FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Aplikace pro distribuci not v orchestru
Student:	Tomáš Vahalík
Vedoucí:	Ing. Josef Pavlíček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Navrhněte a implementujte aplikaci pro distribuci not v orchestru dle níže uvedených požadavků:
- bude sloužit pro přípravu na koncerty a zkoušky,
- bude rozlišovat 2 druhy uživatelů – administrátory a běžné členy,
- noty budou perzistentně uloženy v databázi,
- administrátor bude v aplikaci vytvářet „události" (koncert, zkouška, ...) ,
- administrátor bude moci provádět CRUD operace s notami,
- členové orchestru mají v aplikaci přidělené nástroje,
- aplikace bude zobrazovat seznam nadcházejících událostí,
- detail události bude reflektovat nástroje (a noty) příslušného uživatele,
- aplikaci bude napsaná v jazyku Java, pro frontend bude použit framework Vaadin.

Postupujte v těchto krocích:
1. Formalizujite a kompletujte funkční a nefunkční požadavky.
2. Zpracujte stručnou rešerši obdobných nástrojů.
3. Na základě analýzy požadavků navrhněte vlastní řešení.
4. Řešení implementujte, zdokumentujte a otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 9. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Aplikace pro distribuci not v orchestru

Tomáš Vahalík

Katedra softwarového inženýrství
Vedoucí práce: Ing. Josef Pavlíček, Ph.D.

12. května 2019

Poděkování

Děkuji panu doktorovi Pavlíčkovi, vedoucímu mé práce za pomoc při vytváření této aplikace a za poskytnutí serveru, na kterém mohla být nasazena. Dále děkuji Monice Ungrové, Václavu Šolcovi a Elišce Hryzlíkové za poskytnutí rozhovorů potřebných pro rešeršní část této práce.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 12. května 2019 ……

Odkaz na tuto práci

Vahalík, Tomáš. *Aplikace pro distribuci not v orchestru*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zabývá implementací webové aplikace, která pomůže členům orchestru připravovat se na koncerty tím, že umožní snadno distribuovat noty mezi jednotlivé hudebníky.

Budu zkoumat, jak se noty mezi členy orchestru sdílejí v současné době a identifikuji problémy, které tyto postupy mají. Na základě této rešerše navrhnu funkční požadavky, které aplikace bude splňovat, aby nedostatky současného stavu odstraňovala.

Aplikaci následně implementuji a otestuji v provozu. Budou ji používat členové Pražského filmového orchestru k přípravám na koncerty.

Klíčová slova Java, webová aplikace, noty, orchestr

Abstract

This thesis deals with the implementation of a web application that helps orchestra members to prepare for concerts by allowing them to easily distribute sheets among individual musicians.

I will examine how the sheets between the members of the orchestra are currently shared and identify problems, that these solutions bring. Based on

them I will describe functional requirements that the application will support to eliminate problems of the current state.

Then the application will be implemented and tested in production. It will be used by the members of the Prague film orchestra for preparations for the concerts.

Keywords Java, web application, sheet, orchestra

Obsah

Seznam obrázků

Úvod

Tématem mé bakalářské práce je aplikace pro distribuci not v orchestru. Noty jsou pro orchestr zásadním pracovním nástrojem, bez včasné a přesné distribuce not orchestr nemůže plnit svůj účel. Jednou z důležitých rolí v orchestru je archivář not. Ten se stará o to, aby každý člen orchestru měl noty, které potřebuje pro svůj nástroj na plánované koncerty. Archivář musí na základě programu koncertu připravit noty pro všechny skladby a všechny nástroje. Ty se potom doručí všem účinkujícím. Je také zodpovědný za to, aby noty existovali ve více exemplářích pro případ, že se některé ztratí. Práce archiváře prováděná ručně je tedy velmi náročná, časově i organizačně. V dnešní době lze noty uložit do databáze a kategorizovat je podle skladeb a nástrojů. Tato činnost se dá tedy zautomatizovat.

Cílem této práce je navrhnout a implementovat systém, který usnadní distribuci not jednotlivým členům. Noty budou uloženy v elektronické podobě. Budou tak kdykoli k dispozici k vytisknutí a budou také zálohovány. Archivář nebude muset posílat noty hudebníkům jednotlivě. Bude stačit, když do systému zadá, jaké skladby jsou na programu. Každému hudebníkovi systém zobrazí pro daný koncert seznam not pro jeho nástroj. Toto řešení ušetří práci a čas archiváři. Hudebníci se budou moci připravovat na koncert efektivněji, protože nebudou muset čekat, až jim archivář předá všechny noty osobně.

Toto téma jsem si zvolil, protože je mi velmi blízké. Jsem členem Pražského filmového orchestru, kde dosud noty distribuujeme ručně a zefektivnění této činnosti pro mě bylo výzvou.

Kapitola 1

Cíl práce

Hlavním cílem této práce je vytvořit webovou aplikaci, která pomůže členům orchestru připravovat se na koncerty tím, že umožní snadno distribuovat noty mezi jednotlivé hudebníky. K tomu bude potřeba splnit následující dílčí cíle:

1. Seznámit se s technikami, které se v orchestrech používají pro distribuci not mezi jednotlivé členy v současné době.
2. Identifikovat problémy současného stavu.
3. Seznámit se s technologiemi vývoje enterprise webových aplikací v jazyce Java
4. Na základě rešerše navrhnout funkční požadavky aplikace tak, aby nedostatky současného stavu odstraňovala.
5. Aplikaci implementovat a otestovat v provozu.

Kapitola 2

Teoretická část

2.1 Rešerše existujících řešení

Zde se budu zabývat postupy, které se používají v orchestrech k distribuci not dnes.

2.1.1 Pražský filmový orchestr

V Pražském filmovém orchestru se pro sdílení not používá disk Google, na kterém má archivář také zálohovánu většinu not (některé noty jsou však archivovány pouze v papírové podobě). Pokud hudebníkovi před koncertem chybí nějaké noty, musí si je vyhledat ve zkušebně, kde jsou noty uskladněny. Může také napsat archiváři a ten pro něj vytvoří na Google disku speciální složku s notami, kterou se členem orchestru potom sdílí.

2.1.2 Smyčcový orchestr Primavera–Hradec Králové

V tomto orchestru se o archivování not stará sám dirigent. Noty existují pouze v papírové podobě, elektronicky uloženy nejsou. Pokud člen orchestru potřebuje nějaké noty, vyžaduje to osobní setkání s dirigentem. Ten originály nakopíruje a předá žadateli.

2.1.3 Orchestr ZUŠ Jižní město

Zde má archivář noty uložené na počítači lokálně na disku. Pokud člen orchestru potřebuje sehnat nějaké noty, musí osobně požádat archiváře o jejich vytisknutí. Budou mu pak předány osobně na zkoušce.

2.1.4 Existující aplikace

Zkoumal jsem, zda nějaký software specializující se na distribuci not existuje. Objevil jsem aplikaci **forScore**. Ta umožňuje archivovat a sdílet noty v digi-

tální podobě. Je určena pro hudebníky, kteří používají tablety pro zobrazování not. Umožňuje jim noty popisovat, otáčet stránky klepnutím na obrazovku a mnoho dalších věcí. Nicméně aplikace se nespecializuje na usnadnění distribuce not mezi větší množství členů, sdílení pořád vyžaduje osobní přístup ke každému členovi – vytvoření a odeslání složky každému hudebníkovi [1]. Navíc hudebníků, kteří při vystoupení hrají z elektronických zařízení místo not je zatím menšina.

2.2 Přehled použitých technologií a postupů

2.2.1 Třívrstvá architektura

Třívrstvá architektura je návrhový vzor rozdělující funkčnosti aplikace na tři spolupracující moduly (vrstvy). Tato architektura umožňuje každou vrstvu vyjmout a nahradit bez nutnosti zásahu do vrstev ostatních – například změna uživatelského rozhraní neovlivní výpočty, které aplikace provádí. V třívrstvé architektuře se objevuje vrstva **datová**, **Aplikační** a **Klientská**.

Datová vrstva zajišťuje ukládání, zpřístupnění a konzistenci dat. Vystavuje API, které umožňuje pracovat s daty bez nutnosti vytváření závislostí v ostatních vrstvách. Nejčastěji tuto vrstvu tvoří databázové systémy, v nichž jsou data uložena.

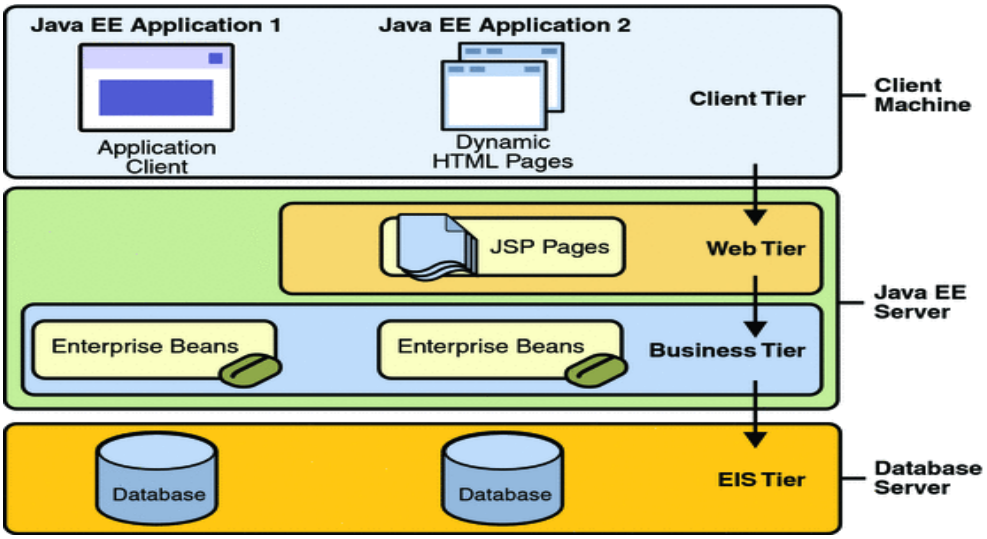
Aplikační vrstva (též „Business“ vrstva) obsahuje většinu funkčností aplikace a manipulaci s daty. Přijímá požadavky od klientské vrstvy a na jejich základě získává a zpracovává data z vrstvy datové. U webových aplikací tuto vrstvu tvoří komponenty nasazované na aplikační server.

Prezentační vrstva (též „Klientská“ vrstva) se stará o zobrazení aplikace klientovi. Zpracovává uživatelovy požadavky a předává je aplikační vrstvě. Výsledky, které obdrží následně prezentuje uživateli. Ve webových aplikacích je tato vrstva tvořena obsahem generovaným webovými prohlížeči. [2]

2.2.2 JavaEE (Enterprise Edition)

JavaEE je sada rozhraní API, které má za cíl usnadnit vývojářům tvorbu *Enterprise aplikací*. To je aplikace

- Robustní, bezpečná, rozšiřitelná a škálovatelná.
- Obsluhující velké množství klientů.
- Pracující s různě velkými a rozličnými daty uloženými v různých datových systémech.
- Komunikující s dalšími systémy pomocí definovaných rozhraní. [3]



Obrázek 2.1: Třívrstvá architektura podle JavaEE specifikace [4]

JavaEE využívá vícevrstvou architekturu. Aplikační logika je rozdělena na komponenty podle funkce. Tyto komponenty jsou instalovány na různých zařízeních v závislosti na vrstvě, do které patří. Na obrázku 2.1 je zobrazená architektura podle specifikace JavaEE. Přestože se na obrázku vyskytují čtyři vrstvy, považuje se architektura za třívrstvou. Komponenty jsou provozovány ve třech prostředích – databázový server, aplikační server a klientův počítač. Já ve své aplikaci budu využívat technologii JSP (Java Server Pages) pouze pro vytvoření úvodní obrazovky, pro zbytek uživatelského rozhraní využiju framework Vaadin.

Klientská vrstva může být naprogramována i v jiném jazyce než v Javě. Jsou dva druhy klientů – **Webový klient** a **Klientská aplikace**.

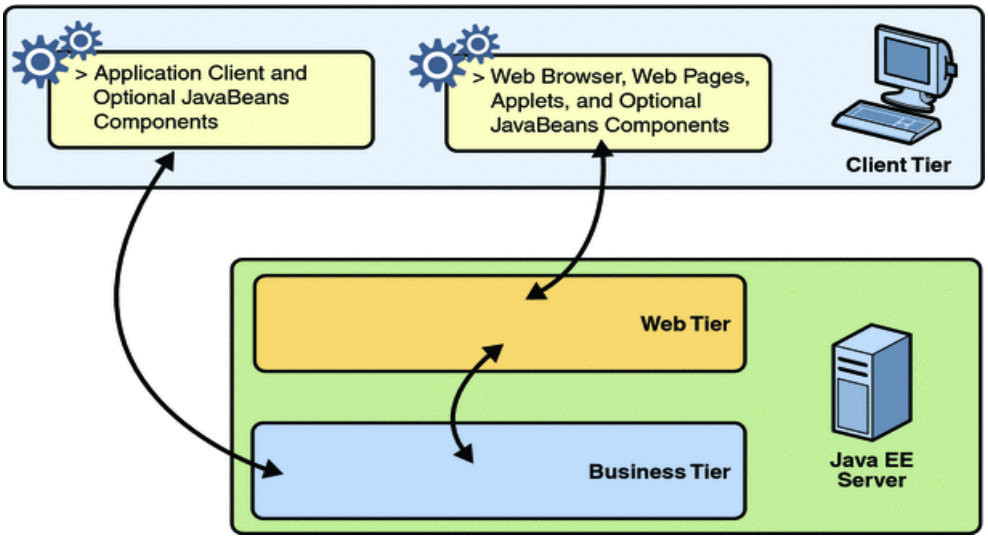
„*Webový prohlížeč komunikuje s webovou vrstvou pomocí HTTP/S protokolu. Webová vrstva následně interpretuje klientské požadavky business vrstvě, která je obsluhuje. Odpověď (response) pak využívá stejný protokol a případně sezení (session).*“[2] Takovému klientu se říká „tenký klient“.

