom místě.

Oddělený repozitář se samotnou knihovnou jsem ponechal, jelikož obsahuje další soubory, jako například XSD specifikaci formátů. Repozitář jsem nahrál na fakultní instanci GitLabu a předal jej vedoucímu práce, aby jej měli k dispozici budoucí studenti, kteří na aplikaci budou pracovat.

3.6 Úpravy Kytarového zpěvníku

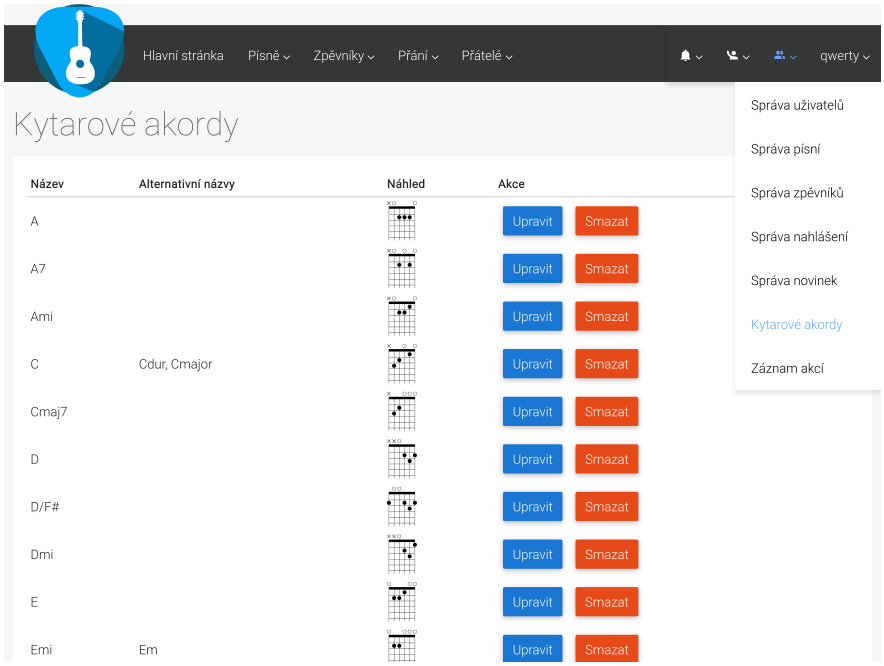
V předchozí sekci jsem se zabýval integrací knihovny do aplikace z pohledu backendu, nyní se dostávám k úpravám klientské části, tedy frontendu. Vysvětlím, jaké změny jsem musel v kódu aplikace provést, aby se nově implementovaná funkcionalita projevila v uživatelském rozhraní.

3.6.1 Administrace akordů

Administrace kytarových akordů představuje zcela novou sekci webu, proto jsem vytvořil nový presenter `ChordAdministrationPresenter` a k němu šablony pro stránku s tabulkou definovaných akordů a stránku s editačním formulářem akordu.

Tabulka definovaných akordů je řešená stejně jako zbytek podobných výpisových tabulek v aplikaci pomocí knihovny DataGrid. V presenteru se knihovně předá definice

sloupců, zdroj dat a soubor se šablonou tabulky. Prostřednictvím šablony lze přizpůsobit vykreslení dat – v tomto případě šablona převádí kolekci alternativních názvů akordů na prostý text oddělený čárkou. DataGrid se pak postará o vykreslení tabulky ve webovém rozhraní, umožňuje nastavit stránkování, řazení či filtrování dat.



■ **Obrázek 3.9** Administrace akordů

Formulář pro vytvoření nového akordu nebo editaci existujícího po domluvě s vedoucím práce obsahuje pouze textové pole pro vložení definice akordu v XML formátu. Po odeslání formuláře presenter data předá třídě `ChordService`, která založení nebo úpravu akordu zpracuje.

3.6.2 Změna generování HTML kódu s textem písně

V souvislosti se změnami popsanými v sekci 3.5 bylo po nahrazení původní implementace generování HTML kódu s textem písně novým řešením nutné změnit i implementaci presenteru, a to na stránce s detailem písně a na stránce s editačním formulářem.

Pro vygenerování HTML kódu na stránce s detailem písně jsem v presenteru použil novou metodu `toHtml` třídy `SongService`. Její výsledek pak presenter předává do Latte šablony, která HTML kód bez dalších změn vypíše.

Při zpracovávání editačního formuláře bylo nutné vyřešit nejprve jeho datový formát, jelikož v původní implementaci `SongPresenter` sám text písně konvertoval do editačního formátu a při zpracovávání odeslaného formuláře text opět parsoval. Takové řešení jednak nebylo optimální z hlediska návrhu, a v tuto chvíli už ani nebylo potřeba. Kód v presenteru jsem nahradil jednoduchým voláním metody `toText` třídy `SongService`, která převod do editačního formátu zajistí. Vzhledem k tomu, že upravená třída `SongService` nově přijímá text písně právě v editačním formátu, není další konverze v presenteru potřeba.

Vedle editačního formuláře se nachází náhled zadaného textu písně, který se v reálném čase aktualizuje. Původní implementace náhledu byla implementována v jazyce JavaScript a převod do HTML kódu se odehrával ve webovém prohlížeči uživatele. Vzhledem k tomu, že došlo k podstatnému rozšíření formátovacích možností editačního formátu

a tím i k nárůstu složitosti jeho parsování, jsem se rozhodl tuto duplicitní implementaci odstranit a převod provádět na serveru pomocí knihovny stejným způsobem, jakým se HTML kód generuje pro stránku s detailem písně.

JavaScript nyní po každé změně v textovém poli odešle požadavek se zadaným textem písně do presenteru. Ten zavolá metodu `textToHtml` třídy `SongService` a v odpovědi na požadavek vrátí vygenerovaný HTML kód. JavaScript jej pak vloží na místo náhledu.

Abych zabránil zbytečnému přetěžování serveru, použil jsem techniku tzv. *debounce*. Na základě této techniky skript zdrží odeslání požadavku na server na 1 vteřinu po tom, co v textovém poli přestaly probíhat změny. V praxi to znamená, že pokud uživatel napíše 10 znaků a neudělá mezi nimi pauzu delší než 1 vteřina, na server se odešle pouze jeden požadavek po napsání 10. znaku. Tím pádem server nemusí zpracovávat 10 požadavků po každém napsaném znaku, k čemuž by docházelo bez použití debounce.

