FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Přepracování aplikace Kytarový zpěvník do MVP architektury a návrh gamifikace aplikace
Student:	Jan Holan
Vedoucí:	Ing. Zdeněk Rybola
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Cílem bakalářské práce je přepracování stávající aplikace Kytarový zpěvník do MVP architektury, a s tím související aktualizace dokumentace zadání, architektury, vnitřní struktury aplikace a testů. Změny v aplikaci budou probíhat souběžně s bakalářskou prací dalšího studenta, která se bude zaměřovat na vzhled a UX aspekty aplikace. Součástí práce bude také rozšíření aplikační logiky o další požadavky stanovené v průběhu vedoucím práce a analýza a návrh gamifikačních prvků.

Pokyny pro vypracování:
- popište stávající architekturu aplikace Kytarový zpěvník
- proveďte analýzu a návrh potřebných změn pro přechod na MVP architekturu a navržené změny realizujte; zachovejte přitom nezbytné REST API využívané mobilní aplikací pro Android
- proveďte analýzu, návrh a realizaci nových požadavků na aplikaci
- výslednou aplikaci patřičně otestujte, včetně testů pokrytí kódu
- připravte programátorskou příručku pro následný rozvoj aplikace
- analyzujte a navrhněte vhodné prvky gamifikace aplikace

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 29. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Přepracování aplikace Kytarový zpěvník do MVP architektury a návrh gamifikace aplikace

Jan Holan

Katedra softwarového inženýrství
Vedoucí práce: Ing. Zdeněk Rybola

14. května 2018

Poděkování

Rád bych poděkoval vedoucímu své práce Ing. Zdeňku Rybolovi za časté konzultace a ochotu zabývat se mojí bakalářskou prací i ve svém volném čase. Další poděkování patří mému kamarádovi/spolustudentovi jménem Nikolai Baranov, který odvedl na aplikaci výbornou práci (zdokumentovanou v jeho vlastní bakalářské práci). A mé poslední jmenovité poděkování patří kamarádovi/kolegovi Jakubovi Červenkovi, který si ohledně webových aplikací ví rady se vším. Nakonec chci poděkovat za trpělivost své rodině a především mým přátelům, kteří mě po celou dobu studia podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2018 .....

Odkaz na tuto práci

Holan, Jan. *Přepracování aplikace Kytarový zpěvník do MVP architektury a návrh gamifikace aplikace.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Objektem mé bakalářské práce je webová aplikace Kytarový zpěvník, která již pár let vzniká v rámci předmětů SP-1 a SP-2. Hlavním záměrem aplikace je ulehčit organizaci písní pro hráče na kytaru a pomoci mu nalézt nové písně k zahrání. Účelem mé bakalářské práce je provést analýzu aktuálního stavu Kytarového zpěvníku, který díky použitým zastaralým technologiím není aktuálně schopný konkurovat podobným aplikacím. Na základě výstupu tohoto průzkumu pak tvořím návrh převedení aplikace na MVP architekturu a použití frameworku Nette pro frontend a v závěru své práce se věnuji implementaci mnou navrhnutých kroků. Zabývám se i porovnáním s konkurencí a konkrétním návrhem toho, jak by principy gamifikace mohly pomoci Kytarovému zpěvníku při uvedení do živého provozu. Vytvořené řešení slouží jako podklad pro práci dalších studentských týmů.

Klíčová slova Webová aplikace, Kytarový zpěvník, Backend, MVP architektura, MVC architektura, Nette, PHP

Abstract

The core of this bachelor thesis is web application Kytarový zpěvník, that was developed during BI-SP1 and BI-SP2 courses. Main goal of application is to help musicians with organizing their music and with discovery of new songs to play. Aim of my thesis is to perform analysis of application's current state, because due to usage of outdated technologies, the application is not able to withstand compared to competition. Results of analysis serve as input for next phase, during which I design new version of application, which will be using MVP architecture pattern and Nette framework for it's frontend. The third step is implementation of said design. Last step of thesis is centered about gamification phenomenon. My main focus is to research possible ways, how gamification could help application ease it's start on the market. Results of this chapter will serve as starting point for other teams working on Kytarový zpěvník.

Keywords Web application, Guitar, Songbook, Backend, MVP architecture, MVC architecture, Nette, PHP

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Webové aplikace jsou v dnešní době všude kolem nás a jejich tvorba je vhodnou úvodní kariérní možností pro nejednoho studenta informatiky. V éře mobilních zařízení a skoro všudypřítomné dostupnosti internetového připojení jsou takové aplikace dostupné kdykoli a tak není divu, že jsou primárním zaměřením nespočtu IT firem, ať už komorních start-upů či velkých korporací.

Zaměřením mé bakalářské práce je aplikace Kytarový zpěvník, která vzniká už několik let pod vedení Ing. Zdeňka Ryboly a to nejen v průběhu předmětů BI-SP1 (softwarový projekt 1) a BI-SP2 (softwarový projekt 2), ale byla již předmětem i dvou závěrečných prací. Aplikace je zaměřena na hráče na kytaru bez ohledu na úroveň schopností a nabízí hlavně komplexní systém správy písní a jejich sdružování do zpěvníků.

Já sám jsem vedl po dva předchozí semestry tým, který se na vývoji aplikace podílel. Udělal jsem na ní s ostatními za tu dobu velký kus práce a nyní má Kytarový zpěvník už skoro všechny funkce, které potřebuje pro uvedení do ostrého provozu.

Reálnému používání ovšem brání fakt, že když aplikace v roce 2014 vznikla, sázela na poněkud netradiční architekturu a experimentální technologie. Ty bohužel za roky vývoje vyšly z módy a výsledkem je nepřehledný kód v programovacím jazyce *Dart*, který má navíc na svědomí nízkou rychlost aplikace a některé uživatelsky nepohodlné interakce. A protože jsem na Kytarovém zpěvníku poslední rok pracoval, rád bych projekt osobně dovedl až do zdárného konce.

Hlavním cílem mé bakalářské práce je přepracování Kytarového zpěvníku do MVP (Model-View-Presenter) architektury a zároveň zachování a zefektivnění všech stávajících funkcí. Také je mým úkolem dostat projekt do stavu, kdy bude pro následující týmy pracující na něm v budoucnosti mnohem jednodušší do aplikace zasahovat a provádět údržbu či její rozšiřování o nové funkcionality.

Prvním krokem práce je analytická fáze, kde zkoumám současnou podobu

aplikace a způsoby řešení, které využívá. Zároveň v ní také představuji do detailů konkrétní důvody pro takto radikální přepracování projektu. Součástí kapitoly s analýzou je i uvedení nových požadavků od vedoucího bakalářské práce. Tyto požadavky byly formulovány v průběhu mé práce právě na základě výsledků provedené analýzy.

Následuje návrhová část, kde prezentuji nově navržené řešení aktuálních problémů aplikace. Vysvětluji zvolenou architekturu a výsledné rozdělení jednotlivých tříd, které doplňuji patřičnými diagramy a také odůvodněním, proč jsem daná rozhodnutí učinil.

Předposlední částí je Realizace, která se týká průběhu uskutečnění nutných kroků, během kterých byla aplikace převedena do finální podoby. Obsahem je i výčet použitých nástrojů, programátorská příručka a vytvoření vhodného počtu automatizovaných testů, které mají na starost ověřování správného fungování Kytarového zpěvníku.

Výsledkem výše zmíněných kapitol je funkční webová aplikace ve verzi 2.0, připravená k nasazení do živého prostředí.

Čtvrtá a zároveň i závěrečná kapitola pojednává o fenoménu gamifikace a způsobu, jakým by její principy mohly být využity ve spojitosti s Kytarovým zpěvníkem. Po nasazení na veřejný server bude potřeba utvořit aktivní komunitu uživatelů, kteří budou vytvářet obsah. Aby se něčeho takového dosáhlo, je třeba nabídnout nějaké netradiční funkce oproti stávající konkurenci a nebo se pokusit žádané chování uživatelů podpořit zmíněnou gamifikací. Výsledek této části poslouží jako podklad pro budoucí týmy pracující na Kytarovém zpěvníku v rámci předmětů BI-SP.

Tato práce je psána ve spolupráci se studentem Nikolaiem Baranovem z FIT ČVUT, členem mého týmu z předmětů BI-SP1 a BI-SP2. Nikolai se ve své práci *UI aplikace Kytarový zpěvník s ohledem na UX* věnuje designu nové podoby aplikace.

1.1 Cíl

Hlavním cílem rešeršní části práce je analýza a zdokumentování aktuální podoby aplikace Kytarový zpěvník. Výsledky této rešerše poslouží jako vstup pro praktickou část mé práce, při které navrhnu postup pro převedení aplikace do MVP architektury při zachování stávajících funkcionalit a REST API (Representational State Transfer Application Programming Interface) využívané mobilní aplikací.

Součástí práce je i vykonání nezbytných kroků pro převod aplikace, analýza, návrh a implementace nových požadavků, které poskytne zadavatel. Výsledkem práce bude otestovaná aplikace ve verzi 2.0, patřičně pokryta automatickými testy včetně programátorské příručky pro následný rozvoj aplikace.

Posledním cílem práce je analýza a návrh možností gamifikace aplikace za účelem vybudování aktivní komunity při nasazení do produkčního prostředí.

Kapitola 2

Analýza

Ještě než dojde k napsání jakékoli řádky kódu, je prvním logickým krokem analýza aktuálního stavu systému. Její výsledek poslouží nejen jako výchozí bod pro další kapitolu (která se týká návrhu), zároveň mi ale také umožní čtenáři co nejsrozumitelněji vysvětlit, v jakém stavu se aplikace vlastně nachází a co všechno dokáže. Pro větší přehled jsem se rozhodl analýzu rozdělit do tří sekcí.

V té první, která nese název Stávající funkčnost aplikace, Kytarový zpěvník představím a zmíním jeho stručnou historii. Poté se důkladněji zaměřím na jednotlivé obchodní procesy, okolo kterých je aplikace postavena.

V kapitole Aktuální způsob řešení se na současný stav Kytarového zpěvníku podívám z více technického hlediska. Zabývám se v ní architekturou aplikace a technologiemi, které byly pro vývoj doposud používány. Ve třetí a poslední části analýzy popíšu, co všechno je aktuálně se systémem špatně a zároveň uvedu specifikace nových požadavků, který mi byly zadány vedoucím práce.

2.1 Stávající funkčnost aplikace

Než se pustím do psaní o nových funkcích, je důležité představit nezainteresovanému čtenáři ty staré. Protože ovšem aplikace nesestává jen ze spousty funkcionalit, přiblížím mu i to, jaké týmy studentů a kolik práce vlastně za Kytarovým zpěvníkem stojí.

2.1.1 Úvod do aplikace a její historie

Projekt Kytarový zpěvník je relativně rozsáhlý, co se funkcí týče. Důvod pro to je jednoduchý, práce na něm začaly už v roce 2014. Tehdy byl projekt Ing. Zdeňkem Rybolou založen a v rámci předmětů BI-SP1 (a posléze BI-SP2) se na něm podílel první tým studentů pod vedením Tomáše Markacze. Ten vytvořil fungující jádro aplikace, na kterém poté Jiří Mantlík, jeden z členů týmu, založil svoji bakalářskou práci nazvanou Rozšíření aplikace Kytarový

zpěvník.[1] V rámci ní implementoval nové funkcionality, jako třeba možnost transpozice akordů písně a nebo nové způsoby vkládání textů.

O rok později na práci navázal nový studentský tým, který Kytarový zpěvník doplnil o mobilní aplikaci pro operační systémy Android. I když se nejednalo o plnohodnotnou alternativu k webové verzi, aplikace umožnila uživatelům mít obsah vytvořený na webu vždy po ruce. Dva členové tohoto týmu, Tomáš Havlíček (frontend) a Gabriela Melingerová (backend), pokračovali v projektu nadále i v roce 2016 v rámci svých závěrečných prací [2] [3].

Následující rok se už správy a vylepšování aplikace ujal mnou vedený tým ve složení Nikolai Baranov, Julia Evseenko, Denis Karlov a Vojtěch Stuchlík. Náplní naší práce krom všeobecné optimalizace a aktualizace dokumentace po předchozích týmech bylo i navržení a implementace systému nahlašování nevhodného obsahu, možnosti exportu do PDF a několik dalších funkcí. Tým vydržel dva semestry skoro v původním složení a nakonec jsme se spolu s Nikolaiem rozhodli navázat bakalářskými pracemi. Stejně jako dvojice studentů před dvěma lety, i my jsme si rozdělili práci na aplikaci na frontendovou [4] a backendovou část.

2.1.2 Aktuální možnosti aplikace

2.1.2.1 Uživatelské profily

Aplikace obsahuje standardní proces registrace uživatele a možnost obnovení ztraceného hesla pomocí vygenerovaného tokenu zaslaného na e-mail spojený s uživatelským účtem. Kytarový zpěvník je možné využít i jen jako návštěvník, je pak ale možné pouze obsah prohlížet a nikoli tvořit či upravovat. Pokud je uživatel přihlášen, otevřou se mu další sekce aplikace, jako je třeba seznam přání. Zároveň dostane možnost editovat svůj osobní profil, který si ostatní uživatelé můžou zobrazit.

2.1.2.2 Písně

Základem jsou texty s akordy, které budu v dalším textu označovat jako písně. Ty se dělí na soukromé a veřejné. Kromě základních CRUD (create, read, update a delete) operací je možné je i komentovat, hodnotit (komentáře i hodnocení opět disponují CRUD operacemi), nahlašovat, transponovat akordy a nebo používat automatické posouvání stránky (autoscroll) pro lepší pohodlí během hraní.

K dispozici je i trojice funkcí určených k šíření obsahu. Soukromou píseň lze nasdílet jinému uživateli a pro toho se poté bude chovat jako píseň veřejná. Jakýkoli veřejný nebo nasdílený obsah je poté možné převzít, což uživateli umožní přiřadit mu vlastní soukromé tagy (o těch později) a také si ji řadit do vlastních zpěvníků. Nemá ovšem žádnou kontrolu nad obsahem, která zůstane původnímu majiteli. A v poslední řadě je možné veřejný, nasdílený či převzatý

obsah zkopírovat, čímž se vytvoří naprosto identická kopie, ovšem s novým uživatelem jako vlastníkem.

2.1.2.3 Zpěvníky

Písně je možné řadit do zpěvníků, které stejně jako písně můžou být privátní či veřejné. Uvnitř zpěvníků je možné jednotlivé písně manuálně seřadit a při zobrazení nějaké písně ze zpěvníku je uživateli umožněno zpěvníkem virtuálně listovat a přecházet tak na následující či předchozí píseň. Zpěvník také disponuje většinou funkcí, jaké jsou dostupné u písní – je tedy možné ho okomentovat, hodnotit, nahlašovat a využívat všech tří funkcí na šíření obsahu.

2.1.2.4 Tagy

V systému také existuje možnost obsahu přidávat tagy, opět se dělící na veřejné a soukromé. Ty představují určitou kategorii, do které obsah spadá. Může se jednat třeba o označení žánru (folk/pop/rock atd.) a nebo třeba jazyku textu písně. Slouží především pro usnadnění organizace obsahu, protože se podle nich dá vyhledávat.

2.1.2.5 Přehledy

Písně i zpěvníky disponují přehledem, na kterém si může uživatel prohlédnout všechen svůj a nebo všechen veřejný obsah v systému. Je mu umožněno si nechat zobrazený obsah filtrovat, například podle jména či autora. Jinak jsou ovšem přehledy jednou ze slabších stránek aplikace, protože zobrazovací tabulky neovládají stránkování a skutečné vyhledávání (jedná se pouze o filtrování již zobrazených dat) a práce s nimi není příliš uživatelsky přívětivá.

2.1.2.6 Přání

Když v systému uživateli nějaká konkrétní píseň chybí, může ji zanést na svůj seznam přání. Pokud se pak v budoucnu obsah se stejným názvem a správným autorem v Kytarovém zpěvníku objeví, bude uživatel na tuto skutečnost upozorněn pomocí systémové notifikace.

2.1.2.7 Export

Pro dostupnost vytvořených zpěvníků offline disponuje aplikace i exportem do formátu PDF za účelem tisku. Krom jednotlivých písní je možné exportovat více písní z jakéhokoli jejich přehledu a také souhrnným exportem zpěvníku, který přidá titulní stranu, obsah a číslování stránek. Formát je částečně dynamický, takže se systém pokusí díky úpravám šablony vždy využít co nejmenší počet stránek. Například pokud se píseň nevejde na stránku do jednoho sloupce, bude zobrazena ve dvou úzkých sloupcích vedle sebe.

2.1.2.8 Administrátorská část

Jestliže má uživatel roli administrátora, je mu zpřístupněna navíc další část aplikace. V administrátorském menu může mazat a obnovovat zpěvníky a písně, povyšovat ostatní uživatele na administrátory a nebo řešit nahlášení nevhodného obsahu. Ty mají vždy více možností finálního verdiktu. Nahlášení může být zamítnuto a poté nejsou podniknuty žádné další kroky. Další variantou je schválení nahlášení s odstraněním závadného obsahu (nebo jeho vynuceném odebrání z veřejných). A v poslední řadě je možné obsah rovnou smazat spolu se zablokováním uživatele, který tím pádem přijde o možnost se do systému přihlásit a bude na tuto skutečnost při dalším pokusu o přihlášení upozorněn.

2.1.2.9 Notifikace

Poslední zmínka patří systému notifikací, který je v Kytarovém zpěvníku implementován. Spousta z akcí zmíněných výše svým provedením odešle notifikaci, která na dění příslušného uživatele upozorní. U notifikací je zaznamenáno, jestli je uživatel již zobrazil a zároveň má možnost označit za zobrazené všechny. Notifikace také umožňují jediným kliknutím přejít přímo na stránku obsahu, o kterém je řeč, popřípadě si zobrazit profil uživatele, který akci vykonal.

2.1.3 Mobilní aplikace

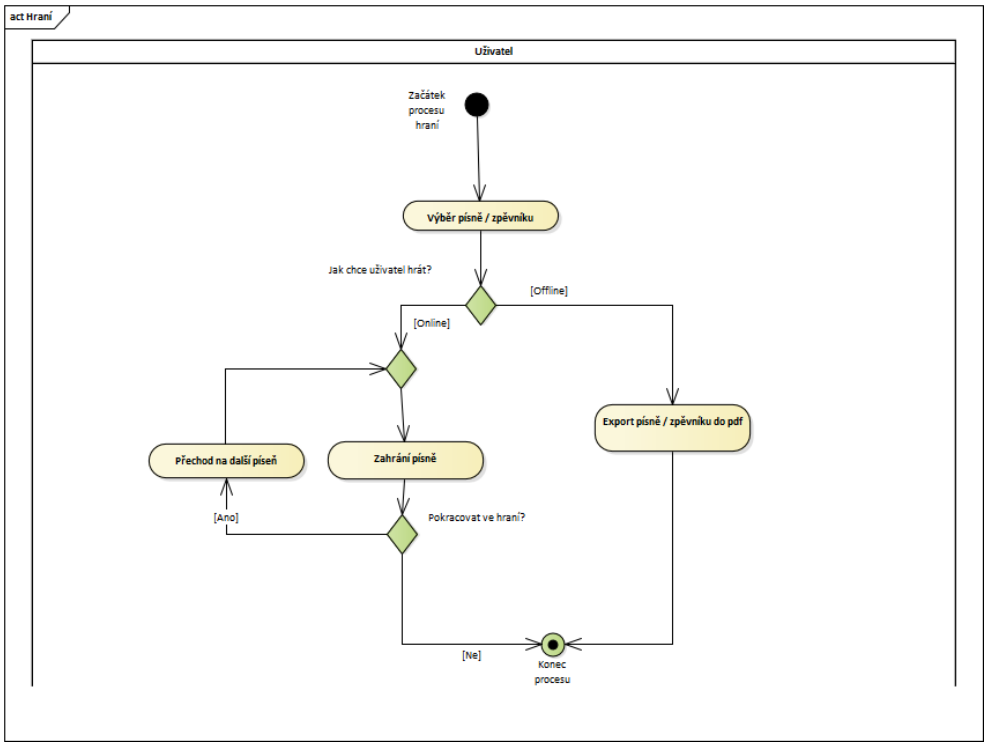
Doprovodná Android aplikace ke Kytarovému zpěvníku se prozatím nenachází na Google Play a tak je pro její stáhnutí nutné zavítat na úložiště projektu na fakultní verzi *GitLabu*. Jejími klíčovými funkcemi je zobrazování/vyhledávání uložených písní (bez možnosti editace) a také kompletní správa jejich zařazení do zpěvníků. Oproti webové aplikaci toho ta mobilní stále většinu postrádá a jedná se tak skutečně jen o doprovodný nástroj a ne plnohodnotnou náhradu. Existence tohoto mobilního společníka se nepřímo dotýká i mé bakalářské práce a vytyčuje tak jisté hranice, co se architektury týče – při převodu na novou verzi je třeba zachovat v celém rozsahu REST API, pomocí kterého spolu aplikace pro Android a webová aplikace komunikují.

2.1.4 Obchodní procesy

Obchodní procesy jsou řetězce aktivit, které uživatel vykonává za určitým cílem za pomoci aplikace. Jejich identifikace je velmi důležitou součástí začátku vývoje, protože dokáže poukázat na klíčové části systému (na které je třeba se více zaměřit), lépe pochopit zadanou problematiku a v neposlední řadě slouží i k tomu, aby byla potenciálně problémová místa aplikace včas rozpoznána. Pro znázornění obchodních procesů v rámci Kytarového zpěvníku

se už od jeho začátku využívá UML (Unified Modeling Language) diagram aktivit, modelovaný v nástroji *Enterprise Architect*.

Protože projekt má po svých čtyřech letech ve svém repozitáři namodelované velké množství diagramů, vychází všechny uvedené v této kapitole z práce provedené během předchozích let. Jmenovat je všechny by bylo zbytečné a tak zmíním pouze ty, jejichž myšlenky jsou dle mě nejdůležitější.