Klientské aplikace oproti tomu mohou komunikovat přímo s Enterprise Java Beany a přeskočit tak vrstvu webovou. Pokud je to vyžadováno, může klientská aplikace komunikovat i s webovou vrstvou přes protokol HTTP. Takovému klientu se říká „Tlustý klient“.[4]

Na obrázku 2.2 je znázorněno, jak tyto druhy klientů komunikují se zbytkem aplikace.

Přehled technologií JavaEE použitých v této aplikaci:

Enterprise Java Beans (EJB) EJB je komponenta, která se stará o implementaci části obchodní logiky.



Obrázek 2.2: Druhy klientů [4]

```
import javax.ejb.EJB
import myPackage.MyBean
public class MyClass{
    @EJB
    MyBean bean;

    bean.method();
}
```

Výpis kódu 1: Získání reference na EJB

Podle [5] EJB můžeme dělit na **Session beans** a **Message driven beans**

Session beany rozdělujeme na stavové a bezstavové.

„Bezstavové beany neuchovávají stav relevantní pro klienta mezi obsluhou jeho jednotlivých požadavků. Pro obsloužení každého požadavku klienta je mu vždy na serveru přidělena samostatná instance, která je vyhrazena pouze tomuto klientovi v rámci jednoho daného požadavku – z tohoto důvodu jsou bezstavové beany přirozeně vláknově bezpečné. V rámci jednoho volání metody (požadavku) lze použít i datové atributy instance beanu, ale není zaručeno, že se jejich obsah při dalším volání nezmění. Bezstavové beany se na serveru obvykle sdružují v poolu, ze kterého jsou odebírány pro obsloužení požadavku a následně se do něj zpět vrací. Třída představující bezstavový bean musí být anotována `@Stateless`

Stavové beany oproti tomu uchovávají svůj vnitřní stav. Každému kli-

entovi je přidělena jedna stavová beana, která se pro daného klienta používá pro obsluhu všech jeho požadavků. V rámci životního cyklu stavového beanu může dojít k jeho pasivaci (uložení pomocí persistentní vrstvy) prováděnou kontejnerem za účelem uvolnění paměti na serveru. Třída představující stavový bean musí být anotována `@Stateful`"

Message driven beans jsou bezstavové beany umožňující asynchronně zpracovávat zprávy. Fungují podobně jako *action listener*, ale místo událostí přijímají JMS zprávy. Zprávu může poslat jakákoli Java EE komponenta, ale i aplikace, která nepoužívá technologii java.[6]

V této aplikaci používám pouze bezstavové session beany.

Pokud chceme v aplikaci získat referenci na EJB, nepoužíváme klíčové slovo `new`, ale využijeme dependency injection. Pokud označíme třídu anotací `@EJB`, můžeme k beaně přistupovat a volat její funkce (viz. ukázka kódu 1). [7]

JAX–RS Jax–RS je rozhraní usnadňující tvorbu RESTových webových služeb „*Webová služba je softwarový systém umožňující interakci dvou strojů na síti. Pro komunikaci s webovou službou se používá protokol SOAP nebo technologie REST. Protože ale firewally nedůvěřují příchozím zprávám, je třeba využít tzv. tunelování. Mechanismus tunelování funguje tak, že SOAP zpráva se zabalí do obecného protokolu (např. HTTP, SMTP nebo MQseries), kterému firewall důvěřuje. Po průchodu firewallem se příjemci dostane požadovaná zpráva.*"[5]

REST je způsob komunikace s webovou službou, který využívá protokolu HTTP. Pomocí anotací se k metodám webové služby přiřazují HTTP operace (GET, PUT, POST, DELETE).

V ukázce kódu 2 je ukázané, jak se používají webové služby s Jax–RS. Anotace `@Path` nad třídou upřesňuje, že třída bude webová služba a specifikuje její URI relativně vůči kořenové URI aplikace. URI lze dále rozšiřovat, aby byl zajištěn přístup k jednotlivým metodám. Zaslání HTTP požadavku GET na */targetPath* by tedy spustilo metodu *getAll*, kdybychom chtěli zavolat metodu *getSpecial*, musel by být požadavek zaslán na */targetPath/special*.

Jak se píše v [8] Pomocí závorek v anotaci `@Path` můžeme specifikovat argumenty pro volanou metodu, se kterými pak můžeme pracovat. Požadavek */targetPath/item/666* by tedy spustil metodu *getSingleItem* s argumentem *id=666*.

Bezpečnost O službách poskytovaných JavaEE ohledně bezpečnosti se více zmíním v sekci „Aplikační servery"

JNDI (Java Naming and Directory Interface) je API, které umožňuje programátorům vyhledávat data a zdroje na základě jejich jmen. V této

```
@Path("/targetPath")
public class MyService{

    @GET
    String getAllItems(){
        ...
    }

    @GET
    @Path("special")
    String getSpecialItem(){
        ...
    }

    @GET
    @Path("/item/{id}")
    String getSingleItem(@PathParam("id") int id){
        ...
    }

}
```

Výpis kódu 2: Použití restové webové služby

aplikaci se služba JNDI bude používat zejména k vyhledání JDBC zdroje a security realmu.

2.2.3 Aplikační servery

Protože je JavaEE pouze rozhraní, je potřeba jeho implementace. Knihovny, které implementují funkcionality specifikované JavaEE jsou označovány pojmem **aplikační server**. „*Kromě toho poskytuje aplikační server další klasické služby jako například administrátorskou konzoli, logování a podobně.*

Na aplikační server se nasazují takzvané komponenty, což jsou základní jednotky, ze kterých je složena výsledná aplikace. “[2]

Komponent existuje několik druhů, pro tuto aplikaci jsou důležité převážně **Webové komponenty** – servlety, JSP soubory a JSF soubory a **Enterprise JavaBeans (EJB) komponenty** - javovské třídy, které tvoří logiku aplikace a manipulují s daty (viz. sekce „JavaEE“).

„*Předtím než může být komponenta spuštěna, musí být přiřazena do takzvaného kontejneru. Každý kontejner spravuje komponenty určitého typu a poskytuje jim svou funkcionalitu. Funkcionalita jednotlivých kontejnerů je dána specifikací Java EE.*“ [2]

Aplikační server nám také poskytuje prostředky například pro konfiguraci datových zdrojů a zabezpečení aplikace.

2.2.3.1 Vytvoření datového zdroje

1. Administrátor od výrobce databáze získá JDBC driver a ten nainstaluje do AP serveru (často stačí jej nakopírovat do složky, ve které jsou sdílené knihovny).

2. Nastaví connection pool – nakonfiguruje: user, password, url, název databáze (a řadu dalších, výrobcem definovaných parametrů).

3. Nastaví jdbc zdroj a ten nasměruje na požadovaný connection pool

„*Tvůrce aplikace pak nezajišťuje připojení k databázi (user, password, port, url) ani neřeší zda connection bylo již vytvořeno, či bude teprve vytvořeno. Neřeší výkon databáze. Pro tvůrce je důležité uvést klíčový název datového zdroje, a JNDI služba již předá požadované připojí aplikaci sama.*“ [2]

2.2.3.2 Zabezpečení aplikace

Aplikace Java EE se skládají z komponent, které mohou obsahovat chráněné i nechráněné prostředky. Často je třeba chránit zdroje, aby k nim mohli přistupovat pouze oprávnění uživatelé. Aplikační servery nám se zabezpečením pomáhají. Lze vytvořit **security realm**, kde bude specifikována množina uživatelů. Těm budou následně přiřazeny role. (Ukázka vytvoření realmu bude v praktické části této práce). Při pokusu o přístup k chráněnému zdroji bude aplikace vyžadovat uživatelské jméno a heslo. Po zadání těchto údajů je informace předána aplikačnímu serveru. Ten údaje zkontroluje a povolí nebo zamítne přístup k chráněnému zdroji. [9]

2.2.4 Objektově relační mapování

Jak se píše v [10]: „*ORM je technika, která nám umožňuje mapovat objekty (používané v objektových programovacích jazycích) do relační databázové struktury. Přináší to efekt práce s „virtuální databází“, kde k datům uloženým v relačních strukturách přistupujeme jako k běžným objektům použitých v daném programovacím jazyce.*

Jádro celého problému mapování objektů na relační struktury tkví v tom, že relační datovou jednotkou je tabulka (entita), která je fakticky dvourozměrným polem–sloupce jsou domény a řádky jsou hodnotami těchto domén. Objekt je oproti tomu vícedimenzionální struktura (referenční datový typ). Ta, díky vlastnosti skládání, je téměř vždy složena z řady dalších (menších) objektů a elementárních (primitivních) datových typů. Takovou strukturu není možné

fakticky uložit do databáze jinak než jako celek (Oracle, Informix, DB2 umožňují ukládat objekty přímo do speciálního datového typu „Object“), to je ale z hlediska nezávislosti dat na vrstvě, která s daty pracuje nevhodné.

Abychom se vyhnuli komplikacím s uložením dat a jejich transformací, zavedeme mechanismus mapování relačních entit na objekty, které zachovává datovou bezpečnost (nemůže dojít ke změně dat v databázi, aniž by změna nebyla potvrzena a korektně dokončena díky transakčnímu zpracování) – je tedy perzistentní. “

V Javě se pro ORM používají knihovny implementující **JPA**(Java Persistence API), které poskytuje programátorské rozhraní pro mapování. Existuje několik implementací JPA, například knihovny EclipseLink, Hibernate nebo OpenJPA. V této aplikaci jsem se rozhodl použít technologii **EclipseLink**, protože se jedná o referenční implementaci JPA.

Mapování spočívá ve vytvoření entitních tříd. Instance entitní třídy reprezentuje jeden řádek v tabulce relační databáze, jednotlivé atributy představují sloupce tabulky. Pomocí anotací můžeme detailněji specifikovat, jakým způsobem má být mapování provedeno. Ukázka kódu pro vytvoření a používání entitní třídy bude v praktické části této práce.

Některé atributy entitní třídy nebudou používané často. Pro zlepšení výkonu můžeme specifikovat, že se budou z databáze načítat pouze data, která budou potřeba. Ostatní atributy se načtou až v okamžiku, kdy k nim bude vyžadován přístup. Této technice se říká **Lazy loading**. [11]

JPA také specifikuje vlastní dotazovací jazyk–**JPQL**. To je jazyk velmi blízký jazyku SQL, ale dotazuje se nad objektovým modelem místo databázového. Knihovna implementující JPA dotaz napsaný v JPQL před provedením přeloží do jazyka SQL. Ten už umí zpracovat databáze, v níž aplikace data ukládá. Chceme–li tedy změnit databázi, není potřeba dotazy přepisovat, budou správně přeloženy automaticky.[12]

Objektově relační mapování umožňuje programátorovi efektivně vytvořit datový model. Na základě entitních tříd, jejich atributů a vztahů mezi nimi se automaticky vygenerují databázové tabulky. Díky tomu je možné vytvořit datový model pouze pomocí programovacího jazyka, který vývojář používá.

2.2.5 Maven

Maven je nástroj usnadňující sestavení a správu projektů. Podporuje hlavně jazyk Java. Jeho tvůrci určili pět oblastí, které by měl Maven pokrývat.

1. Usnadnění procesu buildování
2. Jednotný systém buildování
3. Poskytování kvalitních informací o projektu
4. Poskytování direktiv pro „best practices“

5. Poskytnutí transparentního přidávání nových funkcí.

„ *Základním principem fungování Mavenu je popsání projektu pomocí Project Object Model (POM). Tento model popisuje softwarový projekt nejen z pohledu jeho zdrojového kódu, ale včetně závislostí na externích knihovnách. popisu procesu buildování a různých funkcí s tím spojených (jako je spouštění testů, sbírání informací o zdrojových kódech a podobně).*“ [13]

Project Object Model se zapisuje ve formátu XML. Klíčová část souboru popisující POM je seznam externích knihoven, na kterých projekt závisí (dependencies). Maven pak automaticky vyhledá a nainstaluje potřebné knihovny. Samotné vyhledávání probíhá v definovaných úložištích (repository). Vyhledávání závislostí je tranzitivní – společně s knihovnami které jsme definovali se stáhnou i jejich dependencies. Díky tomu stačí do souboru napsat seznam pouze těch knihoven, na kterých náš projekt závisí přímo.[14]

2.2.6 Vaadin

Vaadin je framework, který umožňuje vytvořit webové uživatelské rozhraní kompletně v jazyce Java, jako při vývoji běžné desktopové aplikaci. Kód uživatelského rozhraní je provozován na aplikačním serveru společně s business vrstvou, následně je přeložen do javascriptu a interpretován pomocí **GWT (Google Web Tooliktu)** v internetovém prohlížeči uživatele. Pro programátora to znamená, že se pro tvorbu uživatelského rozhraní nemusí učit javascript ani jiné webové technologie (HTML, CSS, …), vše může napsat v Javě.[15]

Vaadin je komponentně řízený framework. „*Obsahuje velkou sadu běžně používaných komponent pro tvorbu webových aplikací. K dispozici je seznam a online ukázky včetně zdrojových kódů. Komunita vývojářů přidává neustále nové komponenty a sdílí je s ostatními jako takzvané doplňky (add-ons). Využívá lazy loading (líné načítání). Data nejsou nahrána ihned, ale až když jsou potřeba. Tím je dosaženo rychlých odezev při komunikaci mezi klientem a serverem.*“ [16]

2.3 Shrnutí výsledků rešerše

Z rešerše vyplývá, že ve zkoumaných orchestrech se k distribuci not nevyužívá žádný speciální software. Někde jsou noty uloženy elektronicky a mohou být posílány e-mailem, ale převažuje archivace not v papírové podobě. Takové řešení má několik nevýhod. Pokud si členové orchestru berou noty z fyzických složek, snadno se ztratí přehled o tom, kolik výtisků od dané skladby je ještě k dispozici, a nebo jestli jsou už všechny rozebrané. Pokud potom přijde do orchestru nový člen, nemusí už potřebné noty ve skladišti najít. Je vhodné, aby noty byly uloženy alespoň v nějaké elektronické podobě. Pokud elektronická záloha není a fyzické kopie se ztratí, musí se noty znovu nakoupit a to stojí

výdaje. Rovněž elektronické uložení—třeba i v lokálním úložišti—umožňuje distribuci not způsobem, který nevyžaduje fyzické setkání.