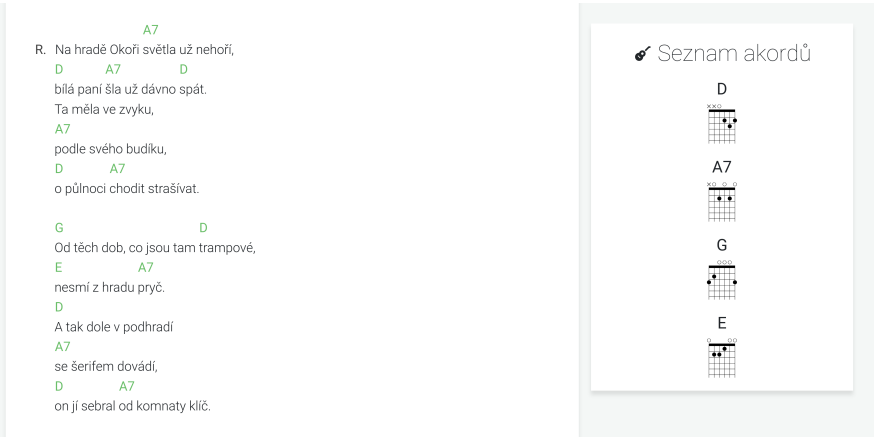
3.6.3 Zobrazení schémat akordů v detailu písně

Na stránce s detailem písně se má podle požadavku F3 zobrazovat seznam použitých akordů včetně jejich schémat. Dále mají být názvy akordů v textu písně aktivní a po kliknutí na akord se má objevit jeho schéma.

V sekci 3.5 jsem implementoval ukládání schémat akordů do databáze, pro jejich zobrazení ve webovém rozhraní jsem musel připravit nový presenter `ChordPresenter`. Ten má dvě metody:

1. Vrací SVG obrázek (cachovaný v entitě `Chord`, vizte sekci 3.5.1) se schématem akordu na základě předaného ID entity `Chord`.
2. Na základě předaného názvu akordu zjistí ID entity `Chord`, která ho má nastavený jako primární nebo alternativní název.

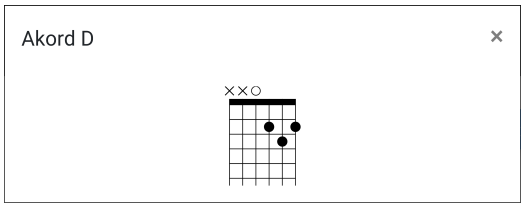
Ke stránce s detailem písně jsem pak přidal nový skript v jazyce JavaScript, který v textu písně najde použité akordy (akordy jsou v HTML kódu označené zvláštní třídou) a sestaví jejich seznam, který umístí do pravého postranního sloupce. Skript pro každý akord v seznamu vyšle požadavek na server, kterým zjistí jeho ID a na základě něj vloží do seznamu obrázek schématu akordu (respektive hlášku o tom, že akord není v evidenci).



■ **Obrázek 3.10** Detail písně se seznamem akordů v pravém postranním panelu

Zároveň skript začne u každého názvu akordu v textu písně naslouchat na událost kliknutí, po kterém otevře modální dialog se schématem akordu. Do dialogu se během jeho otevření předá reference na element, který otevření způsobil, a podle toho se rozhodne,

pro jaký akord se schéma zobrazí. Zjišťování ID entity `Chord` na stránce s detailem písně probíhá pro každý název akordu pouze jednou, skript zjištěné údaje cachuje.



Obrázek 3.11 Dialog se schématem akordu

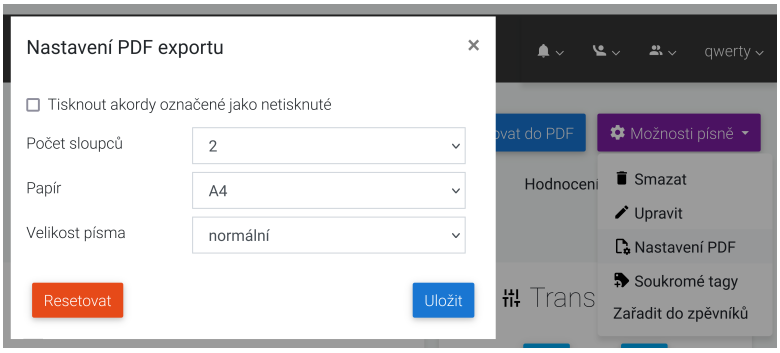
Webové rozhraní navíc umožňuje provádět transpozici akordů (vizte sekci 1.1.4), při které se akordy v textu písně změní. Skript, který transpozici provádí, jsem upravil tak, aby po provedení transpozice znovu zavolal funkci pro detekci akordů, která zajistí aktualizaci seznamu použitých akordů. Z tohoto důvodu jsem také sestavení seznamu akordů ponechal na klientský JavaScript, vzhledem k tomu, že se akordy mohou na stránce měnit bez vědomí serveru.

3.6.4 Přizpůsobení exportu písně do PDF

Pro splnění požadavku F5 bylo potřeba na stránku s detailem písně přidat nový dialog, který umožní nastavit parametry exportu dané písně do PDF formátu. Připomínám, že implementaci na backendu jsem připravil v rámci sekcí 3.4 a 3.5.

Nejprve bylo třeba ošetřit, aby k nastavení exportu měli přístup pouze oprávnění uživatelé. Toho bylo docíleno testem, zda si uživatel prohlíží jím převzatou píseň (v takovém případě se konfigurace exportu ukládá k entitě převzetí) nebo píseň, kterou vlastní (konfigurace se uloží přímo k písni).

Do Latte šablony s detailem písně jsem do menu *Možnosti písně* přidal novou položku *Nastavení PDF*, která po kliknutí otevře konfigurační dialog. V dialogu je formulář s políčky pro nastavení velikosti papíru, velikosti písma, počtu sloupců a zaškrtávátko pro tisk i akordů označených jako netisknuté. Pod formulářem se nachází tlačítko pro uložení nastavení a tlačítko reset, které konfiguraci vrátí na výchozí hodnoty.



Obrázek 3.12 Dialog s konfigurací exportu do PDF

Podpora modálních dialogů je již v aplikaci implementována, proto jsem ji využil a pouze přidal nový dialog.

Kapitola 4

Testování a nasazení

Předposlední fází mé závěrečné práce je otestování implementovaného řešení a ověření splnění specifikovaných požadavků. Funkčnost řešení jsem otestoval pomocí jednotkových testů, statické analýzy kódu a uživatelského testování. Následně jsem prošel všechny funkční a nefunkční požadavky specifikované v sekci 1.6 a zkontroloval, že jsou po realizaci úspěšně splněny.

Na závěr jsem pak sestavil balíček s aplikací připravený k nasazení na produkční prostředí a také jej nahrál do testovacího prostředí.