Obrázek 2.1: Obchodní proces hraní písní

2.1.4.1 Hraní písní

Hraní tvoří alfu a omegu celé aplikace. Diagram 2.1 znázorňuje v obecné rovině pořadí kroků, které hudebník při hraní za použití Kytarového zpěvníku využívá.

Proces započne tím, že uživatel nalezne cílovou píseň, kterou si chce zahrát. V tuto chvíli má před sebou dvě možnosti. Pokud si chce obsah zahrát bez přítomnosti elektronického zařízení (například si ho vzít s sebou na cesty), může si danou píseň (popřípadě celý zpěvník) nechat vyexportovat do formátu PDF (Portable Document Format). V opačném případě si píseň rovnou zahraje za pomoci akordů a textu zobrazeného aplikací. Rozhodne-li se pro dru-

hou možnost, může využít výhod aplikace jako je třeba transpozice a nebo autoscroll. Když je píseň dohrána, má uživatel možnost přejít na další obsah.

2.1.4.2 Tvorba zpěvníku

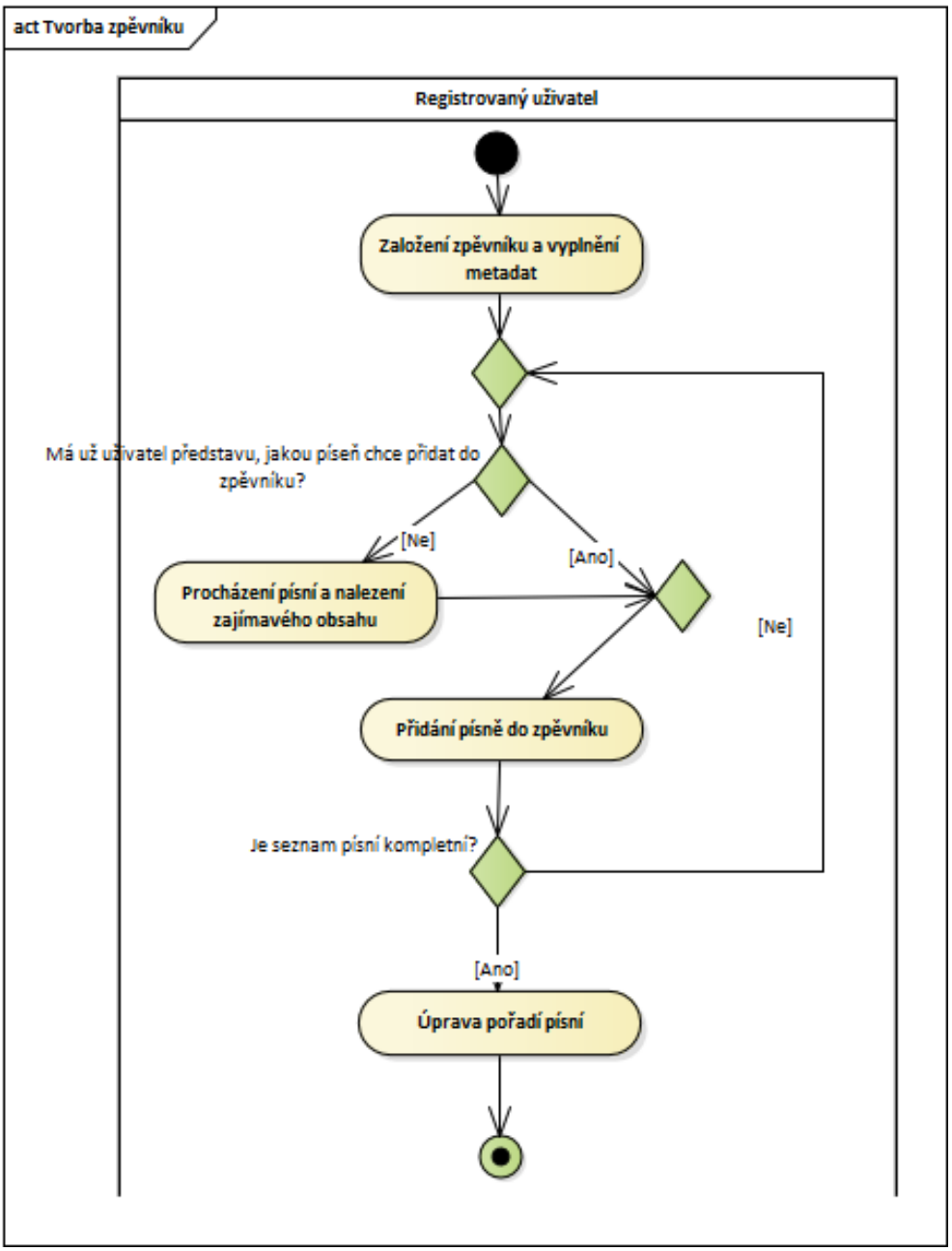
Protože proces vložení nové písně je velmi přímočarý, rozhodl jsem se zde zmínit jen proces tvorby zpěvníku a doprovodit ho diagramem 2.2. Ten totiž sestává hned z několika kroků, které je nutné vykonat v předem určeném pořadí. Samotnému uložení zpěvníku do systému předchází jeho pojmenování a vyplnění dalších metadat, jako jsou třeba tagy a nebo poznámka.

Dalším krokem je výběr vhodných písní, který se dá realizovat z detailu zpěvníku a nebo přímo z detailu písně, kterou chce uživatel do zpěvníku zařadit. Na závěr je autorovi umožněno změnit pořadí jednotlivých písní a ovlivnit tak listování ve zpěvníku či jeho vzhled při exportu do formátu PDF.

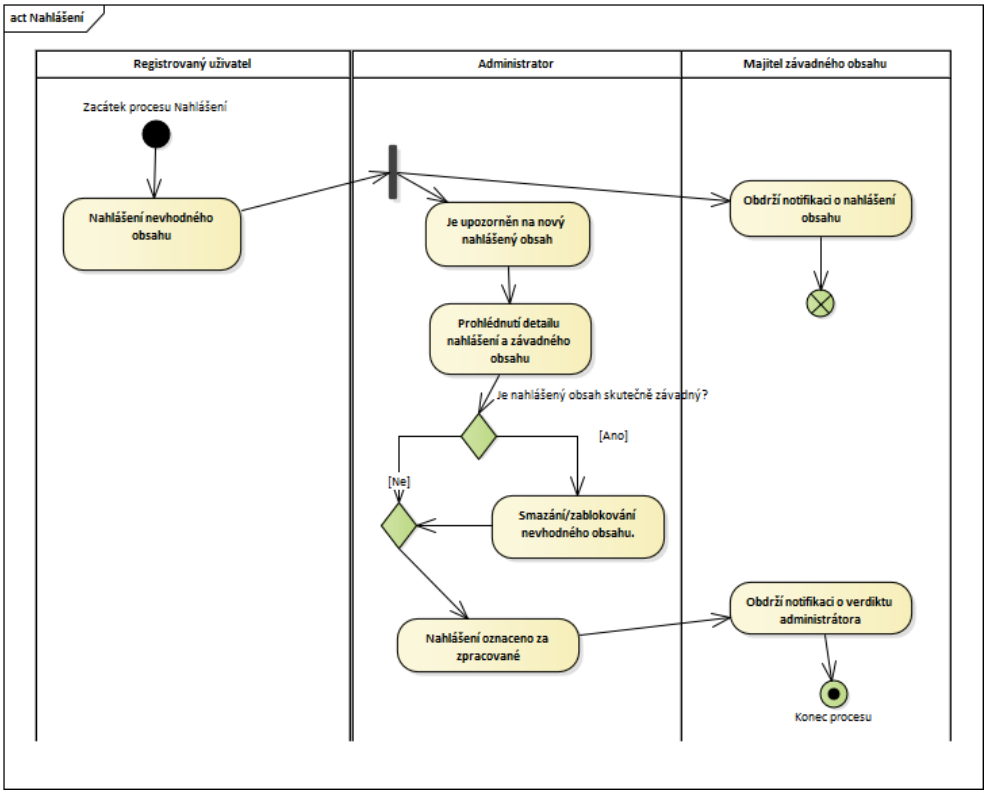
2.1.4.3 Nahlášení nevhodného obsahu

Systém stojí primárně na obsahu tvořeném registrovanými uživateli a proto je nutné poskytnout administrátorům možnost, jak regulovat kvalitu obsahu. Protože ale systém odstraňování písní na základě nízkého hodnocení není úplně pružný a nalezení závadného obsahu tímto způsobem by mohlo trvat zbytečně dlouho, vytvořili jsme s týmem během předmětu BI-SP1 a BI-SP2 funkci nahlašování.

Její průběh znázorňuje obrázek 2.3. Pokud registrovaný uživatel narazí při procházení aplikací na nevhodný obsah (aktuálně podporované kategorie jsou píseň/zpěvník/uživatel/hodnocení/komentář), má možnost ho nahlásit. O této skutečnosti je poté informován majitel závadného obsahu, ale také administrátoři Kytarového zpěvníku. Administrátor má možnost si závadný obsah zobrazit a posléze vynést verdikt ohledně toho, jak bude nahlášení vyřešeno. Majitel obsahu je opět o výsledku nahlašovacího procesu informován.



Obrázek 2.2: Obchodní proces tvorby zpěvníku



Obrázek 2.3: Obchodní proces nahlášení nevhodného obsahu

2.1.5 Doménový model

Doménový model je další nedílnou součástí analýzy, většinou se znázorňuje za pomoci diagramu tříd UML. Slouží primárně pro pochopení entit, které se v dané doméně nachází, jejich významů, atributů a také vztahů, které mezi jednotlivými objekty systému existují. Doménový model neobsahuje žádné detaily vztahující se k způsobu implementace, slouží ale jako výchozí bod pro modelování databáze a návrh konkrétních tříd. Diagramy 2.4 a 2.5 znázorňují aktuální stav, v jakém se aplikace nachází před provedením jakýchkoli plánovaných změn. Pro větší přehlednost byly elementy nahlášení a notifikace vyčleněny do vlastního diagramu, protože mají příliš mnoho vztahů s dalšími objekty.

2.1.5.1 Uživatel

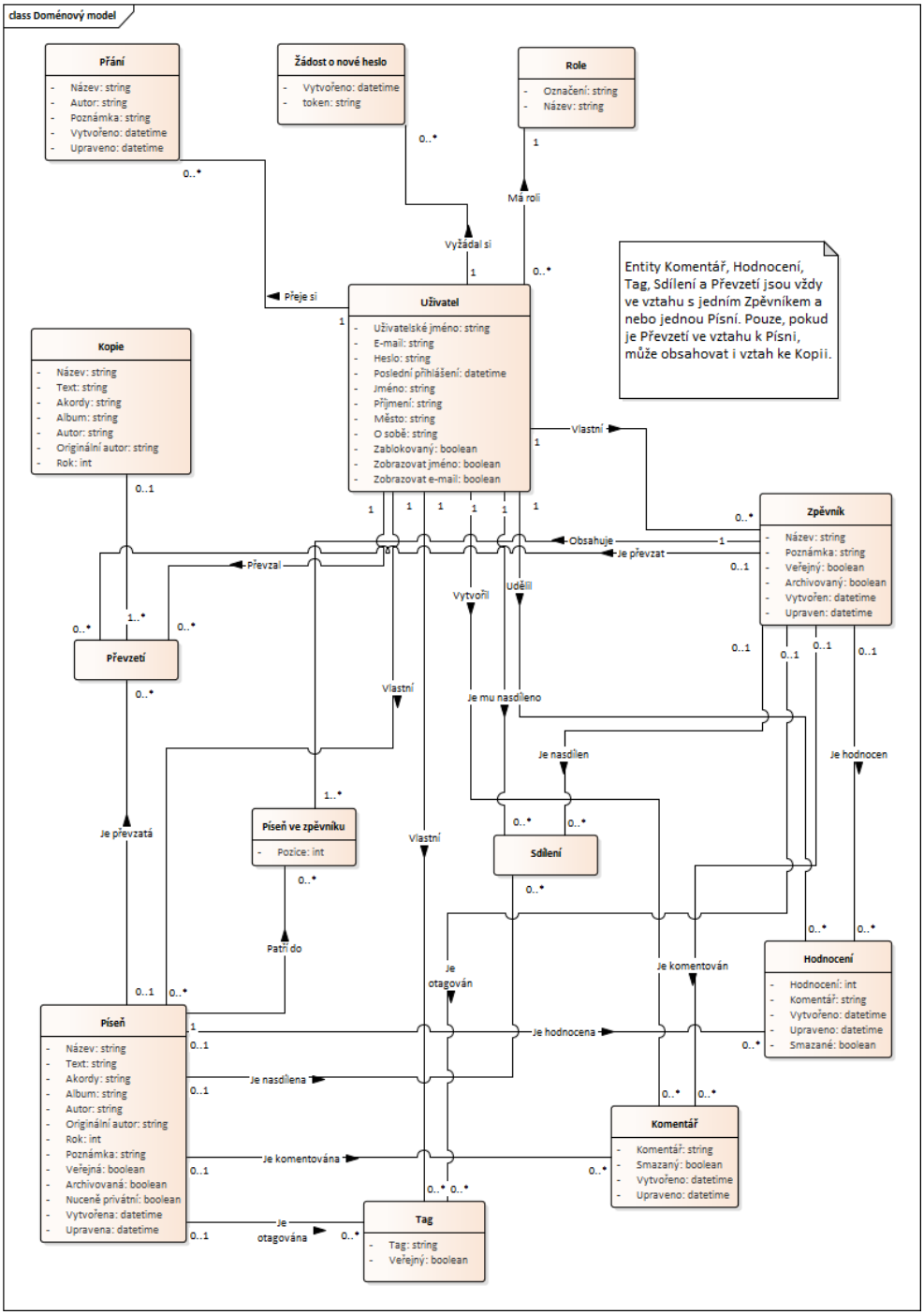
Třída reprezentuje uživatele systému. Od poslední aktualizace doménového modelu přibyly atributy sloužící k upravování a prezentování uživatelského profilu (Jméno, Příjmení, Město, O sobě, Zobrazovat e-mail, Zobrazovat jméno), ty ovšem můj tým přidal už v průběhu BI-SP2.

2.1.5.2 Role

Třída představuje různé uživatelské role, které jsou v systému zastoupeny. V současné době se rozlišuje pouze nepřihlášený uživatel, přihlášený uživatel a administrátor. Nedá se ale vyloučit, že bude v budoucnu k dispozici rolí více. Pokud nemá uživatel aplikací přidělenou roli, je považován za nepřihlášeného uživatele. Role má svůj název a také jeho ekvivalent bez diakritiky, uživatel má vždy přesně jednu roli.

2.1.5.3 Žádost o nové heslo

Když uživatel požádá o resetování svého hesla pro přihlášení, je třeba to zaznamenat – a přesně k tomu slouží tato třída. Datum vytvoření slouží pro ověření platnosti žádosti (je platná pouze 48 hodin) a token slouží pro jednoznačnou identifikaci žadatele.



Obrázek 2.4: Doménový model aplikace

2.1.5.4 Přání

Systém přání umožňuje uživateli vyjádřit, že mu nějaká píseň v systému chybí. Pro zaznamenání této skutečnosti slouží stejnojmenná třída. Obsahuje údaje o chybějící písni a datum vytvoření a poslední úpravy. Uživatel může do systému zadat neomezeně mnoho přání.

2.1.5.5 Píseň

Každá píseň sestává z názvu, textu, akordů a dalších metadat a zároveň má vždy jednoho uživatele/vlastníka. Jediným atributem potřebující vysvětlení je boolean *nuceně privátní*. Ten nabývá pozitivní hodnoty v případě, že píseň byla na základě nahlášení násilně odebrána z veřejných a není tudíž možné ji opět zveřejnit.

2.1.5.6 Zpěvník

Zpěvník samotný tvoří jen název, poznámka, vlastník a pár stavových boolean atributů (stejné jako u entity píseň). Zařazování písní do něj řeší samostatná třída.

2.1.5.7 Píseň ve zpěvníku

Tato pomocná vztahová třída představuje vztah mezi zpěvníkem a písněmi v něm umístěnými. Zároveň tento vztah blíže upřesňuje, protože stanovuje pořadí, v jakém jsou písně do zpěvníku zařazeny.

2.1.5.8 Komentář

Entita představuje komentář, který uživatel v systému přidal ke konkrétní písni a nebo zpěvníku. Oproti starší verzi doménového modelu přibyl boolean symbolizující, že je komentář smazaný a tudíž nemá být zobrazován uživatelům. V aplikaci se komentáře dělí na komentáře písně a zpěvníku, v rámci doménového modelu to není nutné.

2.1.5.9 Hodnocení

Třída reprezentuje hodnocení udělené písni či zpěvníku uživatelem. Krom počtu hvězdiček v rozsahu 1-5 je součástí hodnocení i nepovinný komentář, kterým může uživatel svoje hodnocení odůvodnit. V aplikaci se hodnocení dělí na hodnocení písně a zpěvníku, v rámci doménového modelu to není nutné.

2.1.5.10 Tag

Tag symbolizuje štítek, jakým je možné v aplikaci označit píseň či zpěvník, a který slouží k ulehčení vyhledávání a organizace. Tagy můžou být veřejné, jako

je třeba jazyk písně či žánr, či soukromé, kterými si může uživatel poznačit obsah jen pro vlastní potřebu.

2.1.5.11 Sdílení

Sdílení je čistě vztahová *manyToMany* třída. Ztvárňuje skutečnost, že uživatel může svůj soukromý zpěvník či píseň sdílet jinému uživateli a pro toho se pak sdílený obsah chová jako veřejný.

2.1.5.12 Převzetí

Uživatel si může jakýkoli veřejný obsah převzít a je mu poté umožněno si tento obsah označit soukromými tagy a nebo zařadit do vlastních zpěvníků, pokud je převzatým obsahem píseň. Pokud má uživatel převzatý zpěvník, chovají se pro něj i všechny písně ve zpěvníku jako veřejné.

2.1.5.13 Kopie

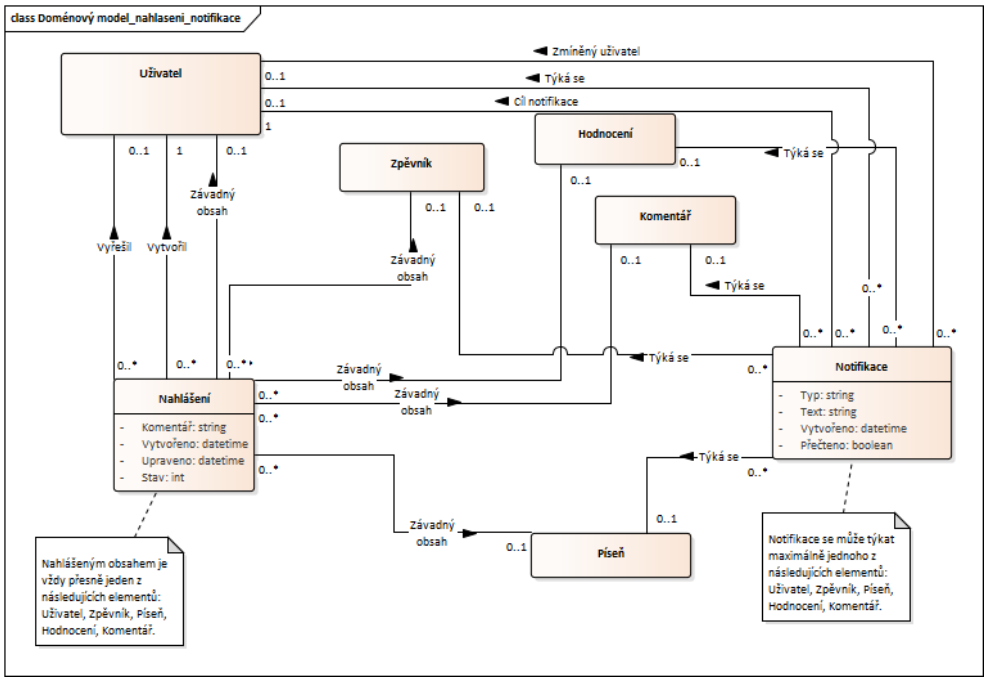
V případě, že má uživatel převzatou píseň a její vlastník ji upraví, vytvoří se kopie původního historického stavu písně (převzatá je ovšem nadále nově upravená píseň). Na tuto skutečnost je poté uživatel upozorněn a má možnost si kopii nechat zobrazit. V případě potřeby si ji může uložit jako vlastní píseň, pokud mu stará verze vyhovuje více, než verze po editaci. Třída tedy uchovává chvilkově historii změněné písně pro případ, že by se ostatním uživatelům nová verze nelíbila.

2.1.5.14 Notifikace

Entita notifikace představuje všechna upozornění, která uživatel v systému obdrží. Může se vázat na píseň, zpěvník, komentář, hodnocení a nebo přímo na jiného uživatele a po přečtení notifikace má uživatel možnost zobrazit si zmíněný obsah. Zároveň v ní může být zmíněný jiný uživatel (může se jednat třeba o toho, kdo akci vyvolal), na jehož profil je možné z notifikace přejít. Důležitou součástí notifikace je boolean symbolizující, jestli ji už uživatel vzal na vědomí a nebo ještě ne.

2.1.5.15 Nahlášení

Třída nahlášení slouží k monitorování závadného obsahu. Sestává ze vztahu na závadný obsah (je to vždy píseň, zpěvník, komentář, hodnocení a nebo uživatel) a z popisu nahlášení. Zároveň je vždy v relaci s jedním uživatelem, který nahlášení vytvořil. Nepovinně může být také v relaci s uživatelem s administrátorskými právy, který nahlášení zpracoval.



