Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Systém pro správu interních školení

Plzeň 2015 František Koleňák

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 24. června 2015

František Koleňák

Abstract

Systém pro správu interních školení

V práci jsem navrhl a realizoval aplikaci pro správu interních školení. Aplikace bude zjednodušovat správu a poskytovat generovaná hlášení o uskutečněných školeních. Tato hlášení pak budou sloužit pro generování dat, která budou použita k dalšímu zpracování. Jednotlivá školení budou rozdělena tak, aby prohlížení a správa školení byla co nejpřehlednější. Školení slouží pro seznámení s tématem, na které jsou zaměřena. Účastníci mají možnost se na školení přihlásit a tím dát najevo svůj zájem.

Klíčová slova

interní školení, role účastníků, uživatelské rozhraní, vkládání závistlostí, objektově relační mapování

The system for managing of internal trainings

In the thesis I have designed and implemented an application for internal training management. The application will simplify management and provide reports about past trainings. Reports will be used for generating data. Generated data will be used for further processing. Trainings will be divided that viewing and management would be as easy as possible. Training serves to introduce the topic on which they are focused. Participants have the opportunity to sign the training and thus express their interest.

Key words

internal trainings, participants role, user interface, dependency injection, object relational mapping

Obsah

1 Úvod

Práce se zabývá návrhem a implementací aplikace pro správu interních školení. Ta slouží pro seznámení s tématem, na která jsou zaměřena. Některá školení jsou ukončena certifikátem. Účastníci mají možnost se na školení přihlásit a tím dát najevo zájem. Správa školení je důležitá, neboť může snadněji identifikovat školení, o která je zájem, a podle těchto údajů naplánovat další termíny konání.

Aplikace musí umět školení vytvářet a následně je zobrazit. Na vytvořené školení se bude zájemce moci přihlásit. U každého školení bude informace o počtu přihlášených zájemců, počtu volných míst a datu, dokdy je možno se ze školení odhlásit. Aplikace bude dále nabízet hlášení o počtu realizovaných školení včetně počtu účastníků. K aplikaci musí být vytvořena uživatelská dokumentace.

Pro tvorbu aplikace bude použita Java a na vytvoření uživatelského rozhraní bude použit framework Vaadin.

V práci bude provedena analýza zadání a popsány všechny hlavní funkce. V další části práce budou představeny a vybrány frameworky, které budou použity při vývoji aplikace. U použitých frameworků pak bude popsáno, jakým způsobem se nastaví. Další kapitola se bude týkat popisu backendu aplikace. Předposlední kapitola bude zaměřena na frontend aplikace, zde budou popsány hlavní třídy aplikace a budou popsány složitější funkce. Poslední kapitolou bude testování aplikace s jednoduše popsaným procesem vývoje a připomínkami od zadavatele.

2 Analýza zadání

2.1 Rozdělení uživatelů

Pro správnou funkci systému bude potřeba rozdělit uživatele do několika rolí. Rozdělení slouží k zamezení zneužití funkcí, které by měl dělat pouze manager či lektor. Každý uživatel, který poprvé navštíví aplikaci, získá roli účastníka, jeho roli bude možno později změnit. Uživatele rozdělím do následujících tříd:

1. Účastník
2. Lektor
3. Manager

2.1.1 Role účastník

Účastník je základní role, kterou dostane každý uživatel při prvním přihlášení. Tento uživatel může pouze zobrazovat seznam vypsaných školení, seznam školení na která je zapsán a která absolvoval. Dále má možnost zapsat se na vypsaná školení a po zapsání se odhlásit.

2.1.2 Role lektor

Lektor má stejné funkce jako účastník, ale navíc může spravovat dva seznamy. Jeden je pro přehled, kde se zobrazí seznam školení, ve kterých je zapsán jako lektor. Ve druhém seznamu se zobrazí pouze školení, která již proběhla a kde byl zapsán jako lektor. Tento seznam slouží k zapsání docházky.

2.1.3 Role manager

Manager může provádět vše co účastník. Z funkcí, které má lektor, manager může pouze zapisovat docházku a to u všech již proběhlých školení. K tomuto účelu bude mít k dispozici seznam školení splňující podmínku. Manager má

možnost měnit role uživatelů, kde role může být změněna každému uživateli na libovolnou. Manager má na starosti správu kategorií školení, zakládání šablon a zakládání nových školení. Dále si může zobrazit seznam přihlášek na dané školení a přihlášku může potvrdit nebo zamítnout. Jeho další funkcí je možnost generování reportů.

2.1.4 Diagram užití

Podle navržených rolí jsem vytvořil diagram užití, viz Diagram 2.1.

Diagram 2.1: Diagram užití

2.2 Funkce aplikace

V této kapitole jsou popsány hlavní funkce aplikace. U každé funkce je uvedeno, jaké jsou požadavky na provedení a případně co bude potřeba kontrolovat. Aby funkce mohly správně pracovat, musí být připojeny k databázi. Pro lepší komfort uživatele by měla být data vrácená z databáze seřazena podle nějakého sloupečku.

2.2.1 Zobrazení školení

Zobrazení školení patří k jedné z hlavních funkcí systému. Program bude schopný zobrazit všechna školení, na která se uživatel může přihlásit. Každý uživatel si bude moci prohlédnout, jaká školení už absolvoval. Lektoři a manageři by měli mít k dispozici seznamy školení, díky kterým budou mít snadný přístup ke školení, u kterých je potřeba udělat docházku, potvrdit přihlášky nebo upravit stávající školení.

Zobrazovaný seznam školení bude záviset na roli uživatele. Běžný uživatel uvidí školení, na kterých je přihlášen, na které se může přihlásit a školení, která již absolvoval. Lektor vidí vše co běžný uživatel, ale navíc má seznam školení, u kterých byl lektorem. U těchto školení bude moci zapisovat docházku. Manager může vidět vše co vidí běžný uživatel, ale navíc má ještě možnost správy docházky u školení, která již proběhla.

U jednotlivých školení se budou zobrazovat pouze důležité informace, ostatní informace budou přístupné v detailu školení. Mezi důležité informace patří například název, vyučující, začátek a místo konání.

2.2.2 Vytváření školení

K vytvoření nového školení bude potřeba vyplnit všechny požadované informace. Pokud některá z požadovaných informací, jako třeba název školení, bude chybět, aplikace školení nenechá vytvořit. Zároveň bude potřeba kontrolovat, zda jsou zadány v požadovaném tvaru. Informace nutné k vytvoření jsou název školení, popis, kategorie, místnost konání, lektor a začátek školení. Doba, do které se půjde odhlásit, bude záviset na začátku školení a bude pevně daná (např. jeden den před začátkem). K vytvoření nového ško-

lení bude použita šablona, a to z důvodu snadnějšího přehledu šablon, které jsou dostupné. Po vytvoření školení bude možnost se na něj okamžitě přihlásit a nebo ho případně upravit. Pokud bude při úpravě zmenšen počet volných míst, musí se kontrolovat, jestli se tam vejdou již přihlášení uživatelé.

2.2.3 Přihlášení na školení

Každý uživatel aplikace bude mít možnost se přihlásit na vypsané školení. Platí jedna výjimka, a to pokud je uživatel zároveň lektorem daného školení, nemůže se na školení přihlásit. Pro správné chování bude potřeba uživatelům zamezit podání více jak jedné přihlášky na jedno školení.

2.2.4 Odhlášení ze školení

Každý uživatel přihlášený na školení s potvrzenou přihláškou bude mít možnost se do určitého data ze školení odhlásit. Datum, do kterého se bude možnost odhlásit, bude záviset na datu, na který je vypsáno školení. Bude nutné kontrolovat, zda uživatel má ještě možnost se odhlásit ze školení (uzávěrka odhlášení) a jestli mu přihláška nebyla již zamítnuta.

2.2.5 Přihlášky na školení

Každý uživatel má možnost podat si přihlášku na školení, která by chtěl absolvovat. Aby mohl na dané školení jít, musí mu manager přihlášku potvrdit. Po potvrzení se uživateli dané školení zobrazí v seznamu školení, na která je přihlášen. Pokud manager přihlášku odmítne, uživatel na školení jít nesmí, tedy se mu ani nikde nesmí zobrazit. Databáze bude vracet seznam přihlášek, které manager bude moci potvrdit. Potvrzené přihlášky budou poslány na aktualizaci do databáze.

2.2.6 Kategorie školení

Úpravy kategorií školení může provádět pouze manager. Manager má možnost změnit nějakou informaci u stávající kategorie. Má možnost přidat kategorii novou, při přidání nové kategorie musí být nově vytvořená unikátní.

Dále může kategorii smazat, smazání bude možné pouze pokud daná kategorie nebyla nikde použita. Funkce bude muset kontrolovat správně zadané údaje a také jestli je kategorie unikátní při vytváření nové nebo editaci staré.

2.2.7 Šablony školení

Úpravy šablon bude moci provádět pouze manager. Mezi hlavní informace, které šablona má, patří název, kategorie a popis, tyto údaje jsou povinné. Mezi nepovinné údaje patří místnost a počet volných míst. Manager bude mít možnost měnit nějakou informaci u šablon s tím, že název šablony je unikátní. Manager vytváří nové unikátní šablony a nebo je maže. Smazání půjde pouze v případě, že šablona ještě nebyla nikde použita. Šablony budou sloužit u reportů také jako "typ" reportu, kde šablona bude udávat jaký je to typ školení. Jednotlivá školení můžou mít rozdílné údaje oproti šabloně (např. kategorie je Excel, název šablon jsou Excel nováček, Excel pokročilí a školení jsou pak jednotlivé běhy podle šablony - Excel nováček první část). U funkce bude nutné kontrolovat, zda všechny povinné údaje jsou vyplněné a také jestli všechna políčka mají správný formát.

2.2.8 Správa rolí uživatelů

Úpravy rolí bude provádět pouze manager. Manager může měnit roli každému uživateli aplikace, kromě sebe. Sám sobě změnit roli nemůže z toho důvodu, aby vždy zbyl manager, který by mohl měnit role uživatelům. Také by mělo být zamezeno změnění práv lektorovi s vypsaným a ještě neproběhlým školením, protože jinak by po skončení nemohl provést docházku. Každý uživatel bude mít v záznamu v databázi svoji roli.

2.2.9 Vytváření reportů

Vytvářet a zobrazovat reporty může pouze manager. Reporty mohou být zobrazeny rovnou jako html stránka nebo půjdou vygenerovat do *.csv* souboru. Reporty půjdou vygenerovat celkem čtyři. A to report o lektorech, školení, přehledu školení a o statistikách školení.

Lektor report

Tento report se týká lektorů. Bude možné vybrat jednoho či více lektorů zároveň. Po vybrání jednotlivých lektorů, kterých se report bude týkat, se zadá datum od kdy, do kdy bude report brát výsledky. Po zadání se vygeneruje obsah podle zadaných kritérií. Výstupem tohoto reportu je tabulka se školeními, které daný lektor měl. U jednotlivých školení budou následující informace: jméno lektora, začátek, název a účast.

Školení report

Report bere jako vstupní parametry kurz nebo kurzy a od kdy, do kdy má brát výsledky. Po zadání potřebných informací se vygeneruje obsah, kde se zobrazí všechny případy s potřebnými detaily. Detaily u každého školení jsou: kategorie, typ (název šablony), název, lektor, místo konání, kapacita a začátek.

Přehled školení report

Tento report se zaměřuje na přehled kurzů, kde nezobrazujeme jednotlivé případy, ale zobrazíme celkový počet účastníků a kolikrát se daný kurz konal během určeného časového rozmezí. Jako vstupní parametry se berou názvy typů kurzů, u kterých chceme generovat report. Výstupem je tabulka s názvem kategorie, názvem typu kurzů, celkovým počtem konání a celkovým počtem účastníků.

Statistiky školení report

Report je zaměřený na statistiky jednotlivých kurzů. Statistiky zobrazujeme pro jednotlivé případy konání kurzu. Vstupem jsou názvy kurzů a časový úsek, pro který chceme zobrazit výsledky. Výstupem je tabulka s názvem kategorie, typem, názvem, datem konání a počtem účastníků školení.

2.3 Vzhled aplikace

Aplikace bude rozdělena na 6 stránek, kde každá stránka bude zobrazovat jen určité prvky tak, aby byly jednotlivé stránky co nejjednodušší. Jsou jimi přihlašovací stránka, domů, školení, správa, lidé a reporty. Nepřihlášený uživatel uvidí pouze stránku na přihlášení.

2.3.1 Přihlašovací stránka

Funkcí této stránky je přihlašování. Na stránku bude moci každý uživatel. Po přihlášení nastane přesměrování na hlavní stránku aplikace.

2.3.2 Stránka domů

Stránka je přístupná každému přihlášenému uživateli. Obsah této stránky bude mít každý uživatel jiný. Na stránce bude seznam školení, na která je uživatel přihlášen a seznam školení, kterých se již zúčastnil. Pro uživatele, který je lektorem, bude navíc ještě zobrazen seznam školení, kde je zapsán jako lektor, u těchto školení bude moci evidovat docházku.

2.3.3 Stránka školení

Stránka je přístupná každému přihlášenému uživateli. Uživatel typu účastník zde uvidí pouze seznam školení, na která se může zapsat.

Manager má navíc v seznamu vypsaných školení možnost zobrazení, potvrzování či zamítání přihlášek na školení. Manager má přístup ke školením, která již proběhla. U těchto školení může zapsat docházku.

2.3.4 Stránka správa

Na této stránce bude možno spravovat vše týkající se školení. Tuto stránku má přístupnou pouze manager.

Správa kategorie

Na tomto místě půjdou kategorie vytvářet, upravovat a mazat. Každá kategorie má unikátní název. Kategorie, která již byla někde použita, nepůjde odstranit. Jinak by vznikly šablony/školení s nevyplněnou kategorií.

Správa šablon

Šablony půjdou vytvářet, upravovat nebo mazat. Pro šablony platí to samé jako u kategorií, tedy název musí být unikátní a již použitá šablona nepůjde smazat.

Správa školení

Upravit školení půjde pouze v případě, že školení ještě neproběhlo. Změnit půjde pouze místnost a počet volných míst. Jelikož školení závisí na šabloně, tak úprava názvu nebo popisu půjde přes změnu šablony.

2.3.5 Stránka lidé

Na této stránce půjde měnit role uživatelů aplikace. Tuto stránku má přístupnou pouze manager. Manager bude mít k dispozici seznamy lidí rozdělených podle jejich rolí (viz 2.1).

2.3.6 Stránka reporty

Tato stránka bude sloužit pro vygenerování reportu podle vlastních kritérií. Na tuto stránku bude mít přístup pouze manager. U každého reportu bude možnost vybrání vstupních dat, které daný report vyžaduje. Dále bude obsahovat tlačítko na generování reportu. Reporty, které půjdou generovat, a jejich popis naleznete v 2.2.9.

2.4 Drátěné modely

Na základě výše popsaných funkcí a stránek jsem navrhl drátěné modely. Každý model představuje jednu stránku aplikace. U některých modelů je také navrženo modální okno, které se objeví po kliknutí na určité tlačítko. Všechny modely lze nalézt v přílohách v části Drátěné modely.

3 Frameworky

Kapitola se zaměřuje na jednotlivé části aplikace, uživatelské rozhraní a databázi. V případě možnosti výběru je uvedeno více frameworků.

3.1 Uživatelské rozhraní - Vaadin

Framework, který má být použit na tvorbu uživatelského rozhraní, je pevně dán. Zadavatel měl v požadavcích, že uživatelské rozhraní musí být vytvořeno pomocí frameworku Vaadin.

Vaadin používá Javu pro tvorbu webových aplikací, které fungují bez úprav na spoustě podporovaných internetových prohlížečích či zařízeních. Zdrojový kód frameworku je napsán v Javě, takže programátor se vždy může podívat, jak je daná třída napsána. Použití frameworku je velice jednoduché, neboť framework má veliké množství komponent, díky kterým není nutné psát u každé nové komponenty úplné základy, ale stačí použít již vytvořené a ty si případně upravit podle libosti. Chování každé komponenty si lze přečíst jak ze zdrojového kódu, tak i z knížky [8], kde je popsáno nejen obecné chování komponent, většinou i s jednoduchým příkladem použití, ale i jakým způsobem je daná komponenta složena, co se týče *html* tagů a jejich *css* tříd. Knížka poskytuje pouze přehled o důležitých tématech. Kompletní přehled o všech třídách, rozhraních a jejich metodách je ve Vaadin API [14].

Vaadin na klientské straně podporuje Javu. Pomocí integrovaného GWT (Google Web Toolkit) Java-to-JavaScript překladače se Java přeloží do JavaScriptu. Výsledný JavaScript je následně interpretován v prohlížeči.