4.1 Jednotkové testy

První formou automatizovaného testování v Kytarovém zpěvníku jsou jednotkové (anglicky *unit*) testy. Tyto testy mají za cíl ověřit správnou funkčnost určité třídy izolovaně od zbytku aplikace. Testovaná třída místo obvyklých implementací svých závislostí (například třídy umožňující přístup k databázi) obdrží objekt, který reálné chování vhodně simuluje (v příkladu by místo načtení dat z databáze objekt vrátil sadu předdefinovaných záznamů). Programátor testů připraví vhodná vstupní data a nakonfiguruje simulované závislosti, které se pak předají do testované třídy. Správnost chování určuje definováním očekávaného výstupu, případně deklarováním očekávané interakce se závislostmi (například testovaná třída má zavolat metodu závislosti pro uložení záznamu). [47]

Ze sekce 1.3.3.2 zopakuji, že Kytarový zpěvník z historických důvodů používá dva testovací frameworky: Nette Tester a PHPUnit. Nette Tester zůstává pro starší testy, nové jsou napsané pro PHPUnit. Během testování své implementace jsem pracoval s testy v obou frameworcích, ale nové testy jsem vytvářel jen pro PHPUnit.

4.1.1 Oprava existujících testů

Změny provedené během realizace měly pochopitelně za následek, že některé dosud funkční (před zahájením implementace byly funkční všechny) testy přestaly procházet. Po dokončení implementace tedy prvním krokem bylo takto rozbité testy opravit. Rozbité testy se týkaly pouze tříd pracujících s písněmi a zpěvníky, zbytek dle očekávání zůstal funkční.

Zprovoznění existujících testů spočívalo nejprve v opravě vytváření testovaných tříd či jejich závislostí, protože u některých z nich další závislosti (například na převodnících mezi různými formáty pro zápis písně) přibyly.

Další změnou, která způsobila nekompatibilitu, bylo úplné nahrazení existujícího způsobu ukládání textu písně a akordů. Tam, kde testy u entity písně původně počítaly

s atributem pro text a atributem pro akordy, jsem je upravil tak, aby místo nich pracovaly s atributem pro XML. Testy zaniklých metod jsem zrušil úplně.

4.1.2 Přidané jednotkové testy

Pokud se podíváme na vytvořené a upravené třídy v aplikaci během realizace, které jsou vhodné k testování, můžeme je rozdělit na servisní třídy a převodníky formátů.

4.1.2.1 Servisní třídy

Nově implementovaná servisní třída `ChordService` obsahuje CRUD operace pro práci s entitami akordů. Zapouzdřuje přístup k DAO třídě a provádí extrakci a aktualizaci metadat o akordu z XML formátu. Testování metod pro čtení a editaci akordu jsem implementoval simulací DAO třídy a kontrolou, zda `ChordService` vrací očekávané entity. Metody pro vytvoření a smazání akordu jsou otestovány tak, že se kontroluje, zda byly zavolány příslušné metody na DAO třídě.

Další nově vytvořená třída `PdfExportConfigService` jen zapouzdřuje přístup k DAO třídě, proto se v jejích testech pouze kontroluje volání správných metod na DAO.

Třída `SongService` již má rozsáhlou sadu existujících testů. Během realizace jsem nově implementoval metody zapouzdřující přístup k převodníkům mezi různými formáty textu písně. Převodníky budu testovat zvlášť, zmíněné metody žádnou logiku navíc neobsahují, proto jsem se rozhodl je netestovat. Vnitřní implementace se zásadně změnila u metody pro nahrazení akordů v písni jinými, která se používá pro uložení provedené transpozice. Metoda původně nebyla vůbec testována, vzhledem k tomu, že nyní pro nahrazení akordů musí interagovat s knihovnou, jsem se rozhodl test přidat.

4.1.2.2 Převodníky formátů

Za převodníky formátů pro účely této sekce považuji třídy, které se starají o převod XML formátu pro definici akordu nebo textu písně do objektové reprezentace (u textu písně jde i o parsování editačního a API formátu) a všechny převodníky z objektové reprezentace do jiného formátu.

Testování těchto tříd jsem pojal jako *black box* testování, to znamená, že netestuji jejich vnitřní chování (například volání správných metod visitorů), ale pouze očekávaný výstup. Pro každý druh převodníku jsem připravil sadu testovacích dat, samotný test probíhá předáním testovacího vstupu a porovnáním výstupu testované třídy s očekávaným výstupem.

Protože testuji pouze správnost výstupu, testovací data jsem sestavil tak, aby používala všechny podporované konstrukce a struktury testovaných formátů. Tím jsem dosáhl vysoké jistoty v tom, že jednotkové testy odhalí i případné drobné regrese.

4.1.3 Testování knihovny

Implementace knihovny obsahuje třídy pro objektovou reprezentaci, parser XML a převodníky do dalších formátů (vizte sekci 3.4.1). Třídy s objektovou reprezentací samy žádnou logiku neobsahují, proto jsem je z testování vynechal. Testování parseru a převodníků jsem implementoval na stejném principu jako u kódu Kytarového zpěvníku v sekci 4.1.2.2.

4.2 Statická analýza

Statická analýza kódu je další forma automatického testování, která se v projektu používá. Jedná se o pasivní způsob testování, kdy aplikace ani její části nejsou žádným způsobem

aktivní (aplikace není spuštěná, nevolají se funkce ani metody tříd). Místo toho externí program skenuje zdrojový kód a hledá v něm problémy. Skener například umí zkontrolovat, zda kód nevolá neexistující metody, nepoužívá nedefinované proměnné, nepředává do metod argumenty se špatným datovým typem a podobně. Kromě těchto zjevných chyb detekuje i tzv. *code smell*, což jsou nežádoucí konstrukce, které ale nemají vliv na funkčnost programu, například nedodržování programovacích konvencí (odsazení, pojmenování proměnných a podobně), duplicitní kód, zbytečný nebo nedosažitelný kód a další. [48]

Kytarový zpěvník ke statické analýze kódu používá nástroje PHPStan a SonarQube. PHPStan umí detekovat podmnožinu problémů co SonarQube (vizte sekci 1.3.3.2), proto zde popíši výstupy ze SonarQube. Zároveň se omezím jen na mnou upravený nebo přidaný kód (do analýzy jsem zahrnul jak kód aplikace, tak vytvořené knihovny).

4.2.1 Výsledky analýzy

Během vývoje mi statická analýza pomohla odhalit několik chyb v kódu, kdy jsem používal nedefinované proměnné, protože jsem v rámci průběžného refaktorování zapomněl některé výskyty přejmenovat. Dále mi pomohla odstranit několik druhů code smell:

- matoucí názvy proměnných (např. kolize lokální proměnné s atributem třídy),
- zbytečně složité konstrukce (např. podmínky, které šly zjednodušit),
- nepotřebné přiřazování do proměnných (např. přiřazení a okamžité vrácení proměnné),
- napevno vložené řetězce bez použití konstant.