Všechna zkoumaná řešení vyžadují osobní komunikaci s archivářem. Ten má potom hodně práce a nemá možnost noty distribuovat hromadně. I sdílení not přes Google disk vyžaduje osobní přístup ke každému hudebníkovi. Vzniká tak velké množství složek, které je pak potřeba promazávat.

Vytvoření aplikace, která se na danou problematiku specializuje tedy usnadní život jak archivářům not, tak hudebníkům. Ti budou mít noty před koncertem k dispozici dříve a tím pádem budou moci déle cvičit.

Kapitola 3

Analýza a návrh

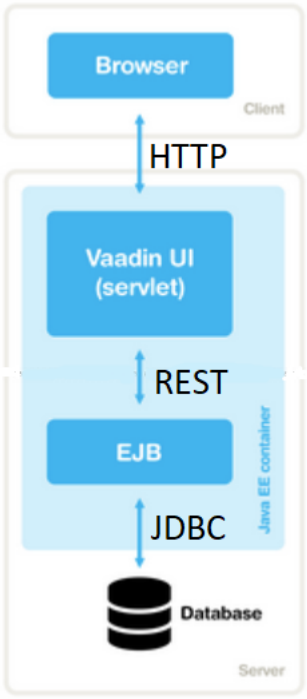
3.1 Návrh aplikace

Aplikaci naprogramuji pomocí technologií popsaných v teoretické části této práce.

Implementačním jazykem bude Java. Jako aplikační server jsem se rozhodl použít Glassfish, protože se jedná o referenční implementaci rozhraní JavaEE. Pro tvorbu většiny uživatelského rozhraní použiji framework Vaadin. Úvodní obrazovku obsahující přihlašovací formulář vytvořím pomocí technologií HTML a CSS. Jako datové úložiště bude použita databáze Derby. Pro objektově-relační mapování bude použita knihovna EclipseLink.

Bude použita třívrstvá architektura. Datovou vrstvu bude realizovat databáze Derby a entitní třídy. Business vrstvu budou tvořit Enterprise Java Beany (EJB), které budou manipulovat s daty z databáze. S datovou vrstvou budou komunikovat pomocí JDBC driveru. Business vrstva bude vystavovat RESTové rozhraní pro klientskou vrstvu. Klientskou vrstvu budou tvořit třídy volající RESTové služby, Vaadin servlety a třídy generující obsah webových stránek. Na obrázku 3.1 je znázorněno, jak spolu jednotlivé vrstvy komunikují.

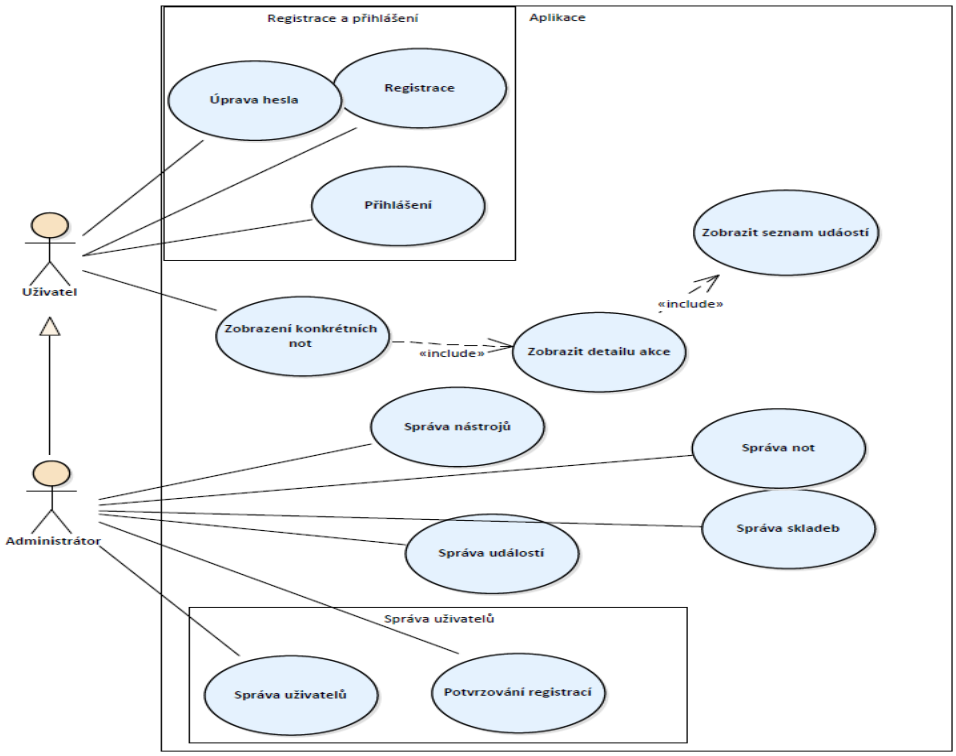
Všechny vrstvy aplikace jsou nasazeny na aplikačním serveru, klient pro používání aplikace potřebuje pouze webový prohlížeč.



Obrázek 3.1: Komunikace mezi vrstvami aplikace [17] (upraveno)

3.2 Analýza funkčních požadavků

Funkční požadavky jsem navrhnul tak, aby odstraňovaly problémy současných postupů distribuce not. Archivář nebude muset komunikovat s každým hudebníkem zvlášť, noty budou distribuovány hromadně. Noty budou uloženy v elektronické formě. Budou tak kdykoli k dispozici a nebude hrozit jejich ztráta. Hudebníci díky rychlé distribuci not budou mít více času se na koncert efektivně připravit. Funkční požadavky detailně zpracuji pomocí případů užití.



Obrázek 3.2: Model případů užití

Na obrázku 3.2 je vidět, že v aplikaci budou dva druhy uživatelů – běžný uživatel a administrátor. Běžný uživatel si bude moci zobrazit seznam nadcházejících akcí a u každé z nich noty pro svůj nástroj. Bude mu umožněno změnit si své heslo. Administrátor bude moci vykonávat vše, co běžný uživatel, ale bude mít k dispozici ještě další funkčnosti. Bude autorizován k provádění CRUD operací nad událostmi, skladbami a notami. Jeho zodpovědnost bude také schvalovat uživatelské registrace. Každému uživateli bude moci přiřadit nástroje, jejichž noty pak bude mít v aplikaci zpřístupněné. Bude také oprávněn k udělení administrátorských práv ostatním uživatelům.

3.2.1 Funkčnosti přístupné běžnému uživateli

UC1–Přihlášení

Cíl: Přihlášení registrovaného uživatele do aplikace.
Hlavní scénář
Výchozí podmínky: Uživatel je na obrazovce pro přihlášení.

1. Uživatel vyplní uživatelské jméno a heslo do příslušných políček.

2. Systém zkontroluje zadané údaje, uživatele přihlásí a zobrazí domovskou obrazovku. Pokud zadané údaje nejsou správné, zobrazí stránku vyzývající k opakování přihlášení.

UC2–Registrace

Cíl: Podání žádosti o přístup do aplikace
Hlavní scénář
Výchozí podmínky: uživatel je na obrazovce pro přihlášení.

1. Uživatel klikne na tlačítko „register“.
2. Systém zobrazí obrazovku pro registraci.
3. Uživatel vyplní své skutečné jméno, přihlašovací jméno, heslo a vybere z nabídky nástrojů ty, pro které bude chtít dostávat noty. Pokud příslušný nástroj není v evidenci, bude uživateli přidělen dodatečně.
4. Uživatelova žádost je nyní podaná, čeká na schválení od administrátora.

UC3–Změna hesla

Cíl: Uživatel změní své heslo pro přístup do aplikace.
Hlavní scénář
Výchozí podmínky: Uživatel je na obrazovce zobrazující jeho profil.

1. Uživatel klikne na tlačítko „Change password“.
2. Systém zobrazí formulář pro změnu hesla.
3. Uživatel do prvního zadá své současné heslo, nové heslo a pro kontrolu nové heslo ještě jednou a klikne na tlačítko „Confirm new password“.
4. Pokud je současné heslo správné a nové heslo bylo zadáno dvakrát stejně, systém uživateli heslo změní. V opačném případě ho upozorní na chybně zadaný údaj.

UC4–Zobrazení seznamu nadcházejících akcí

Cíl: Uživatel si zobrazí seznam nadcházejících akcí seřazený podle data.
Hlavní scénář
Výchozí podmínky: uživatel je na domovské obrazovce.
Seznam nadcházejících akcí je zobrazen na domovské obrazovce.
Alternativní scénář
Výchozí podmínky: uživatel není na domovské obrazovce.

1. Uživatel v menu vybere položku „Home“.
2. Systém zobrazí seznam nadcházejících událostí.

UC5–Zobrazení detailu akce

Cíl: Uživatel si zobrazí podrobnosti o konané akci spolu se seznamem not, které na akci bude potřebovat.
Hlavní scénář
Výchozí podmínky: uživatel je na domovské obrazovce.

1. Uživatel provede UC4–Zobrazení seznamu akcí.
2. Uživatel klikne na tlačítko „Detail“ vedle akce, jejíž podrobnosti si chce zobrazit.
3. Sytém zobrazí podrobnosti o konané akci spolu se seznamem not, které uživatel bude potřebovat.

UC6–Zobrazení konkrétních not

Cíl: Uživatel si zobrazí požadované noty ve formátu pdf.
Hlavní scénář
Výchozí podmínky: uživatel je na domovské obrazovce.

1. Uživatel provede UC5–Zobrazení detailu akce.
2. Uživatel klikne na tlačítko „View“ vedle názvu not, které si chce zobrazit.
3. Systém zobrazí noty ve formátu PDF, ty budou k dispozici ke stažení nebo k tisku.

3.2.2 Funkčnosti přístupné pouze administrátorovi

UC7–Správa nástrojů

Cíl: Spravovat seznam nástrojů včetně zobrazování not pro jednotlivé nástroje.
Hlavní scénář–zobrazit noty pro daný nástroj
Výchozí podmínky: Administrátor je na domovské obrazovce.

1. Administrátor v menu vybere položku ”Instruments”.
2. Systém zobrazí seznam všech evidovaných nástrojů.
3. Administrátor zvolí nástroj, pro který chce zobrazit seznam not.
4. Systém zobrazí požadovaný seznam not, každé noty bude možné zobrazit ve formátu PDF.

Alternativní scénář–přidat nový nástroj
Výchozí podmínky: Administrátor je na obrazovce se seznamem všech evidovaných nástrojů.

1. Administrátor vyplní název nástroje, který chce přidat a klikne na tlačítko „Add instrument“.

2. Systém přidá nástroj do evidence, nový nástroj se ihned objeví v seznamu všech nástrojů.

UC8–Správa skladeb

Cíl: Spravovat seznam skladeb včetně přidávání, upravování a mazání skladeb.
Hlavní scénář–zobrazit seznam not evidovaných pro danou skladbu.
Výchozí podmínky: Administrátor je na domovské obrazovce.

1. Administrátor v menu vybere položku „Musical pieces overview“.

2. Systém zobrazí seznam všech evidovaných skladeb, u každé bude možnost zobrazit noty, které ke skladbě patří, editovat skladbu a skladbu smazat.

3. Administrátor zvolí skladbu, pro kterou chce zobrazit seznam not.

4. Systém zobrazí požadovaný seznam not, každé noty bude možné zobrazit ve formátu PDF.

Alternativní scénář–Přidat skladbu do evidence.
Výchozí podmínky: Administrátor je na domovské obrazovce.

1. Administrátor v menu vybere položku „Create musical piece“.

2. Systém zobrazí formulář pro vytváření skladeb.

3. Administrátor vyplní název skladby. Pokud chce přiřadit ke skladbě noty, které jsou již evidované u jiné skladby, zadá jejich název a klikne na „Add sheet“. Tyto noty budou ve výsledku přiřazeny pouze k nově vytvořené skladbě.

4. Administrátor klikne na „Save musical piece“.

5. Systém přidá skladbu do databáze.

UC9–Správa not

Cíl: Spravovat seznam not.
Hlavní scénář–Přidat noty do evidence.
Výchozí podmínky: Administrátor je na domovské obrazovce. V evidenci již je nástroj, pro který jsou noty určeny a skladba, ke které noty patří.

1. Administrátor v menu vybere položku „Create sheet“.

2. Systém zobrazí formulář pro vytváření not.
3. Administrátor ve formuláři vyplní název, pod kterým noty budou uloženy, vybere nástroj, pro který jsou noty určeny a skladbu, ke které noty patří.
4. Administrátor nahraje noty ve formátu PDF a klikne na „Save sheet“.
5. Systém přidá noty do evidence.