Framework je řízený serverem, takže veškerá logika je řešena na serverové části aplikace. To, že je logika řízena serverem, zvyšuje bezpečnost, protože na klientské části aplikace je pouze API na posílání a přijímání HTTP požadavků. Serverová část také kontroluje, zda akce posílaná od klienta je možná (tlačítko, které je vypnuté nesmí nic poslat). Pokud si toho server všimne, tak požadavek zamítne.

Jednotlivé komponenty jsou rozděleny na dvě části, klientskou a serverovou. Klientská část slouží pouze na zobrazení komponenty uživateli a nebo na interakci s uživatelem. Jakmile uživatel nějakým způsobem použije danou

komponentu, ta odešle na serverovou část požadavek. Server poté rozhodne, jaká akce bude provedena a odešle zpět odpověď. Tento způsob ovšem má jistou nevýhodu, a to že posílání a přijímání požadavků na server je časově náročné, takže vždy bude nějaké zpoždění mezi odpověďmi, ale tohoto zpoždění si uživatel většinou ani nevšimne. [8]

3.2 Databáze

Jelikož data budou uložena v relační databázi, bude potřeba framework, který implementuje rozhraní definované v JDBC (Java Database Connectivity). JBDC se stal standardem, který definuje jednotné rozhraní pro přístup k relačním databázím. Pro přístup do databáze a zpracování dotazu bude potřeba framework umožnující spojení s databází a provádění dotazů, nejlépe se správcem transakcí. Jako jádro bude nutný framework, který umí tvořit dotazy a následně je poslat přes framework zajišťující spojení.

Databázové frameworky na tvoření dotazů se dají rozdělit na psaní čistého SQL a na ty, co mají objektově relační mapování (ORM). U frameworků, které používají čisté SQL se využívá dotazů na databázi, které jsou známé z normálního SQL. U těchto frameworků se musí mapování dělat ručně. Výhodou těchto frameworků je to, že programátor má plnou kontrolu nad SQL dotazem, tedy se vyznačují větší rychlostí oproti frameworkům s ORM (znatelné na velkých databázích).

U frameworků, které používají ORM se využívá anotací (př. *@Id*), kterými se třída, představující tabulku v databázi, namapuje (označí). Anotace u objektů ve třídě představují parametry, se kterými se vytváří databáze. Některé frameworky zároveň zajišťují, aby data byla perzistentní. Použití pak spočívá v tom, že již nemusíme znát jméno tabulky/sloupečku v databázi, ale stačí jméno třídy/proměnné. Výhodou je, že programátor nemusí vytvářet při každém dotazu mapování, ale stačí jméno proměnné, kterou framework sám namapuje.

3.2.1 Frameworky bez ORM

Spring Framework JDBC

Spring Framework JDBC je část, která spadá do frameworku Spring. Tento balík poskytuje implementaci JDBC. Řeší vše okolo spojení s databází a skrz třídu *JdbcTemplate* lze také provádět jednotlivé dotazy na databázi.

Uživatel musí pouze definovat parametry, se kterými se připojí. Pokud uživatel chce provádět SQL dotazy, musí ještě napsat daný dotaz, nastavit parametry a případně projít výsledky. Framework za nás řeší otevření spojení, provedení dotazu, zpracování výjimek, transakcí a zavření spojení, dotazu a množiny výsledků. Framework se tedy za nás postará o všechny nízkoúrovňové detaily.

Třída *JdbcTemplate* je hlavní třídou balíčku JDBC. Třída řeší vytváření a uvolňování alokovaných zdrojů, což pomáhá vyhnout se běžným chybám, jako je zapomenutí zavření spojení. Třída také pomáhá s vytvářením a prováděním dotazů, ale samotné SQL a mapování programátor musí vytvořit sám. [3]

Querydsl-SQL

Querydsl je framework, který umožňuje konstrukci typově bezpečných, se syntaxí podobnou SQL, dotazů pro několik backendů, například JPA nebo SQL. Framework mapuje Java kód a z něho vytváří SQL dotaz. Framework umožňuje mít dotazy bez syntaktických chyb, protože dotazy se syntaktickou chybou nepovoluje. Doménové typy a vlastnosti mohou být bezpečně referencovány, nebot' framework nepoužívá žádné řetězce (String). Dotazy jsou psány, místo přes řetězce nebo XML soubory, skrz API (Java kód). [1]

3.2.2 Frameworky s ORM

Java Persistence API (JPA) je specifikace pro programovací jazyk Java. Specifikuje rozhraní pro programování a popisuje, jakým způsobem by měla probíhat správa relačních dat v aplikacích. Všechny uvedené frameworky s ORM implementují tuto specifikaci.

Implementující frameworky tvoří entity (entita reprezentuje data v databázi) pomocí anotací (například třída, která má být entitou, musí být anotována anotací *javax.persistence.Entity*) nebo *.xml* souboru.

Hibernate

Hibernate je ORM framework, který řeší perzistenci dat u relačních databází. Nabízí svojí vlastní API, která implementuje JPA specifikaci. Díky tomu může být použit ve všech prostředích podporujících JPA. Hibernate umožňuje rozvíjet perzistentní třídy, u kterých funguje OOP, tedy dědičnost, polymorfismus, asociace (1:1, 1:N, ...), kompozice (atribut odkazující na jiný objekt) a kolekce. Hibernate nepotřebuje žádné rozhraní nebo základní třídy pro perzistentní třídy a umožňuje jakékoliv třídě nebo struktuře dat být perzistentní.

Hibernate podporuje takzvanou línou inicializaci, díky níž je odezva dotazů mnohem rychlejší. Líná inicializace spočívá v tom, že Hibernate nenahrává úplně všechna data najednou, ale pouze část. V případě, že některá entita, nazvěme ji entita A, obsahuje v sobě jinou entitu, entitu B, tak Hibernate načte pouze entitu A, tedy tu, kterou si programátor vyžádal a zbytek dat, entitu B, načte, až když jsou data v ní žádána.

Dotazy SQL jsou generovány už při inicializaci, což zvyšuje rychlost za běhu aplikace. Sám Hibernate tvrdí, že nabízí vynikající výkon oproti přímému JDBC kódu. [9]

EclipseLink

EclipseLink je framework pro Javu, který implementuje spoustu specifikací včetně JPA či JAXB (Java Architecture for XML binding). JAXB funguje v podstatě na tom samém principu jako mapování v JPA, ale mapování není v Java třídě, ale v *.xml* souboru. EclipseLink může být použit pro interakci s různými datovými službami včetně relačních databází, nerelačních databází aj. EclipseLink podporuje Java perzistenci. Stejně jako u Hibernate je dotaz generován už při inicializaci. EclipseLink je založen na TopLinku od Oraclu. [6]

3.2.3 Vybraný framework

Z frameworků, které by mohly být použity, jsem si vybíral z těch, které podporovaly ORM a to z toho důvodu, že vytvoření entit a následné vytváření dotazů je velice jednoduché a není časově náročné. To, že frameworky s ORM jsou pomalejší než frameworky bez ORM, mi nevadí, protože moje databáze nebude tvořena z desítek tabulek, takže snížení rychlosti nijak nepocítím. Z uvedených kandidátů jsem si vybral Hibernate. Hibernate mi byl doporučen zadavatelem, ale byl vybrán také proto, že podporuje Spring (Spring popsán v 3.4) a má velkou komunitu programátorů, kteří ho používají. Na svých stránkách má spoustu příkladů, jak začít a má rozsáhlou dokumentaci, kde je vše vysvětleno. To, že je kompatibilní se Springem, byla pro mě největší výhoda, protože Spring jsem plánoval použít také.

3.3 Logování

Logování je důležitou součástí každé aplikace. At' už jsou ve formě klasické print line do konzole, nebo více propracované přes frameworky. Frameworky zjednodušují a standardizují logování, z toho důvodu se také používají.

Logování je například u java.util.logging API rozděleno na dvě části. První částí je Logger, který má za úkol zachycení zprávy a poslání do logovacího frameworku. Z Loggeru je pak zpráva poslána do Handleru. Handler rozhodne, na jaký výstup je zpráva poslána. Logger a Handler mají každý svůj filtr. Pomocí filtru se rozhodují, jestli zprávu pošlou dál. Handler má navíc možnost zprávu ještě před posláním na výstup zformátovat pomocí Formatteru.[11]

To, jakým způsobem je provedeno logování u jednotlivých frameworků, se může lišit, ale základní princip je stejný jako u java.util.logging API. Výhodou logovacího frameworku je, že výstup loggeru můžeme snadno změnit, protože stačí změnit jeden údaj v nastavení.

3.3.1 SLF4J

SLF4J je jednoduchý logovací systém, podle návrhového vzoru fasáda. Fasáda zjednodušuje komunikaci mezi uživatelem a systémem tím, že nahradí

komplikovaný systém rozhraním, které tento systém pokrývá. Fasáda tedy poskytuje rozhraní pro skrytí složitosti systému.

SLF4J je abstrakce pro různé logovací frameworky jako je například java.util.logging, logback nebo log4j. Framework umožňuje uživateli použití stejné konstrukce nezávisle na zvoleném výstupu. Výstup nelze měnit za běhu aplikace. [12]

3.4 Vkládání závislostí

Vkládání závislostí (anglicky Dependency injection [DI]) je ve své podstatě nastavování proměnných objektu, tedy vkládání závislostí je použito k tomu, aby jeden objekt mohl používat metody druhého objektu (objekt se nazývá komponenta). Bez použití DI komponenta, která chce použít komponentu druhou, musí komponentu inicializovat nebo získat referenci na již existující. S použitím DI se ale o inicializaci a následnou referenci na komponentu postará takzvaný injector (poskytovatel závislostí). K získání požadované komponenty za běhu stačí mít referenci na injectora, který pak může poskytnou požadovanou komponentu. DI se dá rozdělit na tři typy: Setter injection, Constructor Injection a Interface injection. Nejběžnější z nich je Setter injection. [7]

3.4.1 Spring

Spring je open-source framework zabývající se vkládáním závislostí. Spring pomáhá programátorům provádět transakce bez nutnosti použití transakčních API. Spring umožňuje vytvářet aplikace z běžných Java objektů (plain old Java objects - POJO) a aplikovat neinvazivně podnikové služby (enterprise services) na programovací model. Ačkoliv Java nabízí množství funkcí pro vývoj aplikací, postrádá prostředky na organizování základních bloků do souvislého celku. Tento úkol pak musí zastat programátor použitím návrhových vzorů, jako je abstraktní továrna (Abstract factory), továrna (Factory) aj. Vzory jsou normalizované osvědčené postupy, které se musí implementovat. Springovská složka Inversion of Control (IoC) pak zastává funkci programátora, kde nabízí právě nutné prostředky ke správě aplikace. Programátor pak nemusí implementovat návrhové vzory, ale stačí aby použil již implementované ve Springu.

Objekty se získávají pomocí vložení závislostí, takto získané objekty se nazývají Beans (beany). Spring podporuje všechny typy vkládání závislostí. Objekty jsou tvořeny buď na základě konfiguračního souboru nebo s využitím anotací. Při použití anotací může být Spring nastaven tak, aby prozkoumal pracovní adresář a našel všechny komponenty označené anotací *@Component*, ty pak zaregistruje do seznamu dostupných komponent. Spring využívá již hotových řešení (ostatní open-source frameworky) na řešení jednotlivých podúloh (například přístup k datům). Spring umožňuje využití pouze části jeho modulů.[10]

V případě použití Springu, který využívá anotace pro detekci závislostí, musí být použita anotace *@Autowired*. Tato anotace zajistí, že hned po inicializaci beanu se vloží daná závislost, vložení ale neprobíhá už při zavolání konstruktoru dané komponenty, ale až po provedení konstruktoru. Pro použití vložené závislosti už při inicializaci je nutné vytvořit speciální metodu, zastupující konstruktor, která se označí anotací *@PostConstruct*. Tato anotace zajistí, že se tento "konstruktor" zavolá až při vložení všech závislostí. Popis všech modulů Springu společně s příklady lze nalézt v [2].

3.5 xpoft / Spring-Vaadin

Spring-Vaadin je add-on pro Vaadin. Add-on pomáhá spojit Spring a Vaadin. Definuje novou anotaci *@VaadinView*, která se používá na označení jednotlivých stránek. Tato anotace pak slouží pro nalezení jednotlivých souborů stránek navigátorem *DiscoveryNavigator*. Tento navigátor se nemusí díky tomu nijak nastavovat, protože všechny stránky najde automaticky. [4]

4 Nastavení aplikace

Předtím, než bylo možné použít dané frameworky, bylo nutné je správně nastavit. K tomu, aby výsledná aplikace mohla být spuštěna, bylo nutné vybrat si nástroj na správu a sestavení projektu. K těmto účelům jsem použil Maven. Vybral jsem si ho, protože jsem s ním byl již seznámen z dřívějších projektů. Maven je nástroj pro správu, řízení a automatizaci sestavování aplikace. Základem fungování Mavenu je popsání projektu v souboru Project Object Model (POM). Model se nachází vždy v kořenovém adresáři. POM obsahuje všechny závislosti a popisuje proces sestavení. Maven se stará o dodání nadefinovaných závislostí a pluginů (použitých při sestavování), k tomu účelu existuje repositář, ze kterého čerpá zdroje. Maven má jednotnou strukturu, kde zdrojové soubory jsou vždy v *src/main/...* nebo *src/test/...*, tyto adresáře jsou pak navíc rozděleny na Java kód (**/java*) a konfigurační soubory (**/resources*) . Při sestavování se prohledává adresář *src/test* a když jsou nalezeny nějaké testy, spustí se. Výstup sestavení je většinou generován do souboru *target*. [5]

4.1 Spring

K tomu, aby Spring věděl kde najít všechny komponenty a aby komponenty vůbec hledal, musel být nastaven. První, co se muselo dodat do projektu, byly listenery Springu. Nutnými listenery, které musí být deklarovány v souboru *web.xml* byly *ContextLoaderListener* a *RequestContextListener*. Aby *ContextLoaderListener* fungoval, musel být nastaven ještě parametr context-param, který říká, kde a s jakým jménem má Spring hledat soubory s nastavením. Listener pak slouží pro start a ukončení Springovského kořenového kontextu *WebApplicationContext*. *RequestContextListener* slouží pro odhalení žádostí pro aktuální vlákno. Dalším parametrem, který Spring musí mít v souboru *web.xml* je filtr *DelegatingFilterProxy*, který umožňuje použití modulu Spring security. Listenery Springu prohledají a zaregistrují každou třídu, která má anotaci *@Component*. Při požadavku na získání beanu se navíc může upřesnit, jestli je bean jedináček (singleton) nebo se po každém zavolání vytváří nový (prototype). K tomuto účelu slouží anotace *@Scope* (singleton a prototype nejsou jedinými možnostmi).

Soubory s nastavením jsou uloženy ve zdrojích (*resources*) aplikace. Soubory jsou rozděleny tak, aby každý nastavoval pouze některou část aplikace. Každý soubor s nastavením je pojmenován podle vzoru *spring-*.xml*. Základním souborem je *spring-config*, ten nastavuje základní strukturu souborů (v mém případě: cz.zcu.skoleni), ve které se mají hledat komponenty. Další funkcí tohoto souboru je nahrání *.properties* souborů do *PropertyPlaceholderConfigurer*. Dalším souborem je *spring-locales.xml*, tento soubor má za úkol nastavit pozici souborů s texty aplikace. Pro získávání *SessionFactory* poskytnuté Springem je nutné vytvořit bean, který tuto továrnu správně nastaví. Tento bean se nastavuje v souboru *spring-hibernate.xml*, *spring-ldap.xml* má za úkol nastavit připojovací parametry LDAPu. Nastavení Jobu probíhá v souboru *spring-scheduler.xml*, tento soubor pouze spouští daný plánovač, který se už o vše postará sám. Job je v mém případě metoda, která je označena anotací *@Scheduled*. Tato metoda se potom může nastavit tak, že se spustí v určitý den v týdnu, čas atd. Posledním nastavovacím souborem Springu je soubor *spring-security.xml*, ten nastavuje jakým způsobem a kde má ověřovat přihlašovací údaje.

4.2 Hibernate

Všechny důležité připojovací parametry jsou v *.properties* souboru. Takto uložené parametry nabízí snadnou změnu jejich hodnot. Potřebné údaje pro připojení do databáze jsou jméno třídy, ve které je Driver na databázi, připojovací url, jméno a heslo pro přístup do databáze (admin), dialekt databáze a nakonec co se má stát s databází při spuštění serveru. Ukázkový soubor lze nalézt ve zdrojových souborech aplikace v *src/main/resources/properties/example-database.properties*.