Při interpretování výsledků statické analýzy je důležité přemýšlet nad každým varováním zvlášť a řešit jen ty z našeho pohledu podstatná. Proto jsem se rozhodl neřešit některé instance upozornění na nepoužití konstant, protože šlo o generování HTML kódu a varování doporučovalo extrahovat uzavírací značky elementů (např. `</div>`) do konstanty, což mi přišlo zbytečné. Například názvy HTML tříd jednotlivých atributů ale v konstantách jsou.

Dále jsem ignoroval upozornění na nepoužité argumenty funkcí, které ale nelze odstranit, protože se jedná o *callbacky* předávané do vestavěných funkcí PHP. U argumentů samozřejmě záleží na pořadí, tzn. i když můj kód například první argument ignoruje, nemohu jej ze signatury callbacku odstranit.

Zůstávají také některá varování o příliš velké komplexnosti metod, zejména se to týká převodníku z editačního formátu na objektovou reprezentaci. Bohužel zpracování tohoto formátu je poměrně složité, a komplexnost kódu nelze uspokojivě snížit. V tomto případě jsem alespoň dal zvýšený důraz na přehlednost a srozumitelnost zdrojového kódu, abych tím jeho komplexitu do určité míry kompenzoval.

V přehledu varování také zůstala dvě upozornění na duplicitní kód ve třídách `HtmlExportVisitor` a `PdfExportVisitor`, která jsem po důkladném zkoumání vyhodnotil jako falešná.

4.3 Uživatelské testování

Uživatelské testování slouží k vyzkoušení celé aplikace, provádí se ručně nad kompletně spuštěnou aplikací naplněnou testovacími daty. Uživatel v rozhraní aplikace provádí specifikovanou sadu úkonů (tzv. scénáře) a ověřuje, že aplikace funguje a dává požadované výsledky. Scénáře rozdělím na dvě skupiny: uživatelské a administrátorské.

Na začátku realizace byla aplikace plně funkční, z popisu provedených změn v kapitole 3 vyplývá, že změny byly omezené na práci s písní, zpěvníkem a kytarovými akordy. Ostatní funkčnost zůstala nedotčená, všechny jednotkové testy procházejí a proto se pro účely uživatelského testování omezím jen na změněné části aplikace.

4.3.1 Uživatelské scénáře

Uživatelské scénáře testují používání aplikace běžnými uživateli, pro přístup k testovaným funkcím není třeba mít žádná speciální oprávnění. Pokud scénář vyžaduje přihlášení, je to v něm uvedeno.

► Uživatelský scénář 1. Zobrazení detailu písně

Projít stránky s detailem písní. Všechny se musí bezchybně načíst a text písně musí být správně zformátovaný a vykreslený.

► Uživatelský scénář 2. Režim přehrávání písně

Na stránce s detailem písně zapnout tlačítkem *Zahrát píseň* upravené zobrazení textu písně přes celou stránku s velkým fontem. Text písně musí být zformátovaný stejně jako v běžném webovém zobrazení.

► Uživatelský scénář 3. Seznam použitých akordů

Otevřít stránku s detailem písně. V pravém postranním panelu se musí zobrazit seznam všech akordů, které se v textu písně vyskytují. V seznamu nesmí být duplicitní akordy. Pokud píseň neobsahuje žádné akordy, blok se seznamem musí být skrytý.

V seznamu akordů musí být zobrazená i schémata akordů, případně hláška o chybějícím schématu, pokud akord není v evidenci. Schéma se musí objevit i v případě, že název akordu v písni je uvedený jako jeden z alternativních názvů akordu v evidenci.

► Uživatelský scénář 4. Kliknutí na název akordu

Otevřít stránku s detailem písně a kliknout na název akordu v textu písně. Po kliknutí se musí otevřít modální dialog, v jehož záhlaví je uveden název akordu a obsahuje graficky znázorněné schéma akordu.

V případě, že akord není v evidenci, musí se v dialogu zobrazit místo schématu příslušná informační hláška. Schéma se musí objevit i v případě, že rozkliknutý akord je alternativním názvem v evidenci.

► Uživatelský scénář 5. Transpozice akordů

Na stránce s detailem písně vyzkoušet všechny úrovně transpozice akordů. Akordy v textu písně se musí ihned transponovat a seznam použitých akordů se musí aktualizovat, aby obsahoval právě jen novou sadu akordů. Kliknutí na název akordu v textu písně musí otevřít dialog s nově transponovanou podobou akordu.

► Uživatelský scénář 6. Uložení transpozice

Přihlásit se, otevřít vlastní píseň, provést transpozici akordů a kliknout na tlačítko *Uložit transpozici*. Musí se zobrazit informační hláška o uložení transpozice a po novém načtení stránky akordy musí zůstat transponované.

► Uživatelský scénář 7. Editace textu písně

Přihlásit se a otevřít stránku pro editaci písně. V textovém poli pro text písně musí být text písně vypsaný v editačním formátu a vedle textového pole se musí zobrazit zformátovaný náhled.

Při psaní textu písně musí náhled změnou průhlednosti signalizovat, že je zastaralý. Několik vteřin po dopsání textu se musí náhled aktualizovat, vykreslit nově zadaný text a zrušit svou průhlednost. Náhled se musí aktualizovat i při použití operací vyjmutí nebo vložení textu.

Vyzkoušet všechny formátovací možnosti: zápis akordů, označení sloky, repetici, repetici sloky a označení akordů jako netisknuté a tisknuté.

► **Uživatelský scénář 8. Uložení textu písně**

Přihlásit se, otevřít stránku pro editaci písně a text písně upravit. Uložení musí proběhnout úspěšně a na stránce s detailem písně musí být vidět upravený text tak, jak byl během editace prezentován v náhledu.

► **Uživatelský scénář 9. Export písně do PDF**

Na stránce s detailem písně kliknout na tlačítko *Exportovat do PDF*. Po kliknutí se musí stáhnout PDF soubor s exportem písně. Na první stránce musí být uvedený název a autor písně, případně i název alba, rok a poznámka, pokud jsou vyplněné. Na dalších stránkách musí být v hlavičce uvedený název a autor písně.

Text písně a akordy musí být správně zformátované. Akordy označené jako netisknuté se nesmí zobrazit. Export musí respektovat nastavení, to znamená mít správnou velikost stránky, písma a počet sloupců. Pokud je tisk skrytých akordů povolen, musí se zobrazit i akordy označené jako netisknuté.