Obrázek 2.5: Doménový model aplikace se zaměřením na nahlášení a notifikace

2.2 Aktuální způsob řešení

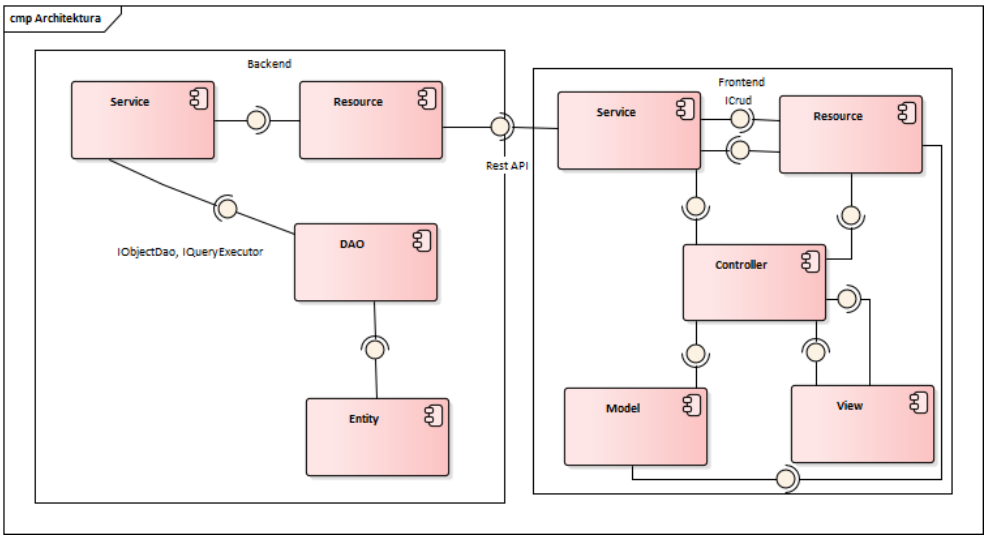
Doposud jsem psal o Kytarovém zpěvníku obecně. O procesech a myšlenkách, které za ním stojí, a o vztazích jednotlivých objektů problémové domény. Neméně důležitým krokem analýzy je ale i pohled na konkrétní použitou architekturu a způsob, jakým je aplikace nyní postavena a jak přistupuje k řešení problémů.

2.2.1 Použité technologie a architektura aplikace

Protože se nejedná o nový projekt, stihly už některé technologie, na kterých je aplikace postavena, značně zestárnout. A ostatně i architektura, která byla v roce 2014 při raných fázích návrhu projektu zvolena, je spíše experimentální. Aby mohly být využité rozdílné technologie, bylo tehdy rozhodnuto, že frontend a backend Kytarového zpěvníku budou dvě navzájem oddělené aplikace, jak je vidět na diagramu 2.6.

2.2.1.1 Backend

Backendová část starající se o veškerou manipulaci s daty je naprogramována v jazyku *PHP*[5] (Hypertext Preprocessor) za použití českého frameworku



Obrázek 2.6: Diagram komponent

Nette.[6] Pro mapování tříd do *MySQL* (Structured Query Language) databáze slouží knihovna *Doctrine*[7], spolu s dalšími závislostmi nainstalovaná pomocí *Composer*[8]. Většina použitých nástrojů na backendu se nachází ve starších verzích. K hromadné aktualizaci při předchozích bězích nedošlo převážně proto, že veliké množství PHP knihoven je závislé na nové verzi samotného jazyka (7 a výše), kdežto v aplikaci je doposud používaná starší verze 5.6. V nové verzi byly ve velkém některé použité konstrukce jazyka označené za *legacy* (tj. při použití způsobují zobrazení *warning* zpráv) a tím pádem se tímto problémem žádný z týmů doposud nezabýval, protože se nejedná o triviální proceduru.

Backendová část využívá třívrstvé architektury, která odděluje prezentační vrstvu (*Resources*, které slouží jako API endpoints a formátují vstupy a výstupy), aplikační vrstvu (*Services*, které mají na starost klíčovou logiku) a datovou vrstvu (zastoupenou *DAO* [Data Access Object] třídami, které zajišťují spolupráci s úložištěm dat). Hlavní výhodou této architektury je lehké testování a zároveň vyměnitelnost jednotlivých částí aplikace.

Na některých místech je ještě tato architektura uvolněná, tj. nekomunikují spolu jen vzájemně sousedící vrstvy. Při implementační části své práce bych rád tuto skutečnosti změnil a převedl architekturu na striktní, jak je zobrazena na obrázku výše. Protože jedinou možností použití backendové části aplikace je použití dostupného API, nejedná se o MVC (Model-View-Controller) ani MVP (Model-View-Presenter) přístup, navzdory použití třívrstvé architektury, protože ve struktuře chybí *View*.

2.2.1.2 Frontend

Frontend Kytarového zpěvníku je psaný v jazyce *Dart*[9] a komunikuje s backendovou aplikací za pomoci HTTP (Hypertext Transfer Protocol) požadavků na specifikované API endpoints. Dart je objektově orientovaným jazykem vyvinutým společností Google, který byl zamýšlený k použití pro webové a mobilní aplikace. Od svého představení v roce 2011 se ale příliš neuchytil a jedinou reálnou možností, jak ho v dnešním světě webových aplikací využít, je překompilovat ho do jazyka *javascript*, protože samotný Dart jako takový podporuje jen prohlížeč *Dartium*[10].

Dart se stará o veškerou logiku FE (frontendu) a vykonává v současné době některé úkoly, kterými by se frontend aplikace podle konvencí zabývat neměl – sám se třeba stará o filtrování písní v přehledu a nebo o proces rozřezávání písně na jednotlivé kusy s přiřazenými akordy. Zbylé technologie použité na FE slouží čistě k vytvoření vzhledu webových stránek, které jsou psané v *HTML* (Hyper Text Markup Language) a doplněny o styly za pomocí jazyka *LESS* (Leaner Style Sheets, rozšířený jazyk CSS[11]) a balíčku *Bootstrap*[12]. Pokud je potřeba dodat nějakou dynamiku do zobrazovaných stránek, občas je místo jazyka Dart využit přímo javascript s knihovnou *jQuery*[13]. O správu všech závislostí se starají *npm*[14] a *pub*[15].

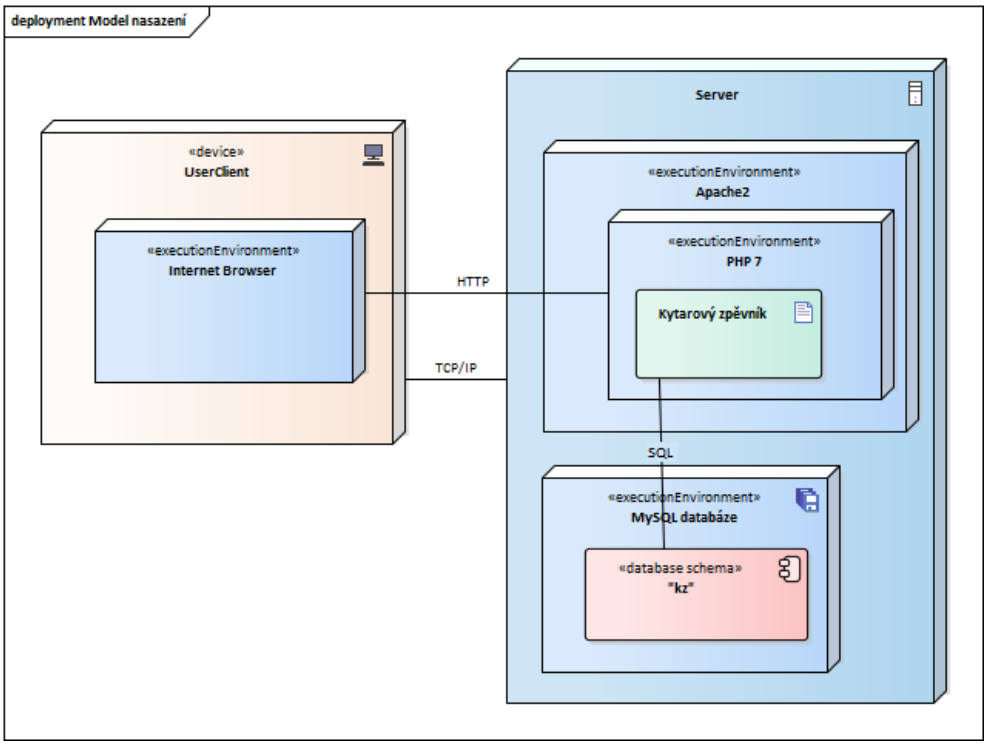
Tato část aplikace sestává z pěti různých druhů komponent, které spolu komunikují podle velmi volných pravidel. *Service* třídy jsou takové, které se starají o přímou komunikaci s backendem. Jsou využívány *Resources* třídami, které obsahují funkce mapující všechny dostupné přístupové body API. *Controllers* představují každý jednu obrazovku Kytarového zpěvníku. Pro získávání dat využívají vztah se *Services* a *Resources*, získanými daty poté naplňují instance *Models* – tříd, které zrcadlí většinu entit používaných v BE (backendové) databázi. Poslední třídou komponent jsou *Views*, které za pomoci kombinace HTML a Dart zobrazují informace dosazené z *Controllers*.

2.2.2 Model nasazení

Pro utvoření konkrétnější představy o aplikaci jsem se rozhodl ještě přidat model nasazení 2.7. Účelem tohoto modelu je zobrazit fyzickou architekturu Kytarového zpěvníku a technologie, které jsou k běhu systému nutné. Krom doposud zmíněných technologií je zde nově zmíněný server *Apache*[16] (vhodná je třeba distribuce *XAMPP*[17]), který byl při vývoji využíván naším týmem.

2.2.3 Co je špatně

Nyní by už měl být čtenář této práce dostatečně s Kytarovým zpěvníkem seznámen. Některé z neduhů, které tento projekt doprovází, jsem už zmínil, rád bych je ale uvedl pro větší přehlednost ještě jednou pohromadě. Dle mého názoru je s aktuální aplikací špatně následující:



Obrázek 2.7: Model nasazení

- Zastaralé/neperspektivní technologie
  - Pokud má aplikace někdy mít reálnou šanci na trhu vůči konkurenci, určitě nemůže sázet na až čtyři roky staré technologie. Naopak potřebuje něco moderního, co má zároveň vyhlídky se do budoucna nadále vyvíjet.
  - Práce s méně známými technologiemi znamená i méně dokumentace a online pomoci, která je pro programátora dostupná. V době psaní této bakalářské práce je na *StackOverflow*[18] 7902 otázek na téma Dart a 1189770 otázek na téma PHP.
- Několik různých programovacích jazyků
  - Na programátory pracující na aplikaci jsou kladeny velké nároky (znalost více rozsáhlých jazyků) a v případě rozdělení na FE a BE programátory je nutná jejich neustálá spolupráce při sebemenší změně. Taková forma spolupráce proces značně zpomaluje.
- Dvě oddělené aplikace

  - Jediným praktickým důvodem, proč by měl být backend Kytarového zpěvníku samostatnou aplikací, je v současnosti případ, kdy by měla běžet mobilní aplikace a zároveň by nebyl nikde spuštěný frontend. Protože ale Android aplikace neumožňuje vkládat nový obsah, bez dostupného frontendu je de facto nepoužitelná.
  - Oddělené aplikace dělají zmatek nejen v Git repozitářích (jedná se totiž o dva rozdělené projekty a je nutné je vždy synchronizovat, aby vše fungovalo; při vracení se na commity z minulosti je náročné dohledat, jaká je správná verze druhé poloviny aplikace), ale také velmi znesnadňují nalézání a opravování chyb.

- Neefektivní a uživatelsky nepohodlný frontend
  - Tato skutečnost má nejspíš na svědomí to, že aplikace obecně není příliš rychlá. Pokud se navíc požadavek na BE nepodaří, FE o tom uživatele většinou nijak neupozorní a pouze donekonečna zobrazuje ikonku načítání.

- Nevhodná architektura
  - Nezanedbatelná část tříd v aplikaci se nedrží žádného známého ověřeného návrhového vzoru, i když backendová část má už velmi blízko k MVP modelu.

Jak je už patrné z názvu této práce, vhodné řešení pro nápravu těchto nedostatků jsem už spolu s vedoucím práce vybral. Protože hlavním kamenem úrazu je frontend napsaný v jazyku Dart a při přepisování aplikace je žádoucí využít co nejvíce již napsaného kódu, přepracuji aplikaci do MVP architektury, která je pro Nette framework charakteristická.

Model a *Service* třídy z backendu budou z velké částí zachovány, jen rozšířeny o nové funkce a zefektivněny. Přidám *Presenters* a *Views*, které poslouží jako frontend nové verze aplikace. Jednotlivé *Resources* též zůstanou zachovány a tak bude i nadále fungovat API vyžadováno mobilní aplikací.

2.3 Specifikace nových požadavků

Přepracování celé aplikace je i dobrou příležitostí pro opravení funkcí, které nefungují ideálně, a nebo pro dodání nových, které bude díky změnám architektury snažší naprogramovat. Na základě požadavků vedoucího práce a vlastní iniciativy v aplikaci navíc provedu následující změny:

- Odebrání redundantních částí
  - Ať už se jedná o nepoužité části kódu a nebo třeba databázové tabulky, které nejsou nikde využity, tak do výsledné nové verze

aplikace nepatří. Takový obsah zbytečně aplikaci znepřehledňuje a mohl by způsobit zmatení týmům, které budou v budoucnu na aplikaci pracovat.

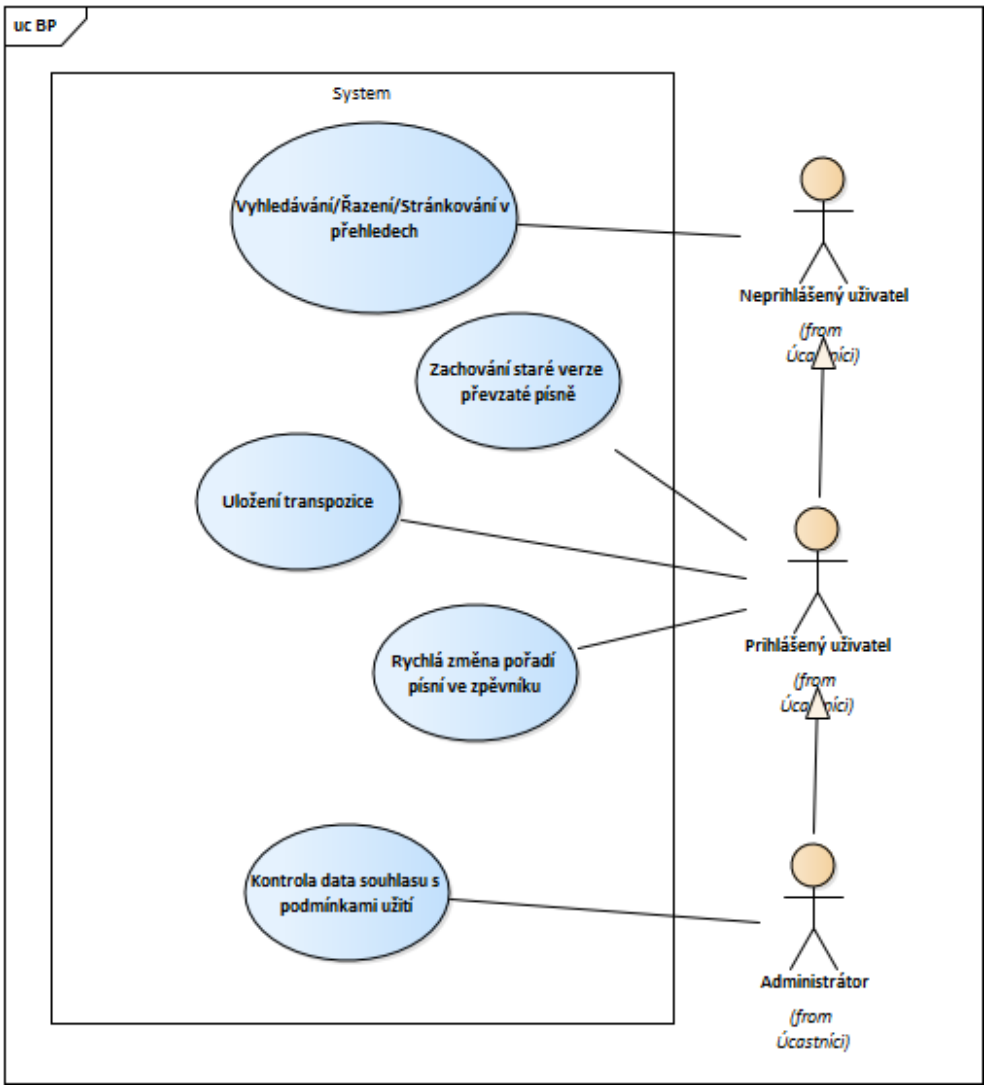
- Aktualizace na nejnovější verzi PHP a aktualizace použitých knihoven
- Zpřísnění pravidel pro komunikaci mezi komponentami
  - Některé *Services* doposud využívají přímého přístupu k *EntityManager* třídě místo *DAO* tříd a jsou tak závislé na implementaci datové vrstvy.
- Zavedení coding standards
  - Na konci poslední iterace BI-SP2 jsme zavedli příručku pravidel pro formátování kódu. Byl bych rád, aby se jí na konci mé práce řídila většina aplikace.
- Dodělání převzetí
  - Podle uživatelské příručky dovoluje převzetí písně v případě její změny uchovat si kopii předešlého stavu. Aktuální verze aplikace to ale ve skutečnosti neumožňuje, i když je na to částečně připravená.
- Zlepšení přehledů
  - Aktuální přehledy písní a zpěvníků nedovolují vyhledávání a nepodporují žádné stránkování. Pouze se z backendu vždy načte dalších deset řádků a zobrazovaná tabulka se zvětší. Takové řešení je uživatelsky velmi nepohodlné. Výsledný přehled bude mít možnost stránkování, řazení a vyhledávání a při přechodu na vyhledanou píseň a následném návratu by se měl nacházet ve stejném stavu.
- Rychlé uložení transpozice
  - Přestože je možné si nechat akordy dynamicky transponovat, tak pokud chce vlastník změnit akordy celé písně trvale, musí je jeden po druhém manuálně přepsat. Vlastník písně by měl disponovat tlačítkem, které umožní aktuálně vybranou transpozicí nahradit všechny uložené akordy.
- Podmínky užívání a souhlas uživatelů
  - Vzhledem k tomu, že aplikace pomalu míří do živého provozu, je z právního hlediska důležité vědět, s čím konkrétně uživatel při registraci souhlasil. Podmínky používání se ale můžou časem měnit.

  - Po domluvě s vedoucím práce bylo rozhodnuto, že zatím stačí implementace v databázi. Do databáze bude možné vkládat smluvní podmínky ve formě HTML a nejnovější budou vždy zobrazeny na webu. U uživatele bude přitom sledováno datum registrace a datum posledního souhlasu s podmínkami (prozatím budou vždy shodné). Na základě těchto údajů bude vždy možné dohledat, kdo s čím přesně souhlasil.

- Změny detailu zpěvníku
  - Písně v detailu zpěvníku by měly být zobrazeny ve stejném formátu jako normální přehled písní se všemi dostupnými funkcemi (řazení, vyhledávání, stránkování). Majitel zpěvníku by měl mít zároveň k dispozici tlačítko „uložit pořadí“, které mu dovolí okamžitě změnit pořadí písní na to, jaké je aktuálně zobrazené. Změnit pořadí všech písní ve zpěvníků tak, ať jsou například řazeny podle alba, bude otázkou jen pár kliknutí.

- Ještě pár drobných změn na závěr:
  - Přidání vyhledávání na úvodní stránku
  - Nejnovější písně a zpěvníky zobrazovat na úvodní stránce
  - Při nutnosti přihlášení vracet poté uživatele na původní stránku
  - Zlepšení efektivity časově náročných funkcí (díky Nette debuggeru bude mnohem lehčí zjistit, které části aplikace jsou pomalé)

Pro lepší představu toho, kterých uživatelů se změny budou týkat, zde uvádím diagram případů užití 2.8 těch nejdůležitějších změn.



Obrázek 2.8: Případy užití

Kapitola 3

Návrh

Po přečtení kapitoly zabývající se analýzou by už čtenář měl mít ucelenou představu o tom, jaký Kytarový zpěvník byl (z pohledu nabízených možností a použitého řešení) a co všechno na něm bylo z mého pohledu špatně. Tato kapitola pojednává o návrhu nové verze a postupně v ní odprezentuji za pomoci vhodně zvolených diagramů to, jak jsem k řešení daných problémů přistupoval.

Nejprve uvedu, jaký je konečný stav architektury a popíšu, z jakých různých komponent aplikace ve verzi 2.0 sestává. Pak přijde řada na databázový model, kde zmíním rozdíly oproti doménovému modelu z předchozí kapitoly a provedené úpravy tabulek v databázi.

Pro lepší pochopení záměrů poté na závěr kapitoly přejdu od obecného ke konkrétnímu a za pomoci třídního modelu a sekvenčních diagramů uvedu jeden příklad přímo z výsledné aplikace, na kterém budou jasně demonstrované principy nově zavedené architektury.