Jelikož využívám Spring na vložení závislostí, měl jsem tu možnost vytvořit si bean *sessionFactory*, který zastává funkci továrny. Továrnou se myslí, že při zavolání funkce vrátí vždy stejně nastavený objekt. V tomto případě se *sessionFactory* využívá na vytváření spojení s databází, které se pak používají na provádění dotazů. Aby *sessionFactory* správně fungoval, je třeba mu dodat pár parametrů. Jedním z nich je umístění entit. Dalšími parametry jsou parametry uvedené v *.properties* popsaném výše. Soubor, ve kterém je vytvořen tento bean, lze nalézt ve zdrojových souborech v *resources/spring-hibernate.xml*, v tomto souboru je také nadefinován manažer transakcí. Manager má jako vstupní parametr *sessionFactory*.

4.3 SLF4J

Tento framework má také jeden soubor s nastavením. Soubor definuje na jaký výstup a v jakém formátu bude výstupní zpráva. Aktuální nastavení frameworku je, že výstup se připojuje do konzole. Formát zprávy je: datum a čas vzniku události, v jakém vlákně se událost stala, jakou úroveň závažnosti událost má, jaký logger byl použit k zápisu a v jaké metodě se událost stala. Na konci je samotná zpráva. Logger je nastaven tak, aby vypisoval všechny debugové a varující zprávy. Soubor s tímto nastavením je v *resource/logback.xml*.

5 Backend

Backend je část aplikace, se kterou se běžný uživatel nesetká. V mém případě je to vše ohledně databáze a LDAPu.

5.1 Návrh a implementace databáze

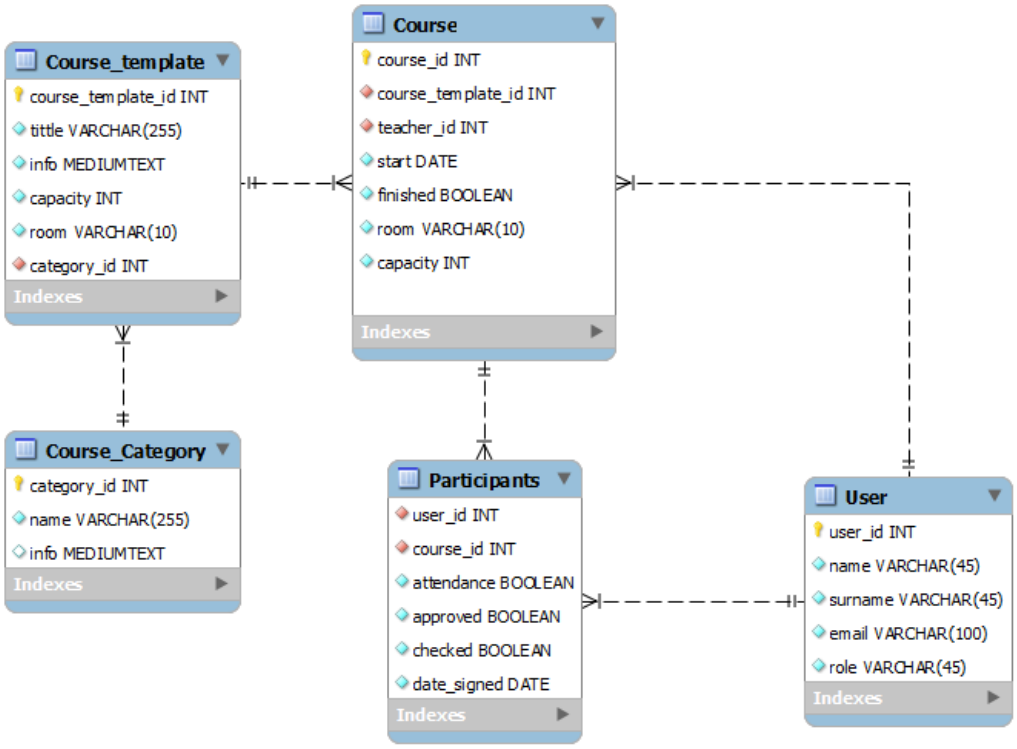
5.1.1 Návrh modelu databáze

Po analýze zadání jsem vytvořil ERA model databáze, viz Obrázek 5.1. Pro uchování všech potřebných dat aplikace mi stačilo celkem pět tabulek. Databáze by mohla mít ještě jednu tabulku navíc, která by obsahovala výčet rolí uživatelů. Tu jsem nakonec vypustil, protože tabulka by byla velice malá a výsledný efekt je pak stejný, jako když si role uložím v nějaké pomocné třídě, kde přidání nebo ubrání role je stejně jednoduché jako přidání nebo odebrání z databáze.

První tabulku, kterou popíši, je tabulka kategorií (**Course_Category**). Tabulka je velice jednoduchá, neboť vše, co je potřeba vědět o kategorii, je jen její název, který je povinný. Popis kategorie (*info*) jsem nastavil jako nepovinný. Plánuji ho zobrazovat jen pro managery při editaci kategorie. Popis pak bude sloužit jen jako dodatečná informace, kdy danou kategorii použít. Samozřejmě tabulka má ID. Hledat v databázi jednu určitou kategorii půjde pomocí unikátních sloupců ID a názvu kategorie(*name*).

Další tabulkou je tabulka uživatelů (**User**). Tabulka uživatelů má všechny sloupce povinné a kromě sloupce ID má unikátní sloupec email (*email*). Každý uživatel by měl být v databázi jen jednou, protože jinak by se mohl na stejné školení přihlásit dvakrát pod stejným jménem. Dalšími povinnými údaji je jméno uživatele (*name*) a příjmení (*surname*). Dále každý uživatel dostane svoje ID a roli (*role*). Role se nastaví při prvním vytvoření uživatele na defaultní hodnotu. Obsah tohoto pole záleží na nastavení konstant uložených v některé ze tříd aplikace.

Třetí tabulkou je tabulka šablon (**Course_template**). Kromě atributu ID obsahuje sloupec s názvem šablony (*tittle*), který musí být unikátní v rámci tabulky. Dalším sloupcem je popis (*info*), který slouží k popisu šablony. Tyto



Obrázek 5.1: ERA model databáze

dva údaje (název a popis) se vztahují už k jednotlivým školením. Dalšími povinnými údaji je standardní (defaultní) místo konání (*room*) a kapacita (*capacity*). Posledním údajem je cizí klíč ID kategorie (*category_id*). Šablony jsou závislé na kategorii a to vazbou N:1.

Další tabulka představuje tabulku školení (**Course**). Tabulka je závislá na tabulce uživatelů, a to vazbou N:1. Uživatelské ID (*teacher_id*) v této tabulce představuje uživatele, který je vypsán u školení jako vyučující. Tabulka má ještě druhou vazbu, kde školení závisí na tabulce se šablonami. Vazba je také N:1, tedy N školení může mít jednu šablonu. Na jaké šabloně je dané školení závislé představuje sloupec *course_template*. Školení touto závislostí získává svůj název a popis. Změna oproti šabloně je pak v tom, že školení má svoje vlastní sloupce s místností (*room*) a kapacitou (*capacity*). Školení má sloupce, kde jeden indikuje, kdy má školení začít (*start*) a druhý (*finished*) říká, zda je školení již ukončeno.

Poslední tabulkou v databázi je tabulka s jednotlivými zápisy na školení (**Participants**). Tabulka má tři sloupce vyjadřující stav přihlášky. Slou-

pec zkontrolováno (*checked*) značí, zda přihláška již byla nějakým způsobem vyhodnocena. Sloupec povoleno (*approved*) značí, zda uživateli byl povolen nebo zamítnut přístup na dané školení. A posledním sloupcem je sloupec účasti (*attendance*). Tento sloupec vyjadřuje, jestli se uživatel zúčastnil daného školení. Navíc má tabulka ještě sloupec říkající kdy se uživatel na školení přihlásil (*date_signed*). Tento sloupec slouží jen k možnosti řazení podle data přihlášení. Tabulka má dvě závislosti. První z nich říká, jaký uživatel podal přihlášku (*user_id*), tato závislost je N:1, tudíž jeden uživatel může mít podaných N přihlášek. Druhou závislostí je závislost vyjadřující na jaký kurz se uživatel zapsal (*course_id*). Závislost je opět N:1.

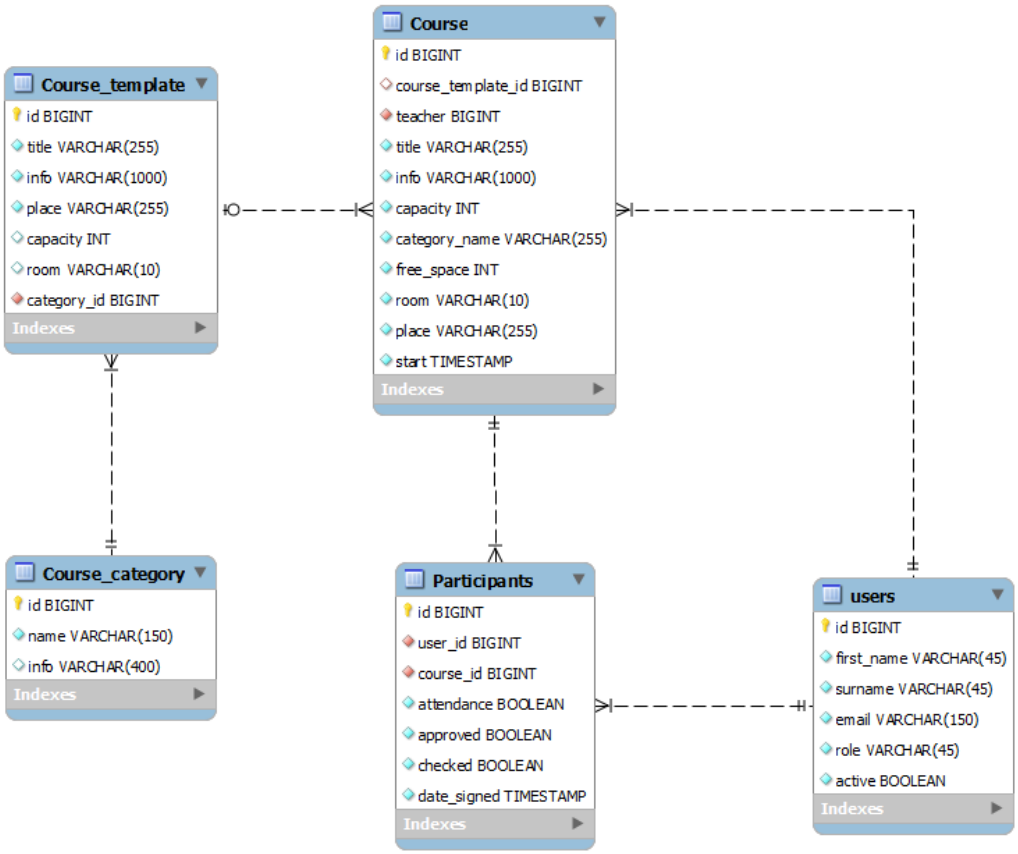
5.1.2 Úpravy modelu databáze

Během vývoje jsem musel databázi ještě trochu pozměnit, neboť nevyhovovala mým potřebám. Zjistil jsem, že některé sloupce jsou zbytečné a zadavatel měl ještě nějaké požadavky navíc. Nový model naleznete pod označením Obrázek 5.2.

Tabulky s kategoriemi a se zápisy na školení zůstaly nezměněné. Tabulka s uživateli se nepatrně změnila, přibyl jeden sloupec (*active*), který značí, zda je uživatel stále aktivní.

V tabulce se šablonami byly provedeny jen malé změny, sloupce kapacita a místnost byly změněny z povinných sloupců na nepovinné. Tato změna je z toho důvodu, že pokud nechceme mít vždy nastavena nějaká základní data, tak nyní máme možnost to tak udělat.

Největší změna byla v tabulce školení, protože zadavatel mi během implementace dodal ještě další požadavek na vytváření školení. První požadavek byl, že školení nemusí být vždy vytvářeny pomocí šablony, ale může se vytvořit 'jednorázové' školení, které nemusí mít nastavenou žádnou šablonu. Na základě tohoto požadavku jsem se rozhodl, že nebudu přidávat novou tabulku jenž by měla školení bez šablon, ale pouze pozměním stávající tabulku školení. Tuto možnost jsem si zvolil, protože tento požadavek jsem dostal až téměř u konce vývoje aplikace a tohle bylo nejjednodušší a nejrychlejší řešení, které mě napadlo. Vazba mezi šablonami a školeními se změnila na nepovinnou. Tím, že šablona již nemusela být povinná, se mi ale ztratil údaj do jaké kategorie školení patří, tedy tento sloupec (*category_name*) jsem doplnil do tabulky. Bohužel název kategorie se pak stává údajem, který v případě vyplněné šablony lze nalézt na dvou místech. Jako druhý požadavek bylo,



Obrázek 5.2: ERA model databáze - konečný stav

že každé školení bude moci mít svůj vlastní název a popis, nezávisle na šabloně. Do tabulky proto přibyly sloupce *title* a *info*, které značí název a popis školení. Posledním požadavkem bylo, že školení by mělo být doplněno ještě o místo konání, proto v tabulce přibyl sloupec místo (*place*). Během vývoje jsem ještě narazil na jednu věc, na kterou jsem při návrhu zapomněl. Zapomněl jsem totiž na počet volných míst, které daný kurz má, proto jsem doplnil tabulku o sloupec *free_space*. Sloupec, v původním návrhu *finished*, jsem vymazal, protože jsem zjistil, že pro mě není potřebný, poněvadž jestli školení bylo již ukončeno, lze určit z data konání.

5.1.3 Vytvoření entit

K vytvoření entit bylo zapotřebí ERA modelu. Podle modelu jsem pak vytvářel jednotlivé třídy. Aby Hibernate poznal, že tato třída představuje tabulku databáze, bylo nutné použít anotaci *@Entity* a anotaci *@Table* s parametrem *name*. Touto anotací se definuje jaké bude mít tabulka v databázi jméno. Pro definování ID entity se použije anotace *@Id*. Jelikož jsem chtěl, aby se ID samo generovalo, připsal jsem navíc ještě anotaci *@GeneratedValue*. Pro jednotlivé sloupce se pak vytvoří proměnné a anotacemi se definuje, jaké vlastnosti mají v databázi mít.

```
1  @Entity
2  @Table(name = "course")
3  public class Course implements Serializable {
4
5    @Id
6    @GeneratedValue
7    private long id;
8
9    @Column(name="title", nullable=false, length=255)
10   private String title;
11
12   @Column(name="capacity", nullable=false)
13   private int capacity;
14
15   @ManyToOne
16   @Cascade({CascadeType.SAVE_UPDATE})
17   @JoinColumn(name="teacher", nullable=false)
18   private User teacher;
19
20   ... Ostatni promenne tridy vynechany...
21
22   ... Getry a setery promennych vynechany...
23 }
```

Kód 5.1 : Ukázka entity

Například v Kód 5.1 je uvedena proměnná *title* (řádek 10). Proměnná je typu *String*, takže v databázi bude typ tohoto sloupce *character varying*. Tento typ je také zvolen na základě definice sloupce (entita *@Column*), ve které se říká jaké jméno (*name*) bude sloupec mít, jestli je sloupec nepovinný (*nullable*) a jakou maximální délku (*length*) může záznam mít. Proměnná na řádce 13 je typu *Integer*, takže v databázi bude tento sloupec mít typ *integer*. Vazba s tabulkou **User** je N:1, k popisu vazby slouží anotace *@ManyToOne*

a anotace *@Cascade* říká, co se stane, pokud nějakým způsobem změníme data v proměnné teacher. Jelikož proměnná teacher je vlastně sloupec v databázi, tak hodnotou *CascadeType.SAVE_UPDATE* se řekne, že pokud je objekt nějak změněn, tak se uloží nebo aktualizuje. Poslední anotací u proměnné je *@JoinColumn*. Tato anotace říká, jak se má jmenovat sloupec v databázi a jestli může být prázdný. Třída není kompletní, chybí ještě zbytek proměnných a metody na získávání a nastavování proměnných. Metody musí být ve třídě uvedeny, jinak by Hibernate neměl jak nastavit proměnné podle databáze a já bych neměl jak tyto hodnoty získat. Všechny anotace jsou z balíku *javax.persistence*

5.1.4 DAO vrstva, business services

DAO vrstva (data acces object) je podle vzoru Data Acces Object. Tento vzor definuje jakým způsobem by se měly vytvořit základní třídy pro přístup dat. Pro vytvoření všech základních dotazů do databáze jsem vytvořil abstraktní rozhraní, které má pouze metody uložení objektu do databáze, aktualizace, nalezení podle ID, vymazání podle ID a nebo podle objektu. Po definování tohoto rozhraní jsem ho implementoval. Implementuje ho abstraktní třída, která funguje stejně na všech tabulkách, funkce budou mít stejnou implementaci pro všechny další DAO třídy. Implementace spočívá v tom, že třída, pro kterou by měla být implementace určena, je neznámá, v Javě se značí $<T>$. Potřeba použití generického názvu třídy byla ve dvou metodách, nalezení podle ID a nalezení všech záznamů v tabulce. Jelikož nalezení je závislé na tom v jaké tabulce se má hledat, bylo nutné zjistit název tabulky. Název tabulky se získá pomocí *GenericTypeResolver*, ten má jako návratovou hodnotu jméno třídy (stejné jako Třída.class). V Kód 5.2 je na řádce 1 uvedena hlavička abstraktní DAO třídy a na řádce 3 je pak implementující třída, která už má místo obecné třídy T skutečnou třídu (v tomto případě entitu *CourseCategory*). Každá metoda otevírající spojení má klauzuli try, catch, finally. V případě, že nastane nějaká chyba, tak chyba se v catch zachytí a loggerem se zaloguje, poté se skočí do finaly, kde se spojení uzavírá.

```
1 public abstract class AbstractDAO<T extends Serializable>
    implements AbstractDAOInterface<T>{...}
2
3 public class CourseCategoryDAO extends AbstractDAO<
    CourseCategory>{...}
```

Kód 5.2 : Hlavičky abstraktní DAO třídy a implementující třídy

Business services jsou třídy, které slouží k oddělení programátora od implementace. V tomto případě oddělení frontendu od backendu. Třída poskytuje metody, které vrací rovnou výsledek. To jakým způsobem se k tomu výsledku dostaneme už není podstatné. Ve Springu je na tento typ tříd připravena anotace *@Service*. V přílohách v sekci UML diagramy lze nalézt UML diagram, pod názvem UML diagram 1, zobrazující abstraktní třídy DAO a service a jejich potomky.