► **Uživatelský scénář 10. Změna nastavení exportu do PDF**

Přihlásit se, otevřít vlastní píseň, kliknout na tlačítko *Možnosti písně* a vybrat položku *Nastavení PDF*. Musí se otevřít dialogové okno s formulářem pro nastavení velikosti stránky, písma, počtu sloupců a s možností aktivovat tisk skrytých akordů. Po stisknutí tlačítka pro uložení změn se musí zobrazit hláška o úspěšném uložení a po opětovném načtení stránky a otevření dialogu musí být ve formuláři vidět nově zadané hodnoty.

► **Uživatelský scénář 11. Reset nastavení exportu do PDF**

Přihlásit se, otevřít detail vlastní písně, která má přizpůsobené nastavení exportu do PDF, kliknout na tlačítko *Možnosti písně*, vybrat položku *Nastavení PDF* a kliknout na tlačítko *Resetovat*. Musí se objevit hláška o resetování nastavení exportu a po opětovném otevření dialogu musí být hodnoty ve formuláři na svých výchozích hodnotách.

► **Uživatelský scénář 12. Změna nastavení exportu do PDF u převzaté písně**

Jako uživatel A otevřít svou vlastní píseň a upravit nastavení exportu. Jako uživatel B otevřít stejnou píseň a převzít ji. Nastavení exportu u převzaté písně musí být stejné, jaké nastavil uživatel A.

Jako uživatel A změnit nastavení exportu. Nastavení exportu u převzaté písně musí zůstat beze změny. Jako uživatel B změnit nastavení exportu převzaté písně. Nastavení exportu písně u originálu musí zůstat beze změny. Za změnu nastavení se počítá i jeho resetování.

► **Uživatelský scénář 13. Export zpěvníku do PDF**

Otevřít detail zpěvníku, kliknout na tlačítko *Exportovat do PDF* a vyzkoušet všechny kombinace možností tisku titulní strany, obsahu a čísel stránek. Po výběru písní a kliknutí na tlačítko *Exportovat* musí být nabídnuto ke stažení PDF, kde musí ve správném pořadí všechny vybrané písně a musí odpovídat aktivovaným možnostem tisku.

Rozložení každé písně musí odpovídat nastavení exportu dané písně, kromě velikosti papíru, ta musí být stejná pro celý zpěvník. Titulní strana (pokud se tiskne) musí obsahovat název a autora zpěvníku.

► **Uživatelský scénář 14. Export sady písní do PDF**

Na stránce *Veřejné a nasdílené písně* kliknout na tlačítko *Exportovat do PDF* a vyzkoušet všechny kombinace možností tisku obsahu a čísel stránek. Po vybrání písní a kliknutí na tlačítko *Exportovat* musí být nabídnuto ke stažení PDF, kde musí být ve správném pořadí všechny vybrané písně a musí odpovídat aktivovaným možnostem tisku.

Rozložení každé písně musí odpovídat nastavení exportu dané písně, kromě velikosti papíru, ta musí být stejná pro celý dokument.

4.3.2 Administrátorské scénáře

Pro provedení administrátorských scénářů se tester musí do aplikace přihlásit pod uživatelem, který má přiřazenou roli administrátora, jinak nebude mít přístup k testovaným sekcím.

► Administrátorský scénář 1. Výpis kytarových akordů

V hlavičce z administrátorského menu vybrat položku *Kytarové akordy*. Musí se zobrazit stránka s tabulkou akordů, které jsou v evidenci. U každého akordu se musí zobrazit jeho grafické schéma.

► Administrátorský scénář 2. Vytvoření akordu

V administraci akordů kliknout na tlačítko *Přidat akord*. Musí se objevit stránka s formulářem pro zadání XML definice akordu. V textovém poli musí být předvyplněno ukázkové XML.

Po kliknutí na tlačítko *Vytvořit* se musí akord přesně podle zadané definice uložit do evidence.

► Administrátorský scénář 3. Editace existujícího akordu

Vybrat existující akord z evidence a kliknout na tlačítko *Upravit*. Musí se objevit stránka s formulářem pro zadání XML definice akordu, na které bude vyplněna aktuální definice editovaného akordu.

Po kliknutí na tlačítko *Uložit* se musí akord upravit přesně podle zadané definice. Původní grafické schéma akordu může být uložené v cachi prohlížeče několik hodin a může být nutné pro jeho obnovení cache vyprázdnit.

► Administrátorský scénář 4. Vložení nevalidního XML

Do formuláře pro vytvoření nebo editaci akordu zadat špatně sestavené XML s definicí akordu. Po odeslání formuláře aplikace musí zobrazit chybovou hlášku a nesmí nevalidní data uložit.

► Administrátorský scénář 5. Kontrola duplicitních názvů akordů

Do formuláře pro vytvoření nebo editaci zadat jako primární nebo alternativní název stejný řetězec, jako je primární nebo alternativní název jiného existujícího akordu. Po odeslání formuláře aplikace nesmí data uložit a musí zobrazit chybové hlášení o duplicitním názvu.

► Administrátorský scénář 6. Odstranění akordu

Vybrat existující akord z evidence a kliknout na tlačítko *Smazat*. Aplikace musí zobrazit hlášku potvrzující smazání a akord musí zmizet z evidence.

4.4 Ověření splnění požadavků

Primárním cílem mé závěrečné práce bylo vytvoření nové verze Kytarového zpěvníku, která bude splňovat požadavky specifikované v sekci 1.6. Po provedení analýzy, návrhu a samotné realizace proto ověřím, že jsem všechny požadavky splnil.

► F1. Netisknuté akordy

Požadavek byl splněn vytvořením nového formátu pro uložení textu písně (sekce 2.5.2), doplněním příslušných tagů do editačního formátu (sekce 3.5.3) a novou implementací PDF exportu (sekce 3.4.4).

Požadavek je testován uživatelským scénářem 7, 9 a jednotkovými testy.

► F2. Schémata akordů

Požadavek byl splněn vytvořením formátu pro definici schématu akordů (sekce 2.5.1), implementací vykreslování obrázku se schématem (sekce 3.4.3) a vytvořením administrace akordů (sekce 3.6.1). Sada základních akordů je součástí skriptu pro aktualizaci databáze aplikace na novou verzi.

Požadavek je testován administrátorskými scénáři 1, 2, 3, 4, 5, 6 a jednotkovými testy.

► F3. Zobrazení schémat akordů

Požadavek byl splněn implementací vykreslování obrázku se schématem akordu (sekce 3.4.3), doplněním seznamu akordů na stránku s detailem písně a zaktivněním názvů akordů v textu písně (sekce 3.6.3).

Požadavek je testován uživatelskými scénáři 3, 4 a jednotkovými testy.

► F4. PDF export písně

Požadavek byl splněn vytvořením zcela nové implementace PDF exportu (sekce 3.4.4). Požadavek je testován uživatelskými scénáři 9, 13, 14 a jednotkovými testy.