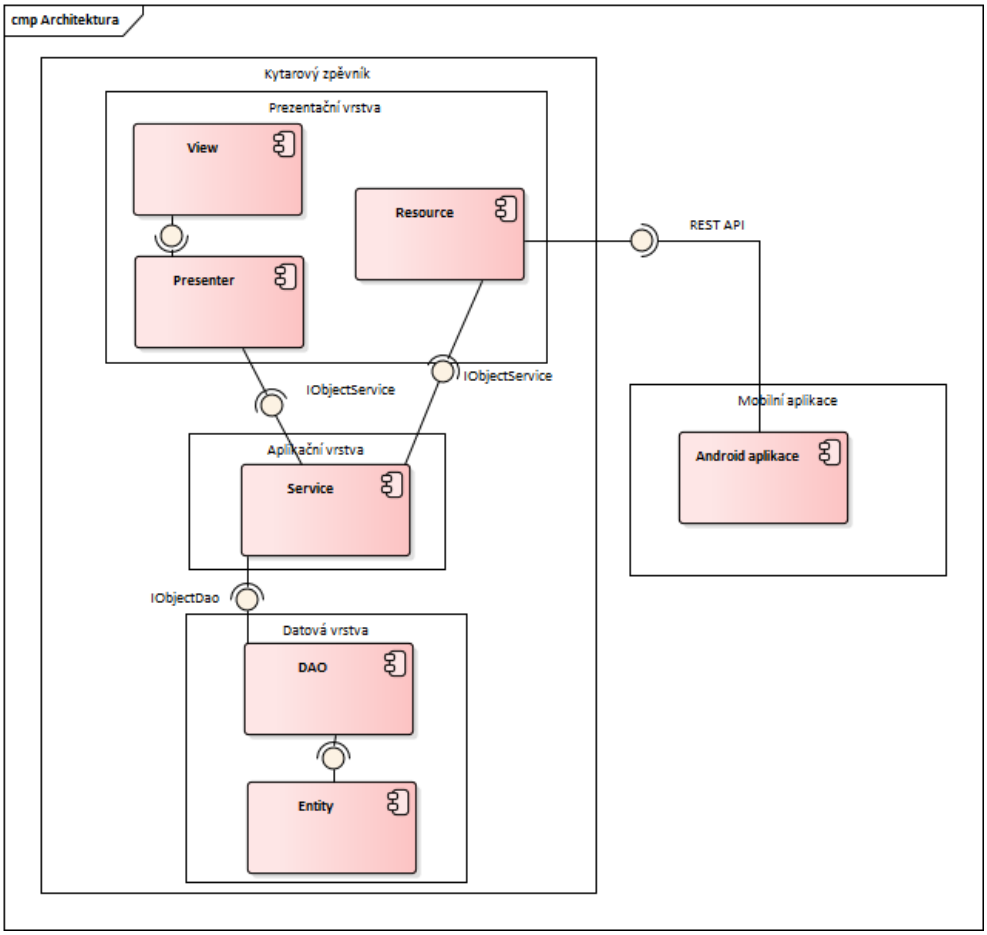
Přestože zamýšlené změny jsou docela rozsáhlé, nový návrh stále vychází z předchozí aplikace a dříve používaných technologií. Klíčová byla snaha o zachování co největší části funkčního kódu, ať není nutné aplikaci vystavět celou od základu. Ze stejného důvodu uvádím v této kapitole také často obrázky, které vycházejí z již existujících diagramů vytvořených v minulosti při práci na projektu.

3.1 Nová architektura

Obrázek 3.1 reprezentuje architekturu komponent, která mi vzhledem k problémům zmíněným v kapitole Analýza přišla nejvhodnější. Při jejím použití se zachovalo největší množství kódu a zároveň se celá aplikace dostala do formálně správného stavu. Nyní je konečně striktně dodržena třívrstvá architektura – třídy v prezentační vrstvě se starají o uživatelský vstup a prezentaci výsledků, třídy v aplikační vrstvě vykonávají centrální logiku a operace na datech v kombinaci se vstupy, a datová vrstva se stará o správu dat a jejich ukládání v databázi. Závislosti mezi jednotlivými vrstvami jsou vyřešeny

pomocí proměnných vyžadujících konkrétní rozhraní (všechny interface třídy mají prefix „I“). Konkrétní instance třídy je pak dosazena frameworkem na základě tříd definovaných v konfiguračním souboru „config.neon“.

Ještě je možné se na aplikaci podívat z pohledu MVP architektury. V takovém případě pod *Views* spadají stejnojmenné třídy, pod *Presenters Presenter* třídy (ale navíc i *Resource třídy*) a všechny ostatní části aplikace jsou označovány jako *Model*.



Obrázek 3.1: Diagram komponent finální aplikace

3.1.1 Entity

Tyto třídy reprezentují objekty nacházející se v Kytarovém zpěvníku. Obsahují jejich definici, přístupové metody a specifikují, jak se má konkrétní třída namapovat na tabulky uvnitř databáze. *Entity* třída sama o sobě s jinou komponentou aplikace nepracuje, naopak je využívána napříč celým zbytkem

serverové aplikace. *Entity* třídy jsou ovšem vytvářeny pouze *DAO* třídami (jak je znázorněno na diagramu), které jejich instance posléze předávají dál jako návratovou hodnotu.

3.1.2 DAO

Jako jediná z komponent se *DAO* třídy starají o manipulaci s vybraným úložištěm a představují tedy datovou vrstvu aplikace. Jejich metody jsou často podobné, ale ne vždy – implementovány jsou pouze ty, které byly nutné pro chod aplikace. Jedinými *DAO* třídami přítomnými v aplikaci jsou třídy používající ORM (Object Relational Mapper) knihovnu *Doctrine*.

3.1.3 Service

Services jsou aplikační vrstvou Kytarového zpěvníku. Veškerá důležitá logika a zajímavé operace se dějí uvnitř nich. Pokud *Service* potřebuje přístup k datům, získá je z příslušné *DAO* třídy, závislý je ale pouze na jejím rozhraní – na implementaci nezáleží, protože v třívrstvé architektuře jsou vrstvy dokonale rozdělené. Jsou využívány *Resources* a *Presenters*, i když samotné *Services* o jejich existenci neví.

3.1.4 Resource

Jedná se o zachované třídy ze staré aplikace, které představují endpoints pro přístup do Kytarového zpěvníku pomocí API. Třídy podstoupily pouze malé změny, aby byla striktně dodržena navržená architektura. Slouží jen pro práci s příchozími a odchozími požadavky, pro jakékoli náročnější operace využívají příslušných *Services*. Jedná se o část prezentační vrstvy (i když data prezentuje pouze v *JSON* formátu).

3.1.5 Presenter

Tyto třídy tvoří spolu s *Views* zbytek prezentační vrstvy aplikace. Starají se o vytvoření komponent nutných pro zobrazení webové stránky, naplňují *Views* potřebnými daty získanými ze *Services* a zároveň zpracovávají veškerý vstup uživatele, který je z *View* delegován právě do *Presenters* a z nich následně do *Services*.

3.1.6 Views

V Kytarovém zpěvníku jsou *Views* zastoupeny *Latte* šablonami, ze kterých se generuje výsledná HTML stránka, která je zobrazena uživateli. Potřebná data získávají z přidružené *Presenter* třídy, které zároveň posílají valnou většinu uživatelských vstupů. Některé minoritní *frontend-only* uživatelské vstupy, jako

je například transpozice akordů, jsou rovnou odchyceny a zpracovány za pomoci javascriptu.

3.2 Databázový model

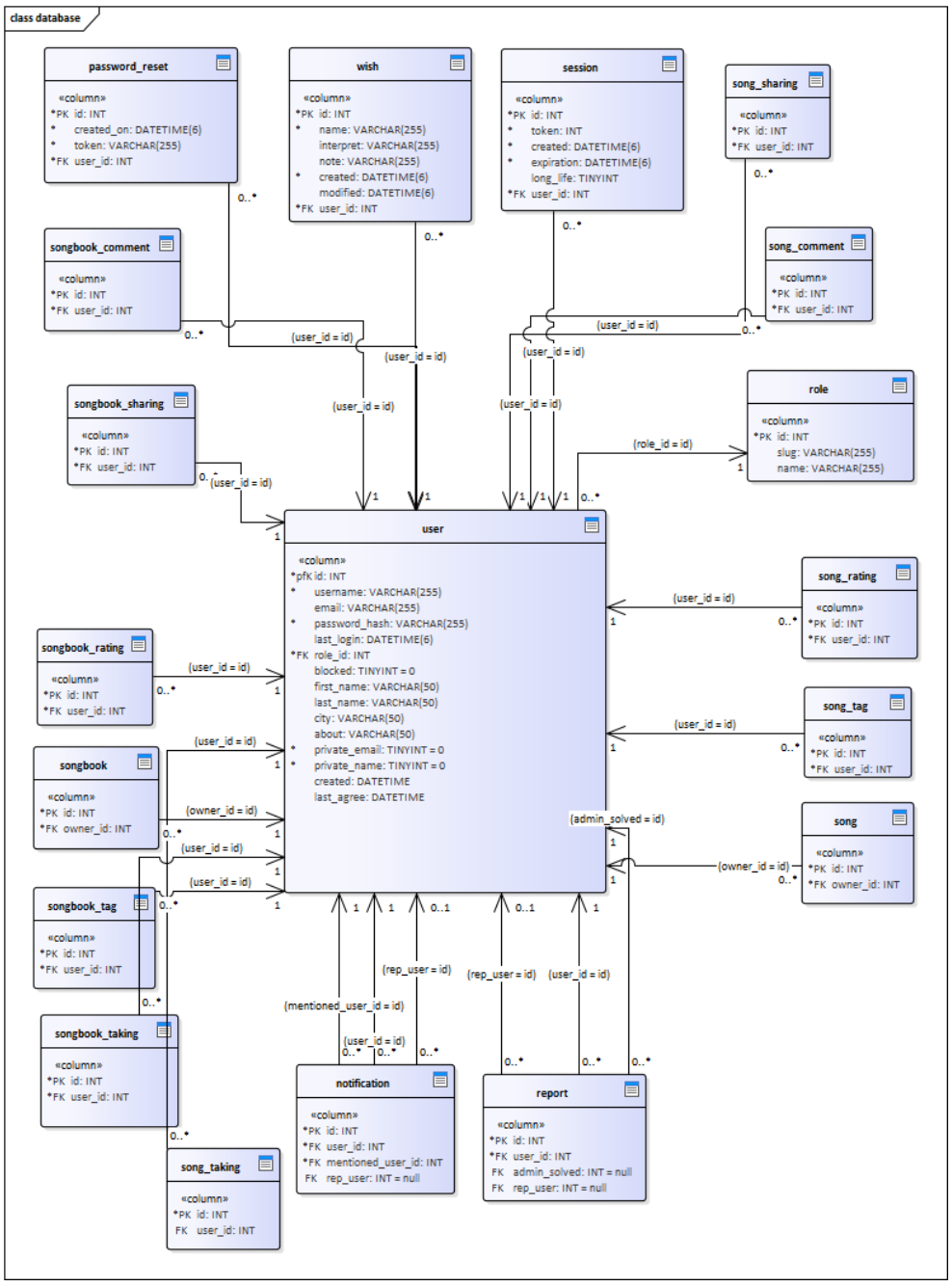
Na obrázcích 3.2, 3.3 a 3.4 je databázový model, znázorněný pomocí UML třídních diagramů. Na rozdíl od doménového modelu obsahuje informace o konkrétní implementaci, vztazích, atributech a klíčích, které tvoří databázi aplikace. Zobrazená verze reflektuje stav, v jakém se struktura nachází po provedení všech zamýšlených změn. Model slouží pro lepší orientaci programátorů ve struktuře aplikace a také jednotlivé tabulky vlastně odpovídají *Entity* třídám, které se uvnitř Kytarového zpěvníku nacházejí.

Valná většina tabulek je samovysvětlujících a přímo odpovídá třídám z doménového modelu, proto je zde nebudu dopodrobna zmiňovat. Protože změn bylo provedeno minimum, pouze jsem upravil již existující diagram. Provedené změny jsou následující:

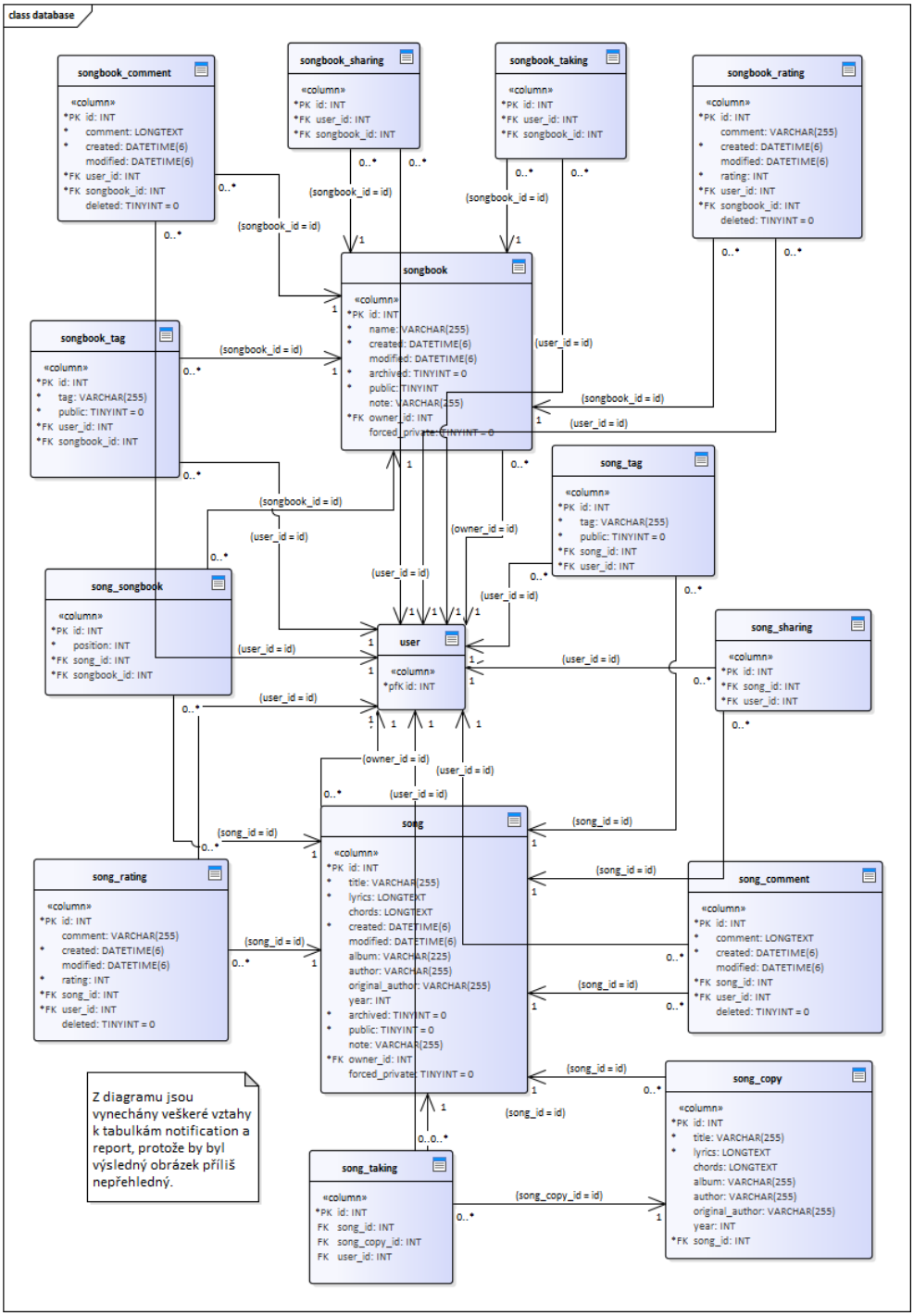
- Byla odebrána tabulka `user_settings`, která nebyla v databázi nijak využívána.
- Byl odebrán sloupec `display_name` v tabulce `user`, který nebyl využíván.
- Byla vyrobena tabulka `terms`, ve které se ukládají aktuální podmínky používání.
- Z důvodu zjištění data souhlasu s podmínkami (a dohledání odpovídajícího znění) byly do tabulky `user` přidány sloupce `created` a `last_agree`.

Dále za zmínku stojí tabulka `session`, která se v doménovém modelu nenacházela, protože s problémovou doménou nesouvisí. V tabulce jsou ukládané vygenerované tokeny nutné pro přístup do aplikace pomocí API. Krom tokenu obsahuje každý řádek i datum jeho vytvoření a datum konce platnosti.

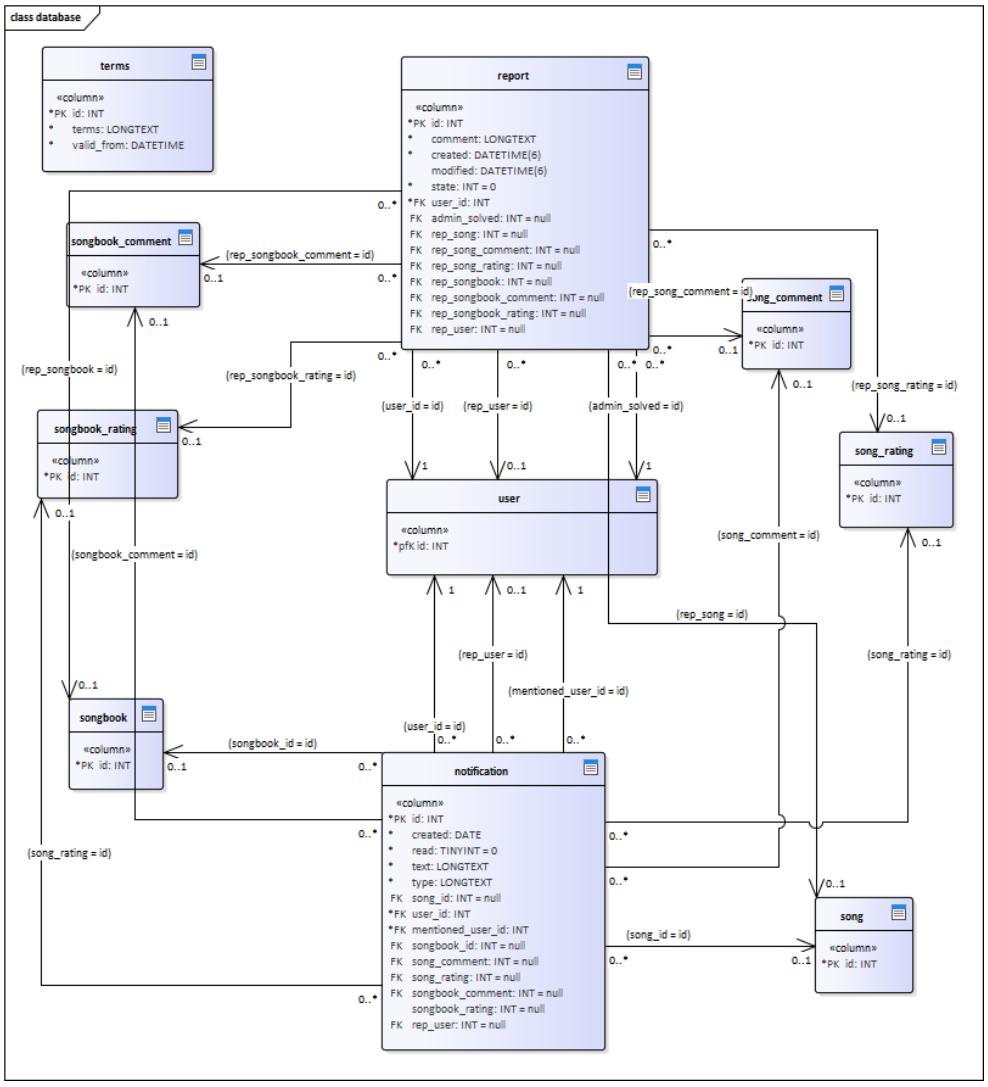
Pro větší přehlednost byl databázový model rozdělen do tří různých diagramů. 3.2 zobrazuje třídy s vazbami na tabulku `user`. Ty třídy, které budou zobrazeny důkladněji později, jsou reprezentovány pouze svým `id` a vztahem k tabulce uživatelů. Diagram 3.3 je zaměřený na třídy `Song` a `Songbook` a všechny tabulky, které jsou s nimi v relaci. Poslední 3.4 pak obsahuje notifikace, nahlášení a podmínky používání. Kompletní databázový model je k dispozici v elektronické podobě na přiloženém médiu.



Obrázek 3.2: Model databáze se zaměřením na třídu `User`



Obrázek 3.3: Model databáze se zaměřením na třídy `Song` a `Songbook`



Obrázek 3.4: Model databáze se zaměřením na třídy `Report`, `Terms` a `Notification`

3.3 Třídní model

Diagramy tříd jsou v praxi nejčastěji používaným typem UML diagramů[19] – zobrazují typy objektů implementovaných v systému a jejich vzájemné vztahy. Na obrázcích 3.5, 3.6 a 3.7 jsou zobrazeny diagramy tříd se zaměřením na třídu `RegisterPresenter`. Méně podstatné třídy, jako je třeba trojice *Service* tříd u `BasePresenter` nejsou uvedeny v jejich plném rozsahu, protože nejsou nutné pro demonstraci principů přiřazení zodpovědnosti. Toto se týká i jednotlivých metod rozhraní, které též nejsou zobrazeny v plné míře. `RegisterPresenter`, který je centrálním bodem diagramů, se stará o zobrazení a zpracování registračního formuláře.

Valná většina tříd v Kytarovém zpěvníku rozšiřuje některý ze základních objektů Nette frameworku. Tyto objekty nejsou v diagramech zobrazeny, protože jeho cílem je přiblížit to, jaké řešení využívá aplikace, nikoli Nette knihovny.

Občas je u návratových typů v diagramu použit otazník před jménem typu (např. `?User`). Takový způsob zápisu vychází z PHP 7 a symbolizuje, že návratovou hodnotou krom specifikovaného objektu může být i `null`.

3.3.1 BasePresenter

Rodič všech *Presenters* v aplikaci, ve kterém se nachází často používané obecné metody, obsahuje tři privátní *Service* třídy, které jsou netradičně vloženy pomocí metody `injectBase()`. Metoda `startup()` ještě před provedením akce potomka ověřuje, jestli má uživatel pro provedení dané akce dostatečné oprávnění, a také se stará o označování notifikací za přečtené (protože něco takového je možné učinit z jakéhokoli místa v aplikaci). Metoda `beforeRender()` pak vkládá do šablony informace o přihlášeném uživateli a odpovídající instanci.