5.1.5 Testovací třída

K testu funkčnosti databáze jsem vytvořil třídu na testování (*HibernateTest*). Třída má za úkol otestovat přístup k databázi a ověřit funkčnost základních metod, uvedených v abstraktním rozhraní (*AbstractDAOInterface*). Každá metoda, která představuje jednotlivé testy má anotaci *@Test*, testy se automaticky spustí při sestavování aplikace.

5.2 LDAP

LDAP (Lightweight Directory Access Protocol) definuje protokolové zprávy použité adresářovými klienty a servery. LDAP se stal standardem, který definuje standardní metody na přístup a aktualizaci informací v adresáři. Data jsou uložena na serveru s adresářovou strukturou a jsou ukládána formou záznamů. Samotná adresářová struktura je definována uživatelem. Adresáře jsou optimalizovány na čtení dat, zápis dat může být limitován pouze na administrátory. Adresáře jsou proto použity tam, kde se data mění jen zřídka. [13]

LDAP je použit jako přídavná databáze, která slouží pouze pro získání nutných informací o uživateli a ověřování přihlašovacích údajů. Každý uživatel má v aplikaci svoje práva nezávisle na již přiřazených právech, které v LDAPu mohou být. Na serveru je uložen seznam lidí, kteří mohou potenciálně aplikaci využívat. Pokud uživatel není na serveru, nemůže se přihlásit do aplikace.

LDAP je použit, protože zadavatel měl požadavek, aby ověřování uživatelských údajů probíhalo právě přes LDAP.

K získání informací ze serveru bylo nutné vytvořit si třídu, do které se nahrají potřebné informace. Třída *Person* má pouze svoje proměnné a k nim odpovídající metody na získávání a nastavování proměnných. Tuto třídu pak využívá třída *CustomUserDetails*, která implementuje rozhraní *UserDetails*. *UserDetails* je rozhraní definující základní metody, které LDAP normálně u vráceného objektu poskytuje. Bohužel byly nutné ještě informace, které normálně nejsou dostupné pomocí standardních tříd. Hodnoty, které byly potřebné, korespondují s databází, tudíž u každého uživatele musí být znám email, jméno a příjmení. Abych naplnil třídu *CustomUserDetails*, musel jsem implementovat a nahradit stávající *UserDetailsContextMapper*. Třída pak musí implementovat dvě metody, jedna, když je přijímán objekt z LDAPu a druhá, když je posílán do LDAPu. Druhou metodu nepotřebuji, protože hodnoty v LDAPu měnit nebudu. První metoda pouze získává ze záznamu v LDAPu hodnoty a nastavuje je do připravených tříd na uchování.

K samotnému získání záznamu z LDAPu byla vytvořena DAO třída, nazvaná *LdapDAO*. Třída poskytuje dvě metody, *loadPeople* a *getPerson*. První metoda slouží na získání všech záznamů lidí, co jsou na serveru. Druhá metoda slouží na získání konkrétního uživatele podle jeho přihlašovacího jména. Metoda *getPerson* je pak využita ve třídě *UserDetailServiceImpl*, která implementuje *UserDetailsService*. Třída má jedinou metodu *loadUserByUsername*, která slouží k načtení informací o uživateli (vrací *CustomUserDetails*).

Metoda *loadPeople* ze třídy *LdapDAO* se využívá při vykonávání Jobu. Metoda (*synchUsers*), která má tuto anotaci, je ve třídě *JobUserSynch* a slouží k synchronizaci uživatelů mezi databází aplikace a LDAPu. V případě, že uživatel je v LDAPu, ale není v databázi, tak se uživatel nahraje do databáze. V opačném případě se uživatel v databázi označí, že není aktivní (sloupec *active*).

6 Frontend

Základ aplikace tvoří třída *MyVaadinUI*, která vytvoří základní rozložení s hlavičkou nahoře, kde jsou ovládací prvky na přepínání a odhlášení. Pod hlavičkou je místo na obsah stránky. Třída po vytvoření hlavičky a místa na obsah vytvoří *DiscoveryNavigator* z add-onu Spring-Vaadin. *DiscoveryNavigator* najde všechny stránky, které jsou v aplikaci definovány speciální anotací *@VaadinView*, jako parametr do této anotace slouží název stránky. Zvolený název bude viditelný v url.

6.1 Základní třídy aplikace

Vytvoření základních tříd je výhodné z důvodů, usnadnění orientace ve zdrojových kódech aplikace a umožňuje použít již jednou napsaný kód.

MessagesLoader

Třída je velice jednoduchá, má pouze jednu metodu na zjišťování hodnoty podle klíče. Všechny texty, které jsou v aplikaci zobrazovány uživateli, jsou uloženy ve speciálním souboru, který má následující strukturu: Klíč = hodnota, z tohoto souboru pak třída čerpá hodnoty. Výhodou tohoto způsobu uložení textů je, že pokud chceme změnit nebo přeložit text do jiné řeči, stačí změnit hodnotu v tomto souboru. Umístění souboru je v *resource/locales/-messages*.properties*, pozice je nastavena v konfiguračním souboru.

ComponentContentBase

Z této třídy dědí každý obsah stránky (pod jednotlivými záložkami), tato třída má pouze vizuální úkol, který vytváří ohraničení obsahu. V přílohách, v sekci s UML diagramy pod názvem UML diagram 8 lze nalézt UML diagram se všemi potomky této třídy, každý potomek třídy pak představuje jeden obsah záložky.

ComponentAddTrainingBase

Tato třída slouží k vytváření školení. Třída vytváří prázdný formulář, díky kterému je možné zadávat potřebné údaje k vytvoření školení. Formulář je validován pomocí *FieldGroup*. Nejdříve se vytvoří potřebná pole a pak se spojí s objektem *FieldGroup*. K tomu, aby objekt mohl uchovávat hodnoty z políček, bylo nutné vytvořit třídu (*ComponentCourseBean*) k tomuto účelu. Tato třída může navíc mít anotace, díky kterým *FieldGroup* dokáže validovat (říct, jestli je hodnota správně zadána) hodnoty uložené v polích. Validace políček je nastavená tak, aby odpovídala potřebným polím v databázi, včetně kontroly typu zadané hodnoty.

Komponenta má dva rozbalovací seznamy, jeden uchovává kategorie a druhý šablony. Pokud je vybrána kategorie, tak se rozbalovací seznamy se šablonami naplní pouze šablonami z dané kategorie.

ComponentTemplatesAddBase

Třída slouží k přidávání nových šablon. Třída vytváří prázdný formulář, do kterého je možné zadávat potřebné údaje k vytvoření šablony. Formulář je validován stejným způsobem jako ve třídě *ComponentAddTrainingBase*, ale objekt, do kterého se ukládají data, je vytvořen podle požadavků šablony.

ComponentTemplatesAddBase má dva potomky. Jeden potomek (*ComponentTemplatesAdd*) je kopií rodiče, ale má navíc jednu metodu *reload*. Druhý potomek (*ComponentTemplatesEdit*) slouží na editaci již vytvořených šablon, třída se liší ve zpracování hodnot z políček a navíc má metodu na nastavení políček podle stávající šablony.

ComponentReportsBase

Abstraktní třída je základem pro vytváření reportů, třída přednastavuje rozmístění prvků a nastavení popisků. Každý potomek musí implementovat nadefinované abstraktní metody. Jelikož některé popisky jsou pokaždé jiné, tyto popisky musí dědící třída nastavit, nastavení popisků je pomocí metod k tomu určených. Další metody slouží k přidání tabulek pro zdroj dat a potřebných funkcí na přehazování prvků v tabulkách. Posledními abstraktními metodami jsou metody na generování reportu a nastavení referencí tabulek. Reference

jsou nastaveny tak, že tabulka na jedné straně má vždy referenci na tabulku na opačné straně (např. levá na pravou). V přílohách v sekci UML diagramy lze najít pod názvem UML diagram 9 UML diagram, vysvětlující strukturu základní třídy, včetně výčtu abstraktních metod a jejích potomků.

Potomkem *ComponentReportsBase* je třída *ComponentReportTrainingBase*. Třída implementuje všechny abstraktní metody kromě jedné, ta nemůže být implementována, protože každý report bude mít tuto metodu jinou.

ComponentPeopleContentBase

Tato třída vytváří tabulku, do které se vkládají uživatelé z databáze. To, jací uživatelé se přidají, určuje metoda *getUserList*. Tato metoda je ve třídě označena jako abstraktní. Potomci pak implementují abstraktní metodu tak, že vrátí jen uživatele s danou rolí.

ComponentTrainingDetailBase

Třída slouží pouze k zobrazení všech informací o školení. Třída má jednu abstraktní metodu pro vytvoření spodní části komponenty. Ve spodní části v běžném detailu není vůbec nic, ale komponenta na přihlašování na školení (*ComponentTrainingSignIn*) vyžaduje v dolní části tlačítko, které zajišťuje přihlašování nebo odhlašování se ze školení.

ComponentTwoButtonsBase

Třída obsahuje pouze dvě tlačítka u kterých je jejich funkce ještě neznámá. Většinou třída slouží jako druhé potvrzení při odebírání školení, šablon nebo kategorií z databáze. Druhým případem použití je u reportů, kde se při generování zobrazí okno. Obsahem okna je tato třída se změněnými popisky a vlastními funkcemi tlačítek. Třída potom slouží na generování reportů do okna nebo do souboru.

ComponentTable

Třída je Vaadinskou tabulkou, kde se v konstruktoru nastavují základní parametry tabulky. V post konstruktoru je použita abstraktní metoda, která má za úkol přidat jednotlivé sloupečky do tabulky. Potomkem této třídy je například *ComponentCourseTable*, ta je základem pro tabulku, jejíž obsahem jsou informace o školení. Třída obsahuje abstraktní metody *createButtons*,*getSubWindow* a *reload*. Metoda *createButtons* vytváří tlačítka k jednotlivým řádkům tabulky. Metoda *reload* slouží pro znovunahrání obsahu a poslední metoda slouží na získání objektu okna (*Window*). Kompletní přehled potomků třídy *ComponentTable* lze nalézt v přílohách v sekci UML diagramy, pod názvem UML diagram 10.

ViewBase

Třída slouží jako základ stránek (view). V post konstruktoru třídy jsou pouze nastaveny různé parametry týkající se vzhledu a je nastaven indikátor stránky (v hlavičce aplikace). Nastavení indikátoru je pomocí abstraktní metody, kterou pak každá stránka implementuje. Dále má implementovanou metodu *enter*, která zajistí, že po příchodu na stránku je uživatel přihlášen.

6.2 Přihlášení a práva

Po příchodu do aplikace se nepřihlášený uživatel dostane na stránku, kde se zadávají přihlašovací údaje. Stránka je jednoduchá, má pouze dvě políčka a tlačítko. Po zadání přihlašovacích údajů a kliknutí na tlačítko se pomocí *LdapAuthenticationProvider* ověří údaje v LDAPu. Nastavení *LdapAuthenticationProvider* je v *xml* souboru *spring-security.xml*, potřebné atributy pro nastavení jsou v souboru *ldap.properties*. Pokud jsou zadané údaje správně zadány, uloží se informace o uživateli do třídy *SecurityHelper*. Tato třída je výjimečná tím, že má scope(*@Scope*) rovný session, v tomto případě se bean vytváří pro každou HTTP relaci. Existují dvě výjimky kdy tomu tak není. První nastane, když uživatel v databázi neexistuje (nestačil proběhnout Job na synchronizaci). Řešením je nahrání uživatele do databáze. Druhá výjimka ukazuje na skutečnost, že uživatel je v databázi nastaven jako neaktivní. V tomto případě se pouze změní hodnota v databázi. Po úspěšném přihlášení a nastavení údajů do *SecurityHelper* se přesměruje na stránku domů.

U každého uživatele, který chce přesměrovat na nějakou stránku, se kontroluje, zda na danou stránku má přístup. Tato kontrola se provede ještě předtím, než se přesměruje na požadovanou stránku. Toho je dosaženo tím, že do *DiscoveryNavigator* se přidá *ViewChangeListener*. V tomto listeneru se kontroluje, zda přihlášený uživatel má roli potřebnou pro přístup na stránku určenou pouze pro managera. Jelikož v tomto listeneru nejde použít přesměrování, každá stránka kontroluje, jestli je uživatel přihlášený. V případě, že není, proběhne přesměrování na přihlašovací stránku.

6.3 Reporty

Generování všech reportů probíhá na stejném principu. Každý report nejdříve získá seznam, podle kterého má daný report vytvořit. Po získání zdrojového seznamu se získá seznam všech záznamů, nejlépe v jedné transakci, odpovídajících kritérií. Z těchto záznamů se report vytvoří. Data reportu se generují hned při otevření okna s volbou zobrazení nebo stažení reportu. To umožňuje mít větší rychlost zobrazení, neboť uživateli bude vždy trvat nějakou dobu, než klikne na tlačítko pro zobrazení/stažení reportu. Pro vytvoření souboru se vytváří postupně řetězec, do kterého se přidávají jednotlivé řádky. Jakmile je řetězec kompletní, vytvoří se dočasný soubor *.csv*. Jméno tohoto souboru se odvíjí od času, kdy je soubor generován. Výsledný název staženého souboru je tvořen názvem dočasného souboru a navíc Vaadin nakonec automaticky přidá svůj vygenerovaný řetězec. Po vytvoření dočasného souboru a naplnění daty se vytvoří *FileDownloader*, který se připojí na stahovací tlačítko. *FileDownloader* pak zajistí stažení souboru ze serveru k uživateli.

6.4 Stránky aplikace

Jak už bylo řečeno, aplikace byla rozdělena na stránky. V přílohách, v sekci UML diagramy jsou UML diagramy všech stránek. UML diagramy, které znázorňují závislosti stránek jsou:

1. Diagram stránky domů
2. Diagram stránky školení
3. Diagram stránky správa

4. Diagram stránky reporty
5. Diagram stránky lidé
6. Diagram přihlašovací stránky

V těchto diagramech je naznačena struktura jednotlivých stránek. Hloubka zobrazených závislostí a podrobnosti o třídách záležely na dostupném místě.

7 Zkušební provoz

Aplikace byla zkoušena během implementace mnou a zadavatelem práce. Zadavatel měl po celou dobu vývoje přístup ke zdrojovým souborům. Pokud byla nalezena nějaká chyba, byla opravena ihned, nebo byla zapsána na pozdější opravení. Jakmile byla aplikace ve stavu, kdy byly všechny funkce implementovány, ale nebyl ještě hotový design (kromě hlavičky, tlačítek a jednotlivých tabů) a nebyla vyřešena práva uživatelů, vyzval jsem zadavatele k podrobnějšímu vyzkoušení. Z tohoto prozkoušení zadavatel přidal jeden funkční požadavek na aplikaci a pár designových(kosmetických). Většina připomínek se týkala designu aplikace, v tomto směru to byly spíše rady. Funkční požadavky se změnily u přidávání školení, kdy zadavatel chtěl, aby nebyla nutná šablona. Z tohoto vyzkoušení mám také následující hlášení od budoucího uživatele aplikace Mgr. Michaely Bolckové:

> „K profilu lektor: je dobré, že lektor má přehled o svých kurzech, které vyučuje, a lidech, kteří mu na kurz chodí. Pokud by tento přístup měl platit pro běžného uživatele, zvážila bych míru pravomocí. Nemyslím si, že by účastník měl možnost potvrdit přijetí do kurzu – to by měl schvalovat lektor a admin. Mnohdy mám pocit, že se role prolínají a běžný uživatel má více možností, jako například přidat kurz. A to by určitě neměl. Z pozice administrátora přehledný způsob vytváření šablon a zadávání kurzů. Musím ocenit možnost reportů, kdy se generují základní data.
>
> Celkově je správa školení praktická a přehledná. Za mě jde o jednoduchý způsob evidence kurzů. Proto, aby se mohla plně používat v praxi, je třeba lépe vymezit roli administrátora x lektora x uživatele a „vyladit“ chyby. Design koresponduje se značkou společnosti.”