Alternativní scénář–Upravit údaje o existujících notách.
Výchozí podmínky: Administrátor je na domovské obrazovce. V evidenci již je nástroj, pro který jsou noty určeny a skladba, ke které noty patří.

1. Administrátor v menu vybere položku „Sheet overview“.
2. Systém zobrazí seznam všech evidovaných not, u každých bude možnost noty zobrazit, editovat údaje o notách a noty smazat.
3. Administrátor klikne na tlačítko „edit“ vedle not, které chce upravovat.
4. Systém zobrazí obrazovku pro vytváření not s předvyplněnými údaji, které jsou nyní u not uloženy.
5. Administrátor provede úpravy a klikne na „Save sheet“.
6. Systém upraví údaje o příslušných notách.

UC10–Správa událostí

Hlavní scénář–Vytvořit novou událost.
Výchozí podmínky: Administrátor je na domovské obrazovce. V evidenci již jsou skladby, které se na události budou hrát.

1. Administrátor v menu vybere položku „Create event“.
2. Systém zobrazí formulář pro vytváření událostí.
3. Administrátor ve formuláři vyplní název události, doplňující informace o události, vybere datum a skladby, které se na události budou hrát.
4. Administrátor klikne na „Save event“.
5. Systém přidá událost do evidence.

Alternativní scénář–Upravit údaje o existující události. Výchozí podmínky: Administrátor je na domovské obrazovce. V evidenci již jsou skladby, které se na události budou hrát.

1. Administrátor klikne na tlačítko „Edit“ vedle události, kterou chce upravit.

2. Systém zobrazí formulář pro vytváření událostí s předvyplněnými údaji, které jsou u události nyní evidovány.

3. Administrátor provede úpravy a klikne na „Save event“.

4. Systém upraví údaje o příslušné události.

UC11–Správa registrací

Hlavní scénář–Potvrdit registraci uživatele.
Výchozí podmínky: Administrátor je na domovské obrazovce.

1. Administrátor v menu vybere položku „Users waiting for approval“.

2. Systém zobrazí seznam registrací, které je třeba schválit. Registraci bude možné přijmout nebo zamítnout.

3. Administrátor klikne na „Approve“ vedle uživatele, jehož registraci chce potvrdit.

4. Daný uživatel se nyní může přihlásit do aplikace. U událostí se mu zobrazují noty na nástroje, které uvedl při registraci.

UC12–Správa uživatelů

Hlavní scénář–Přidělit uživateli nový nástroj.
Výchozí podmínky: Administrátor je na domovské obrazovce. V evidenci je nástroj, který bude uživateli přidělen.

1. Administrátor v menu vybere položku „Existing users“.

2. Systém zobrazí seznam všech uživatelů.

3. Administrátor klikne na „Detail“ vedle uživatele, jemuž chce přidělit nový nástroj.

4. Systém zobrazí profil uživatele spolu s formulářem na přidělování nástrojů.

5. Administrátor vybere nástroje, které chce uživateli přidat (může je i odebírat) a klikne na tlačítko „save“.

6. Uživateli se nyní u události budou zobrazovat noty na všechny přidělené nástroje.

Alternativní scénář–Přidělit uživateli administrátorská práva.
Výchozí podmínky: Administrátor je na obrazovce s profilem uživatele, kterému chce přidělit administrátorská práva.

1. Administrátor klikne na „Set this user admin“.
2. Daný uživatel je nyní také administrátorem a má zpřístupněné všechny funkce a nástroje.

3.3 Analýza nefunkčních požadavků

Identifikoval jsem následující nefunkční požadavky:

1. Noty budou uloženy v databázi ve formátu PDF.
2. Uživatelé se přihlašují uživatelským jménem a heslem.
3. Aplikace bude zdokumentována v uživatelské příručce.
4. Aplikace nebude mít optimalizované uživatelské rozhraní pro mobilní telefony.

Kapitola 4

Implementace

Při implementaci jsem postupoval podle návrhu aplikace. Postupně jsem vytvářel jednotlivé vrstvy třívrstvé architektury. Popis těchto vrstev je v teoretické části této práce a v kapitole *Návrh*.

4.1 Vytvoření projektu

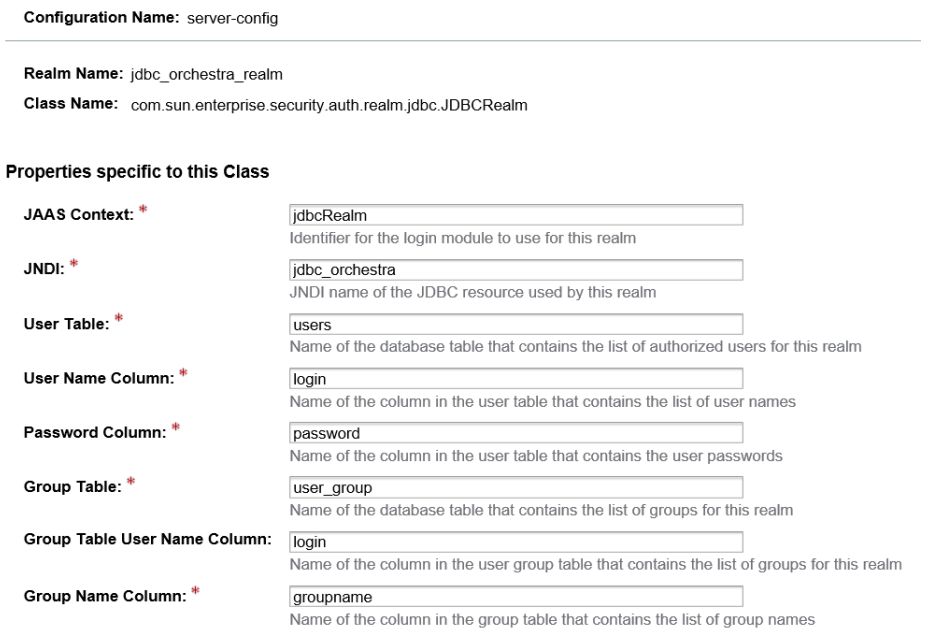
4.1.1 Připojení k databázi

Nejprve jsem vytvořil databázi, ve které aplikace bude persistentně ukládat potřebná data. Pro tento účel jsem vytvořil databázové schéma *Orchestra*. Na aplikačním serveru jsem vytvořil **connection pool** sloužící jako soubor znovupoužitelných připojení k právě vytvořené databázi. V tomto poolu jsou informace o typu databáze, serveru, na němž je databáze provozována, a autorizační údaje, které se použijí při navazování spojení. Vytváření nových spojení je časově velmi náročné. Server si proto v poolu udržuje informace o dostupných připojeních. Kdykoli aplikace vyžaduje spojení s databází, bude jí přiděleno některé z poolu. Po uzavření se spojení do connection poolu vrací.

Dále jsem nad connection poolem vytvořil **JDBC Resource** s názvem *jdbc_orchestra*. Pro propojení vrstvy obchodní logiky s aplikačním serverem je třeba definovat, který JDBC zdroj bude používán. K tomu slouží soubor **persistence.xml**. Tam jsem zadal název používaného zdroje. Také jsem zde nakonfiguroval, že knihovna použitá pro objektově–relační mapování bude **Eclipse Link**.

4.1.2 Autentizace uživatelů

Pro ověřování přihlašovaných uživatelů jsem využil služeb aplikačního serveru. Postupoval jsem podle návodu [18]. Vytvořil jsem **JDBC security realm**. V něm jsou specifikované databázové tabulky, v nichž jsou uložena uživatelská jména, hesla a role. Při přihlašování uživatele aplikační server automaticky



Obrázek 4.1: Formulář na vytvoření security realmu

verifikuje vyplněné údaje v nakonfigurovaných tabulkách. Na obrázku 4.1 je formulář, který je potřeba při vytvoření security realmu potřeba vyplnit.

Potom v souboru *web.xml* specifikuji části aplikace, které budou security realmem chráněny. Také zde specifikuji JNDI název realmu, jenž se má použít při autentifikaci uživatelů. Viz. obrázek 3

Důležitá řádka je `<form-login-page>/login.jsp</form-login-page>` udávající stránku, která obsahuje přihlašovací formulář (konkrétně „login.jsp“). Toto je jediná část uživatelského rozhraní, která nebude vytvořena pomocí frameworku Vaadin, ale použitím technologií HTML a CSS.

4.2 Datová vrstva

Datovou vrstvu realizuje databázové schéma *Orchestra*. Toto schéma obsahuje příslušné tabulky a relační vazby mezi nimi. Celé schéma jsem nechal automaticky vytvořit systémem EclipseLink na základě entitních tříd pomocí objektově relačního mapování. Proto bylo potřeba pečlivě definovat entitní třídy a vazby mezi nimi. Ty jsou popsány anotacemi.

Aby aplikace byla schopná udržovat všechna data potřebná pro realizaci případů užití, navrhnul jsem následující entity:

Instrument Tato třída reprezentuje nástroj.

Musical piece Třída reprezentující jednu skladbu.

```
 <security-constraint>
    <web-resource-collection>
      //Na url /ui se dostanou pouze registrovaní uživatelé
      <web-resource-name>Vaadin application</web-resource-name>
      <url-pattern>/ui/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      //Vpuštěni budou uživatelé, kteří mají roli "User" nebo "Admin"
      <role-name>user</role-name>
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>FORM</auth-method>
    //Používá se realm s tímto jménem
    <realm-name>jdbc_orchestra_realm</realm-name>
    <form-login-config>
      //Zde specifikuji, která stránka zobrazí, když je vyžadováno přihlášení,
      //a která stránka informuje o tom, že přihlášení nebylo úspěšné
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/loginError.jsp</form-error-page>
    </form-login-config>
  </login-config>
}
```

Výpis kódu 3: Soubor web.xml

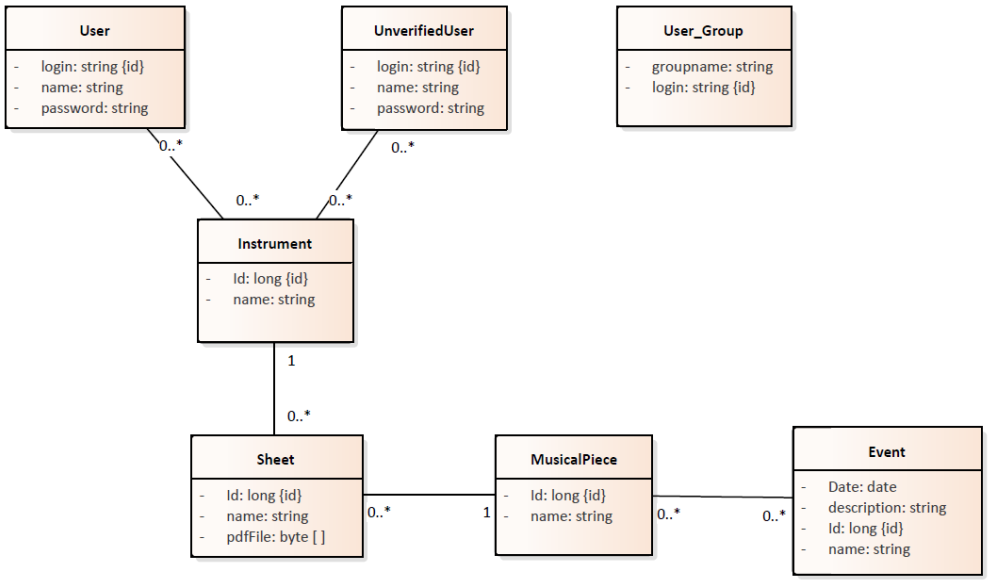
Sheet Třída zastupující jeden výtisk not. Noty u sebe evidují, k jaké skladbě se váží a pro jaký nástroj jsou určeny.

Event Třída reprezentující akci. Akce u sebe eviduje seznam skladeb, které se na ní budou hrát.

User Tato třída reprezentuje uživatele. Uživatel má jméno, heslo a seznam nástrojů, na které hraje.

UnverifiedUser Třída reprezentující uživatele, který se do aplikace zaregistroval, ale jeho žádost ještě nebyla schválena administrátorem.

User_Group Při vytváření security realmu na aplikačním serveru bylo potřeba specifikovat tabulku, ve které je uloženo, do jaké skupiny jsou uživatelé zařazeni. V této aplikaci rozlišuji dvě skupiny uživatelů–administrátory a běžné uživatele.



Obrázek 4.2: Diagram tříd

Na obrázku 4.2 je zachycen diagram znázorňující atributy tříd a vazby mezi nimi. Tento diagram je pouze na konceptuální úrovni, nezachycuje metody, které třídy mají. V ukázce kódu 4 je implementace entitní třídy reprezentující noty. Jedná se o entitní třídu, proto je nad definicí třídy použitá anotace `@Entity`. Třídu budu potřebovat reprezentovat ve formátu XML pro přenos pomocí RESTových služeb. K tomu slouží anotace `@XmlRootElement`. Další anotace specifikují, jakým způsobem se atributy třídy namapují na sloupce v databázi. Anotace `@Id` specifikuje atribut, který bude použitý jako identifikátor. Anotace `@ManyToOne` označují, že atribut představuje relační vazbu a specifikuje její kardinalitu. Atributy realizující relační vazbu musí být také entitní třídy, nebo jejich kolekce (zde konkrétně třídy *MusicalPiece* a *Instrument* musí být entitní.)