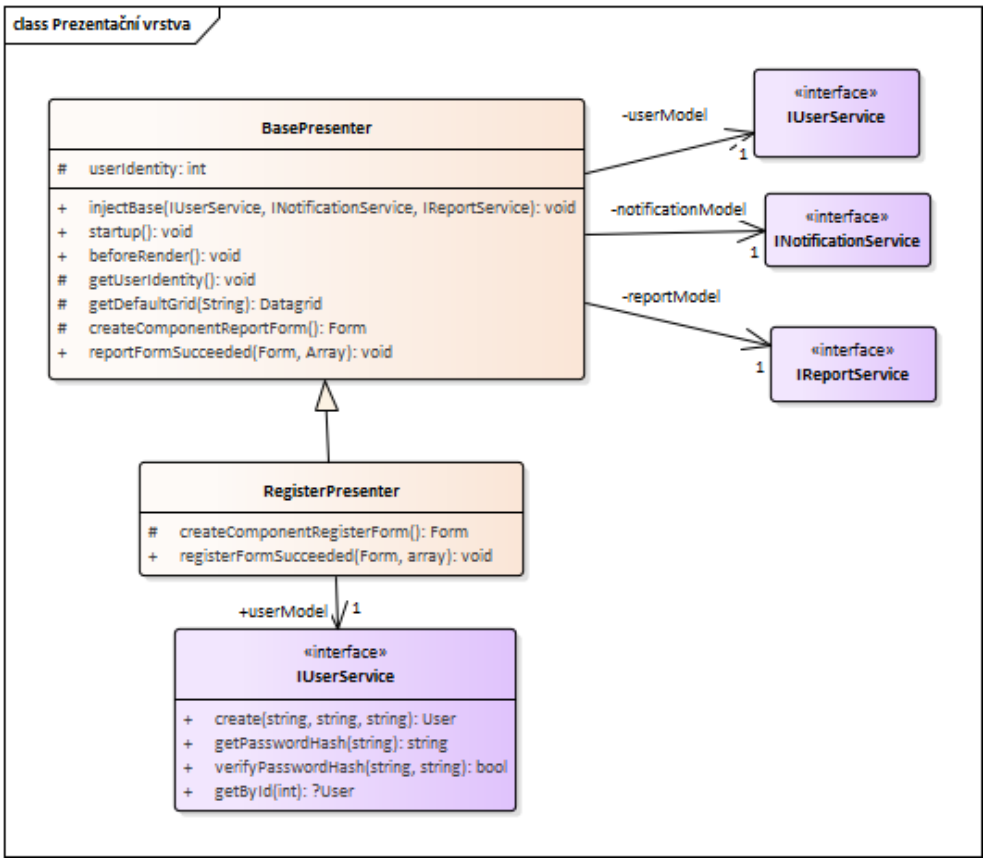
Další obsažené metody slouží k výrobě základní tabulky s českým překladem (metoda `getDefaultGrid()`) a k výrobě a zpracování formuláře nahlášení nevhodného obsahu, který se v aplikaci vyskytuje na více místech, proto jsem se jeho zpracování rozhodl umístit přímo sem.

3.3.2 RegisterPresenter

Tato třída rozšiřuje `BasePresenter` a vlastní jediný *Service* a dvě metody. Ty slouží k výrobě a zpracování formuláře, který se v šabloně registrace nachází. Proměnná `userModel` je veřejná, protože se jedná o předpoklad pro její úspěšné automatické naplnění pomocí Nette DI (dependency injection) na základě specifikovaného rozhraní.

3.3.3 UserService

Presenter volá patřičnou metodu *Service* třídy. Veškerá aplikační logika se potom děje přímo zde – konkrétně v tomto případě v metodě `create()`. Pro

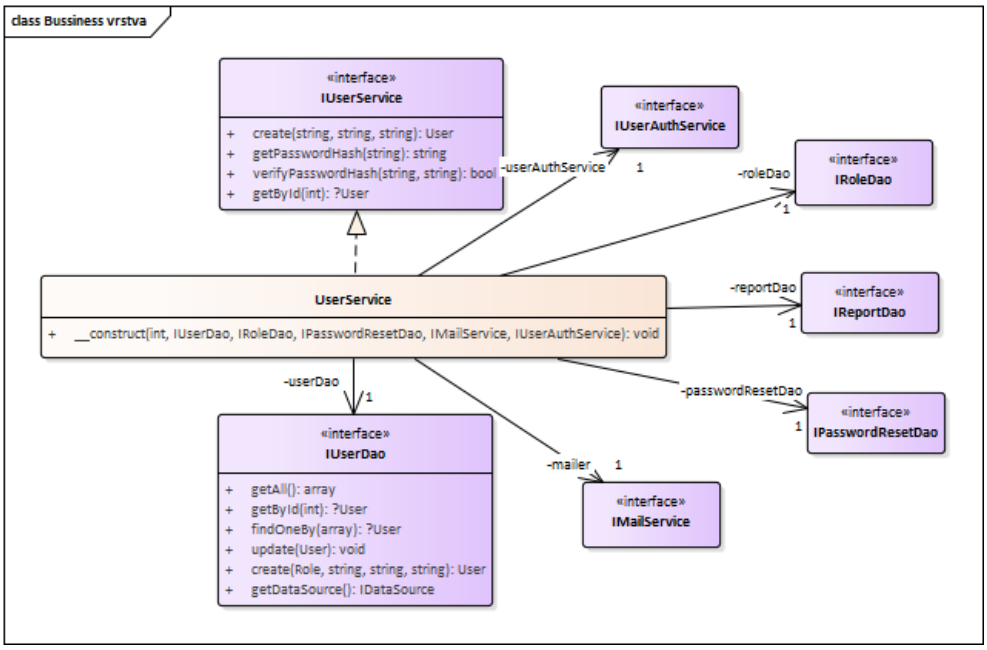


Obrázek 3.5: Třídní model se zaměřením na prezentační vrstu

operace související s ukládáním dat pak poslouží *Service* třídě vložená *DAO* třída. Typem atributu je pouze třída implementující `IUserDao` rozhraní, o její vložení se opět postará DI.

3.3.4 UserDao

Všechny *DAO* třídy v aplikaci slouží aktuálně jen k manipulaci s databází v kombinaci s *Doctrine* knihovnou, protože ale všechny mají specifikovaný svůj vlastní interface, není problém je nahradit jiným způsobem správy dat, pokud stanovené rozhraní bude dodrženo. Každé *DAO* obdrží v konstruktoru třídu *EntityManager*, pomocí které získávání a manipulace s daty probíhá. *DAO* třídy jsou jediné v celé aplikaci, které ví o *Doctrine*.



Obrázek 3.6: Třídní model se zaměřením na aplikační vrstu

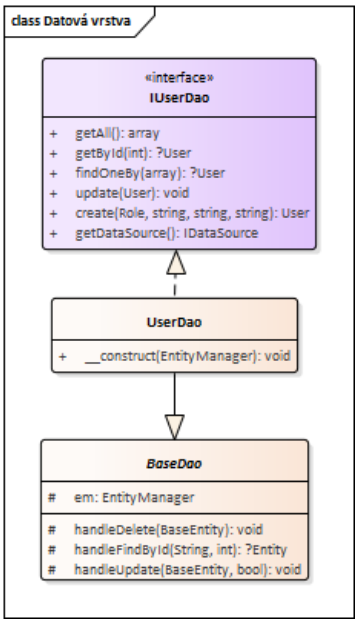
3.3.5 BaseDao

Protože je několik metod uvnitř *DAO* tříd podobných, byly vyčleněny do abstraktní třídy `BaseDao`, která se stala rodičem všech ostatních *DAO*. Protože ale podoba neplatí vždy a například u update operace je vhodné specifikovat přímo konkrétní *Entity* třídu na vstupu, nejsou metody používané potomky přímo. Místo toho mají prefix „handle“ a volají se z `update()`, `delete()` a `findById()` metod konkrétních *DAO* tříd.

3.4 Model komunikace

Aby bylo předvedení architektury kompletní, předkládám ještě komunikační model 3.8 a 3.9, znázorněný pomocí UML sekvenčního diagramu. Je na něm zachycen stejný případ užití jako u třídního modelu – „Registrace nového uživatele“. Protože je centrálním bodem diagramu právě tento proces, nejsou dopodrobna rozepsána volání funkcí, která s ním nesouvisí a nebo nejsou součástí implementovaného řešení, ale například nějaké knihovny. Průběh procesu je následující:

1. Uživatel v šabloně do příslušného formuláře vyplní uživatelské jméno, e-mailovou adresu a heslo (pro kontrolu dvakrát). Poté může formulář odeslat.



Obrázek 3.7: Třídní model se zaměřením na datovou vrstu

2. Šablona zpracování formuláře deleguje do `registerFormSucceeded()` metody patřící *Presenter* třídě, ve které se nejprve zjistí, jestli jsou správná data přijatá z formuláře – tj. jestli byl zaškrtnut souhlas s podmínkami užití a jestli se shodují vložená hesla. Pokud ano, jsou data předána dále do `UserService` metodě `create()`.

3. `UserService` nejprve pomocí dvou dotazů na *DAO* třídu zjistí, jestli se použité uživatelské jméno a nebo e-mail v systému již nenachází. Pokud ano, způsobí vyhození výjimky `DuplicateUsernameException` nebo `DuplicateEmailException`, kterou *Presenter* odchytí a způsobí znovunačtení stránky s patřičnou chybovou hláškou.

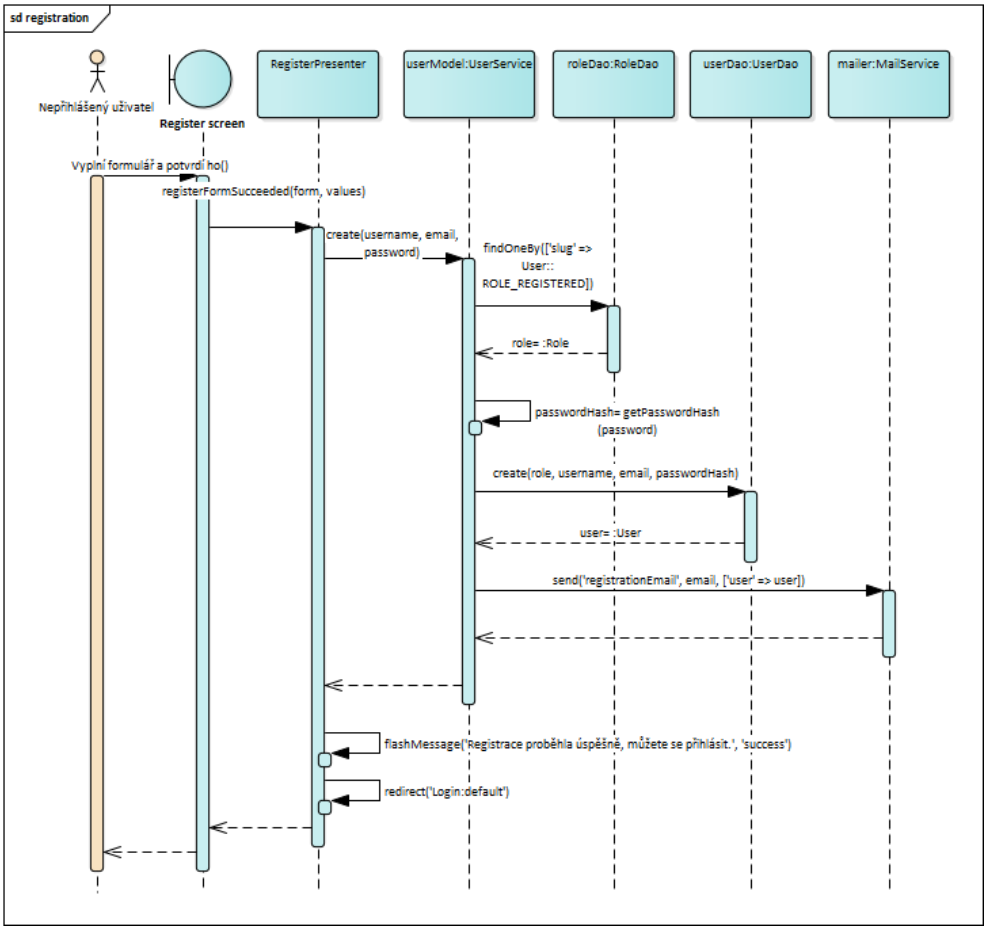
4. Pokud jsou uživatelské jméno i e-mail unikátní, získá servis pomocí `RoleDao` roli reprezentující registrovaného uživatele a také pomocí vlastní metody `getPasswordHash()` zahashuje uživatelem zvolené heslo. Obě tyto proměnné spolu s uživatelským jménem a e-mailem pošle do metody `create()` patřící `UserDao` třídě.

5. V `UserDao` je vyrobena nová `User` třída, je naplněna obdrženými daty a pomocí *EntityManager* zanesena do databáze. Poté je tato nová entita vrácena zpět do `UserService`.

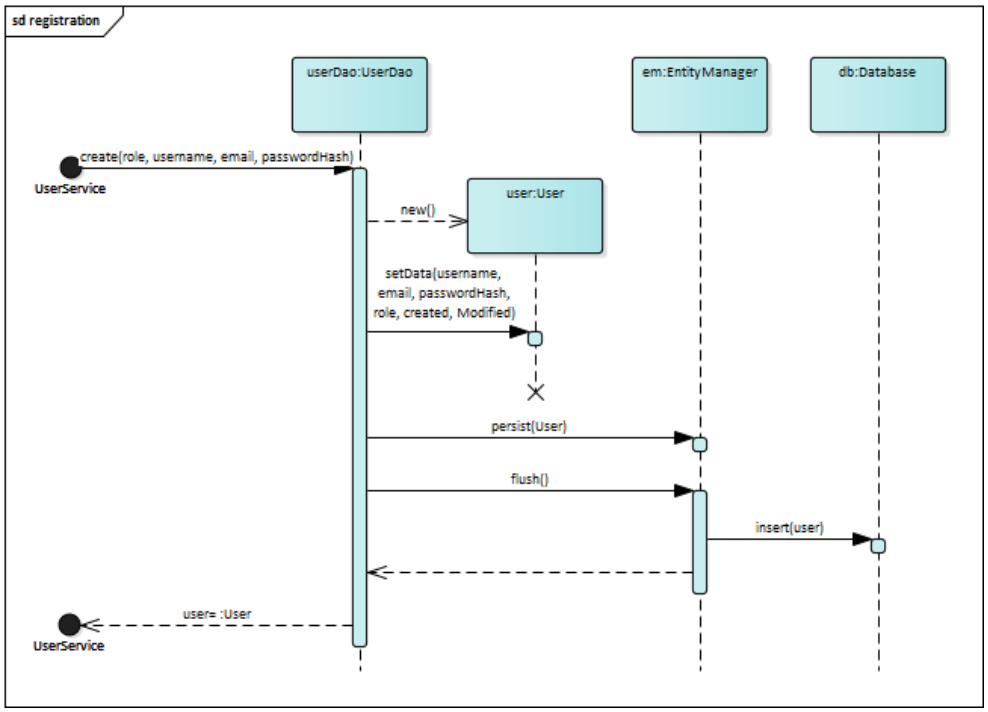
6. `UserService` zašle uživateli pomocí `MailService` mail informující ho o úspěšné registraci. Typ mailu je pevně daný a do jeho šablony je

dosazen nově vytvořený uživatel. Po odeslání zprávy je funkce `create()` ukončena a běh programu se vrátí zpět do `RegistrationPresenter`.

7. V něm je nejprve nastavena vhodná `flashMessage`, která uživateli oznámí úspěch provedené akce, a potom je uživatel přesměrován na akci default do `LoginPresenter`, aby se mohl rovnou přihlásit. Šablona se překreslí, kontrolu obdrží jiný *Presenter*, a proces registrace je zdárně ukončen.



Obrázek 3.8: Sekvenční diagram znázorňující registraci nového uživatele



Obrázek 3.9: Sekvenční diagram znázorňující registraci nového uživatele, zaměření na `UserDao`

Kapitola 4

Realizace

Poslední fází přepracování Kytarového zpěvníku byla realizace kroků, které byly analyzovány a navrhnuty v předchozích dvou kapitolách mé bakalářské práce. Nejprve zde krátce zmíním nové použité technologie, které jsem nakonec pro projekt zvolil. Potom se dopodrobna rozepíšu o všech provedených krocích, které tvořily realizaci, a také zmíním, jak byly nakonec vyřešeny dodatečné požadavky na nové funkce.

Poté uvedu konkrétní zajímavé situace, které jsem při programování aplikace musel řešit, a nakonec se budu věnovat dalším náležitostem, které k hotové aplikaci patří – testování a příručkám.

4.1 Použité technologie

Protože byla při provádění změn většina backendu zachována, nebyl ani důvod měnit stávající technologie. Naopak díky přechodu na PHP 7 mohly být všechny konečně nahrazeny svými nejnovějšími verzemi. Nejvýraznější novou technologií je použití šablonovacího jazyka *Latte*, který je důležitou součástí Nette frameworku. Latte umožňuje elegantní propojení HTML a PHP kódu a dovoluje tak lehce napsat zabezpečené a pěkné *Views*. Další součástí frontendu je nástroj pro správu assetů *Webpack*, pomocí kterého jsou do jednotlivých *Views* efektivně vkládány správné soubory s javascriptem a styly. Více o těchto nástrojích je zmíněno v paralelní bakalářské práci Nikolaie Baranova.

Pár novinek ale přece jen přibylo, i když se jedná převážně o knihovny sloužící ke zlepšení pohodlí programátora. Jedinou nutností na základě nových požadavků byla *Ublaboo/Datagrid* knihovna, která slouží k vytváření inteligentní tabulky zobrazující data, a která ještě bude zmíněna v pozdější sekci. Dále byla pro lepší možnosti testování přidána knihovna *Mockery*, která umožňuje vytvářet objekty na základě zadaného rozhraní, takže můžou nahradit závislosti testované třídy.

Poslední skupinou nových knihoven tvoří knihovny pro kontrolu kvality kódu. *PhpStan* slouží pro statickou analýzu PHP kódu podle obecně sta-

novených konvencí. *Simplify/Easy-coding-standard* a *Nette/Coding-standard* zase slouží pro dodržování stylu kódu stanoveného Nette frameworkem. Výhodou je to, že tyto dvě knihovny dokážou nalezené chyby ve většině případů i automaticky opravit. Všechny způsoby analýzy kódu se spouštějí pomocí příslušných Composer skriptů a trvají maximálně pár desítek vteřin.

4.2 Kroky potřebné pro přechod na MVP architekturu

Protože jsem prováděl v aplikaci vcelku rozsáhlé změny, rozepsal jsem si celý postup do několika po sobě jdoucích kroků. Jejich pořadí bylo stanoveno na základě mých osobních zkušeností a úsudku. Výchozím stavem byl PHP backend, frontendovou část s Dartem bylo možné úplně opustit.

1. Aktualizace technologií
    - Update používané verze PHP na 7.1.7 místo 5.6.
    - Díky tomu bylo možné aktualizovat i používané knihovny na jejich nejnovější verze, včetně samotného Nette frameworku. Protože tento krok způsobil spoustu *warning* zpráv, byly dočasně potlačeny pomocí `error_reporting( ~E_USER_DEPRECATED)` (než došlo k aktualizaci všech součástí aplikace) a později byly opraveny.
    - Update souboru *.htaccess*, který určuje možnosti HTTP přístupu do složek projektu. Nově mohou uživatelé jednotlivé části bývalého backendu navštěvovat přímo a tak bylo nutné zabezpečit, aby se nedostali někam, kam nemají.
    - Upravení Nette config souborů, protože s přechodem na novou verzi se změnila forma zápisu některých parametrů.
2. Úklid projektu a formátování
    - Smazání nepotřebných souborů, které se ve složce nacházely. Jednalo se například o různé testy neexistujících částí aplikace, zastaralé SQL dump soubory a různé nepoužívané konfigurační soubory.
    - Přejmenování API složky z implementation1 na implementation, protože pro jiný název není žádný důvod – další verze implementace se v dohledné době neplánuje.
    - Odstranění nepoužívaného/zakomentovaného kódu a atributů/entit.
    - Přejmenování jmenného prostoru napříč celou aplikací z `App` na `KZ`. Původně používaný jmenný prostor byl příliš obecný.
    - Konverze všech použitých čtyřmezer na tabulátory, protože jen tak je to správně (tady je prostor pro hádku programátorů, ale nyní vedu vývoj já).

- Instalace nových knihoven pro statickou kontrolu formátu kódu a dodržování Nette coding standards. Opravení existujícího kódu tak, aby dodržoval nově stanovená pravidla.
- Odstranění API přístupových bodů, které už nebudou nutné. Například generování do PDF mobilní aplikace nepoužívá, tak nebylo potřeba mít ve webové aplikaci nadále třídu `PDFResource`, poskytující daný API endpoint.
- Opravení existujících *Service* tříd tak, aby žádná z nich nepřistupovala přímo do databáze, ale používala *DAO* třídy. U některých částí aplikace už oprava proběhla při posledním běhu BI-SP2, ne však u všech.
- Vyžadovat ve všech konstruktorech interfaces místo konkrétních tříd, pokud je to možné. Stanovit u metod aplikační a datové vrstvy typy vstupních parametrů a návratových hodnot za pomoci nových možností PHP 7.

3. Tvorba nové struktury, vytvoření základu MVP

   - Vytvoření nové struktury složek, která je přehlednější a lépe reflektuje, které třídy spolu souvisí.
   - Vyrobení nového routeru, který přesměrovává HTTP požadavky na jednotlivé *Presenters*, zároveň ale nenarušuje funkčnost požadavků na API.
   - Přidání `BasePresenter` s obecnými funkcemi pro ověření identity uživatele a dalších specializovaných *Presenters*, které od něj dědí.
   - Přidání základního Latte layoutu a dalších specifických *Views*, které ho rozšiřují.
   - Přidání *Webpacku* (provedl Nikolai Baranov v rámci své BP) a potřebných knihoven pomocí něj. Na začátku procesu implementace byly potřebné knihovny dočasně vložené pomocí odkazů na jejich zdrojové soubory online.
   - Přidání `Authenticatoru`, který se bude starat o přihlašování a přidělování identit uživatelům. Následovalo přidání `Authorizatoru`, který monitoruje dostupné stránky a dává pozor na to, aby uživatel měl přístup pouze tam, kam mu to jeho role v systému dovoluje.

4. Postupné naplnění Presenters starými funkcemi

   - Aby se na nic nezapomnělo, posloužila jako předloha stará aplikace, přístupové body API a také uživatelská příručka.
   - Tyto změny se týkaly víceméně všech tříd v aplikaci.

5. Zapracování nových požadavků do systému

6. Vytvoření automatických testů

4.3 Specifikace nových požadavků

Některé z požadavků na nové funkce, které jsem již zmínil v závěru Analýzy, nebyly úplně triviální. Aby bylo možné řádně zkontrolovat, zda byly požadavky v plné míře splněny, rozhodl jsem se zde některé z nich blíže specifikovat a zároveň čtenáři vysvětlit, jak byly implementovány.