Samozřejmě všechny poznámky ohledně práv uživatelů byly vyřešeny. Chyby, o kterých je řeč, jsou spíše výjimky (speciální případy). Všechny, na které jsem přišel, byly opraveny. Požadavek na přidávání školení bez šablony byl také vyřešen. Po vyřešení všech nalezených problémů jsem se pustil do designu aplikace. Většinou jsem jen řešil pozicování prvků na stránce a jejich odsazení od jiných prvků, či okrajů. Po dokončení vzhledu aplikace jsem vyzval zadavatele k dalšímu prozkoušení. Z toho už ale nemám žádnou zprávu. Další výhrady jsem nedostal, takže lze říct, že aplikace je už hotová.

8 Závěr

Hlavním cílem práce bylo navrhnout a realizovat aplikaci pro správu interních školení společnosti. Aplikace má umět školení vytvářet a následně je zobrazit. Na vytvořené školení má zájemce mít možnost se přihlásit. U každého školení je uvedena informace o počtu přihlášených zájemců, počtu volných míst a datu, dokdy je možno se ze školení odhlásit. Aplikace dále nabízí hlášení o počtu realizovaných školení včetně počtu účastníků. K aplikaci je vytvořena uživatelská dokumentace.

Během vývoje aplikace přibyly další požadavky. V původní analýze bylo stanoveno, že školení půjde vytvářet jen podle šablony. To se změnilo tak, že školení lze vytvořit i bez šablony. Dalším novým požadavkem bylo přidání více druhů reportů. Tento požadavek se změnil už při analýze, takže s ním bylo v implementaci počítáno. Všechny výše uvedené cíle a přidané požadavky byly splněny. Během implementace jsem přišel na několik speciálních případů, které bylo nutné ošetřit.

Analýza zadání byla podle mého názoru provedena do dostatečných podrobností. Během implementace jsem se řídil analýzou, kde jsem věděl jak přesně mají funkce fungovat. Design aplikace jsem vytvářel pomocí navržených drátěných modelů. V těch se ale objevila jedna chyba, kde bylo zapomenuto na úpravu školení. Výsledná aplikace byla řádně otestována na půdě zadavatele.

Práci je možné rozšířit o emailového klienta, který má na starosti posílání emailů uživatelům. Dále je možné aplikaci rozšířit o nové reporty. Například důležitým reportem bude sledování docházky uživatelů a tím identifikování uživatelů, kteří se sice zapíší, ale nepřijdou na školení.

Seznam zkratek

API Application programming interface

DAO Data access object

DI Dependency injection

GWT Google web toolkit

HTTP Hypertext transfer protocol

IOC Inversion of control

JAXB Java architecture for XML binding

JDBC Java database connectivity

JPA Java persistence API

LDAP Lightweight directory access protocol

OOP Object-oriented programming

ORM Object-relational mapping

POJO Plain old Java object

POM Project object model

SQL Structured query language

UML Unified modeling language

XML Extensible markup language

Literatura

[1] Querydsl Reference Guide, 2015. Dostupné z: http://www.querydsl.com/static/querydsl/4.0.1/reference/html_single/. [Online; navštíveno 19. 06. 2015].

[2] Spring Framework Reference Documentation, 2014. Dostupné z: http://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf. [Online; navštíveno 17. 06. 2015].

[3] Data access with JDBC, 2015. Dostupné z: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/jdbc.html. [Online; navštíveno 19. 06. 2015].

[4] Spring Vaadin Integration, 2015. Dostupné z: http://vaadin.xpoft.ru/. [Online; navštíveno 19. 06. 2015].

[5] FOUNDATION, A. S. Apache Maven, 2015. Dostupné z: https://maven.apache.org/index.html. [Online; navštíveno 19. 06. 2015].

[6] FOUNDATION, E. Understanding EclipseLink. Dostupné z: http://www.eclipse.org/eclipselink/documentation/2.6/concepts/general001.htm#CHDIJJGA. [Online; navštíveno 20. 06. 2015].

[7] FOWLER, M. Inversion of Control Containers and the Dependency Injection pattern, 2004. Dostupné z: http://www.martinfowler.com/articles/injection.html. [Online; navštíveno 20. 06. 2015].

[8] GRöNROO, M. *Book of Vaadin*. Vaadin Ltd, 2015. Dostupné z: https://vaadin.com/book. [Online; navštíveno 17. 06. 2015].

[9] HAT, R. Hibernate ORM. Dostupné z: http://hibernate.org/orm/. [Online; navštíveno 20. 06. 2015].

[10] JOHNSON, R. – HOELLER, J. et al. Introduction to the Spring Framework. Dostupné z: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html. [Online; navštíveno 20. 06. 2015].

[11] ORACLE. Java™ Logging Overview. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/logging/overview.html. [Online; navštíveno 20. 06. 2015].

[12] QOS.CH. SLF4J user manual. Dostupné z: http://www.slf4j.org/manual.html. [Online; navštíveno 20. 06. 2015].

[13] TUTTLE, S. – EHLENBERGER et al. *Understanding LDAP - Design and Implementation*. IBM Redbooks. IBM Redbooks, 2006. Dostupné z: https://books.google.cz/books?id=aT2rAgAAQBAJ. ISBN 9780738497860.

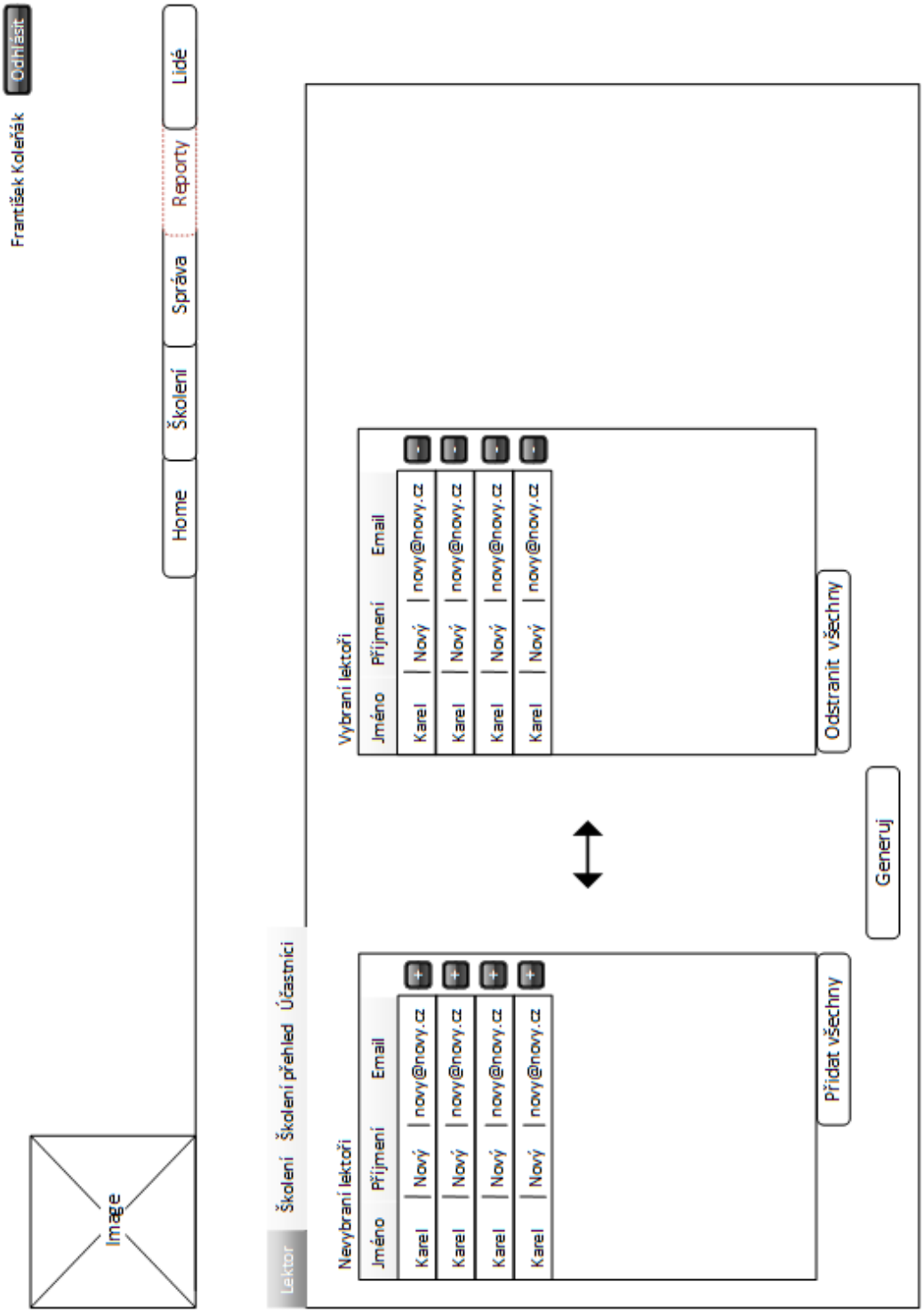
[14] *Vaadin API*. Vaadin Ltd. Dostupné z: https://vaadin.com/api/. [Online; navštíveno 17. 06. 2015].