Speciální anotaci také potřebujeme, pokud pracujeme se sloupcem typu *Lob*. LOB znamená Large Object a sloupce s tímto typem se používají k ukládání velkého množství dat (zde se do atributu pdfFile ukládá soubor PDF s notami). Přístup k těmto sloupečkům vyžaduje speciální volání **JDBC Driveru** [11] (to je komponenta, která vytváří spojení s databází, přes které je pak možné vykonávat CRUD operace).

Knihovna EclipseLink potom na základě entitních tříd, jejich atributů a vzájemných vztahů vygeneruje tabulky databázi. Na obrázku 4.3 je znázorněno, jaké tabulky se v databázi vytvořily. Obecně jako identifikátory entit používám automaticky vygenerované hodnoty typu *Long*, pouze entity **User** a **UnverifiedUser** jsou identifikovány atributem „login“ typu *Varchar*. Vztahy typu *Many to Many* byly dekomponovány na dvě vazby typu

```
@Entity
@XmlRootElement
public class Sheet implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Lob
    @Column(columnDefinition="BLOB")
    private byte[] pdfFile;
    private String name;
    @ManyToOne
    @JoinColumn(name = "piece")
    private MusicalPiece piece;
    @ManyToOne
    @JoinColumn(name = "instrument")
    private Instrument instrument;
    .
    .
    Getters and Setters
    .
    .
}
```

Výpis kódu 4: Entitní třída pro noty

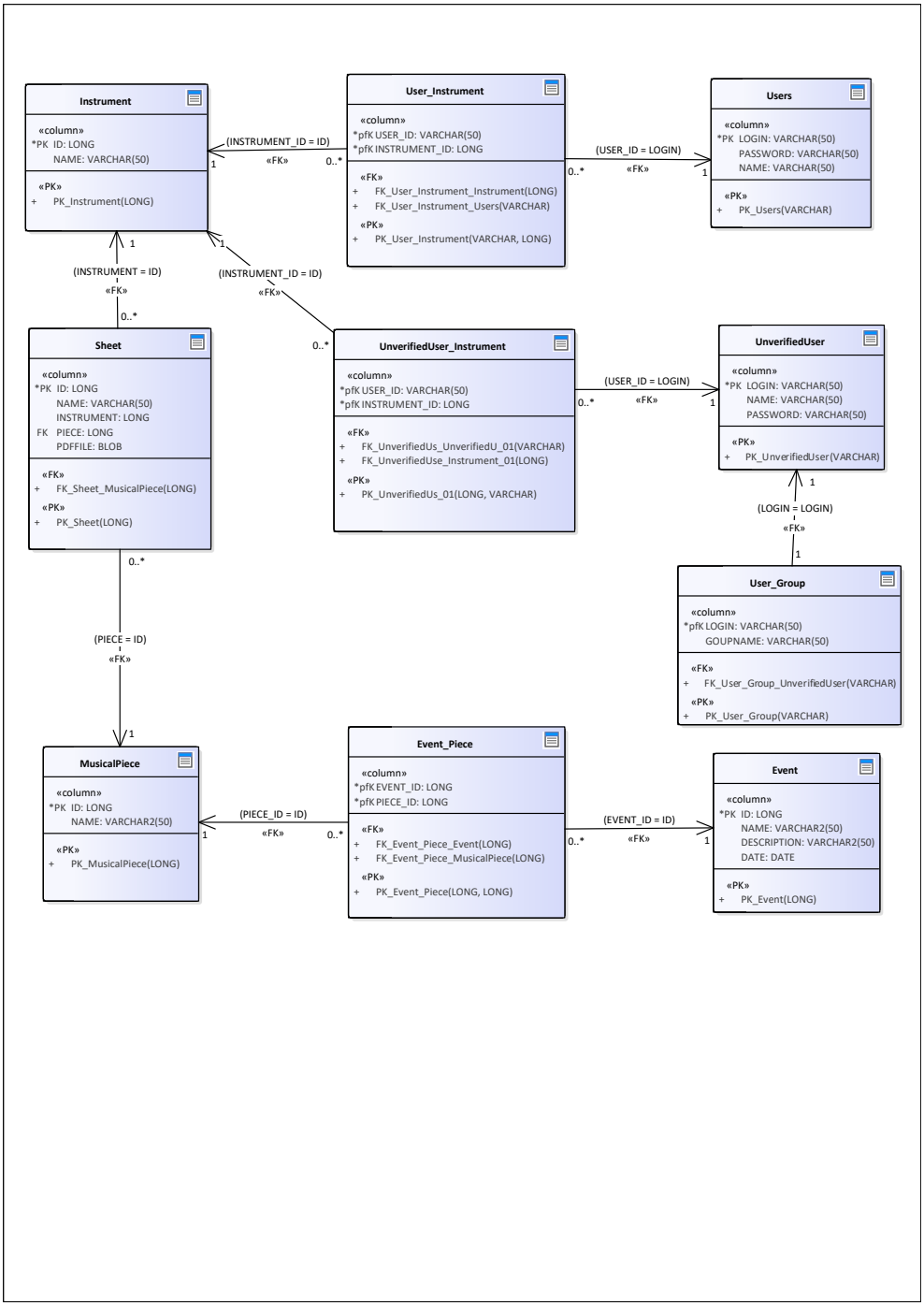
One to Many a byly pro ně vytvořeny vazební tabulky.

4.3 Business vrstva

Na základě mnou navržených entitních tříd jsem vytvořil generickou třídu **AbstractFacade**. Ta obsahuje obecný přístup k manipulaci s entitami. Jsou zde předdefinované metody realizující základní CRUD operace. Tyto metody můžu dále podle potřeby rozšiřovat. Tato třída je na ukázce kódu 5.

Pro každou entitní třídu jsem vytvořil enterpirse Java Bean. Ten bude použit jako RESTová webová služeb. To nám umožní anotace *@Path*. Tyto beany rozšiřují třídu **AbstractFacade** a konkretizují ji pro práci s jedním typem entit. Do každé beany jsem doplnil formou vytvoření nových metod funkcionality specifické pro jednotlivé typy entit. Pro každou entitu jsem přidal možnost vyhledání podle jména (atributu „name“).

V beaně manipulující se skladbami jsem upravil metodu pro mazání. Je zajištěno, aby se smazaná skladba odstranila i z programů všech plánovaných akcí.



Obrázek 4.3: Databázový model

```
public abstract class AbstractFacade<T> {
    private Class<T> entityClass;
    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }
    protected abstract EntityManager getEntityManager();
    public void create(T entity) {
        getEntityManager().persist(entity);
    }
    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }
    ...
}
```

Výpis kódu 5: Generická třída pro manipulaci s entitami

Nejvíce funkcionalit bylo potřeba naimplementovat do beany poskytující metody pro manipulaci s notami. Doplnil jsem možnost vyhledat noty podle nástroje a skladby, k níž jsou noty přiřazeny. Je k dispozici i metoda, která pro zadanou akci najde všechny noty, jenž se váží na skladby hrané na události. Nejdůležitější metoda je *getPdf*, která vrací noty ve formátu PDF. Tato metoda je v ukázce kódu 6.

Do beany pracující s uživateli jsem přidal metodu, která se stará o přidělení uživateli administrátorská práva. Všechny uživatelské registrace schvaluje administrátor. Při prvotním spuštění aplikace ale žádný administrátor neexistuje – nedalo by se tedy do aplikace zaregistrovat. Vytvořil jsem pro tento účel metodu, která vytvoří administrátora s předdefinovaným uživatelským jménem a heslem za předpokladu, že v aplikaci žádný administrátor není.

4.4 Klientská vrstva

Implementaci klientské vrstvy jsem zahájil vytvořením tříd, které zaobalují volání RESTových služeb business vrstvy. V ukázce kódu 8 je vidět kus jedné takové třídy. Za povšimnutí stojí řádka

```
webTarget = client.target(BASE_URI).path("...")
```

Zde se nastavuje adresa, na kterou se budou zasílat požadavky. Tuto adresu můžeme nadále rozšiřovat v jednotlivých metodách. Požadavek se pak zašle takto:

```
return resource.request(MediaType.APPLICATION_XML).get(responseType);
```

Toto je požadavek GET, ostatní druhy požadavků můžeme zaslat podobně pomocí `.post()`, `.put()` a `.delete()`.

```
@Stateless
@Path("eu.cz.fit.vahalto1.orchestraapplication.sheet")
public class SheetFacadeREST extends AbstractFacade<Sheet> {
    @PersistenceContext(unitName = "OrchestraAppPersistence")
    private EntityManager em;
    @EJB
    private InstrumentFacadeREST ifr;
    public SheetFacadeREST() {
        super(Sheet.class);
    }
    ...
    @Path("/{id}/pdf")
    @Produces("application/pdf")
    public Response getPdf(@PathParam("id") Long id){
        byte [] result = (byte [])em.createQuery(
        "SELECT s.pdfFile FROM Sheet s WHERE s.id = :id")
        .setParameter("id", id)
        .getSingleResult();
        ByteArrayInputStream is = new ByteArrayInputStream(result);
        Response.ResponseBuilder responseBuilder = Response.ok((Object) is);
        responseBuilder.type("application/pdf");
        responseBuilder.header("Content-Disposition", "filename=test.pdf");
        return responseBuilder.build();
    }
    ...
}
```

Výpis kódu 6: RESTová služba pro manipulaci s notami

```
@XmlRootElement
public class Events {
    private List<Event> events = new ArrayList<>();
    public List<Event> getEvents() {
        return events;
    }
    public void setEvents(List<Event> events) {
        this.events = events;
    }
}
```

Výpis kódu 7: Třída zaobalující kolekci událostí

```
public class EventClient {
    private WebTarget webTarget;
    private Client client;
    private static String BASE_URI;
    public EventClient(String baseURI) {
        BASE_URI = baseURI;
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget = client.target(BASE_URI).
        path("webresources/eu.cz.fit.vahalto1.orchestraapplication.event");
    }
    ...
     public <T> T find_XML(Class<T> responseType, String id){
        WebTarget resource = webTarget;
        resource = resource.path(java.text.MessageFormat.format("{0}",
        new Object[]{id}));
        return resource.request(javax.ws.rs.core.MediaType.APPLICATION_XML)
        .get(responseType);
    }
    ...
}
```

Výpis kódu 8: Třída volající RESTové rozhraní

Metody těchto tříd potom volám, když potřebuji získat data nebo s nimi manipulovat.

Při volání požadavku GET je potřeba upřesnit, jaký objekt je očekáván v odpovědi (response type). Často je potřeba získat kolekci objektů, ale kolekci nelze jako response type použít (nelze napsat `.get(List<Event>.class)`). Vytvořil jsem tedy třídy reprezentující kolekce entitních tříd (například pro entitní třídu „Event“ jsem vytvořil třídu „Events“, která je v ukázce kódu 7). Response type pak bude `Events.class` a ke kolekci pak získám přístup pomocí `Events.getEvents()`.

Uživatelské rozhraní je realizováno pomocí dvou Vaadin servletů. První představuje rozhraní pro registraci uživatele a je dostupný po kliknutí na tlačítko „register“ na úvodní obrazovce. Druhý servlet prezentuje všechny ostatní funkcionality aplikace a pro jeho zobrazení je vyžadováno přihlášení uživatele.

Obrazovky jsem navrhnul tak, aby korespondovaly s případy užití. Vytvořil jsem toolbar, který slouží k navigaci mezi obrazovkami. Počet položek v toolbaru se odvíjí od toho, zda je přihlášený uživatel administrátor. Administrátor má oproti běžnému uživateli zpřístupněné obrazovky umožňující provádění CRUD operací nad skladbami, notami, akcemi a ostatními uživateli.

Pro každou obrazovku jsem vytvořil třídu, která generuje její obsah. Po kliknutí na položku v toolbaru se zavolá třída zodpovědná za generování pří-

slušné obrazovky. Plocha stránky se pak vyplní novým obsahem. Toto řešení používá pro zobrazení uživatelského rozhraní pouze jednu stránku, jejíž obsah se dynamicky mění. To znamená, že uživatelské rozhraní je celé na jednom URL.

Největší problém při tvorbě uživatelského rozhraní pro mě byla manipulace se pdf soubory. Bylo třeba zajistit, aby uživatel mohl nahrát soubor, ten se správně uložil do databáze a následně ho bylo možné zobrazit.

Vaadin nám poskytuje komponentu zobrazující dialog pro výběr souboru, který má být nahrán. Pro její použití je třeba naprogramovat „receiver“. Ten říká, co se má se souborem po nahrání stát. Klíčové části této třídy jsou v ukázce kódu 9. Tlačítko pro nahrávání souboru pak vytvořím takto:

```
Upload upload = new Upload("Upload pdf file", new FileUploadReceiver());
```

Zobrazování pdf souboru jsem vyřešil tak, že otevřu nové okno prohlížeče. Přesměruji ho na adresu, kde je RESTová služba, která pdf soubor vrací (popsaná v ukázce kódu 6 výše). Prohlížeč se o zobrazení postará sám. Nevýhoda tohoto řešení je, že prohlížeč může zablokovat vyskakování nového okna a uživatel ho musí explicitně povolit.

Jak jsem zmínil v sekci **Autentizace uživatelů**, úvodní obrazovka aplikace není vytvořená pomocí frameworku Vaadin, ale použitím HTML a CSS. Aby styl stránky vypadal podobně jako u zbytku aplikace, aplikoval jsem na komponenty stejné CSS styly, jaké používá Vaadin. Obrázek hudebníků použitý na této obrazovce pro mě vytvořil pan Petr Netík.

Ukázka uživatelského rozhraní je uvedena v příručce k aplikaci, která je v příloze této práce.