4.3.1 Podmínky užívání a souhlas uživatelů

Po domluvě s vedoucím práce bylo rozhodnuto, že zatím stačí, aby bylo datum souhlasu uložené v databázi. U uživatele je sledováno datum registrace a datum posledního souhlasu s podmínkami. Prozatím jsou vždy shodné, ale v budoucnu je očekávána implementace procesu, který při změně smluvních podmínek požádá uživatele v systému o nový souhlas.

Zároveň je možné do databáze do příslušné tabulky vkládat smluvní podmínky napsané v HTML, spolu s datem začátku jejich platnosti. Řádek tabulky s nejnovějším datumem je vždy zobrazen přímo na webu. Žádné mechanismy pro změnu podmínek a nebo kontrolu souhlasu za pomoci aplikace implementovány nebyly, administrátor bude v případě potřeby přistupovat přímo do databáze.

4.3.2 Přidání vyhledávání na úvodní stránku

Na úvodní stránce je nově možnost rovnou vyhledávat obsah, konkrétně písně dle názvu, tagu či autora. Výsledku bylo dosaženo jednoduchým způsobem – po zadání dotazu a jeho odeslání je uživatel přesměrován na přehled písní, kde je hledaný řetězec vložen jako filtr do tabulky. Tabulka ale má filtry pro jednotlivé vlastnosti písně a hledat tak podle autora, tagu i písně najednou, nebylo vzhledem k jejímu zamýšlenému designu vhodné.

Situace byla vyřešena následovně – Při zadání dotazu se systém nejprve pokusí najít píseň s autorem, který hledanému řetězci odpovídá. Pokud je nalezen alespoň jeden výsledek, je uživatel přesměrován na přehled s vyplněným autorským filtrem. Pokud ne, tak systém zjistí, jestli hledaným řetězcem je tag. Následuje stejný proces a pokud hledání tagu i autora bylo bez výsledku, použije se v zobrazené tabulce filtr podle názvu písně.

4.3.3 Obnovení převzaté písně

Pokud je píseň smazaná či upravená a nějaký uživatel ji má v tu chvíli převzatou, bude vytvořena její kopie a uživatel bude na tuto skutečnost upozorněn za pomoci systémové notifikace. Při kliknutí na upozornění je přesměrován na detail kopie původní podoby písně a dostane jednorázovou možnost si nechat zobrazit v novém okně novou verzi a starou si popřípadě zkopírovat pro své potřeby.

Při opětovném kliknutí na notifikaci mu už tato možnost nabídnuta není – je totiž nutné v určitý moment finálně rozhodnout, jaká z písní má být nakonec tou uživatelem převzatou a zároveň toto rozhodnutí uživateli sdělit. Při prvním zobrazení staré verze se tedy vztah mezi převzetím a kopií automaticky zruší, uživatel je o tomto kroku informován a pokud s ním nebude souhlasit, má až do dalšího načtení stránky možnost si píseň zkopírovat.

4.3.4 Změny detailu zpěvníku

Důvodem pro tyto změny byl proces, který uživatelé budou často vykonávat – přidání písně do zpěvníku, který je abecedně řazený. Protože ale je přidání písně samostatný krok, oddělený od řazení, je nově přidaná píseň vždy přidána na konec zpěvníku. Uživatel pak musí píseň manuálně přesunout na vhodnou pozici.

Protože ale přehled zpěvníku už stejně disponuje chytrou tabulkou, byla by škoda ji nevyužít pro ulehčení správy písní. Pokud si tedy zpěvník nechá zobrazit jeho vlastník, má vedle tabulky navíc tlačítko „Uložit pořadí písní“. Po jeho stisknutí proběhne kontrola, zda-li jsou zobrazeny všechny písně – pokud by například byl v tabulce zrovna aplikovaný nějaký filtr a nebyly zobrazeny všechny, nebylo by možné jednoznačně stanovit pořadí všech písní – a pokud ano, jsou jejich pozice přečíslovány vzhledem k tomu, jak jsou zrovna zobrazeny uvnitř tabulky.

4.4 Konkrétní řešené problémy

Během implementační části své bakalářské práce jsem napsal nespočet řádků kódu. Přestože se na mnou vykonanou práci v historii *Git* repozitáře může podívat kdokoli, uvedl bych zde rád alespoň pár zajímavých překážek, se kterými jsem se musel vypořádat.

4.4.1 Routing

Jedním z problémů, na které jsem při začátku práce na implementaci narazil, bylo routování (směrování HTTP požadavků na vhodné *Presenter* třídy). Podle dokumentace Nette je doporučeným přístupem napsání si vlastní `RouterFactory`, která ve statické metodě `createRouter()` vytvoří, naplní a vrátí `RouteList`. Podobně ostatně bylo routování v aplikaci implementované již dříve.

Vyrobit novou třídu, která bude přesměrovávat požadavky na konkrétní *Presenter*, nebyl problém. Ten spočíval až v nekompatibilitě nové `RouterFactory` a staré `ApiRouterFactory`. Celý dříve používaný systém routování byl totiž naprogramován jedním z předchozích studentů, Tomášem Markaczem. Ten se ale rozhodl vše napsat sám a stará verze směrování fungovala bez využití nabízených možností Nette frameworku.

Aby nebylo se starým routováním zbytečně manipulováno a nedošlo k rozbití funkčnosti mobilní aplikace, jsou všechny požadavky obsahující řetězec „api“ přesměrovány na zvláštní `ApiPresenter`, jak je vidět v ukázce kódu 1.

```
class RouterFactory
{
  use Nette\StaticClass;

  public static function createRouter(): RouteList
  {
     $router = new RouteList;
     $router[] = new Route('<presenter api>[/<path .+>]', [
        'path' => [
           Route::VALUE => null,
           Route::FILTER_IN => null,
           Route::FILTER_OUT => null,
        ],
     ]);
     $router[] = new Route('<presenter>[/<action>][/<id>]',
             'Homepage:default');
     return $router;
  }
}
```

Listing 1: Ukázka třídy `RouterFactory`

V `ApiPresentru` jsou pak všechny požadavky předávány dále třídě `RestApp\Application` z knihovny Tomáše Markacze, která už je zpracuje za použití `ApiRouterFactory` – takže stejným způsobem, jako požadavky na API fungovaly před přepracováním celého Kytarového zpěvníku. Tento proces je vidět v ukázce 2.

4.4.2 Příliš mnoho požadavků na databázi

Jednou z výhod používání Nette frameworku je i přítomnost jeho ladícího nástroje jménem *Tracy*. Ten krom času načtení stránky, identity uložené v session a dalších užitečných informací zobrazuje i všechny SQL dotazy, které byly nad připojenou databází vykonány. Díky tomu se mi podařilo zjistit, že třeba při načtení písní a zpěvníků na úvodní stránce aplikace je takových dotazů více než sto. Toto množství bylo zaviněné jediným důvodem – zobrazováním průměrného hodnocení.

Třídy `Song` a `Songbook` totiž mají kolekce (obdoba pole), pomocí kterých je možné v kódu získat přístup ke všem hodnocením, které k nim náleží, i navzdory tomu, že vztah je v databázi uložen v tabulce hodnocení. Při získávání nejlépe hodnocených písní ale byly neefektivně všechny písně proiterovány

```
class ApiPresenter implements Application\IPresenter
{
  /** @var RestApp\Application @inject */
  public $apiApplication;


  public function run(Application\Request $request): void
  {
     RestApp\ErrorHandlers::register($this->apiApplication);
     $this->apiApplication->run();
  }
}
```

Listing 2: Ukázka třídy `ApiPresenter`

(přičemž bylo vždy nutné udělat nový dotaz na získání jejich hodnocení) a až poté seřazeny. Problém tak netkvěl ve složitosti dotazů a nebo množství výsledných dat, ale čistě v počtu provedených požadavků. Řešením se ukázal býti kód 3, umístěný před iterací nad písněmi, za účelem získání jejich hodnocení.

```
$ids = \Nette\Utils\Arrays::map($songs, function (Song $song) {
                return $song->getId();
        });

$this->em->getRepository(Song::class)
        ->createQueryBuilder('song')
        ->leftJoin('song.songRatings', 'ratings')
        ->select('PARTIAL song.{id}, ratings')
        ->whereCriteria(['id' => $ids])
        ->getQuery()
        ->getResult();
```

Listing 3: Ukázka načtení hodnocení písní do mezipaměti

Knihovna Doctrine totiž načtená data v rámci jednoho HTTP požadavku ukládá a když ví, že entitu se správnou třídou a daným `id` už jednou získávala, sáhne pro ni do paměti, místo do databáze. Přestože tedy výsledek kódu výše není přiřazený do žádné proměnné, je proveden dotaz na databázi a jsou načtena všechna hodnocení (join proveden pouze s `id` písně, aby nebyla ve výsledku zbytečná data). Když je potom později v rámci jednoho požadavku počítáno průměrné hodnocení, stačí se podívat do cache.

4.4.3 Datagrid

Jedním z klíčových požadavků na aplikaci byl i praktický přehled písní a zpěvníků. Takový, ve kterém bude možné vyhledávat, řadit a který bude ovládat stránkování. Chytrým tabulkám pro zobrazování dat se říká datagrid, a protože nebyl důvod vynalézat kolo, rozhodl jsem se využít nějaké již existující řešení. Protože jsem chtěl, aby výsledná knihovna byla kompatibilní s používaným frameworkem, hledal jsem na stránkách Componette, kde se vhodné komponenty nacházejí.

Výsledkem hledání byly tři knihovny – *Nextras\Datagrid*, *O5\Grido* a *Ublaboo\Datagrid*. Všechny splňovaly požadavky na potřebné funkce a tak na konci bylo rozhodnuto na základě osobních preferencí. Původně jsem chtěl zvolit *Nextras\Datagrid*, protože jsem ji již na jednom projektu použil, ale nakonec padla volba na *Ublaboo\Datagrid*. Byla totiž jako jediná ze všech knihoven neustále vyvíjena a cílem mé práce přece jen je se zbavit zastaralých technologií, neměl bych tedy ty neaktuální sám přidávat.

Vytvořit tabulku za pomoci použité knihovny bylo jednoduché, níže přikládám ukázku kódu, který tabulku tvoří. Komponenta je vytvořena v příslušné *Presenter* třídě (ukázka 5), pro získání základní podoby tabulky si ovšem volá funkci `getDefaultGrid()` z `BasePresenter` (ukázka 4), který je rodičem všech *Presenter* tříd. O něco níže je ještě vidět příklad toho, jak vypadá šablona přehledu písní 6 a jak je manuálně možné v šabloně předefinovat způsob, jakým jsou jednotlivé sloupce vykresleny 7.

Vybraný datagrid obsahuje několik různých možných zdrojů dat, přičemž jedním z nich je i `DoctrineDataSource`. Výhodou této třídy je, že při filtrování a řazení manipuluje pomocí *DQL* (Doctrine Query Language) a SQL přímo s databází a tudíž načítá vždy pouze ta data, která jsou aktuálně zobrazena. Brzy se ale ukázalo, že není možné tuto třídu v její aktuální podobě použít – díky své obecnosti nebyla schopna se automaticky vypořádat s takovými úkony, jako je třeba řazení podle hodnocení (tj. seřazení na základě průměru hodnot z *ManyToOne* vztahu) či vyhledávání podle tagů.

Vznikla tak vlastní třída `DoctrineSongDataSource` (a další obdoby pro třídy `Report` a `Songbook`), která je potomkem `DoctrineDataSource` a která rozšiřuje funkce `applyFilterText(FilterText $filter)` a `sort(Sorting $sorting)`. Třídy byly umístěny v aplikaci do namespace `KZ\Model\DataSource`. Pro ilustraci funkcí níže uvádím kód konstruktoru 8, řazení 9 a část vyhledávání dle tagů 10.

Aby zároveň bylo zachované rozdělení na prezentační, aplikační a datovou vrstvu, volá *Presenter* při požadování `DataSource` pro datagrid funkci *Service* třídy, která si o `DataSource` řekne příslušné *DAO* třídě. Ta už je ovšem specifická (využívá knihovnu Doctrine) a je v ní možné vrátit nově vyrobený `DoctrineDataSource`.

```
protected function getDefaultGrid($name)
{
  $grid = new DataGrid($this, $name . 'Grid');
  $grid->setTemplateFile(__DIR__ . '/../templates/grid/'
           . $name . '.latte');
  $grid->setDefaultPerPage(20);

  $translator =
           new \Ublaboo\DataGrid\Localization\SimpleTranslator([
     'ublaboo_datagrid.no_item_found_reset' =>
            'Žádné položky nenalezeny. Filtr můžete vynulovat',
     'ublaboo_datagrid.no_item_found' =>
            'Žádné položky nenalezeny.',
     'ublaboo_datagrid.here' => 'zde',
     'ublaboo_datagrid.items' => 'Položky',
     'ublaboo_datagrid.all' => 'všechny',
     'ublaboo_datagrid.from' => 'z',
     'ublaboo_datagrid.reset_filter' => 'Resetovat filtr',
     'ublaboo_datagrid.group_actions' => 'Hromadné akce',
     'ublaboo_datagrid.show_all_columns' =>
            'Zobrazit všechny sloupce',
     'ublaboo_datagrid.hide_column' => 'Skrýt sloupec',
     'ublaboo_datagrid.action' => 'Akce',
     'ublaboo_datagrid.previous' => 'Předchozí',
     'ublaboo_datagrid.next' => 'Další',
     'ublaboo_datagrid.choose' => 'Vyberte',
     'ublaboo_datagrid.execute' => 'Provést',
     'ublaboo_datagrid.show_filter' => 'Filtry',
     'ublaboo_datagrid.filter_submit_button' => 'Filtrovat',
     'Name' => 'Jméno',
     'Inserted' => 'Vloženo',
  ]);
  $grid->setTranslator($translator);
  return $grid;
}
```

Listing 4: Funkce `getDefaultGrid()` z `BasePresenter`

```
protected function createComponentSongGrid()
{
  $grid = $this->getDefaultGrid('song');
  $grid->setRememberState();
  $grid->setDataSource($this->songModel->
          prepareSongGridData($this->getUserIdentity(),
                  $this->type));

  $grid->addColumnText('title', 'Název')
     ->setSortable()
     ->setSortableResetPagination();
  $grid->addColumnText('author', 'Autor')
     ->setSortable()
     ->setSortableResetPagination();
  $grid->addColumnText('album', 'Album')
     ->setSortable()
     ->setSortableResetPagination();
  $grid->addColumnText('rating', 'Hodnocení')
     ->setSortable()
     ->setSortableResetPagination();

  /* zkraceno */

  $grid->addFilterText('title', 'Vyhledat', ['title'])
     ->setPlaceholder('Název...');
  $grid->addFilterText('author', 'Autor', ['author'])
     ->setPlaceholder('Autor...');
  $grid->addFilterText('album', 'Album', ['album'])
     ->setPlaceholder('Album...');
}
```

Listing 5: Funkce `createComponentSongGrid()` z `SongPresenter`

```
{block content}
<div  class="float-right">
    <button class="btn btn-primary export-songs-btn"
            style="color:white;font-weight: 300!important;" >
        <i class="mdi mdi-18px mdi-download"></i>
        Exportovat do PDF
    </button>
</div>

<h1 n:inner-block="title">Veřejné písně a nasdílené písně</h1>
<div id="export-list-dialog" class="export-dialog"
        title="Písně k˜Exportování">
    <div id="export-list-content">
        <div class="sortable-export" id="export-list">
        </div>
    </div>
</div>
{control songGrid}
```

Listing 6: Šablona *default.latte* patřící k `SongPresenter`, ve které se tabulka vykresluje

```
{extends $original_template}

{define col-title}
  <a href="{$presenter->link('Song:view', ['id' =>
  $item->getId(), 'backlink' =>
  $presenter->getParameter('action')])}">
          {$item->getTitle()}
  </a>
{/define}

{define col-owner}
  <a href="{$presenter->link('Profile:default', ['id' =>
  $item->getOwner()->getId()])}">
          {$item->getOwner()->getUsername()}
  </a>
{/define}
```

Listing 7: Ukázka ze šablony datagridu *song.latte*, kde se definuje podoba jednotlivých sloupců

```
public function __construct(QueryBuilder $data_source,
        $primary_key, User $user = null, $type,
        $songbook = null)
{
  parent::__construct($data_source, $primary_key);
  $this->user = $user;
  $this->type = $type;

  if ($this->type ===
         SongSongbookPresenter::FILTER_PUBLIC_AND_SHARED) {
     $this->data_source->andWhere('song.public = true');
     $this->data_source->andWhere('song.archived = false');
     if ($this->user) {
        $this->data_source->leftJoin('song.songShares',
                'shares', 'WITH', '(shares.song = song AND
                shares.user = :user)')
           ->setParameter('user', $this->user);
        $this->data_source->addGroupBy('song.id');
        $this->data_source->orWhere('shares.id IS NOT NULL');
     }
  } elseif ($this->type ===
         SongSongbookPresenter::FILTER_OWNED_AND_TAKEN) {
     $this->data_source->andWhere('song.archived = false');
     $this->data_source->leftJoin('song.songTakes',
             'takes', 'WITH', '(takes.song = song AND
             takes.user = :user)')
        ->addGroupBy('song.id')
        ->andWhere('song.owner = :user')
        ->setParameter('user', $this->user)
        ->orWhere('takes.id IS NOT NULL');
  } elseif ($this->type ===
         SongSongbookPresenter::FROM_SONGBOOK) {
     $this->data_source->leftJoin('song.songbooks',
             'songSongbooks');
     $this->data_source->andWhere('song.archived = false');
     $this->data_source
             ->andWhere('songSongbooks.songbook = :songbook')
             ->setParameter('songbook', $songbook);
     $this->data_source->groupBy('song.id');
     $this->data_source->orderBy('songSongbooks.position',
     'ASC');
  }
}
```

Listing 8: Ukázka konstruktoru třídy `DoctrineSongDataSource`

```
public function sort(Sorting $sorting)
{
  $sort = $sorting->getSort();
  if ($sort && $this->type ===
          SongSongbookPresenter::FROM_SONGBOOK) {
     $this->data_source->resetDQLPart('orderBy');
  }
  if (key($sort) === 'rating') {
     $sorting = new Sorting($sorting->getSort(),
             function ($dataSource, $sort) {
                $dataSource->leftJoin('song.songRatings',
                        'ratings', 'WITH', '(ratings.song =
                        song AND ratings.deleted = 0)');
                $dataSource->addSelect('COALESCE(
                        AVG(ratings.rating), 0)
                        AS HIDDEN avgRating');
                $dataSource->addGroupBy('song.id');
                $dataSource->orderBy('avgRating',
                        $sort['rating'] ?? 'ASC');
             });
  } elseif (key($sort) === 'owner') {
     $sorting = new Sorting($sorting->getSort(),
             function ($dataSource, $sort) {
                $dataSource->leftJoin('song.owner', 'users');
                $dataSource->orderBy('users.id',
                        $sort['owner'] ?? 'ASC');
             });
  } elseif (key($sort) === 'position') {
     $sorting = new Sorting($sorting->getSort(),
             function ($datasource, $sort) {
                $datasource->orderBy('songSongbooks.position',
                        $sort['position'] ?? 'ASC');
             });
  }
  parent::sort($sorting);
  return $this;
}
```

Listing 9: Ukázka funkce `sort` starající se o řazení

```
if ($c === 'song.tags') {
  foreach ($words as $key => $word) {
     $this->data_source->leftJoin('song.tags', 'tags_' . $key,
             'WITH', '(tags_' . $key . '.song = song AND
             (tags_' . $key . '.public = true OR tags_' .
             $key . '.user = :user))')
        ->andWhere('tags_' . $key . '.tag LIKE :word_' . $key)
        ->addGroupBy('song.id')
        ->setParameter('word_' . $key, '%' . $word . '%');
  }
  $this->data_source->setParameter('user', $this->user);
  continue;
}
```

Listing 10: Ukázka filtrování dle tagů ve třídě `DoctrineSongDataSource`

4.4.4 Cyklická závislost

Když jsem se rozhodl vytvořit speciální *Service* pro posílání emailů, vznikla v systému cyklická závislost. `Authenticator` (třída sloužící pro ověřování identity uživatele) totiž vyžadoval `UserService`. Ten zase ke svému fungování potřeboval `MailService`, protože nově vytvořeným uživatelům posílal mail o potvrzení registrace. Aby `MailService` mohl vytvořit šablonu posílané zprávy, používal `Latte\TemplateFactory`, která ovšem byla využitá i třídou `Application`, která celý systém zaštiťuje. A konečně `Application` při svém běhu používala třídu `Security\User`, která ale pro ověření identity používala právě `Authenticator`, zmíněný na začátku tohoto odstavce.

Protože roztrhnout cyklickou závislost někde uvnitř Nette frameworku by nebylo moudré řešení, bylo možné operovat pouze s mnou napsanými třídami `Authenticator`, `UserService` a `MailService`. Konkrétně mezi prvními dvěma zmíněnými nakonec došlo k roztrhnutí cyklu – `Authenticator` používal servis k získání uživatele pomocí konkrétního mailu a tato jediná metoda byla vyčleněna ven do samostatné třídy, viz. ukázka 11.

Aby se předešlo duplikování kódu, bylo volání stejné funkce upraveno i v `UserService` 12, který nyní pouze provádí přesměrování do externího *Service*.

4.5 Dostupné manuály

Nová verze aplikace obsahuje i trojici manuálů. Jedná se o mírně aktualizované příručky z předchozí verze, pouze ta vývojářská se dočkala razantnějších změn.

4.5.1 Uživatelská příručka

Slouží pro seznámení nového uživatele se všemi důležitými pojmy a funkcionalitami Kytarového zpěvníku. Protože je velmi nepravděpodobné, že by si jakýkoli uživatel před použitím systému chtěl stahovat a pročítat doprovodnou příručku, je umístěna přímo v samotné aplikaci.

4.5.2 Vývojářská příručka

Obsahuje popis kroků, které je nutné provést, aby mohl programátor aplikaci stáhnout, zprovoznit a následně rozšiřovat. Kromě toho také zmiňuje použité technologie a instrukce pro spouštění automatických testů a nebo automatické kontroly formátu napsaného kódu. Je součástí Git repozitáře s projektem a také jednou z příloh této bakalářské práce.