Přílohy

A Drátěné modely

Drátěný model 1: Model stránky domů

Drátěný model 2: Model stránky správa

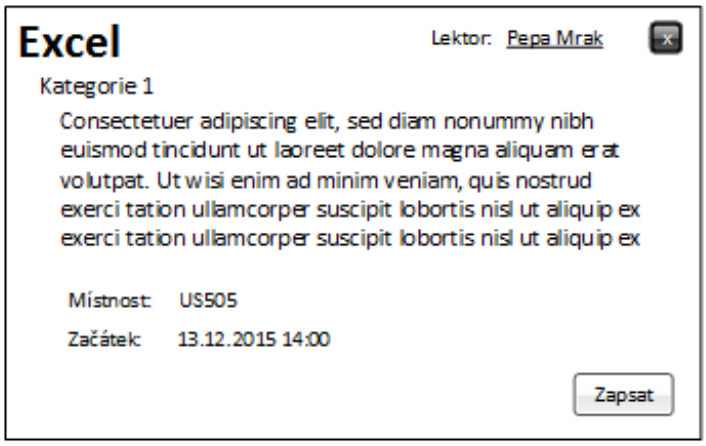


Drátěný model 3: Model stránky reporty

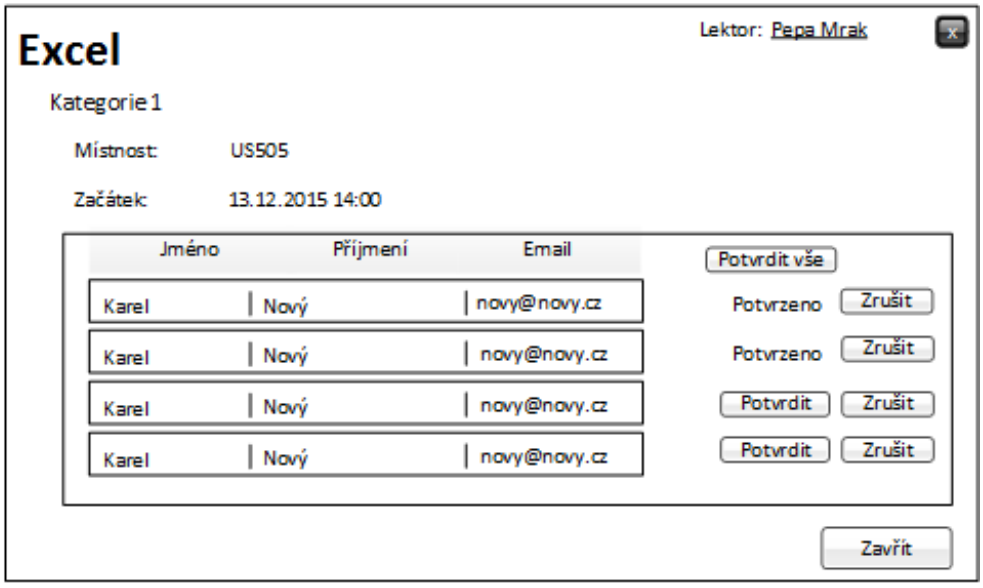
Drátěný model 4: Model stránky školení

Drátěný model 5: Model stránky lidé

Popup školení - zápis



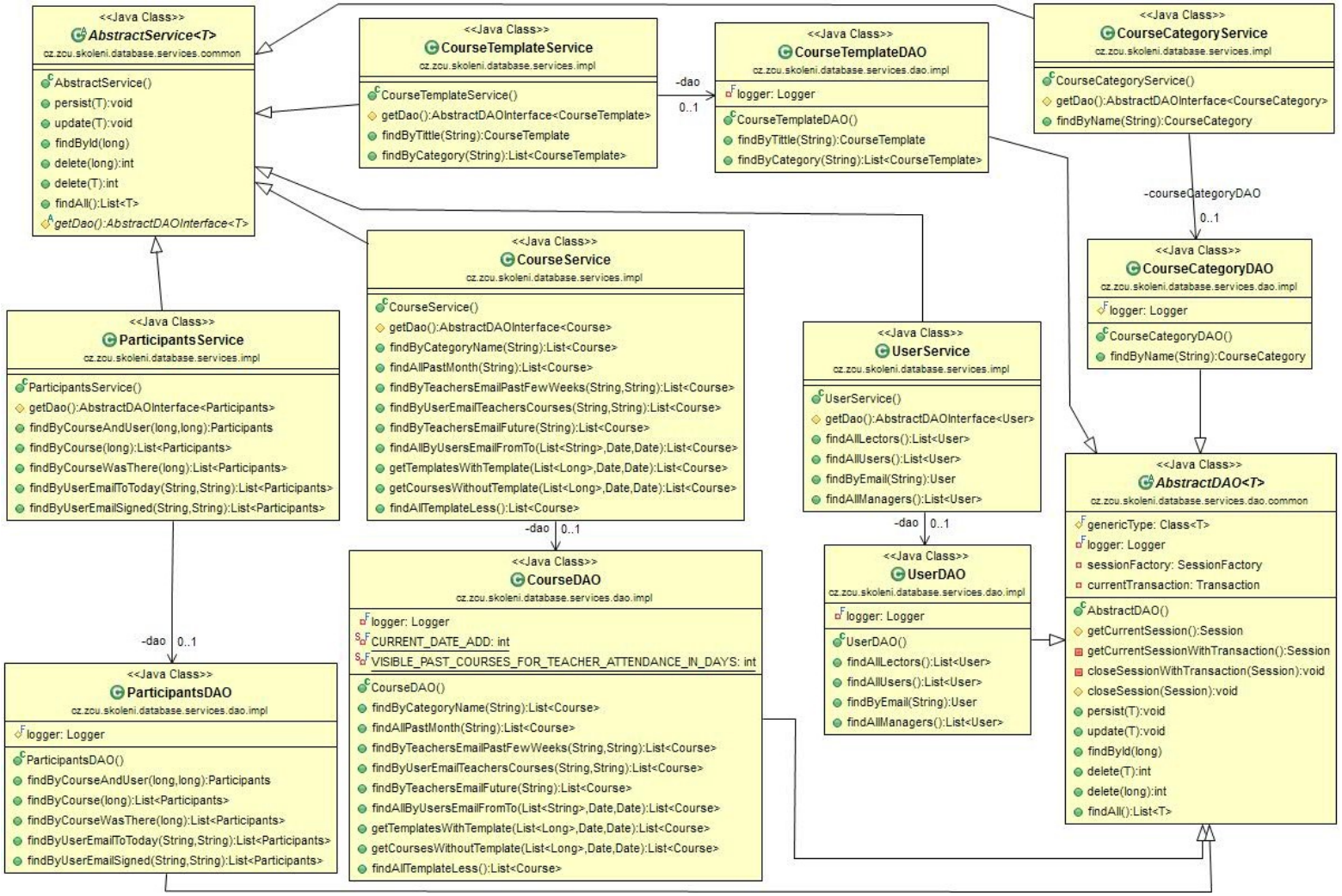
Popup školení - přihlášky



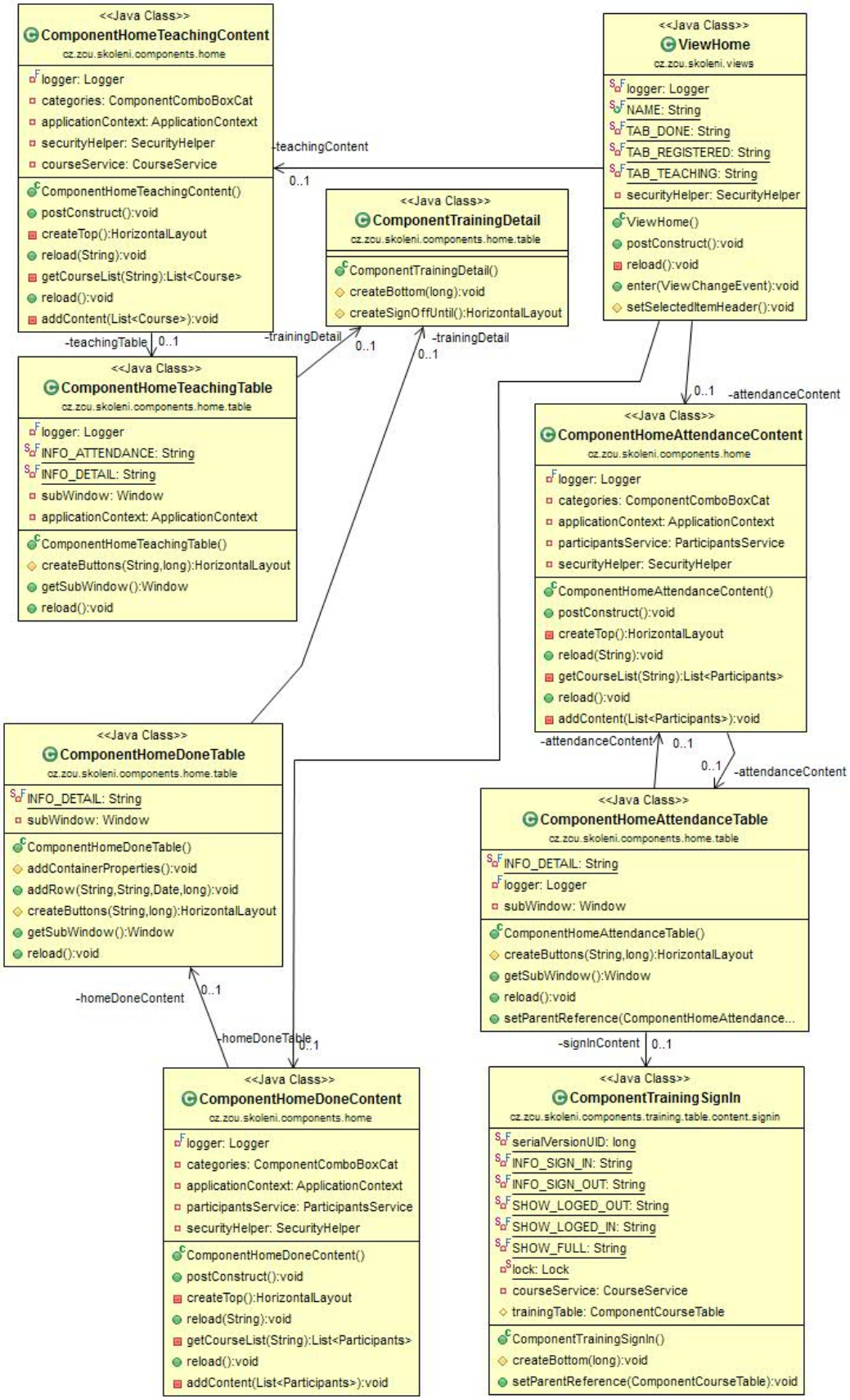
Drátěný model 6: Model oken po kliknutí na tlačítka ve stránce školení

Drátěný model 7: Model oken po kliknutí na tlačítka ve stránce školení

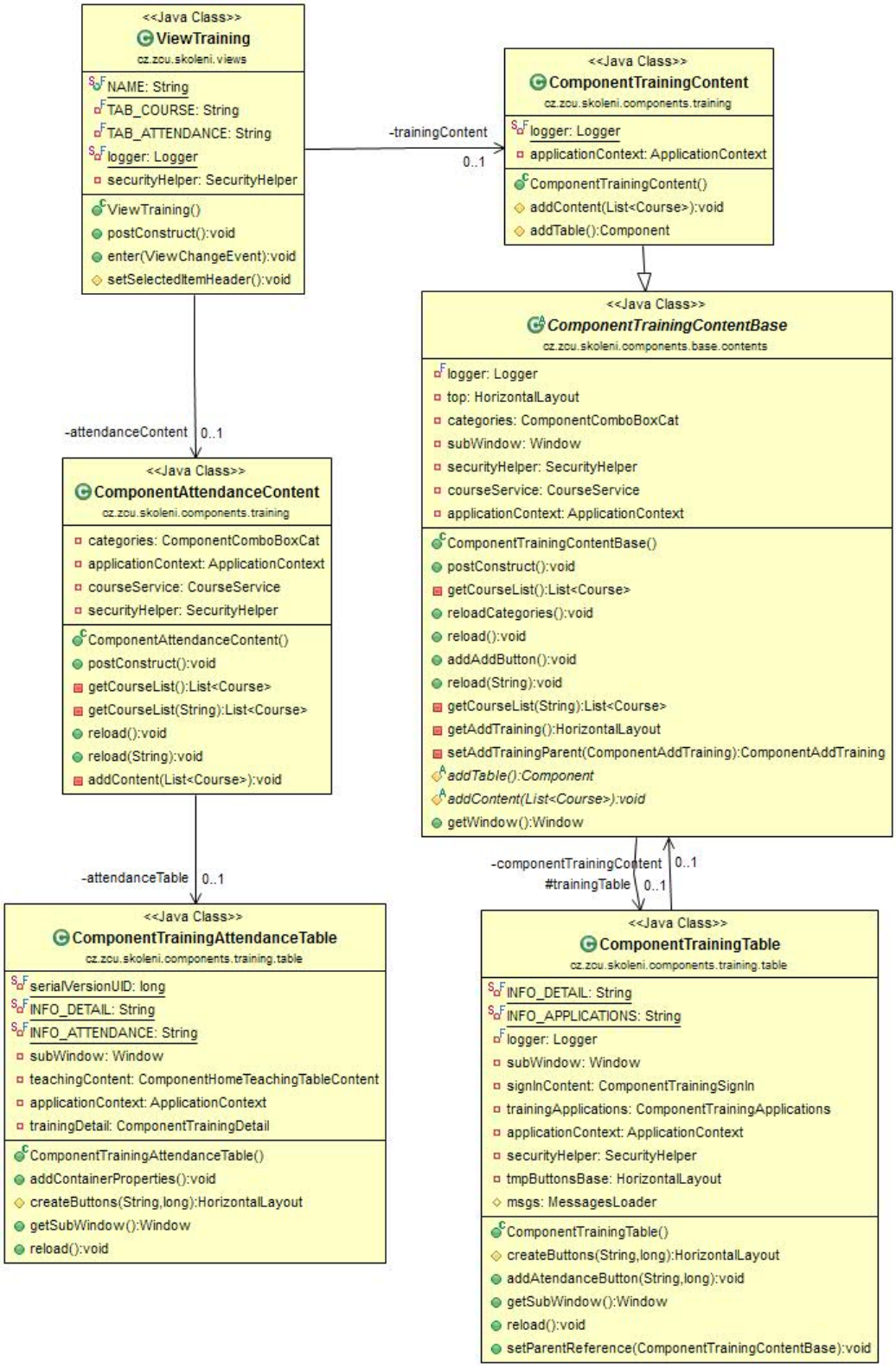
B UML diagramy



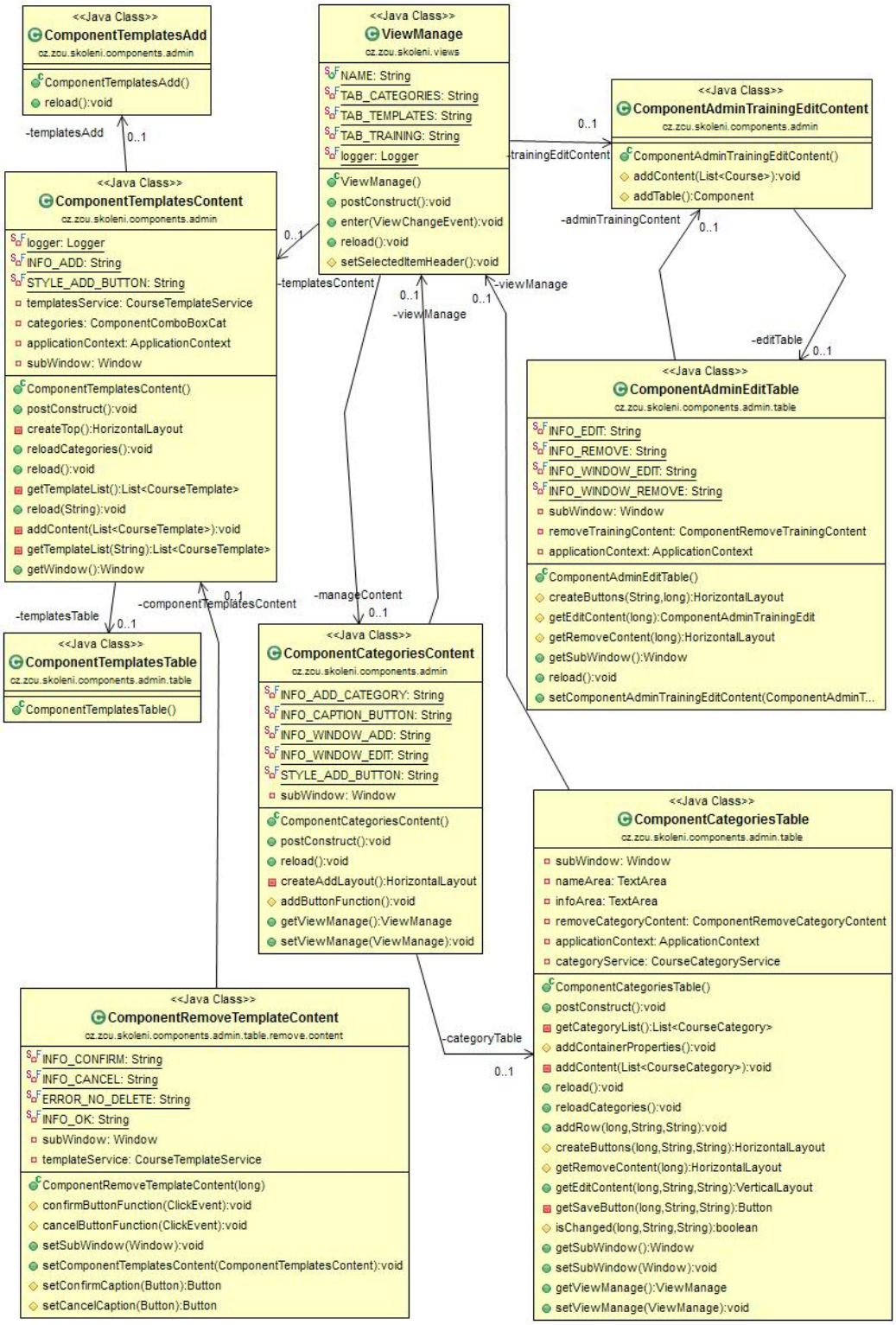
UML diagram 1: UML diagram databázové části aplikace