4.5 Výsledek implementace

Aplikace je vytvořená a pokrývá všechny specifikované případy užití. Je nasazená na aplikačním serveru, který mi poskytla společnost **Uspin** a je připravena pro používání v praxi. Fungování aplikace je zdokumentováno v uživatelské příručce, která je dostupná v příloze této práce.

```
public class FileUploadReceiver implements Upload.Receiver{

private String fileName;
private ByteArrayOutputStream baos = new ByteArrayOutputStream();

public ByteArrayOutputStream getStream(){
    return baos;
}

@Override
    public OutputStream receiveUpload(String filename, String mimeType) {
        this.fileName = filename;
        baos.reset();
        return baos;
    }
}

n
```

Výpis kódu 9: Upload receiver

Kapitola 5

Testování

5.1 Lokální testování

Aplikaci jsem otestoval manuálním prováděním případů užití. Nalezl jsem několik chyb, které bylo potřeba opravit (například zobrazení detailu akce, na které se nehrály žádné skladby, končilo chybou).

Dále jsem testoval, jak aplikace funguje v různých webových prohlížečích. Zpočátku jsem měl problém vytvořit úvodní obrazovku tak, aby se správně zobrazovala v prohlížeči Internet Explorer, ale i to se nakonec podařilo. Aplikaci jsem testoval v prohlížečích **Google Chrome** (verze 73.0.3683.103), **Mozilla Firefox** (verze 65.0.2), **Microsoft Edge**(verze 17.17134) a **Internet Explorer** (verze 11.0.120). Ve všech těchto prohlížečích aplikace funguje.

5.2 Uživatelské testování

Po odstranění chyb nalezených při manuálním testování jsem aplikaci zpřístupnil svým kolegům z Pražského filmového orchestru, aby ji mohli používat. Protože v aplikaci dosud nebyla žádná data, archivář do ní musel všechny skladby a noty zadávat ručně. Toto je časově velmi náročné, a tak v systému nejsou všechny plánované akce. Pokud už v systému potřebná data jsou, vytvoření události a distribuce not mezi jednotlivé členy je velmi jednoduchá. Po nahrání not do systému aplikace práci archiváři značně urychlí. V systému jsou registrovaní i někteří hudebníci, kteří nemají administrátorská práva a aplikaci používají pouze pro zobrazování plánovaných akcí a not.

Při testování s větším množstvím dat se ukázalo, že některé akce (například zobrazení seznamu všech not) se při větším množství zpracovaných dat zpomalují. Problém byl způsoben tím, že při vyžádání seznamu not od RESTové služby se přenášely i soubory pdf. To přenos velmi zpomalovalo. Přenášet pdf soubory není při zobrazení seznamu not potřeba. Pdf soubor bude získán, až když bude vyžadováno jeho zobrazení. K tomu slouží výše zmíněná metoda

getPdf (ukázka kódu 6). Toto řešení zobrazení seznamu všech not výrazně urychlilo.

5.3 Vyhodnocení testování a další vývoj aplikace

Od uživatelů jsem dostal několik námětů na vylepšení uživatelského rozhraní, žádné další chyby ve fungování aplikace ale nalezeny nebyly. Aplikace se uživatelům líbí, po zadání not a skladeb distribuci not opravdu zefektivňuje. Ukázalo se tedy, že aplikace je provozuschopná na skutečných datech a svůj účel plní.

V budoucím vývoji aplikace se zaměřím na zlepšení uživatelského rozhraní. Dále bych chtěl aplikaci rozšířit tak, aby ji mohlo používat více orchestrů zároveň.

Závěr

Cílem této práce bylo navrhnout a vytvořit aplikaci, která bude umožňovat snadnou distribuci not mezi jednotlivé členy hudebního orchestru.

Nejprve jsem se seznámil s postupy, které se používají k distribuci not v orchestrech v dnešní době. Z této rešerše vyplynulo, že se v orchestrech ke sdílení not mezi hudebníky žádný speciální software nepoužívá a tato aplikace usnadní život jak hudebníkům, tak i archivářům not, kteří hudebníkům noty posílají. Na základě rešerše jsem navrhnul funkčnosti aplikace a popsal je pomocí případů užití.

Dále jsem se seznámil s technikami používanými pro tvorbu webových aplikací v jazyce Java. Pomocí těchto technologií jsem aplikaci naimplementoval a nasadil do produkce na aplikační server.

Aplikace rozděluje uživatele do dvou skupin. Administrátor může vytvářet v aplikaci události, k nimž přiřazuje skladby, které se na ní budou hrát. Dále spravuje databázi not, skladeb a nástrojů, které jsou v aplikaci evidovány a schvaluje žádosti uživatelů o přístup do aplikace. Běžný uživatel si pak v aplikaci může zobrazit seznam nadcházejících akcí a u každé bude mít přístupné noty na nástroje, na které hraje.

Aplikace nyní prochází uživatelským testováním, aby se ukázalo, zda je použitelná v praxi. Používají ji členové Pražského filmového orchestru pro přípravu na koncerty. Budu rád, pokud se aplikace bude i nadále používat a usnadní život jak hudebníkům, tak archivářům not.

Práce na této aplikaci mě bavila a byla pro mě cennou zkušeností. Jsem rád, že jsem si vyzkoušel projít celým procesem vývoje webové aplikace od návrhu přes implementaci po testování. Vyzkoušel jsem si nové technologie pro tvorbu jak backendu, tak frontendu aplikací. Většinu znalostí potřebných pro tvorbu této aplikace jsem získal na předmětu BI-TJV vyučovaném ve druhém ročníku a při konzultacích se svým vedoucím. Získané zkušenosti mi pomůžou další aplikace tvořit rychleji a efektivněji.

Bibliografie

1. *forScore manual*.
   Dostupné také z: `https://forscore.co/about-power/`.
2. PAVLÍČEK, Josef. JDBC, Aplikační servery a konfigurace zdrojů. In: [online]. Dostupné také z:
   `https://docs.google.com/presentation/d/12jr7jnWbx_S4sIwLd0v9HmUyX1uuAP3-z6HOdIDp7WA/edit?usp=sharing`.
   [Přednáška z předmětu BI-TJV na FIT ČVUT, cit. 2019-04-10].
3. PAVLÍČEK, Josef. Úvod, OOP v Javě. In: [online].
   Dostupné také z: `https://docs.google.com/presentation/d/18tb1b0YEvc6TLXslKA0fVl9kNPs7q5wF9IMB3TmHcME/edit?usp=sharing`.
   [Přednáška z předmětu BI-TJV na FIT ČVUT, cit. 2019-03-28].
4. *The Java EE 5 Tutorial* [online]. Dostupné také z:
   `https://docs.oracle.com/javaee/5/tutorial/doc/bnaay.html`.
   [cit. 2019-04-13].
5. PAVLÍČEK, Josef. EJB a Webové služby (SOAP,REST). In: [online].
   Dostupné také z: `https://docs.google.com/presentation/d/1gsY5kUuBmDRbCt2vyNE-YBNYq6lfRIuOeH0AQpSFjCI/edit?usp=sharing`.
   [Přednáška z předmětu BI-TJV na FIT ČVUT, cit. 2019-04-08].
6. *The Java EE 6 Tutorial* [online]. Dostupné také z:
   `https://docs.oracle.com/javaee/6/tutorial/doc/gipko.html`.
   [cit. 2019-04-12].
7. REESE, Richard M. *EJB 3.1 Cookbook*. Packt Publishing, 2011. ISBN 9781849682381.
8. BURKE, Bill. *RESTful Java with JAX-RS 2.0: designing and Developing Distributed Web Services*. O'Reilly Media, 2013. ISBN 9781449361341.

9. *The Java EE 6 Tutorial* [online]. Dostupné také z: https://docs.oracle.com/javaee/6/tutorial/doc/gkbaa.html. [cit. 2019-04-12].

10. PAVLÍČEK, Josef. Práce s databázemi JPA - ORM. In: [online]. Dostupné také z: https://docs.google.com/presentation/d/1XkHIKutZPBVycJyiccJuV09TCink4eH1FA91WNkNPTk/edit?usp=sharing. [Přednáška z předmětu BI-TJV na FIT ČVUT, cit. 2019-04-1].

11. KEITH, Mike; SCHINCARIOL, Merrick; KEITH, Jeremy. *Pro JPA 2: Mastering the Java Persistence API*. Apress, 2009. ISBN 9781430219569.

12. JANSSEN, Thorben. *What's the difference between JPA, Hibernate and EclipseLink* [online]. Dostupné také z: https://thoughts-on-java.org/difference-jpa-hibernate-eclipselink/. [cit. 2019-03-27].

13. PAVLÍČEK, Josef. Sestavení aplikace Ant, Maven. In: [online]. Dostupné také z: https://docs.google.com/presentation/d/1VegRpgPRDuKGVul1sy4P-Ycu2Oksk7V9X4367HIPhC4/edit?usp=sharing. [Přednáška z předmětu BI-TJV na FIT ČVUT, cit. 2019-04-3].

14. *POM reference* [online]. Dostupné také z: https://maven.apache.org/pom.html#Dependencies. [cit. 2019-04-11].

15. GRONROOS, Marko. *Book of Vaadin*. 4. vyd. Vaadin Ltd, 2012. ISBN 9789529267538.

16. PAVLÍČEK, Josef. Vaadin, Spring MVC. In: [online]. Dostupné také z: https://docs.google.com/presentation/d/10hz_2biG4V0qeq76Fe4yfWqNYT_4vGDCuqd2zL6YF4M/edit. [Přednáška z předmětu BI-TJV na FIT ČVUT, cit. 2019-04-15].

17. Dostupné také z: https://vaadin.com/static/content/vaadin-docs/8/framework/articles/img/ejbswingvaadin.png.

18. *Glassfish Form Based Authentication Example* [online]. Dostupné také z: https://javatutorial.net/glassfish-form-based-authentication-example. [cit. 2019-02-16].

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CSS Cascading Style Sheet

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

EJB Enterprise Java Beany

JSP Java Server Pages

JDBC Java Database Connectivity

POM Project Object Model

REST Representational State Transfer

UC Use Case

XML Extensible markup language

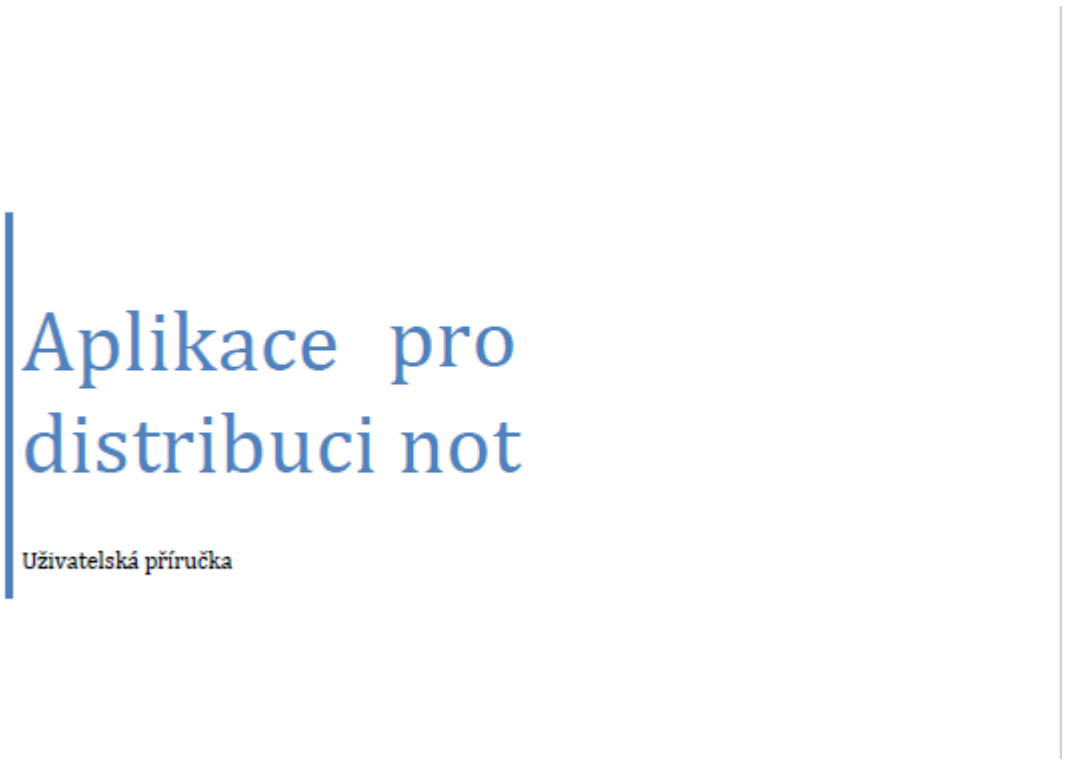
Příloha B

Obsah přiloženého CD

- `src` ..... zdrojové kódy implementace
- `thesis`
  - `latex` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Uživatelská příručka

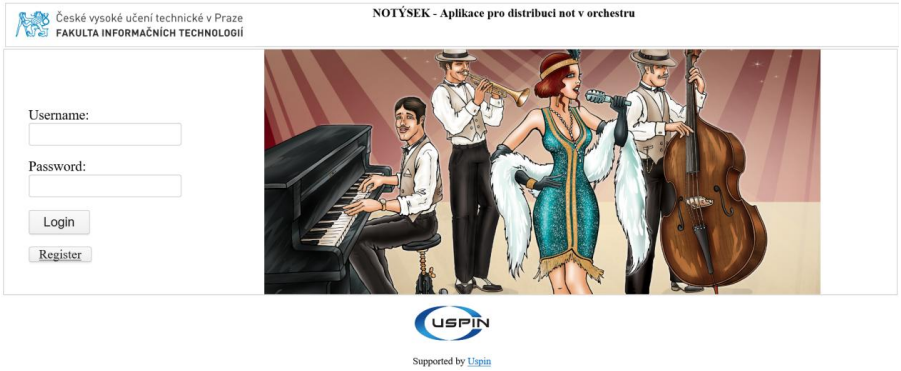


Obsah

O aplikaci

Aplikace slouží členům orchestru pro zálohování not v elektronické podobě a pro efektivní přípravu na koncerty a zkoušky. Zobrazuje hudebníkům seznam nadcházejících akcí. U každé akce si hudebník může zobrazit noty, které bude potřebovat. Zobrazí se pouze noty na nástroje, které má uživatel v aplikaci přidělené.