4.5.3 Příručka pro nasazení

Poslední příručka je ta, která poslouží v případě, kdy cílem je pouhé nasazení již hotového release balíčku na server. Protože není záměrem aplikaci jakkoli

```
class UserAuthService extends Object implements IUserAuthService
{

  /** @var IUserDao */
  private $userDao;


  /**
   * UserService constructor.
   * @param IUserDao $userDao
   */
  public function __construct(
     IUserDao $userDao
  ) {
     $this->userDao = $userDao;
  }


  /**
   * @param string $username
   * @return User
   */
  public function getUserProfile($username)
  {
     return $this->userDao->
             getFirstByColumn('username', $username);
  }
}
```

Listing 11: Třída `AuthUserService`

```
public function getUserProfile($username)
{
  return $this->userAuthService->getUserProfile($username);
}
```

Listing 12: Funkce `getUserProfile` ze třídy `UserService`

upravovat, je vyžadováno mnohem méně nainstalovaných technologií. Příručka obsahuje detailní návod, jak nainstalovat celou aplikaci z release balíčku ve verzi 2.0 a také jak aktualizovat již funkční aplikaci verze 1.4.0. Příručka je součástí release balíčku a také jednou z příloh této bakalářské práce. Obsahuje i jednoduchý UML diagram nasazení 2.7.

4.6 Testování

Kytarový zpěvník je rozsahem už dávno za hranicí toho, co by bylo možné testovat manuálně, a tak využívá automatické testy. Jedná se o způsob, jak ověřit splnění požadavků na aplikaci, a zároveň také rychlou kontrolu toho, jestli programátor při provádění rozsáhlejších úprav omylem nerozbil nějaký z již existujících procesů. K jejich správě a spouštění v aplikaci slouží PHP knihovna `Nette\Tester`, která je součástí Nette frameworku.

Existuje spousta způsobů, jak k testování aplikace přistupovat. Ke kytarovému zpěvníku byly už dříve vyvinuty dva různé druhy testů. Prvním z nich jsou naivně napsané testy ověřující za pomoci požadavků funkčnosti API a druhým typem jsou pak krátké unit testy sloužící k ověření funkčnosti aplikační logiky. Protože jsem nakonec zasahoval do obou typů testů, rozepíšu se o nich nyní podrobněji.

4.6.1 Testy API a jejich scénáře

Testy z této skupiny vznikly už dávno. Během let se na ně zapomnělo a můj tým se v rámci BI-SP2 věnoval jejich obnovení v předchozí podobě. Jednalo se ovšem o naivní implementaci, kde kód byl členěn jen na úrovni souborů, ale ne třeba do funkcí či tříd. Zároveň také každý test před svým spuštěním prováděl smazání a vytvoření celé databáze, aby vždy začínal „s čistým štítem“. Tím pádem ale provedení testů zabíralo zbytečně mnoho času a vytrácela se možnost rychlé kontroly toho, jestli programátor změnami něco důležitého omylem neovlivnil.

Obsah testů jsem se po domluvě s vedoucím práce rozhodl neměnit. Žádné testovací scénáře jsem tedy nepřidal ani neodebral a testy jsem ponechal „implementation specific“ (pro co nejpřesnější kontrolu manipulovaných dat využívají přístupu do databáze přímo přes Doctrine, nikoli přes *DAO* třídy). Změnil jsem ovšem formu, jakou testy mají – podle testovaného přístupového bodu API jsem je rozdělil do tříd a všem přidal společného abstraktního rodiče `KZTestHelper`, který má funkce pro některé často opakované části kódů, jako je například tvorba requests. Sám dědí od třídy `TestCase` z knihovny `Nette\Tester`. Díky tomu se při jeho spuštění vykonají všechny jeho metody začínající na slovo „test“ a před každou z nich se provede metoda `setUp()`.

V této metodě potomek `KZTestHelper` vykoná načtení jednoho ze souborů s SQL, který se váže ke konkrétnímu *Resource*. Jsou tak vždy do čistého stavu přivedeny jen ty tabulky databáze, kterých se právě vykonávané testy týkají, a nikoli celá databáze. Ta se načte pouze jednou při spuštění testování. Čas provedení všech testů se pohybuje okolo 80 vteřin, zatímco dříve nebylo neobvyklé čekat i více než pětinásobek času, než vše stihne doběhnout.

Třída `TestCase` umožňuje jednotlivé testovací metody spouštět paralelně ve více procesech a urychlit tak samotné testování. Protože ale všechny testy zasahují do databáze a `Nette\Tester` negarantuje pevné pořadí testů (naopak

dynamicky vždy prvně provádí ty, které naposled selhaly), není takové chování žádoucí a je zakázáno přepínačem v souboru *run-api.sh*, pomocí kterého se testy z root složky projektu spouští.

Pro úplnost ještě uvádím výčet scénářů, které API testy pokrývají:

- Písně
  - Získání všech písní
  - Získání všech hodnocení písně
  - Získání všech komentářů písně
  - Hodnocení
    - Vytvoření
    - Přečtení
    - Upravení
    - Smazání
  - Komentář
    - Vytvoření
    - Přečtení
    - Upravení
    - Smazání
- Zpěvníky
  - Získání všech zpěvníků
  - Získání všech hodnocení zpěvníku
  - Získání všech komentářů zpěvníku
  - Hodnocení
    - Vytvoření
    - Přečtení
    - Upravení
    - Smazání
  - Komentář
    - Vytvoření
    - Přečtení
    - Upravení
    - Smazání
- Upozornění
  - Získání všech upozornění

  - Označení všech upozornění za přečtené
- Uživatelé
  - Získání všech uživatelů
  - Registrace uživatele
  - Povýšení na administrátora
- Přání
  - Smazání přání
  - Získání všech přání
  - Přečtení jednoho přání
  - Vytvoření přání
  - Upravení přání

4.6.2 Service Unit Testy

Druhou kategorií testů, které už byly v malé míře v Kytarovém zpěvníku přítomny, jsou takzvané unit testy. Jedná se postup, při kterém jsou testované jednotlivé metody jedné izolované třídy. Veškeré závislosti a volání metod jiných tříd jsou nahrazené mock třídami. Zatímco některé jsou vyrobené manuálně (například *DAO* třídy, které musejí poskytovat falešná data bez napojení na databázi), jiné jsou dodané za pomoci PHP knihovny *Mockery*.

Unit testy jsou cíleny pouze na *Service* třídy. Původně byly testy pokryté jen *Service* starající se o generování PDF a nahlašování uživatelů (tj. třídy, na kterých dělal v minulých semestrech můj tým). Opět se jednalo jen o naivní implementaci, tak jsem testy přepsal do jednotlivých tříd a přidělil jim společného předka s často používanými metodami. V současné době jsou pokryté následující třídy:

- NotificationService
- PDFService
- ReportService
- SessionService
- SongbookService
- SongService
- TermsService
- UserService

- WishService

Nejedná se o kompletní pokrytí kódu. Takový přístup by byl příliš časově náročný, přesto je aktuálně testy pokryto 47 % řádků *Service* tříd. Pro lepší přehled o pokrytí poslouží patřičný přepínač třídy `Tester`, která za pomoci nainstalovaného PHP rozšíření *XDebug* (nejedná se o knihovnu pro konkrétní projekt, ale o knihovnu pro celý používaný PHP server) dokáže vygenerovat podrobné hlášení o stavu. Pro spuštění unit testů slouží skript *run-model.sh*, který se spouští z root složky celého projektu.

Kapitola 5

Gamifikace

5.1 Úvod

Gamifikace bývá definována jako proces využívající prvků z her, věrnostních programů a behaviorální ekonomie. Je trendem posledních let, a to z dobrého důvodu. Než mladí američané vyrostou a začnou pracovat, stráví odhadem více než 10 000 hodin hraním videoher [20]. Přitom kdyby se člověk po podobnou dobu věnoval nějakému koníčku, byl by v něm pravděpodobně tou dobou už dávno mistr. A tak je logickým krokem, že se designéři často snaží svět upravit tak, aby měl uživatel možnost zažívat podobné pocity, jako kdyby si hrál.

S gamifikačními prvky, které se snaží do různých činností přinést odměny či zábavu navíc, se jistě setkal už každý. Tím nejběžnějším případem jsou věrnostní programy v obchodech. Tuto mechaniku zmodernizoval třeba řetězec Starbucks [21] či společnost Nike [22]. Od doby těchto začátků se obecné principy i používané metody hodně vyvinuly a gamifikace je dnes skoro všude kolem nás.

5.1.1 Proč ji použít?

Základním kamenem aplikace je hudba a říká se, že ta sama o sobě dokáže dělat divy. Ostatně – „Nic není hrubé, tvrdé a vzteklé dost, aby to hudba nedovedla změnit“[23]. To ale bohužel nestačí pro to, aby se Kytarový zpěvník stal úspěšným. Aplikace to zkrátka při spuštění v živém prostředí nebude mít jednoduché. Bude zaostávat za konkurencí a to primárně díky dvěma důvodům. Tím prvním bude dozajista nedostatek uživatelů oproti dalším podobným webům, které mají mnohaletý náskok v budování komunity. A tím druhým nedostatek obsahu, který jde ruku v ruce s nízkým počtem uživatelů. Cílem této části mé bakalářské práce je vysvětlit, proč si myslím, že by mohla gamifikace tyto problémy řešit a navrhnout konkrétní řešení.

5.2 Průzkum konkurence

Protože právě díky konkurenci to bude mít Kytarový zpěvník na trhu tak těžké, je vhodné začít průzkumem toho, jestli gamifikaci používají podobné webové aplikace, a pokud ano, tak jakým způsobem. Přestože reálnou konkurenci představují prozatím jen české weby, podívám se v této sekci kvůli inspiraci i na stránky zahraniční.

- Velký zpěvník.cz
  - 16308 písní v databázi k 8.2.2018.
  - Nové písně může vložit úplně kdokoli, i když proces je přehnaně komplikovaný a má spoustu vysvětlivek. Za popularitu může pravděpodobně fakt, že při hledání slova „zpěvník“ pomocí vyhledávače Google se odkaz na web zobrazí úplně jako první.
  - Žádné gamifikační prvky.
- Písničky-akordy.cz
  - Zobrazuje se seznam top 30 zpěvníků a uživatelé si často do názvů zpěvníků dávají své jméno (pro získání uznání komunity).
- Karaoke texty
  - Na webu jsou k dispozici pouze texty (bez akordů), takže aplikace nemá úplně stejný záměr jako Kytarový zpěvník. Uvádím ji ovšem proto, že gamifikaci využívá. Konkrétně se jedná o tyto dva mechanismy:
    - Body
      - Jsou získávány za určitou aktivitu uživatelů. Nejméně jich je za přidání textu nebo přidání fotky k interpretovi. Střední množství bodů se dá získat přidáním překladu nebo videoklipu a nejvíce jich je za přidání karaoke a nebo nahrání profilové fotky.
      - K dispozici je žebříček uživatelů s nejvíce body, ale zároveň se na hlavní stránce zobrazují uživatelé s největším množstvím bodů za poslední týden. Díky tomu mají i nově registrovaní možnost si zažít svoji chvilku slávy.
    - Odznáčky
      - Když uživatel splní určité úkony, obdrží na svůj profil odznáček. Každý z odznáčků má tři stupně dle obtížnosti, vždy se jedná o stejný úkon, jen ve větším množství. Příkladem činností pro získání odznáčku je třeba určitý čas strávený na webu, ohodnocování playlistů, za přihlašování a nebo označování jiných uživatelů za své přátele.

· Odznáčky mají vlastní globální žebříček a zároveň se na titulní stránce webu zobrazuje seznam uživatelů, kteří jako poslední získali nějaký odznáček. Získané úspěchy jsou také zobrazeny na profilu daného uživatele.

- Supermusic.sk
  - Přestože se jedná pravděpodobně o jeden z nejnavštěvovanějších online zpěvníků u nás (dle jejich počítadla okolo 18 tisíc unikátních uživatelů za den), tak jediným prvkem soutěživosti na stránkách jsou žebříčky uživatelů dle počtu vložených písní a článků.
- Ultimate-guitar.com
  - Systém této zahraniční stránky je velmi podobný přístupu na webu Karaoke texty. Uživatelé za určité akce získávají body a také trofeje, které se zobrazují na profilu (nemají několik různých stupňů, ale některé jde získat vícekrát).
  - Pokud je schválené určité množství nových vložených písní, dostane uživatel nová speciální práva. Od určité chvíle má tak možnost upravovat jakoukoli cizí píseň (pokud mu přijde, že akordy jsou špatně) a jeho hodnocení má větší váhu.
  - Zajímavostí jsou progress bary které zobrazují procenta vyplnění uživatelského profilu a jeho jednotlivých sekcí. Včetně checklistů konkrétních kroků, jaké uživatel musí podniknout, aby získal 100 %.

Navzdory intenzivnímu průzkumu se mi podařilo najít pouze velmi malé množství prvků gamifikace na hudebních webech napříč internetem. Důvod se po chvíli přemýšlení zdá býti docela jednoduchým – většina velkých zavedených stránek funguje dlouho a v době jejich vzniku nebylo vkládání herních prvků do systému ještě tak populární. A protože nyní už mají dostatečnou uživatelskou základnu i množství obsahu, nepotřebují aktivně uživatele v určitých činnostech podporovat.

Než se ovšem pustím do konkrétních úvah ohledně toho, jak by se nasbíraná inspirace dala použít ve spojení s Kytarovým zpěvníkem, je vhodné si ještě ve zkratce představit základní principy.

5.3 Žádoucí chování uživatele

Gamifikace není produkt sama o sobě. Jedná se o proces a vždy se musí vázat k určitému chování uživatele, které se snaží podpořit. Aplikace Kytarový zpěvník stojí převážně na množství a kvalitě obsahu a proto by měli dle mého názoru být uživatelé odměňováni převážně za následující aktivity:

- Tvorba vlastních veřejných písní/zpěvníků

  - Je důležité, aby šlo o tvorbu nového obsahu a ne jen převzetí či zkopírování cizího. Bonusové body by mohly být za to, kdyby píseň s daným názvem ještě v systému nebyla a nebo kdyby se uživatel při tvorbě nové písně trefil do přání nějakého jiného uživatele.
  - Větší odměna by měla následovat v případě, že vytvořená píseň či zpěvník dosáhne určitého průměru hodnocení (s nějakým minimálním počtem udělených hodnocení). To bude sloužit jako motivace k tomu, ať je vyráběný obsah kvalitní.

- Hodnocení veřejných písní/zpěvníků
  - Je třeba nějak hlídat kvalitu obsahu a nejlehčí variantou je, že to uživatelé udělají sami. Žádoucí by mělo být udělování rozdílného počtu hvězdiček, aby se předešlo tomu, že uživatel nakliká pár náhodným písním maximální hodnocení a bude za to odměněn.
- Nahlašování nevhodného obsahu
  - Podobná je i aktivita nahlašování nevhodného obsahu. V tomto případě by měl být uživatel odměněn pouze v tom případě, že nahlášený obsah byl skutečně závadný a na základě nahlášení ho administrátor odstranil.
- Pozvání nového uživatele
  - Jedna z nejčastějších praktik napříč celou historií gamifikace je nabádání uživatele k tomu, aby do systému zapojil své přátele. Kytarový zpěvník prozatím žádnou možnost pozvání nového uživatele (nejspíše dle emailové adresy) neobsahuje, do budoucna by se ale mělo jednat o zásadní krok pro růst komunity.
- Sdílení obsahu na sociálních sítích
  - Alternativou k předchozí myšlence získávání nových uživatelů může být podporování sdílení obsahu na sociálních sítích. Ty největší sociální sítě mají návody pro rychlou integraci sdílení obsahu a například zobrazení odkazu na web na zdi uživatele Facebooku má mnohem větší dopad, než pozvání jediné konkrétní osoby mailem.
- Přihlášení se do aplikace
  - Když už se uživatel jednou nachází na stránce aplikace, existuje šance, že v ní bude nějakým způsobem aktivní – ať už si třeba zahraje písničku, napíše komentář a nebo ohodnotí zpěvník.
  - Určitě je tedy žádoucí motivovat ho k přihlášení co nejčastěji. Podobný záměr je velmi běžně k vidění u mobilních aplikací (převážně her), které poskytují odměny za zapnutí aplikace každý den a pokud uživatel hraje několik dnů za sebou, odměny se neustále zvětšují.

Samostatnou kapitolou pak je možnost podporování socializace, která je pro určitý typ uživatelů (o rozdělení budu mluvit v další kapitole) tím nejdůležitějším lákadlem a zároveň největší odměnou. Protože ale aplikace teprve nedávno získala možnost zobrazování soukromých profilů a jakýkoli systém přidávání přátel či posílání zpráv je věcí daleké budoucnosti, nebudu se mu prozatím věnovat.

5.4 Typy uživatelů

Protože se přidávání herních prvků do běžného prostředí často prolíná s uměním herního designu, dovolím si v této kapitole trochu utéct jiným směrem. Při designování nové části systému je vždycky dobré vědět, pro jaké uživatele je určena, aby se dalo simulovat jejich chování a zajistit, že systém funguje přesně tak, aby uspokojil jejich potřeby.

Existuje několik typologických dělení hráčů počítačových her, z nichž některé zachází velmi do detailů. Pro mé účely ovšem bohatě postačí rozdělení, které definoval v roce 1996 britský profesor Richard Bartle[24], a pro které se vžil obecný název *Bartle taxonomy of player types*. Přestože původním záměrem bylo popsat pouze hráče MMORPG (Massive Multiplayer Online Role Playing Games), je možné tento přístup alespoň v omezené míře aplikovat při tvorbě gamifikovaného prostředí. dokud bude mít designér stále na paměti, že se nejedná o primární použití této teorie.

Podle Bartleho existují 4 kategorie, které ovšem nejsou rigidní. Přístup každého člověka pak sestává ze znaků z více kategorií, jedna ale vždy převládá. Základní dělení je následující:

- The Achiever
  - Tito uživatelé rádi dosahují úspěchů a prohlubují si svůj sociální status, kterým se následně nezapomenou náležitě chlubit. Přímo na ně jsou cílené gamifikační prvky jako odznaky, u kterých je nejdůležitější, aby byly někde na uživatelském profilu vystavené. Klíčový totiž není úspěch jako takový, ale fakt, že ostatní uživatelé ví, že uživatel úspěchu dosáhl.
  - Alternativou odznakům jsou získatelné tituly či jiné modifikace osobního profilu, které všichni ostatní uvidí na první pohled.
  - Podle Bartleho prvotního výzkumu se jedná přibližně o 10 % populace.
- The Explorer
  - Mezi průzkumníky spadají ti, kterým stačí pro pocit uspokojení jen objevování nových věcí. Samotný fakt nalezení něčeho nového jim plně nahrazuje jakoukoli odměnu. Pro tuto kategorii uživatelů bude

nejtěžší nalézt vhodné prvky, které by jim imponovaly a zároveň zapadaly do konceptu Kytarového zpěvníku.

  - Naštěstí je tento typ rozšířen přibližně stejně jako Achiever a tak se jedná pouze o 10 % lidí.

  - Běžnými prvky používanými ve vztahu k těmto uživatelům jsou odemykání sekcí/obsahu, tajné odznaky a nebo indikátory postupu (takzvané progress bary) naznačující, kolik toho už prozkoumali.

- The Socializer

  - Nejpočetnější skupina čítá přibližně okolo 80 % uživatelů. Jejich hlavní motivací je navázání sociální kontaktu s dalšími hráči a nebo zkrátka jen pocit toho, že se v aplikaci nepohybují v úplně anonymním prostředí, ale mají podvědomí i o dalších konkrétních uživatelích. Zároveň jim ale často socializace nestačí jako jediná odměna (ostatně i na Facebooku je uživatel odměněn počtem „lajků“ a ne jen samotným sdílením informace).

  - Tradiční herní mechanikou pro komunitně založené hráče je možnost vypomoci někomu jinému za patřičnou odměnu a nebo třeba spolupracovat na společném cíli.

- The Killer

  - Motivace této poslední skupiny stojí čistě na pocitu nadřazenosti nad ostatními hráči. V prostřední kompetitivních online her se jedná o ty hráče, kterým nejde tak o to vyhrát, jako spíš porazit soupeře. Nestačí jim pocit z toho, že oni jsou dobří, ale ještě potřebují vědět, že ostatní jsou horší.

  - Jedná se o staticky nejmenší skupinu (odhad okolo 1 %) a nejlehčí cestou vedoucí k naplnění jejich motivace jsou žebříčky.

5.5 Konkrétní gamifikační prvky a jak je využít

Po seznámení se se základními principy je dalším krokem pohled na konkrétní mechaniky v kontextu KZ (Kytarového zpěvníku). Nejprve uvedu ty, které jsem označil za „základní“ a které považuji za vhodné minimum a pak uvedu i „pokročilé“ nápady, které budou náročnější na implementaci a provedení, ale také dle mého názoru stojí za zvážení. U každého prvku krom popisu fungování zmiňuji i způsob provedení a to, jak přesně by mohl být aplikaci prospěšný.

5.5.1 Základní prvky

5.5.1.1 Body a úrovně

Body jsou alfou a omegou každého systému, který se rozhodne využívat prvky gamifikace. Jsou totiž nejlehčí na implementaci a také na design – stačí si vzít inspiraci ze skoro jakéhokoli RPG (Role Playing Game) a nebo většiny věrnostních programů. Ostatně v širším slova smyslu by se i peníze daly považovat za „body“, podle kterých by se dali lidé porovnávat, kdyby stav účtu byl veřejnou informací.

Každá z akcí ze sekce „Žádoucí chování uživatele“ by měla být bodově ohodnocena. Za základní akci se dá požadovat přidání písně do systému – Této akci přidělíme hodnotu 10 bodů a všechny další akce budou ohodnoceny podle vztahu k této. Můj první návrh přidělování bodů je shrnut v tabulce 5.1.