► F5. Přizpůsobení exportu do PDF

Požadavek byl splněn přidáním podpory pro ukládání nastavení exportu (sekce 3.5.2), vytvořením konfiguračního dialogu (sekce 3.6.4) a novou implementací PDF exportu (sekce 3.4.4).

Požadavek je testován uživatelskými scénáři 9, 10, 11, 12, 13, 14 a jednotkovými testy.

► N1. Programovací jazyk

Celá realizace a testování probíhaly s nainstalovaným PHP 7.1, nebyly použity žádné jazykové konstrukce nebo knihovny, které by vyžadovaly novější verzi. Nové JavaScript skripty taktéž nepoužívají žádné funkce z novějších verzí jazyka, které by současné prohlížeče nepodporovaly. Požadavek byl splněn.

► N2. Architektura implementace

Během realizace jsem respektoval zavedené programovací konvence a existující architekturu aplikace, proto požadavek považuji za splněný. Navíc je zčásti testován statickou analýzou kódu (sekce 4.2).

► N3. Kompatibilita se sdíleným webhostingem

Nově přidané knihovny a závislosti na PHP rozšířeních byly vybírány tak, aby měly co nejmenší požadavky na hosting, kde se bude aplikace provozovat. Splnění požadavku jsem empiricky ověřil v rámci nasazení do testovacího provozu na stejném webhostingu, kde běží produkční verze.

► N4. Automatické testy

Automatickým testům jsem se podrobně věnoval v sekcích 4.1 a 4.2, požadavek byl splněn.

▶ **N5. Dokumentace**

Během realizace jsem dodržoval standardní dokumentační postupy (vizte sekci 3.1, dokumentaci doplnil a zanesl do ní provedené změny. Požadavek byl splněn.

▶ **N6. Kompatibilita s Android aplikací**

Rozhraní REST API jsem žádným způsobem neměnil, ale proběhla změna ve formátu textu písně. Pro zachování původního rozhraní jsem proto implementoval převodník mezi novým formátem a datovou strukturou používanou v API (sekce 3.5.5). Požadavek byl splněn.

Požadavek je testován jednotkovými testy. Navíc jsem funkčnost ověřil manuálním sestavením HTTP požadavků na API a kontrolou provedených akcí a vrácených dat.

4.5 Nasazení

Poslední fází mé závěrečné práce bylo připravení balíčku s aplikací pro nasazení do testovacího, později produkčního prostředí.

Protože během realizace došlo k úpravě databázového schématu, bylo nutné připravit SQL skript, který upraví databázi starší verze aplikace. Kromě vytvoření nových tabulek a úpravy sloupců ale také došlo ke změně formátu, ve kterém jsou uložené texty písní. Existující texty proto bylo nutné převést do nového formátu. K tomu jsem vytvořil PHP skript, který pomocí převodníků implementovaných během realizace zkonvertuje text písně do XML formátu. Oba skripty jsou součástí výsledného balíčku a v instalační příručce je popsané, jak je spustit.

Samotné sestavení balíčku s aplikací probíhá automaticky v CI pipeline (vizte sekci 3.2.3). Aktualizační skripty, instalační příručku a zdrojový kód aplikace tedy stačí nahrát do repozitáře na GitLab a během několika minut se aplikace zkompiluje a zabalí do archivu.

```
app.zip ..... archiv se zdrojovým kódem aplikace
extract.php ..... skript pro rozbalení archivu s aplikací
INSTALL.pdf ..... instalační příručka
sql ..... složka s aktualizačními skripty
```

Výpis kódu 4.1 Obsah balíčku s aplikací

Administrátor aplikace si pak balíček nahraje na svůj webový server a postupuje podle instalační příručky.

Pro účely testování aplikace během realizace mi vedoucí práce poskytl subdoménu na svém webhostingu (kde běží i produkční verze Kytarového zpěvníku), kam jsem novou verzi aplikace nahrál.

Závěr

Cílem práce bylo rozšířit funkcionalitu webové aplikace Kytarový zpěvník v ohledu zobrazování, editace a tisku písní. Během práce byla provedena analýza současného stavu s důrazem na zmíněnou funkcionalitu a následně ve spolupráci s vedoucím stanoveny konkrétní požadavky na vylepšení. Požadavky se týkaly vytvoření evidence kytarových akordů a zaznamenání jejich schémat v grafické podobě, rozšíření editačních možností textů písní a vylepšení a oprava chyb tiskového exportu písní.

Na analýzu navázal návrh řešení, kde jsem dospěl k rozhodnutí vytvořit nové XML formáty pro definici schématu akordu a uložení textu písně a kód pro práci s těmito formáty implementovat v rámci samostatné knihovny. V realizační části jsem provedl implementaci vytvořeného návrhu, následovalo testování a příprava na nasazení nové verze aplikace do produkčního prostředí.

Výstupem mé práce je nová verze aplikace, včetně vytvoření a úprav existující dokumentace a využívání nástrojů pro řízení vývoje a dokumentaci, které používali vývojáři aplikace přede mnou a budou používat budoucí studenti při práci na aplikaci.

Implementace požadavků byla ověřená vedoucím práce a schválená, nová verze aplikace je připravená k nasazení. Cíle práce jsem tedy splnil.

Bibliografie

1. EUROPEAN PARLIAMENT, DIRECTORATE-GENERAL FOR PARLIAMENTARY RESEARCH SERVICES; MILDEBRATH, Hendrik. *Internet access as a fundamental right: exploring aspects of connectivity*. European Parliament, 2021. Dostupné z DOI: 10.2861/261938.
2. ČESKÝ STATISTICKÝ ÚŘAD. *Internet používá přes 80 % obyvatel Česka* [online]. 2020. [cit. 2022-03-23]. Dostupné z: https://www.czso.cz/csu/czso/internet-pouziva-pres-80-obyvatel-ceska.
3. JÁNSKÝ, Petr. *Já, písnička 1*. Music distribuce, 2008. ISBN 80-706517-7-0.
4. RICHARDS, Mark. *Software Architecture Patterns: Understanding Common Architecture Patterns And When To Use Them*. Sebastopol (California, USA): O'Reilly Media, 2015. ISBN 978-1-491-92424-2.
5. MICROSOFT PATTERNS AND PRACTICES TEAM. *Microsoft Application Architecture Guide*. 2. vyd. Microsoft Press, 2009. ISBN 978-0735627109.
6. INGENO, Joseph. *Software Architect's Handbook*. Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-406-0.
7. HOLAN, Jan. *Přepracování aplikace Kytarový zpěvník do MVP architektury a návrh gamifikace aplikace*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Bakalářská práce.
8. NETTE FOUNDATION. *Nette* [online]. 2022. [cit. 2022-03-29]. Dostupné z: https://nette.org/.
9. FOWLER, Martin. *Patterns of Enterprise Application Architecture*. Crawfordsville (Indiana, USA): Addison-Wesley Professional, 2002. ISBN 0-321-12742-0.
10. BACK, Ian. *mPDF* [online]. 2015. [cit. 2022-03-28]. Dostupné z: https://mpdf.github.io/.
11. LEDERBUCHOVÁ, Ladislava. *Slovník literárních pojmů: Co se skrývá za slovy*. Plzeň: Fraus, 2006. ISBN 80-7238-620-4.
12. FULLER, Ted; HAYMAN, Julian. *Kytarové akordy*. Svojtka & Co, 2008. ISBN 978-80-256-0171-6.
13. WIEGERS, Karl; BEATTY, Joy. *Software Requirements (Developer Best Practices)*. 3. vyd. Redmond: Microsoft Press, 2013. ISBN 978-0-7356-7966-5.
14. AURUM, Aybüke; WOHLIN, Claes. *Engineering and Managing Software Requirements*. Berlin: Springer, 2005. ISBN 978-3-540-28244-0.