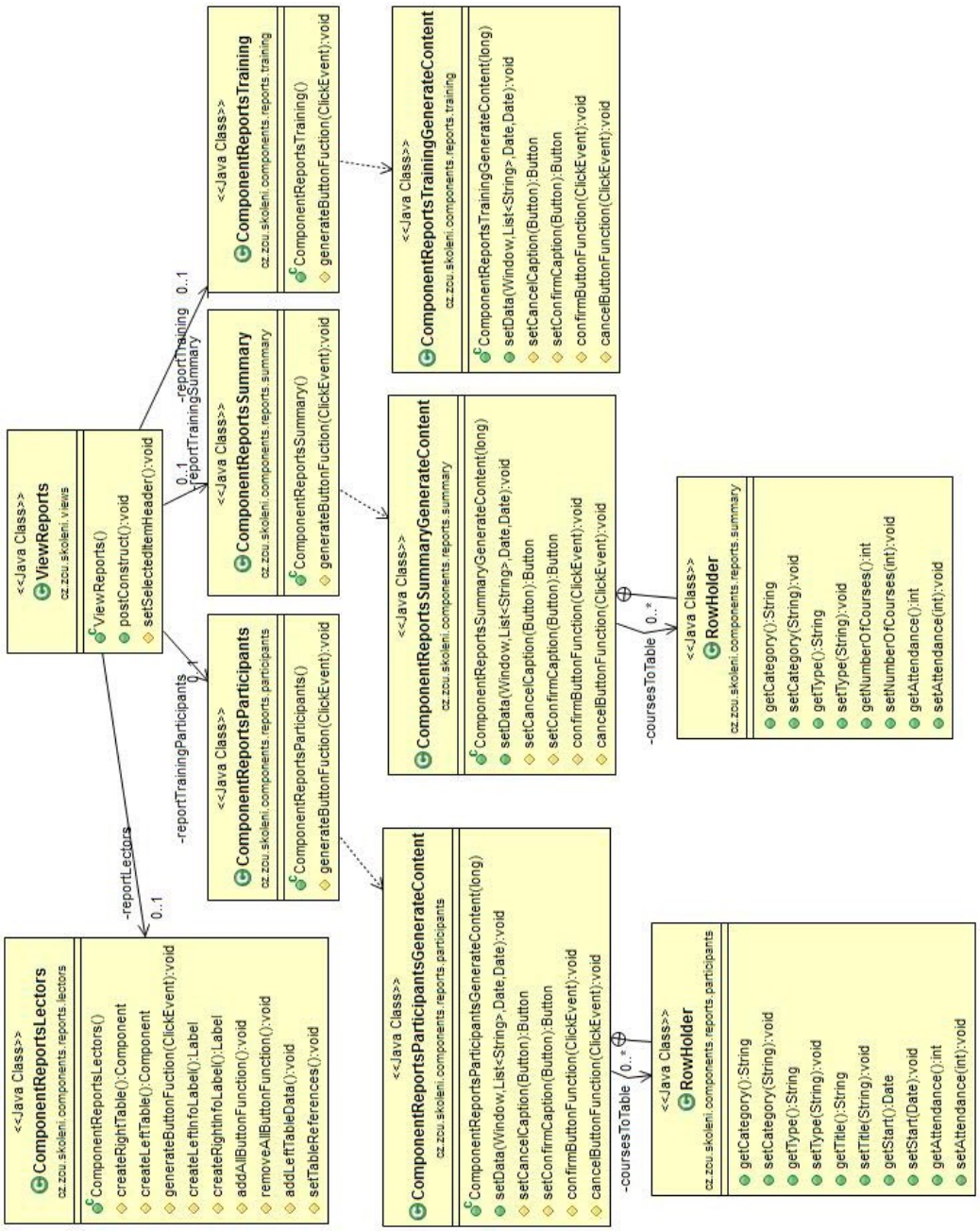
UML diagram 2: Diagram stránky domů



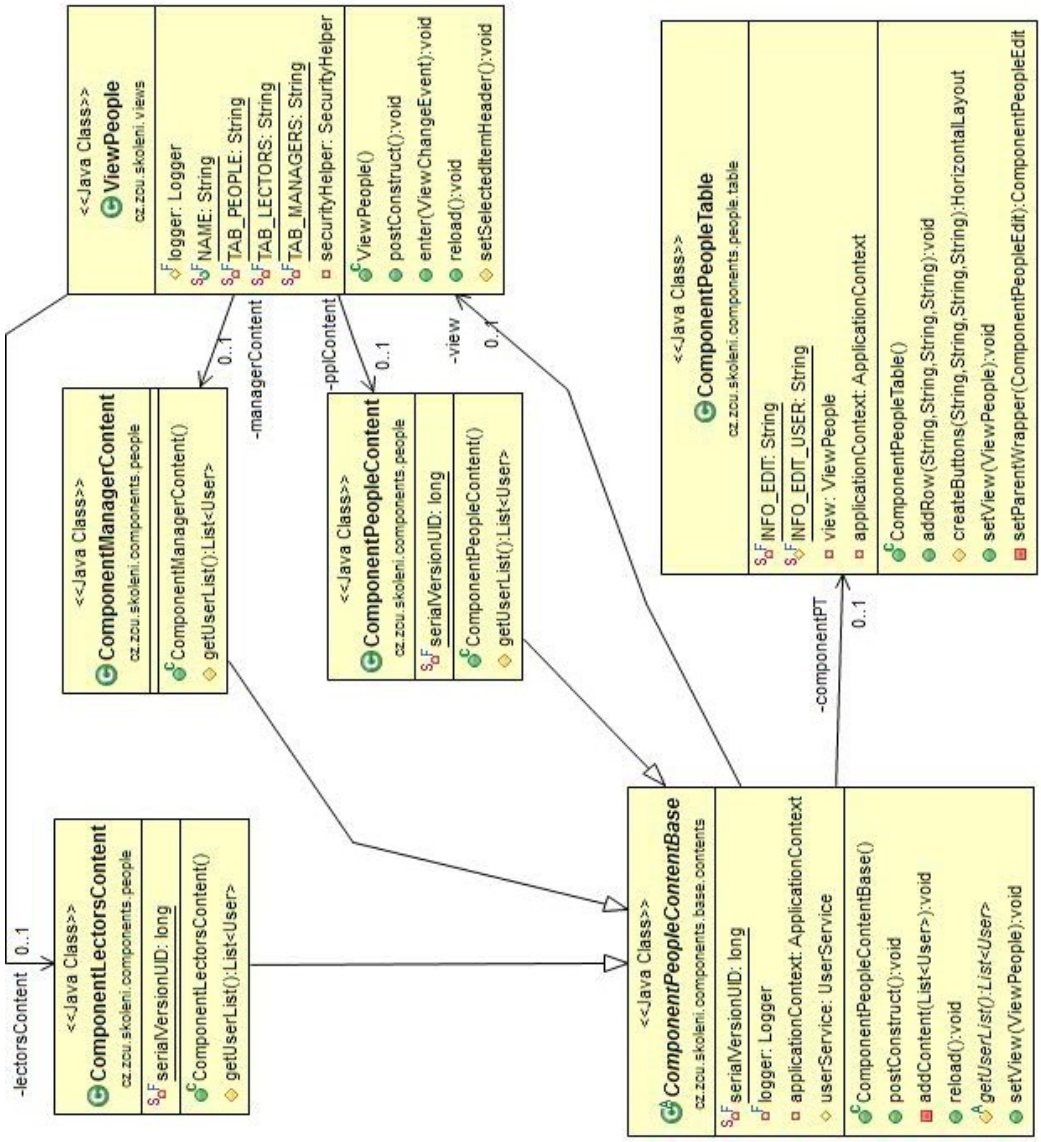
UML diagram 3: Diagram stránky školení



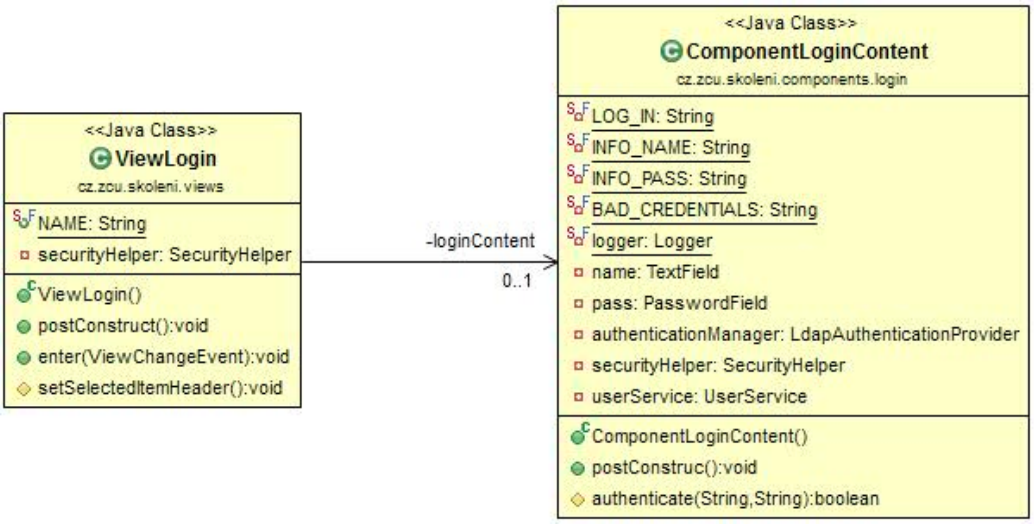
UML diagram 4: Diagram stránky správa



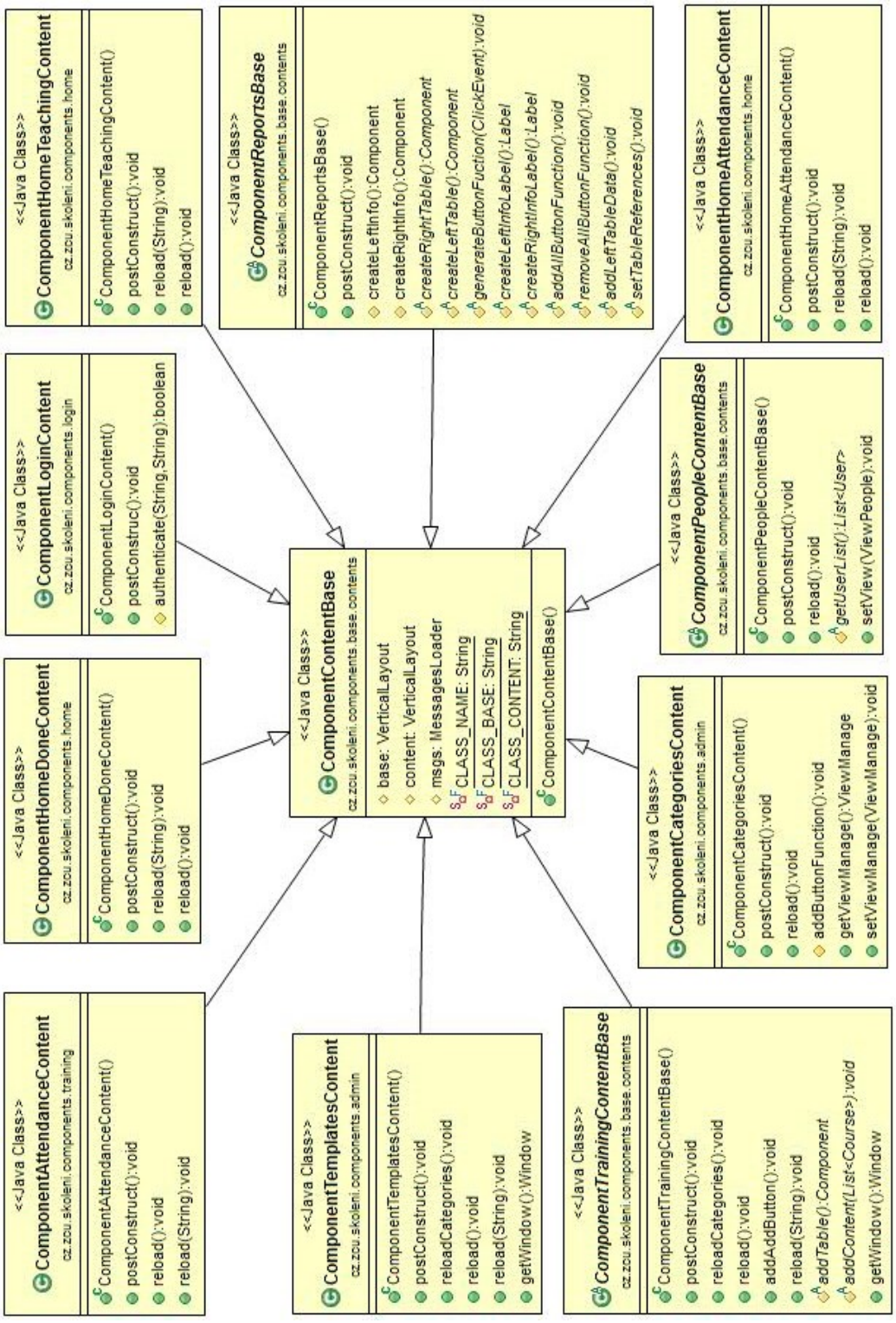
UML diagram 5: Diagram stránky reporty



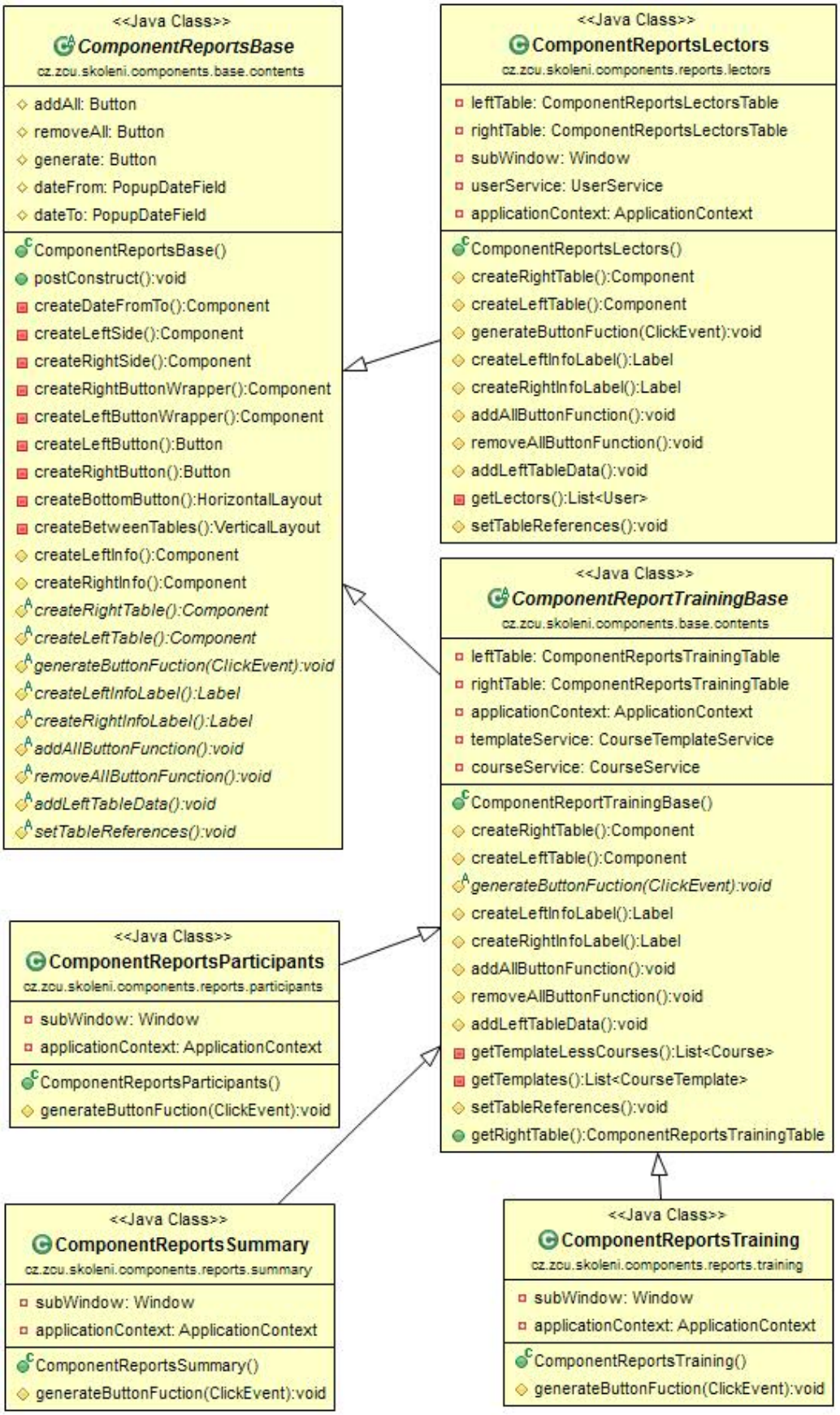
UML diagram 6: Diagram stránky lidé



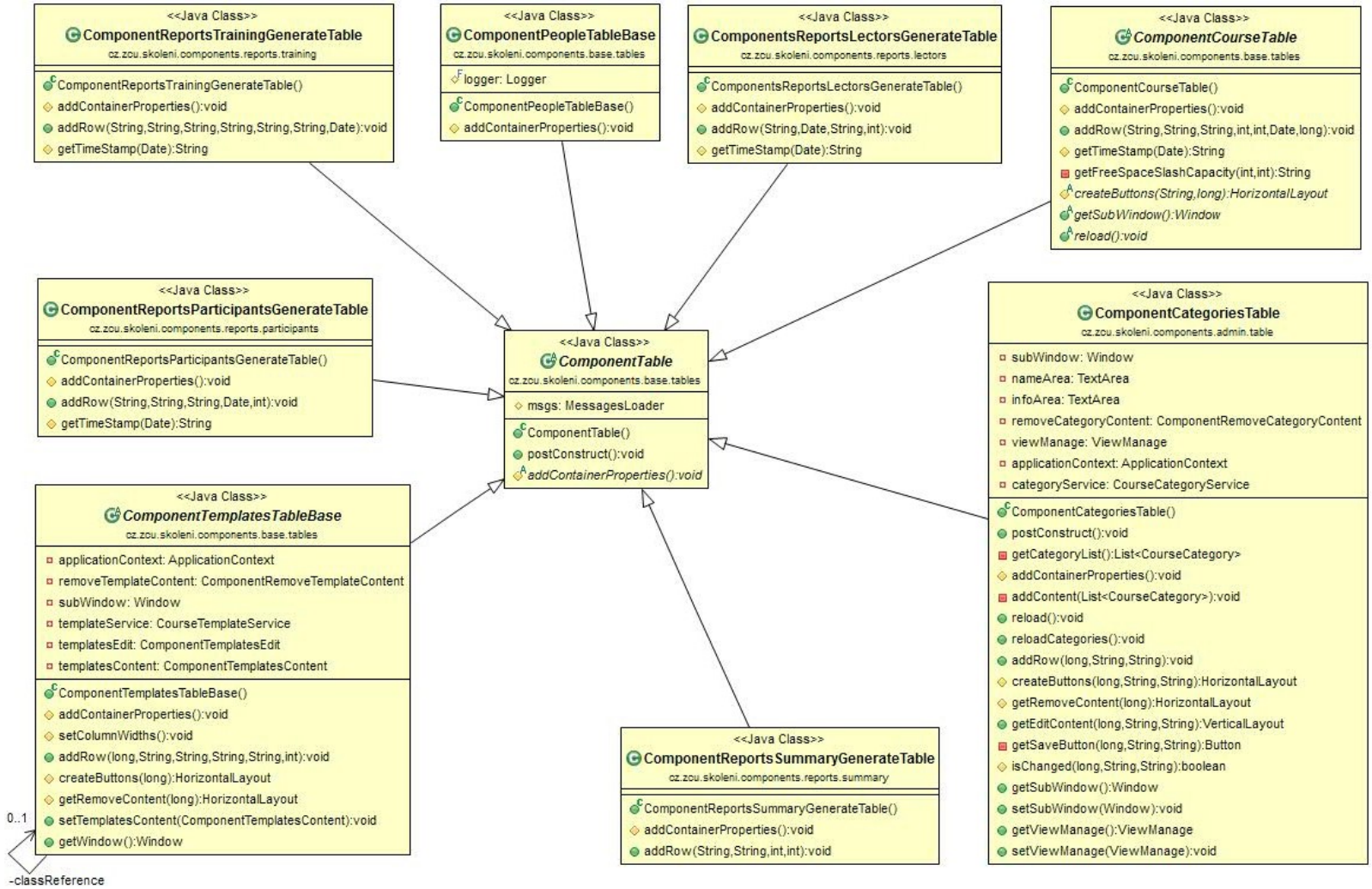
UML diagram 7: Diagram přihlašovací stránky



UML diagram 8: UML diagram potomků třídy *ComponentContentBase*



UML diagram 9: UML diagram - hierarchie reportů



UML diagram 10: UML diagram potomků třídy *ComponentTable*

C Uživatelská dokumentace

Uživatelská dokumentace

Systém pro správu interních školení

Plzeň 2015 František Koleňák

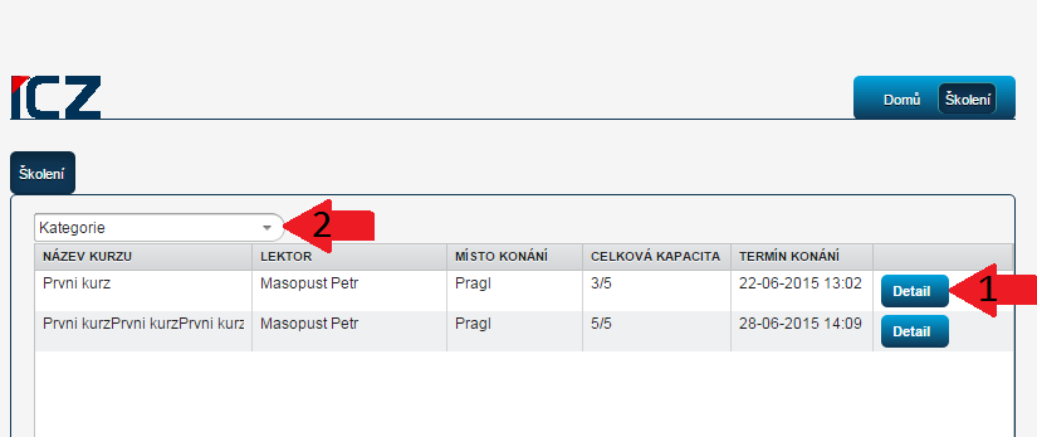
Obsah

1 Role účastník

Každý uživatel, který prvně navštíví aplikaci, dostane roli účastník. Tato role je omezena pouze na základní funkce. Uživatel bude mít přístup pouze na dvě stránky, na stránky **Domů** a **Školení**.