Krom bodů je vhodné (po vzoru různých RPG her) implementovat i úrovně uživatelů, které se budou od získaných bodů odvíjet. Úroveň uživatele bude reprezentována tématickou ikonkou vedle jména uživatele (popřípadě třeba barvou jména). Na první pohled by tak každý mohl poznat, jestli je třeba autor akordů písničky zkušený veterán (volba slova není náhodná, ostatně se jedná o takovou obdobu insignií v armádě) či nováček.

Důležité je zvolit vhodné rozestupy jednotlivých úrovní. Nesmí se dát získat příliš lehce, aby neztratily na váze, zároveň ale nesmí působit nedosažitelně. Jako počáteční hranici pro získání následující úrovně mi přijde vhodné 25 bodů (tj. přidání dvou písní / pozvání jednoho kamaráda a pár hodnocení). Mé návrhy na další úrovně shrnuje tabulka 5.2.

Musí existovat nějaká maximální úroveň, v případě vývoje komunity by zároveň ale nemělo být problémové vymyslet další úrovně se stejným tématem.

5.5.1.2 Odznaky

Přestože se často skrývají za spoustou alternativních názvů (třeba achievementy/trofeje/medaile), lze je najít skoro v každé gamifikované aplikaci. Jedná se o barevné odznaky získané za určité unikátní činnosti, popřípadě za provedení nějaké činnosti několikrát. Protože se jedná o prvek cílený převážně na Achievery, nesmí chybět možnost se získanými odznaky pochlubit – Uživatel by měl na profilu jiného uživatele vidět výběr několika nejlepších odznaků (každý by si sám nastavil, jakými odznaky se chce prezentovat) a také by měl mít možnost si prohlédnout všechny úspěchy, kterých daný uživatel dosáhl.

Těmi nejméně nápaditými (ale neméně důležitými) jsou odznaky za vykonávání nějaké z žádoucích činností. Tyto odznaky můžou existovat v několika odstupňovaných variantách, které se můžou graficky odlišovat třeba jen barvou pozadí. Například pro první stupeň bude třeba vložit do systému 5 písní, pro druhý 10, pro třetí 25 a pro čtvrtý 50.

Odznaky zároveň nabízejí velkou volnost, a tak není nutné zůstat u odstupňovaných základních akcí. Uživatel může být oceněn třeba za to, že web

Tabulka 5.1: Návrh akcí a jejich bodového ohodnocení

Akce	Získané body
Přidání nové veřejné písně. Pokud bude píseň označena za neveřejnou, budou body odebrány.	10 bodů • +10 pokud píseň v systému veřejně ještě nebyla • +2 za každého uživatele, co ji měl na seznamu přání
Má píseň obdržela 20+ hodnocení a měla alespoň jednou průměr nad 3,5.	20 bodů
Má píseň obdržela 20+ hodnocení a měla alespoň jednou průměr nad 4.	40 bodů
Má píseň obdržela 20+ hodnocení a měla alespoň jednou průměr nad 4,5 (je možné získat pouze nejvyšší z těchto tří odměn).	60 bodů
Přidání nového veřejného zpěvníku (alespoň 5 písní). Systém s hodnocením stejný jako u písní.	10 bodů
Udělení hodnocení písně/zpěvníku.	1 bod • +1 pokud je k hodnocení přiložen komentář
Nahlášení obsahu, který bude posléze odebrán.	5 bodů
Pozvání nového uživatele.	20 bodů
Sdílení na sociálních sítích.	5 bodů • alternativou je po vzoru mobilních her nabídnout zdvojnásobení získaných bodů při sdílení na sociální síti
Přihlášení se do aplikace.	5 bodů • +1 navíc za každý den, kdy se uživatel přihlásil v řadě, do maxima 20 bodů za přihlášení za den

Tabulka 5.2: Návrhy získatelných úrovní

Úroveň	Potřebný počet bodů	Ikona
1.	0	Žádná
2.	25	Celá nota
3.	60	Půlová nota
4.	100	Čtvrťová nota
5.	150	Dvě spojené čtvrťové noty
6.	250	Osminová nota
7.	500	Šestnáctinová nota
8.	750	32tinová nota
9.	1000	Trsátko
10.	1500	Trsátko s korunou

navštíví 10 dnů v řadě a nebo třeba za strávení více než hodiny s otevřenou jednou písní (to si přímo říká o odznak s názvem „Let me try it just one more time“). Při designování odznaků je ale třeba neustále myslet na to, že není žádoucí pro každý z nich manuálně programovat nespočet nových funkcí. Extrémním příkladem takových jednorázových odznaků jsou pak skryté odznaky, ze kterých by uživatel znal pouze název (samozřejmě musí existovat obrazovka, kde si uživatel bude moci prohlédnout ještě nezískané odznaky a jejich procento splnění) a musel sám přijít na to, jak ho získat. Takový prvek bude pro změnu vyhovovat uživatelům, co rádi objevují.

Všechny odznaky musí mít vhodný název a být konzistentní s tématem webu. Osobně se mi třeba líbí myšlenka toho, že by měly tvar trsátka určité barvy, na kterém vždy bude nakreslena jednoduchá ikona symbolizující splněnou výzvu.

5.5.1.3 Žebříčky

Když už uživatel disponuje nějakým vysokým počtem bodů, je pochopitelné, že se bude chtít pochlubit světu. Aplikace by tak určitě nějaké žebříčky obsahovat měla a to nejen nějaké podrobné, ale i jejich menší verze nejradši přímo na domovské stránce. Podle výzkumu od společnosti Microsoft stráví 90 % uživatelů na stránce průměrně pouhých deset vteřin [25] [26], než se rozhodnou, zdali na stránce zůstanou či nikoliv. A protože systém gamifikace bude jedním z hlavních taháků vůči konkurenci, bylo by vhodné ho nechat patřičně na očích.

Tradiční top 100 žebříček uživatelů s nejvíce body je samozřejmostí, ale pokud by se jednalo o jediný druh žebříčku v aplikaci, neměl by veliký efekt. Vždyť pro jakéhokoli nově příchozího by působil demotivačně – jak by také mohl nový uživatel konkurovat těm, kteří začali aktivně sbírat body již před několika měsíci. Proto je třeba zobrazovat i nějaké porovnání v kratším ča-

sovém úseku, než je celá životnost aplikace. Vhodnou alternativou jsou třeba žebříčky uživatelů s nejvíce body za poslední kalendářní týden, jako například na webu Karaoke texty. Tak má každý uživatel (včetně těch nejnovějších) šanci si vybojovat své místo na výsluní.

Posledním důležitým druhem žebříčku je takový, který je místo na nejlepší uživatele centrován na přihlášeného uživatele a dalších deset uživatelů těsně nad ním a těsně pod ním. Tato obdoba pojmenovaná *User in the middle* je poslední dobou velmi populární – Uživatel má díky tomuto přehledu možnost jednoduše vidět své „soupeře“ a potřebný bodový zisk pro posun v tabulce je tak transparentní.

Nejefektivnější variantou vůbec je zobrazení žebříčku uživatele ve vztahu k jiným uživatelům, které označil za přátele. Tato funkce zatím v Kytarovém zpěvníku není, ale pokud se někdy v budoucnu objeví, příslušný žebříček by neměl chybět.

5.5.2 Další návrhy pokročilých prvků a rozšíření aplikace

5.5.2.1 Uvítací prohlídka a indikátory postupu

Aplikace začíná být docela rozsáhlá a není možné očekávat, že by majorita uživatelů skutečně četla uživatelskou příručku. Mnohem lepším způsobem úvodu nových uživatelů do systému by byl interaktivní tutoriál (alternativou je pouhý odškrtávací seznam úkolů na profilu uživatele), který by nováčky provedl základními funkcemi aplikace a rovnou je bodově odměňoval za různé úkony (uživatel tak zjistí, za jakou akci je kolik bodů).

Když totiž uživatel bude hned od začátku uveden do aplikace a zároveň za to bude pochválen a odměněn body, je mnohem menší šance, že se v Kytarovém zpěvníku nedokáže zorientovat a odejde. Viditelný procentuální indikátor kompletace této „virtuální prohlídky“ ho navíc bude neustále motivovat k tomu, aby splnil všechny úkoly. Podobnou mechanikou používá web Ultimate-guitar.com, ovšem pouze ohledně vyplnění uživatelského profilu. Interaktivní tutoriály jsou také často k vidění u moderních mobilních her.

Návrhy na položky pro úplné pochopení aplikace:

- Založení zpěvníku a přidání první písně do něj
- Ohodnocení cizí písně
- Napsání komentáře
- Vyplnění vlastního profilu
- Přidání vlastní první písně
- První export zpěvníku
- Přidání a převzetí písně

5.5.2.2 Karma

Další z možností rozšíření systému je použití gamifikačního prvku karmy. Její funkcionalita spočívá v tom, že uživatel může příspěvku jiného uživatele přidělit +1 nebo -1 a na základě všech hodnocení je pak vypočítané celkové ohodnocení příspěvku. Nejznámějším příkladem použití tohoto mechanismu je stránka Reddit a v Kytarovém zpěvníku by prvek karmy získal využití hlavně u hodnocení jednotlivých komentářů. Na základě toho by se daly komentáře řadit podle kvality a také by se na jejich karmu daly navázat nějaké odznaky a nebo systém automatického mazání nevhodných komentářů.

5.5.2.3 Denní úkoly

Po vzoru her by bylo možné pro aplikace vytvořit systém denních úkolů. Uživatel by každý den o půlnoci dostal přidělený nový úkol a za jeho splnění by byl krom běžných bodů za provedení akce odměněn i body bonusovými. Při nesplnění úkolu by byl úkol o půlnoci nahrazen úkolem novým. Tento prvek dává velkou designovou volnost při navrhování a tak by se úkoly od obecných zadání „Napiš 5 komentářů“ či „Ohodnoť tři písně“ mohly stát i více specifičtějšími „Ohodnoť 3 písně od Imagine Dragons“ nebo „Napiš komentář delší než 200 znaků“.

Hlavním cílem úkolů by bylo poskytnout uživatelům aktivně pronásledujícím body další způsob, jak je získat, který zároveň bude ovšem zábavný a hlavně nebude stereotypní.

5.5.2.4 Vybrané písně/zpěvníky

Jednou z věcí, které většinou lidé chtějí, je sláva a pozornost. Ta by se alespoň do určité míry dala uživatelům Kytarového zpěvníku poskytnout následovně – Administrátoři by každý týden/měsíc sestavili seznam nejlepších pěti písní a zpěvníků a ty by pak do sestavení dalšího výběru byly zobrazeny na hlavní stránce. Autoři těchto písní/zpěvníků by krom místa ve středu dění získali i odznaky, které budou symbolizovat jejich úspěch. Podobné mechanismy moderovaných výběrů je možné vysledovat napříč celým internetem, používá je třeba crowdfundingová platforma Kickstarter [27] a nebo obchod Google Play [28].

5.5.2.5 Sociální možnosti

Jak vyplývá z dříve zmíněné taxonomie typů hráčů, tak socializace je hlavní odměnou pro nejednoho člověka na internetu. Aplikace ale aktuálně umožňuje pouze psaní komentářů a tam veškeré možnosti končí. Je zřejmé, že se v první řadě nejedná o sociální platformu, alespoň o pár rozšířeních do budoucna by se ale uvažovat dalo.

Základem by měl být systém přidávání kamarádů, kdy uživatel bude mít možnost sledovat aktivitu ostatních vybraných uživatelů. Tím pádem mu například neuniknou nové písně od jeho oblíbeného uživatele. Ruku v ruce se systémem sledování jde i možnost zasílání soukromých zpráv, aby se Kytarový zpěvník skutečně stal místem, kde je možné si najít nové přátele. Dalšími možnostmi rozšíření jsou systémy pozvánek a nebo zájmových skupin, do kterých by se uživatelé mohli sdružovat.

5.5.2.6 Alternativní odměny

Podle Gabe Zichermanna, autora několika knih o gamifikaci, se dají odměny pro uživatele klasifikovat do čtyř kategorií, takzvaných *SAPS* [29] – Status, Access rights (přístup), Power (moc) a Stuff (fyzické věci). Každá z kategorií má přitom větší váhu než další, které následují (sociální status je nejlepší motivací). Návrh gamifikace pro Kytarový zpěvník už odměňuje ve formě sociálního statusu. Reálné odměny rovnou nepřipadají v úvahu, dokud produkt nebude nějak monetizován. Co ale zbylé dvě kategorie?

Přístup je definován jakožto příležitost dostat se k nějakým informacím, kontaktům či zdrojům, ke kterým ostatní uživatelé přístup nemají. Online obchod Gilt například svým nejvěrnějším zákazníkům umožňoval začít prohlížet zboží o 15 minut dříve, než ostatním [30]. V kontextu Kytarového zpěvníku by se dal přístup promítnout pouze v případě, že by nějaké písně byly exkluzivní a skryté. Protože ale hlavně zpočátku je cílem nabídnout nově příchozím co nejvíce obsahu, nejedná se o použitelnou variantu.

Co se moci týče, tak zajímavý přístup používá webová aplikace Ultimate-guitar.com. Ověření uživatelé splňující určité podmínky jsou pasování na jakési sub-správce a postupně získávají částečné administrátorské možnosti uvnitř systému. Například tak můžou dostat schopnost upravovat metadata cizích písní a posléze i jejich text a akordy. Jejich hodnocení má také větší váhu a uživatelé s největším stupněm privilegií můžou dokonce mazat nevhodné komentáře a nebo označovat písně za neveřejné.

Kapitola 6

Závěr

Cílem celé práce bylo dostat po několika letech aplikaci Kytarový zpěvník do stavu, kdy bude v souladu s moderními technologiemi a alespoň do určité míry konkurenceschopná. Začínal jsem analýzou její současné podoby a zdůvodněním, proč je třeba celý systém modernizovat, poté jsem vypracoval návrh přechodu na novou achitekturu a následně vytyčené kroky podniknul. Kromě tohoto tématu se má práce zabývala ještě průzkumem fenoménu gamifikace u podobně zaměřených webových aplikací a možností, jak můžou herní prvky pomoci vstupu aplikace na trh.

Krom modernizace architektury a technologií aplikace došlo i k dalším vylepšením v podobě automatických testů a nových funkcionalit. Všech předem stanovených cílů jsem dosáhl a výsledkem je aplikace ve verzi 2.0, včetně potřebných příruček. Stav aplikace není perfektní a je na ní ještě co zlepšovat (například veškeré konkrétní texty v kódu přesunout do externích souborů pro snadnou lokalizaci v budoucnu), i tak ale Kytarový zpěvník dosáhl radikálně lepšího stavu, než kdy dříve.

Výsledek této bakalářské práce poslouží jako dobrý startovní bod pro další týmy, které budou na aplikaci do budoucna pracovat. Na základě provedených změn pro ně bude mnohem snažší aplikaci dále vyvíjet a výsledky mého průzkumu gamifikace budou moci posloužit jako podklady pro další rozvoj. Pokud o to bude vedoucí práce aktivně usilovat, neměl by být problém aplikaci do jednoho roku nasadit do ostrého provozu a získat jí uživatelskou základnu.

Literatura

[1] Mantlík, J.: Rozšíření aplikace Kytarový zpěvník. Praha, 2016. 73 s. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Zdeněk Rybola.

[2] Havlíček, T.: Kytarový zpěvník pro Android - Frontend. Praha, 2016. 67 s. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.Vedoucí práce Ing. Zdeněk Rybola.

[3] Melingerová, G.: Kytarový zpěvník pro Android. Praha, 2016. 51 s. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Zdeněk Rybola.

[4] Přepracování UI aplikace Kytarový zpěvník s ohledem na UX. Praha, 2016. Nevydáno. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Zdeněk Rybola.

[5] PHP: Hypertext Preprocessor. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: `http://php.net`

[6] Nette Foundation; 2008, c.: Rychlý a pohodlný vývoj webových aplikací v PHP | Nette Framework. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: `https://nette.org`

[7] Doctrine, The Open-Source PHP ORM and Persistence Tools Project. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: `https://www.doctrine-project.org`

[8] Composer. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: `https://getcomposer.org`

[9] Dart programming language. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: `https://www.dartlang.org`

[10] Dartium: Chromium with the Dart VM. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: https://webdev.dartlang.org/tools/dartium

[11] Team, T. C. L.: Getting started | Less.js. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: http://lesscss.org

[12] Mark Otto, Jacob Thornton, and Bootstrap contributors: Bootstrap. Jan 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: https://getbootstrap.com

[13] jQuery Foundation - jquery.org: jQuery. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: http://jquery.com

[14] npm. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: https://www.npmjs.com

[15] pub get. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: https://www.dartlang.org/tools/pub/cmd/pub-get

[16] Documentation Group: Welcome! - The Apache HTTP Server Project. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: https://httpd.apache.org

[17] WAMP, MAMP and LAMP Stack : Softaculous AMPPS. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: http://www.xampp.net

[18] Stack Overflow - Where Developers Learn, Share, & Build Careers. Apr 2018, [Online; accessed 29. Apr. 2018]. Dostupné z: https://stackoverflow.com

[19] Martin Fowler: *Destilované UML*. Praha: Grada, 2009. s. 51.

[20] Prensky, M.: DIGITAL GAME-BASED LEARNING: Chapter 2. Nov 2011: str. 3, [Online; accessed 8. May 2018]. Dostupné z: https://bit.ly/2rsPYLr

[21] McEachern, A.: Loyalty Case Study: Starbucks Rewards. Apr 2018, [Online; accessed 18. Apr. 2018]. Dostupné z: https://blog.smile.io/loyalty-case-study-starbucks-rewards

[22] Nike Fuel Case Study - DigitalAdBlog. *DigitalAdBlog*, Apr 2018, [Online; accessed 18. Apr. 2018]. Dostupné z: http://digitaladblog.com/2013/04/11/nike-fuel-case-study

[23] William Shakespeare: Kupec benátský. Praha: TORST, 1999. III. I. Překlad Martin Hilský.

[24] Richard A. Bartle: Players Who Suit MUDs. Feb 2014, [Online; accessed 26. Apr. 2018]. Dostupné z: http://mud.co.uk/richard/hcds.htm

[25] Liu, C.; White, R. W.; Dumais, S.: Understanding web browsing behaviors through Weibull analysis of dwell time. Jul 2010, doi:10.1145/1835449.1835513.

[26] How Long Do Users Stay on Web Pages? *Nielsen Norman Group*, Apr 2018, [Online; accessed 26. Apr. 2018]. Dostupné z: `https://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages`

[27] Discover ≫ Projects We Love — Kickstarter. Apr 2018, [Online; accessed 26. Apr. 2018]. Dostupné z: `https://www.kickstarter.com/discover/recommended?ref=category_modal&sort=popularity`

[28] Naše tipy v kategorii Aplikace pro Android – Aplikace pro Android ve službě Google Play. Apr 2018, [Online; accessed 26. Apr. 2018]. Dostupné z: `https://play.google.com/store/apps/collection/featured`

[29] Zichermann, G.: Cash is for SAPS. *Gamification Co*, Oct 2010. Dostupné z: `http://www.gamification.co/2010/10/18/cash-is-for-saps`

[30] Top 10 eCommerce Gamification Examples that will Revolutionize Shopping. Apr 2018, [Online; accessed 26. Apr. 2018]. Dostupné z: `http://yukaichou.com/gamification-examples/top-10-ecommerce-gamification-examples-revolutionize-shopping`

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

BE Backend

BI-SP1 Předmět Softwarový projekt 1

BI-SP2 Předmět Softwarový projekt 2

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

DAO Data Access Object

DI Dependency Injection

DQL Doctrine Query Language

FE Frontend

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

IT Informační technologie

JSON JavaScript Object Notation

KZ Kytarový zpěvník

LESS Leaner Style Sheets

MMORPG Massive Multiplayer Online Role Playing Game

MVC Model View Controller

MVP Model View Presenter

ORM Object Relational Mapper

PDF Portable Document Format

PHP Hypertext Preprocessor

REST Representational State Transfer

SAPS Status Access Power Stuff

SQL Structured Query Language

UML Unified Modeling Language

XAMPP Cross-platform, Apache, MariaDB, PHP, Perl

Příloha B

Obsah přiloženého CD

- `readme.txt` ... stručný popis obsahu CD
- `release` ... adresář obsahující výsledný release aplikace ve verzi 2.0
  - `releaseGuide_2.0.pdf` .. příručka s instrukcemi pro nasazení aplikace
  - `sql` ... složka s SQL skripty pro vytvoření a naplnění databáze
  - `kytarovy-zpevnik-release-v2-0` ... soubory nasazení na server
- `src` ... zdrojové kódy implementace
  - `devGuide_2.0.pdf` ... příručka pro vývojáře
  - `app` ... složka s jádrem aplikace
  - `data` ... složka s SQL skripty se základními daty
  - `log` ... složka se zaznamenanými chybami
  - `temp` ... složka pro ukládání dočasných souborů
  - `test` ... složka s testy aplikace
    - `testConfig.neon` ... konfigurační soubor pro testování
    - `run-api.sh` ... skript pro spuštění testů API
    - `run-model.sh` ... skript pro spuštění testů Service tříd
  - `coverage-services.html` . soubor s aktuálním pokrytím kódu Service tříd unit testy
  - `vendor` . složka se závislostmi aplikace nainstalovanými skrz Composer
  - `node_modules` .... složka se závislostmi aplikace nainstalovanými skrze Yarn
  - `www` ... jediná složka přístupná z webu
    - `index.php` ... vstupní bod do aplikace
- `src_thesis` ... zdrojová forma práce ve formátu LaTeX
  - `build` ... složka s výslednými soubory práce
    - `BP_HOLAN_JAN_2018.pdf` ... text práce ve formátu PDF
    - `BP_HOLAN_JAN_2018.tex` ... text práce ve formátu LaTeX
  - `chapters` .... složka s jednotlivými kapitolami práce ve formátu LaTeX
  - `build.bat` ... skript pro zkompilování práce do formátu PDF
  - `clean.bat` ... skript pro vyčištění adresáře
  - `FITthesis.cls` ... šablona BP
  - `header.tex` ... hlavička BP
  - `footer.tex` ... zápatí BP
  - `bibliographysources.bib` ... soubor s použitými citacemi
- `appendix`
  - `database.png` ... kompletní databázový model