15. SUPRSHOP S.R.O. *Supermusic* [online]. 2018. [cit. 2022-03-24]. Dostupné z: https://supermusic.cz/.

16. VAŠEK, Vlastimil. *Písničky s akordy* [online]. 2022. [cit. 2022-03-24]. Dostupné z: https://pisnicky-akordy.cz/.

17. KYNČL, Jiří. *spísničkou.cz* [online]. [cit. 2022-03-24]. Dostupné z: https://spisnickou.cz/.

18. BRNKNI.CZ. *Brnkni.cz* [online]. 2022. [cit. 2022-03-24]. Dostupné z: https://www.brnkni.cz/.

19. THE CONTRIBUTORS TO THE MUSICXML SPECIFICATION. *MusicXML: Version 4.0* [online]. 2021. [cit. 2022-03-25]. Dostupné z: https://www.w3.org/2021/06/musicxml40/.

20. THE AUTHORS OF LILYPOND. *LilyPond: Music notation for everyone* [online]. 2022. [cit. 2022-03-25]. Dostupné z: http://lilypond.org/.

21. TEX USERS GROUP. *TeX Users Group* [online]. 2022. [cit. 2022-03-26]. Dostupné z: https://tug.org/.

22. THE CHORDPRO TEAM. *ChordPro* [online]. 2022. [cit. 2022-03-26]. Dostupné z: https://www.chordpro.org/.

23. WURTZ, Nicolas. *chordpro-php* [online]. 2019. [cit. 2022-03-26]. Dostupné z: https://github.com/nicolaswurtz/chordpro-php.

24. PACKAGIST.IO. *Packagist* [online]. [cit. 2022-03-28]. Dostupné z: https://packagist.org/?tags=pdf.

25. FPDF LIBRARY. *FPDF* [online]. 2021. [cit. 2022-03-28]. Dostupné z: http://www.fpdf.org/.

26. ANDROUTSELLIS-THEOTOKIS, Stephanos; SPINELLIS, Diomides; KECHAGIA, Maria; GOUSIOS, Georgios. Open source software: A survey from 10,000 feet. *Foundations and Trends in Technology, Information and Operations Management*. 2011, roč. 4, č. 3–4. Dostupné z DOI: 10.1561/0200000026.

27. ASUNI, Nicola; TECNICK.COM. *TCPDF* [online]. 2022. [cit. 2022-03-28]. Dostupné z: https://tcpdf.org/.

28. ASUNI, Nicola. *tc-lib-pdf* [online]. 2021. [cit. 2022-03-28]. Dostupné z: https://github.com/tecnickcom/tc-lib-pdf.

29. FREE SOFTWARE FOUNDATION, INC. *GNU Lesser General Public License* [online]. 2007. [cit. 2022-03-28]. Dostupné z: https://www.gnu.org/licenses/lgpl-3.0.html.

30. FREE SOFTWARE FOUNDATION, INC. *GNU General Public License, version 2.0* [online]. 1991. [cit. 2022-03-28]. Dostupné z: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

31. DOMPDF. *dompdf* [online]. [cit. 2022-03-28]. Dostupné z: https://dompdf.github.io/.

32. FREE SOFTWARE FOUNDATION, INC. *GNU Lesser General Public License, version 2.1* [online]. 1999. [cit. 2022-03-28]. Dostupné z: https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html.

33. YOURDON, Edward; CONSTANTINE, Larry L. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. 2. vyd. New York: YOURDON Press, 1978. ISBN 0-917072-11-1.

34. ECMA INTERNATIONAL. *ECMA-404: The JSON data interchange syntax* [online]. 2. vyd. 2017. [cit. 2022-04-09]. Dostupné z: `https://www.ecma-international.org/publications-and-standards/standards/ecma-404/`.

35. DOWNEY, Allen B.; MAYFIELD, Chris. *Think Java: How to Think Like a Computer Scientist*. 2. vyd. Sebastopol (California, USA): O'Reilly Media, 2019. ISBN 978-1-492-07250-8.

36. FOWLER, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. vyd. USA: Addison-Wesley Professional, 2003. ISBN 978-0-321-19368-1.

37. W3C; MIT; ERCIM; KEIO. *XML Schema: Part 1: Structures Second Edition* [online]. 2004. [cit. 2022-04-10]. Dostupné z: `https://www.w3.org/TR/xmlschema-1/`.

38. FUGGETTA, Alfonso. A classification of CASE technology. *Computer*. 1993, roč. 26, č. 12, s. 25–38. Dostupné z DOI: `10.1109/2.247645`.

39. MEMBERS OF THE GIT COMMUNITY. *Git* [online]. 2022. [cit. 2022-05-05]. Dostupné z: `https://git-scm.com/`.

40. RYBOLA, Zdeněk et al. *KZ-server* [online]. 2022. [cit. 2022-05-05]. Dostupné z: `https://gitlab.fit.cvut.cz/rybolzde/KZ-server`.

41. FOWLER, Martin. *Continuous Integration* [online]. 2006. [cit. 2022-04-29]. Dostupné z: `https://www.martinfowler.com/articles/continuousIntegration.html`.

42. THE PHP GROUP. *SimpleXML: Installation* [online]. 2022. [cit. 2022-05-01]. Dostupné z: `https://www.php.net/manual/en/simplexml.installation.php`.

43. THE PHP GROUP. *DOM: Installation* [online]. 2022. [cit. 2022-05-01]. Dostupné z: `https://www.php.net/manual/en/dom.installation.php`.

44. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Westford (Massachusetts, USA): Addison-Wesley Professional, 1994. ISBN 978-0201633610.