1.1 Přihlášení na školení

Přihlásit se na školení lze na stránce **Školení**. Na této stránce je pak seznam školení, která byla vypsána. Tento seznam lze filtrovat pomocí rozbalovacího seznamu (na Obrázek 1.1 pod číslem 2) podle kategorie. Po vybrání musíme kliknout na tlačítko *Detail* (Obrázek 1.1 pod číslem 1). Po kliknutí se otevře modální okno (příklad okna na obrázku Obrázek 1.2), ve kterém jsou všechny detaily o školení. Pokud se chceme přihlásit na školení, stačí kliknout na tlačítko *Přihlásit*. Jakmile manager potvrdí přihlášku, školení se zobrazí v seznamu zapsaných kurzů na stránce **Domů**.



Obrázek 1.1: Stránka školení - přihlášení na školení

Obrázek 1.2: Detail školení s přihlašovacím tlačítkem

1.2 Odhlášení se ze školení

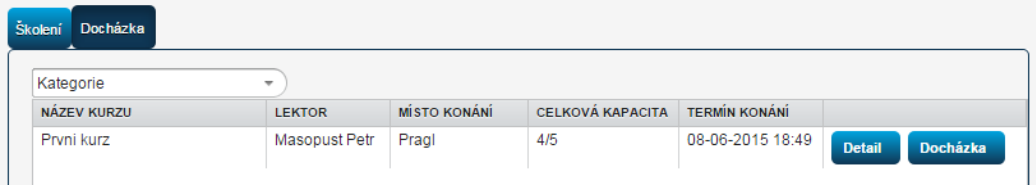
Po zapsání na kurz a potvrzení přihlášky se lze z kurzu do určitého data odhlásit. Datum a čas, do kterého je odhlášení možné, lze nalézt v detailu školení (Obrázek 1.2 řádek *Odhlášení do*). Odhlášení je možné na dvou místech. Na stránce **Domů** na záložce *Zapsané* nebo na stránce **Školení** na záložce *Školení*. Odhlášení je možné v detailu školení.

2 Role lektor

Lektor má všechny funkce stejné jako účastník. Lektor má také přístup pouze na dvě stránky, na stránky **Domů** a **Školení**. Navíc má k dispozici dva seznamy školení, které mu umožňují správu docházky a prohlížení již skončených školení. Seznam s již ukončenými školeními je na stránce **Domů**, na záložce *Vyučované*.

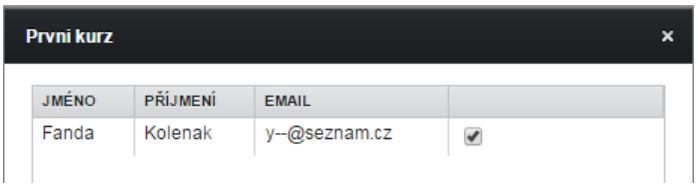
2.1 Správa docházky

Správa docházky je na stránce **Školení**, na záložce *Docházka*. Na stránce (příklad stránky viz. Obrázek 2.1) je seznam všech školení, u kterých je možnost provedení docházky. Detail školení je dostupný po kliknutí na tlačítko *Detail*. Docházka se pak provádí v modálním okně, které je přístupné po kliknutí na tlačítko *Docházka*.



Obrázek 2.1: Stránka se správou docházky

V modálním okně (příklad okna viz Obrázek 2.2) je pak seznam lidí, kteří se měli dostavit na dané školení. Účast se potvrdí zaškrtnutím zaškrtávacího políčka. V případě, že se účastník nedostavil, políčko zůstane volné.



Obrázek 2.2: Okno s potvrzením docházky

3 Role manager

Manager má všechny funkce jako účastník a lektor. Manager může provádět docházku u všech proběhlých školení. Dále má k dispozici stránky **Správa**, **Reporty** a **Lidé**.

3.1 Správa

Stránka **Správa** (viz Obrázek 3.1) slouží pro správu školení, šablon a kategorií. Každá existující šablona a školení se dá zároveň upravit nebo vymazat. Školení, která ještě neproběhla, lze také upravit nebo smazat. Na místě, kam ukazuje šipka 1 (Obrázek 3.1), bude vždy umístěno přidání školení/šablony/kategorie.

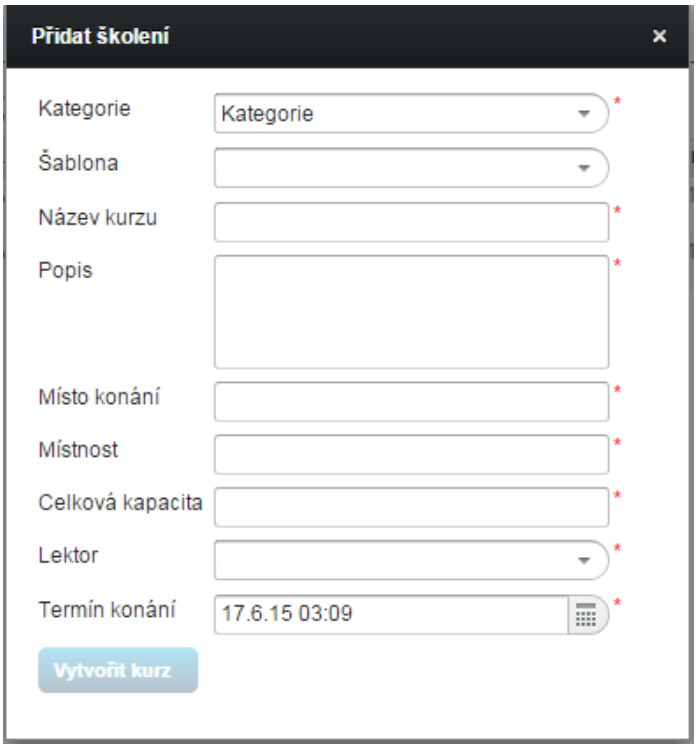
Obrázek 3.1: Stránka správa

3.1.1 Školení

Pro správu školení je nutné přepnout se do patřičné záložky *Školení*.

Vytváření, úprava

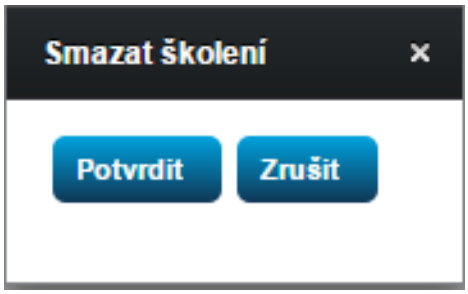
Vytvořit školení půjde tlačítkem *+*. Toto tlačítko otevře modální okno (viz. Obrázek 3.2), ve kterém půjde vytvořit nové školení. Po zadání všech údajů lze kliknout na tlačítko *Vytvořit kurz*. Pro úpravu existujícího školení slouží tlačítko *Úprava*. Po kliknutí na tlačítko se objeví modální okno s předvyplněnými údaji, které lze změnit. Okno je vzhledově stejné jako Obrázek 3.2.



Obrázek 3.2: Okno vytvořit školení

Mazání

Smazat školení půjde tlačítkem *Vymazat*. Po kliknutí na tlačítko se objeví potvrzovací modální okno (viz. Obrázek 3.3). Okno nabízí pouze dvě možnosti: Potvrdit nebo Zrušit. Smazáním školení se smažou všechny podané přihlášky.



Obrázek 3.3: Potvrzovací okno

3.1.2 Šablona

Pro správu šablon je nutné přepnout se do patřičné záložky *Šablona*.

Vytváření, úprava

Vytváření a úprava šablon funguje na stejném principu jako u vytváření nebo úpravy školení.

Mazání

Mazání šablon funguje na stejném principu jako mazání školení s tím rozdílem, že šablona, která již byla použita, nejde vymazat.

3.1.3 Kategorie

Pro správu kategorií je nutné přepnout se do patřičné záložky *Kategorie*.

Vytváření, úprava

Vytváření a úprava kategorií funguje na stejném principu jako u vytváření nebo úpravy školení.

Mazání

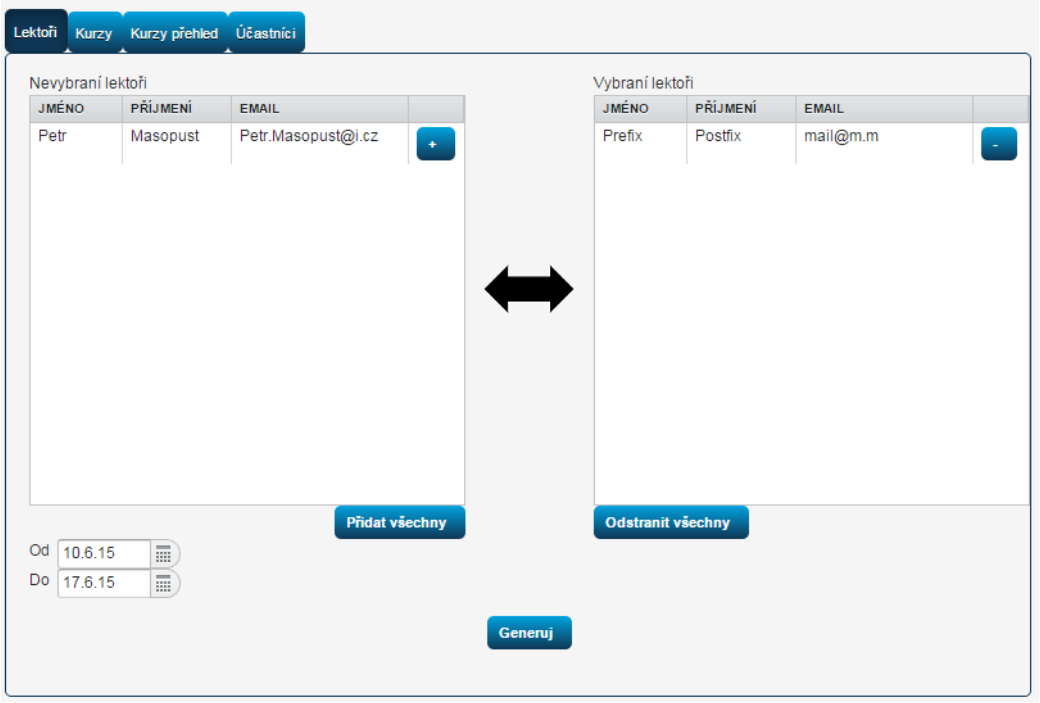
Mazání kategorií funguje na stejném principu jako mazání školení s tím rozdílem, že kategorie, která již byla použita, nejde vymazat.

3.2 Reporty

Generování repotů lze nalézt na stránce **Reporty** (viz. Obrázek 3.4). K dispozici jsou celkem čtyři druhy reportů:

- Lektor report
- Školení report
- Přehled školení report
- Statistiky školení report

Těmto druhům reportů pak odpovídají jednotlivé záložky na stránce (záložka nazvaná *Lektoři* odpovídá reportu Lektor report, záložka *Kurzy* odpovídá reportu Školení report, záložka *Kurzy přehled* odpovídá reportu Přehled školení report, *Účastníci* odpovídá reportu Statistiky školení report).

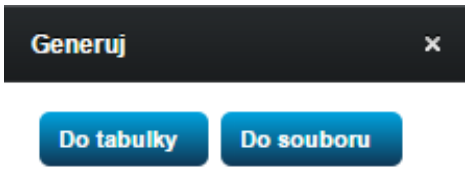


Obrázek 3.4: Stránka reporty

3.2.1 Generování reportu

Každý report se generuje stejným způsobem. Nejdříve se z levé tabulky vyberou všechny záznamy, u kterých chceme report vytvořit.

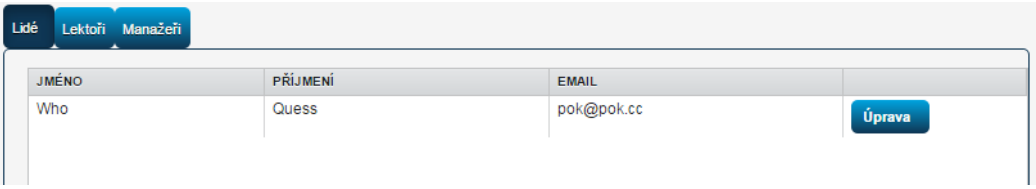
Vybrání záznamu se provede tlačítkem +, odebrání záznamu z vybraných se provede tlačítkem -. Pokud chceme naráz přidat všechny záznamy v tabulce, stačí kliknout na tlačítko *Přidat všechny*. K odebrání všech vybraných záznamů slouží tlačítko *Odstranit všechny*. Po vybrání záznamů můžeme přenastavit datumy od kdy, do kdy nás výsledky zajímají. Po nastavení všech nutných parametrů lze kliknout na tlačítko *Generuj*. Po kliknutí se objeví modální okno (viz. Obrázek 3.5) se dvěma možnostmi. Zobrazení reportu do tabulky, která se zobrazí v okně, nebo stažení souboru, ve kterém budou všechna data.



Obrázek 3.5: Reporty - okno s možnostmi

3.3 Lidé

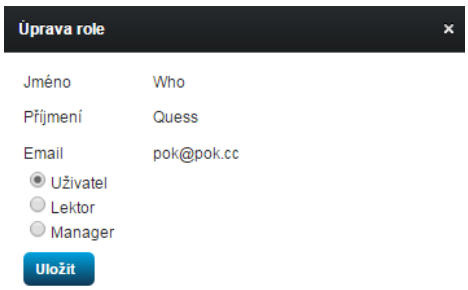
Změna práv uživatelů je možná na stránce **Lidé** (viz. Obrázek 3.6). Na této stránce jsou tři záložky. Každá záložka představuje uživatele aplikace rozdělených podle jejich rolí.



Obrázek 3.6: Stránka lidé

3.3.1 Změna rolí

Změna role je dostupná po kliknutí na tlačítko *Úprava*. Po kliknutí se otevře modální okno (viz. Obrázek 3.7). Změna je možná výběrem jiné role a kliknutím na tlačítko *Uložit*.



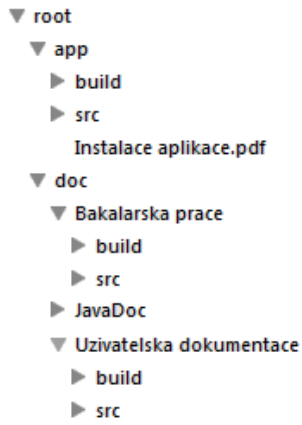
Obrázek 3.7: Lidé - úprava role

4 Instalace aplikace

Pro správný chod aplikace bude nutné mít server s databází, LDAPem a se samotnou aplikací. Ukázkový soubor s nastavením připojení do relační databáze je v *resources/example-database.properties*. V tomto souboru jsou ukázkové parametry pro databázi PostgreSQL s databází s názvem skoleni. Tento soubor se musí upravit podle vlastních nastavení a přejmenovat na *database.properties*. Po nastavení databáze je nutné upravit soubor *resources/ldap.properties* podle vlastního nastavení LDAPu. V první části souboru je struktura LDAPu s nastavením připojovacích údajů. Dalšími údaji v souboru jsou atributy popisující uživatele. Pro správný chod aplikace musí být v LDAPu vyplněno celkem pět atributů: user.id, user.firstName, user.lastName, user.password a user.email. Ostatní parametry v LDAPu se nikde v aplikaci nezobrazí. Poslední částí jsou údaje o adminu LDAPu, přes které se bude aplikace přihlašovat.

Poslední částí zprovoznění aplikace je sestavení aplikace pomocí Mavenu. Spustíme konzoli v kořenovém adresáři a pomocí příkazu *mvn package* sestavíme aplikaci do *.war* souboru. Tento příkaz má základní výstup nastaven na složku *target*. Ve složce *target* pak nalezneme sestavenou aplikaci s názvem *skoleni-1.0-SNAPSHOT.war*. Tento soubor nahrajeme na server (například na TomCat). Další náhradní možností spuštění aplikace (vhodné v případě testování) je spuštění přes konzoli pomocí příkazu *mvn jetty:run*. Příkaz spustí aplikaci na servletu Jetty.

D Struktura DVD



Obrázek 11: Struktura přiloženého DVD