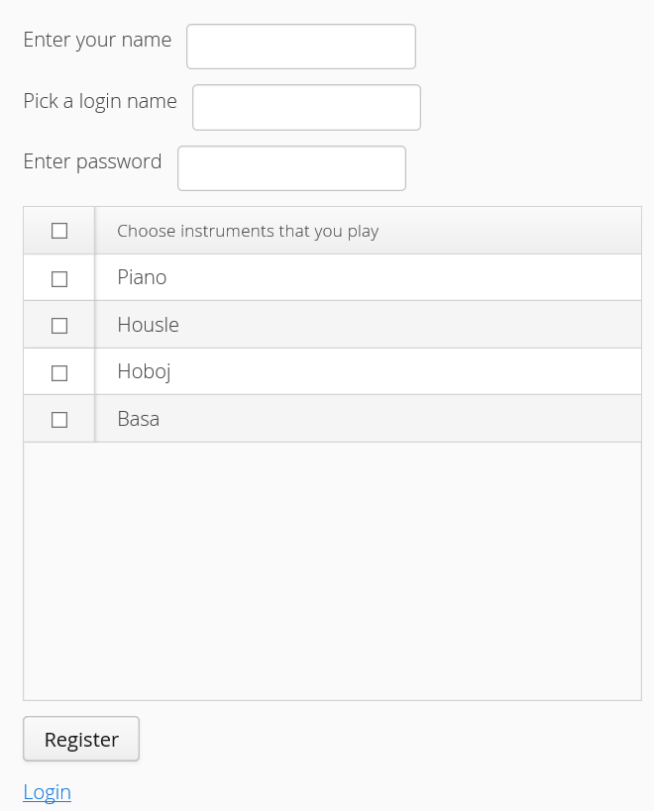
Registrace a přihlášení



Obrázek 1 - Úvodní obrazovka

Pokud v aplikaci ještě nemáte účet, klikněte na úvodní obrazovce na tlačítko „Register“. Ve formuláři vyplňte své celé jméno, zvolte si uživatelské jméno a heslo. Uživatelským jménem a heslem se budete později do aplikace přihlašovat. Dále v tabulce zaškrtněte nástroje, na které hrajete. Noty na tyto nástroje Vám bude aplikace zobrazovat u každé události. (Pokud nástroj, na který hrajete, není v nabídce, nezaškrtávejte nic, pokračujte v registraci a poté se osobně domluvte s administrátorem, aby Vám nástroj přidělil dodatečně). Klikněte na tlačítko „Register“ a vyčkejte, než

administrátor potvrdí vaši registraci**.** (Komunikací mezi uživatelem a administrátorem se tato aplikace nezabývá. Předpokládá se, že člen orchestru má na administrátora kontakt).



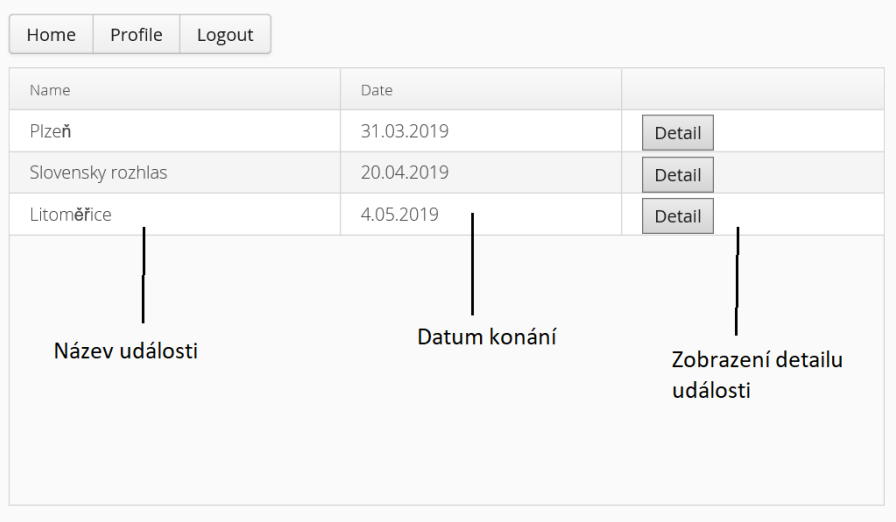
Obrázek 2 - Registrační formulář

Až Váš účet bude schválen, do aplikace se přihlásíte zadáním uživatelského jména a hesla do příslušných políček na úvodní obrazovce.

Funkcionality přístupné běžnému uživateli

Zobrazení seznamu nadcházejících událostí

Seznam nadcházejících událostí uvidí uživatel hned poté, co se do aplikace přihlásí. Tuto obrazovku lze kdykoli zobrazit kliknutím na tlačítko „Home“.

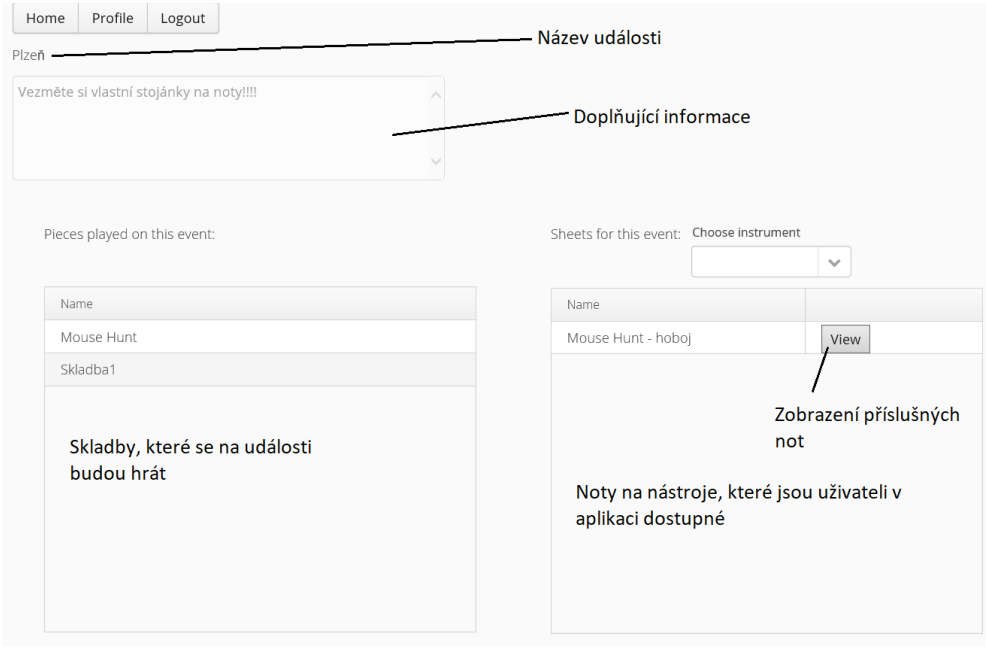


Obrázek 3 - Domovská obrazovka běžného uživatele

Zobrazení detailu akce se seznamem not

Na domovské obrazovce klikněte na tlačítko „Detail“ vedle příslušné akce, kterou si chcete zobrazit.

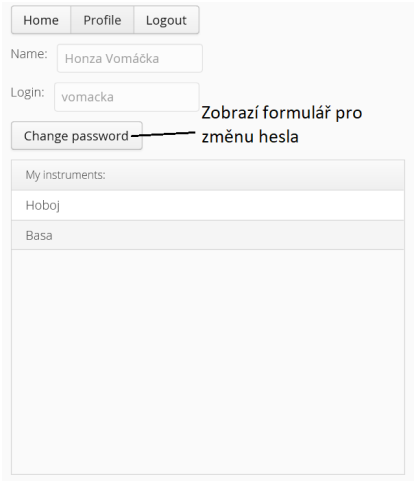
Na obrazovce s informacemi k akci uvidíte název akce, popis, seznam skladeb, které se budou hrát a seznam not, které budete potřebovat. Pokud máte v aplikaci přístup k notám na více nástrojů, můžete noty filtrovat podle nástroje, který si chcete zobrazit. Tlačítko „View“ vedle každých not je zobrazí ve formátu PDF v novém okně prohlížeče.



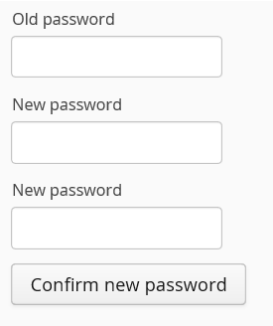
Obrázek 4 - Detail události

Úprava hesla

V sekci „Profile“ si můžete zobrazit své přihlašovací údaje společně s nástroji, ke kterým máte v aplikaci přístup. Chcete-li si změnit heslo, klikněte na „Change password“ a objeví se příslušný formulář. Do prvního políčka napište své současné heslo, do druhého nové heslo a do třetího nové heslo zopakujte. Pro uložení nového hesla klikněte na „Confirm new password“.



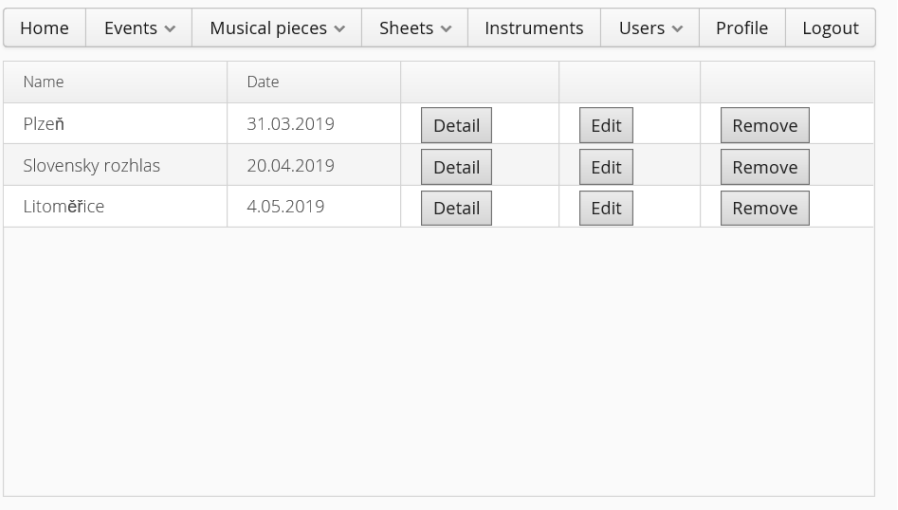
Obrázek 5 - Profil uživatele



Obrázek 6 - Formulář na úpravu hesla

Funkcionality přístupné Administrátorovi

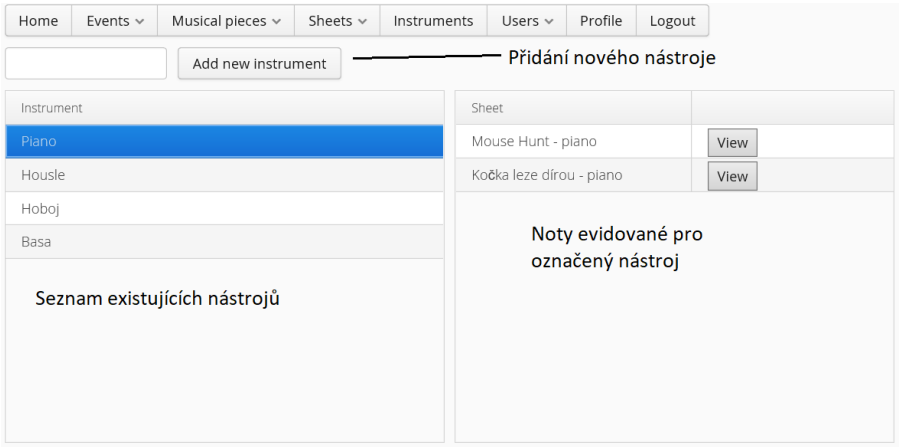
Administrátorova domovská obrazovka vypadá podobně jako u běžného uživatele, ale má v menu zpřístupněno více funkcí.



Obrázek 7 - Domovská obrazovka administrátora

Evidování nástrojů

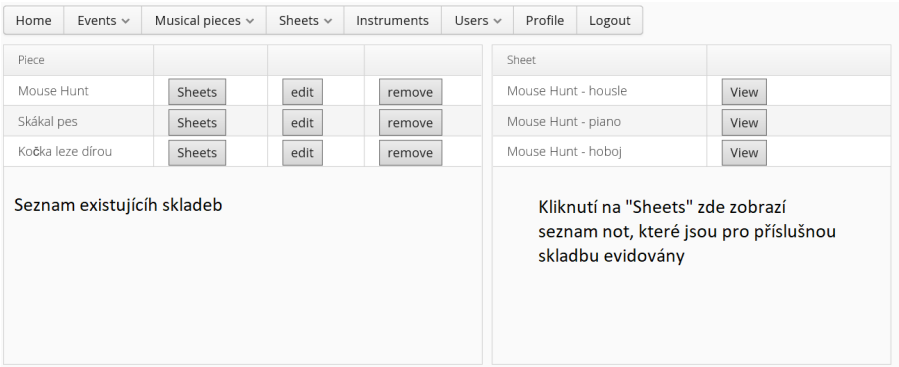
Seznam nástrojů si administrátor může zobrazit kliknutím na tlačítko „Instruments“ v menu. V tabulce vlevo je zobrazen seznam již evidovaných nástrojů, po kliknutí na některý z nich se v tabulce vpravo ukážou všechny noty, které jsou pro daný nástroj k dispozici. Pro vytvoření nového nástroje stačí vyplnit název a kliknout na „Add new instrument“.