45. SETASIGN GMBH & CO. KG. *FPDI* [online]. 2022. [cit. 2022-05-02]. Dostupné z: `https://www.setasign.com/products/fpdi/`.

46. NETTE FOUNDATION. *Nette DI Container* [online]. 2022. [cit. 2022-05-03]. Dostupné z: `https://doc.nette.org/en/dependency-injection/nette-container`.

47. FOWLER, Martin. *UnitTest* [online]. 2014. [cit. 2022-05-06]. Dostupné z: `https://martinfowler.com/bliki/UnitTest.html`.

48. AYEWAH, Nathaniel; PUGH, William; HOVEMEYER, David; MORGENTHALER, J. David; PENIX, John. Using Static Analysis to Find Bugs. *IEEE Software*. 2008, roč. 25, č. 5, s. 22–29. Dostupné z DOI: `10.1109/MS.2008.130`.

Seznam použitých zkratek

CASE Computer-Aided Software Engineering

CI Continuous Integration

CRUD Create, Read, Update, Delete

DAO Data Access Object

E/R Entity–relationship model

IDE Integrated Development Environment

JSON JavaScript Object Notation

MVC Model–View–Controller

MVP Model–View–Presenter

ORM Object–Relational Mapping

UML Unified Modelling Language

WYSIWYG What You See Is What You Get

XML eXtensible Markup Language

XSD XML Schema Definition

Příloha A

Příklady XML dokumentů

Akord

Tento XML dokument popisuje schéma kytarového akordu C (s alternativními názvy Cdur a Cmajor). Hmatník má 6 strun a 5 pražců, začíná pražcem číslo viii. První dvě struny jsou tlumené, stisknutá je struna 3 na pražci 10, struna 4 na pražci 9 a struny 5 a 6 na pražci 8.

```
<?xml version="1.0" encoding="UTF-8"?>
<chord>
  <name>C</name>
  <alt-names>
    <name>Cdur</name>
    <name>Cmajor</name>
  </alt-names>
  <def>
    <def-strings>6</def-strings>
    <def-frets offset="7">12</def-frets>
    <def-note>
      <note-string>6</note-string>
      <note-fret>8</note-fret>
    </def-note>
    <def-note>
      <note-string>5</note-string>
      <note-fret>8</note-fret>
    </def-note>
    <def-note>
      <note-string>4</note-string>
      <note-fret>9</note-fret>
    </def-note>
    <def-note>
      <note-string>3</note-string>
      <note-fret>10</note-fret>
    </def-note>
    <def-mark>
      <mark-string>1</mark-string>
      <mark-type>muted</mark-type>
    </def-mark>
```

```
    <def-mark>
      <mark-string>2</mark-string>
      <mark-type>muted</mark-type>
    </def-mark>
  </def>
</chord>
```

Píseň

XML dokument obsahuje text lidové písně Okoř včetně kytarových akordů. V příkladu je ve druhém verši první sloky vytvořená repetice, poslední verš refrénu je celý v repetici a v závěru písně se refrén opakuje třikrát. Akordy ve druhé sloce se nezobrazí v PDF exportu.

```
<?xml version="1.0" encoding="UTF-8"?>
<song>
  <info>
    <title>Okoř</title>
    <!-- <author>lidová</author> -->
  </info>
  <lyrics>
    <strophe label="1.">
      <paragraph>
        <verse><chord>D</chord>Na Okoř je cesta jako žádná ze
          sta,</verse>
        <verse><chord>A7</chord>vroubená je <repeat>stromama</repeat>
          <chord>D</chord>.</verse>
        <verse>Když jdu po ní v létě, samoten na světě,</verse>
        <verse><chord>A7</chord>sotva pletu nohama
          <chord>D</chord>.</verse>
        <verse><chord>D</chord>Na konci té cesty <chord>D</chord>
          trnité,</verse>
        <verse><chord>E</chord>stojí krčma jako <chord>A7</chord>
          hrad.</verse>
        <verse><chord>D</chord>Tam zapadli trampi, hladoví
          a sešlí,</verse>
        <verse><chord>A7</chord>začli sobě notovat
          <chord>D</chord>.</verse>
      </paragraph>
    </strophe>
    <strophe label="R.">
      <paragraph>
        <verse>Na hradě Okoři <chord>A7</chord>světla už nehoří,</verse>
        <verse><chord>D</chord>bílá paní <chord>A7</chord>šla už dávno
          spát.</verse>
        <verse>Ta měla ve zvyku,</verse>
        <verse><chord>A7</chord>podle svého budíku,</verse>
        <verse><chord>D</chord>o půlnoci <chord>A7</chord>chodit
          strašívat.</verse>
      </paragraph>
      <paragraph>
```

```
      <verse><chord>G</chord>Od těch dob, co jsou tam <chord>D</chord>
        trampové,</verse>
      <verse><chord>E</chord>nesmí z hradu <chord>A7</chord>
        pryč.</verse>
      <verse><chord>D</chord>A tak dole v podhradí,</verse>
      <verse><chord>A7</chord>se šerifem dovádí,</verse>
      <repeat>
        <verse><chord>D</chord>on jí sebral <chord>A7</chord>od
          komnaty klíč.</verse>
      </repeat>
    </paragraph>
  </strophe>
  <strophe label="2.">
    <paragraph>
      <verse><chord print="false">D</chord>Jednoho dne z rána,</verse>
      <verse>roznesla se zpráva,</verse>
      <verse><chord print="false">A7</chord>že byl Okoř
        <repeat>vykraden</repeat><chord print="false">D</chord>.
        </verse>
      <verse>Nikdo neví dodnes,</verse>
      <verse>kdo to tenkrát odnes,</verse>
      <verse><chord print="false">A7</chord>nikdo nebyl
        <chord print="false">D</chord>dopaden.</verse>
      <verse><chord print="false">G</chord>Šerif hrál celou noc
        <chord print="false">D</chord>mariáš</verse>
      <verse>s <chord print="false">E</chord>bílou paní v
        <chord print="false">A7</chord>kostnici.</verse>
      <verse><chord print="false">D</chord>Místo aby hlídal, zuřivě jí
        líbal,</verse>
      <verse><chord print="false">A7</chord>dostal z toho
        zimnici<chord print="false">D</chord>.</verse>
    </paragraph>
  </strophe>
  <repeat count="3">
    <strophe-ref ref="R." />
  </repeat>
 </lyrics>
</song>
```

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `release.zip` ..... balíček pro nasazení aplikace
- `thesis`
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `latex` ..... zdrojová forma práce ve formátu LaTeX
- `src`
  - `app` ..... zdrojové kódy aplikace
  - `lib`
    - `implementation` ..... zdrojové kódy knihovny
    - `specification` ..... specifikace vytvořených formátů