Obrázek 8 - Obrazovka na evidenci nástrojů

Evidování skladeb

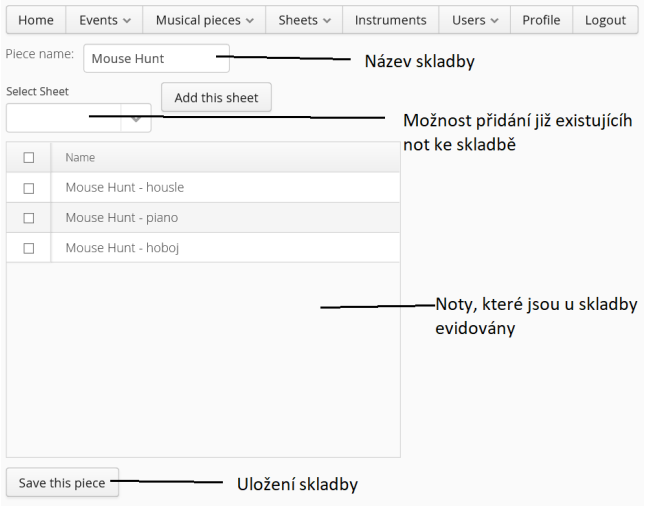
Pro zobrazení existujících skladeb klikněte na tlačítko „Musical pieces“ v menu a následně zvolte možnost „Pieces overview“. V tabulce vlevo se zobrazí všechny dosud evidované skladby. Kliknutím na tlačítko „Sheets“ vedle skladby se zobrazí v tabulce vpravo seznam not, které jsou k dané skladbě evidovány. Tlačítko „edit“ Vám umožní změnit údaje o příslušné skladbě a tlačítko „Remove“ skladbu smaže.



Obrázek 9 - Přehled existujících skladeb

K vytvoření nové skladby opět klikněte na tlačítko „Musical pieces“, ale vyberte možnost „Create piece“. Do formuláře vyplňte název skladby. Můžete k vytvářené skladbě přidat již existující noty, které jsou již přiřazené k jiné skladbě. Vybrané noty se přidávají do seznamu vlevo. Po uložení

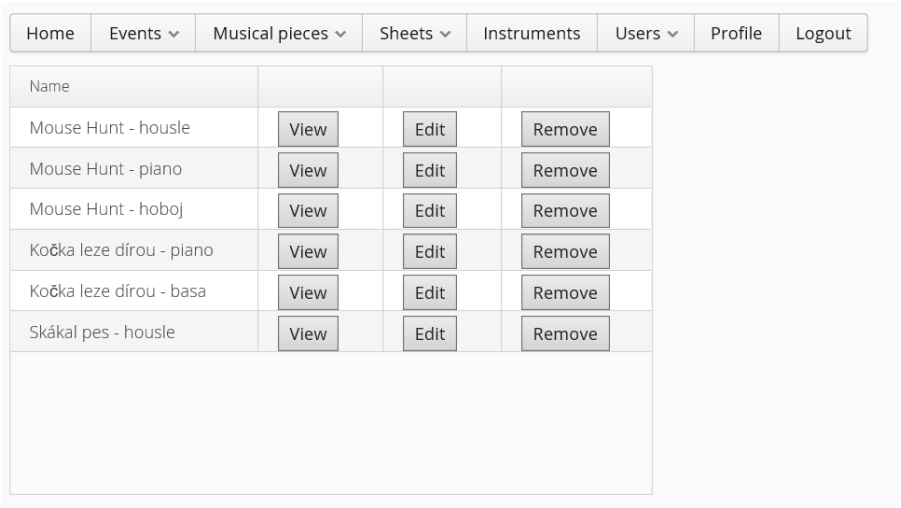
budou vybrané noty přiřazené pouze k nově vytvořené skladbě. Pro editaci již existující



Obrázek 10 - Formulář pro vytvoření / editaci skladby

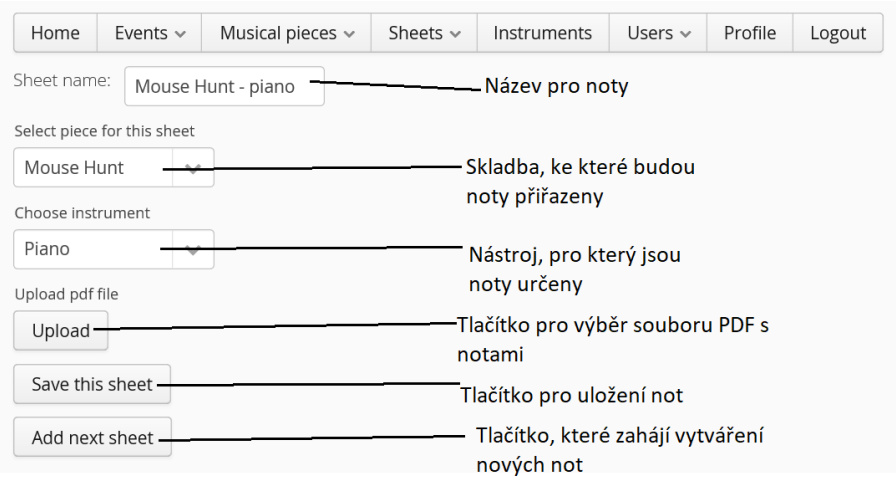
Zálohování not

Seznam již existujících not je přístupný pod položkou „Sheet overview“. Kliknutím na tlačítko „View“ vedle not se zobrazí PDF soubour s notami v novém okně prohlížeče. Tlačítko „edit“ Vám umožní změnit údaje o příslušných notách a tlačítko „Remove“ noty smaže.



Obrázek 11 - Přehled evidovaných not

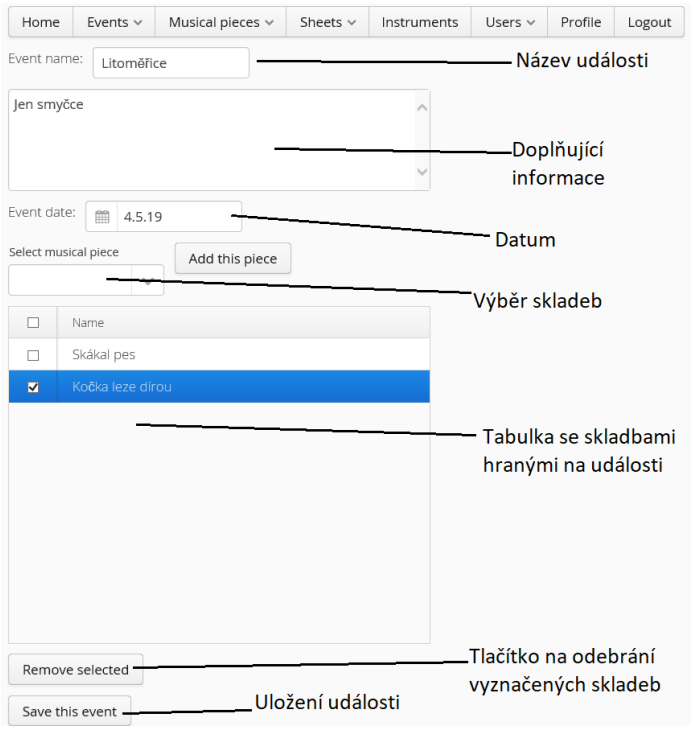
Pro vložení not do evidence klikněte na tlačítko „Sheets“ v menu a následně zvolte možnost „Create sheet“. Při vytváření not již musí být v evidenci skladba, které noty budou patřit a nástroj, pro který jsou určeny. Do formuláře vyplňte název, pod kterým budou noty uloženy, vyberte skladbu, ke které noty patří, a zvolte nástroj, pro který jsou noty určeny. Následně klikněte na tlačítko „Upload“ a vyberte soubor PDF s notami. Pro uložení not klikněte na „Save this sheet“. Chcete-li pokračovat ve vytváření dalších not, klikněte na „Add next sheet“.



Obrázek 12 - Formulář pro vytvoření / editaci not

Vytvoření a editace události

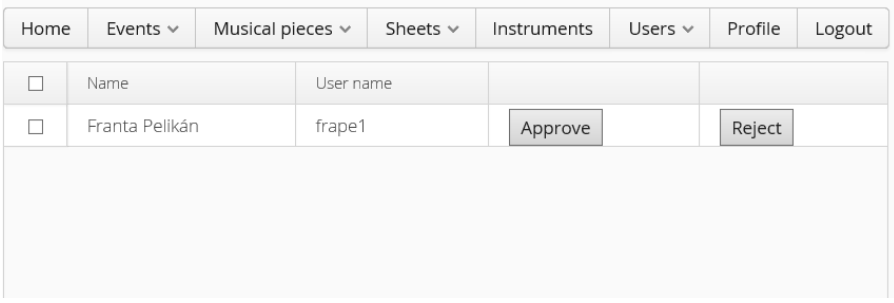
Pro vytvoření nové události klikněte na „Events“ a vyberte položku „Create event“. Do formuláře vyplňte název akce, níže můžete přidat doplňující informace o událost. Dále kliknutím na obrázek kalendáře budete moci vybrat datum události. Nakonec vyberte skladby, které se na události budou hrát (výběr vždy potvrďte tlačítkem „Add piece“). Pokud si přejete některé skladby ze seznamu odstranit, označte je kliknutím do čtverečku vedle jejich názvu a klineˇte na „Remove selected“. Událost uložíte tlačítkem „Save this event“.



Obrázek 13 - Formulář pro vytvoření / editaci události

Potvrdit / odmítnout registraci uživatele

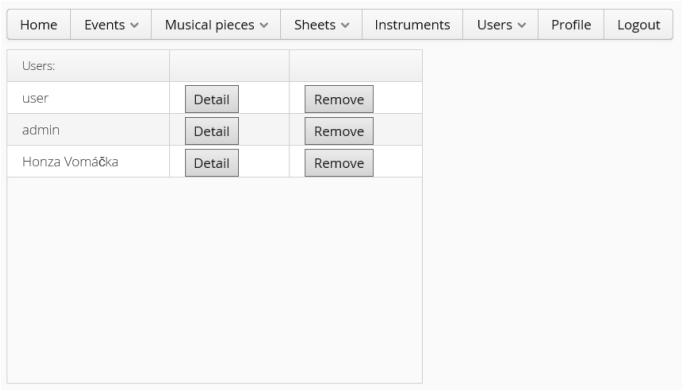
Účet se registrovanému uživateli zpřístupní až poté, co jeho registraci potvrdí některý z administrátorů systému. Seznam uživatelů čekajících na schválení je možné zobrazit kliknutím na možnost „Users“ v menu a následně na „Waiting for approval“. V tabulce je vidět jméno a login uživatele. Tlačítko „Approve“ jeho žádost potvrdí, „Reject“ ji zamítne.



Obrázek 14 - Seznam uživatelů čekajících na schválení

Přidělit uživateli administrátorská práva

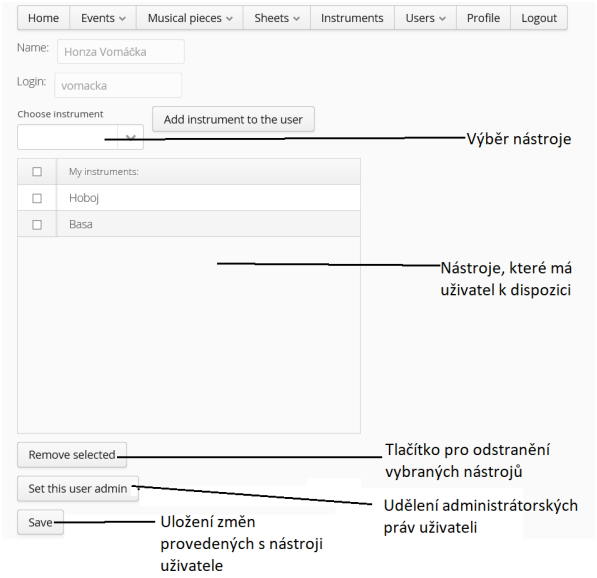
Administrátor si může zobrazit profil jakéhokoli registrovaného uživatele. Pro zobrazení seznamu všech uživatelů klikněte na „Users“ v menu a následně zvolte možnost „Existing users“. V tabulce jsou jména registrovaných uživatelů. Tlačítkem „Detail“ si zobrazíte uživatelův profil, tlačítko „Remove“ uživatele smaže.



Obrázek 15 - Seznam existujících uživatelů

Když si jako administrátor zobrazíte něčí profil, můžete mu přiřadit další nástroje, jejichž noty uživatel pak bude mít zpřístupněné u nadcházejících koncertů. Vyberte nástroj, který chcete přidat a potvrďte výběr tlačítkem „Add instrument to user“. Rovněž můžete uživateli nástroje odebírat. Označte nástroje v tabulce a klikněte na „Remove selected“. Pro uložení změn provedených s nástroji uživatele klikněte na tlačítko „Save“.

Kliknutím na „Set this user admin“ udělíte uživateli administrátorská práva a přístup ke všem nástrojům. Tato akce je nevratná.



Obrázek 16 - Manipulace administrátora s profilem uživatele