

Knihovna pro implementaci vyhledávacích filtrů v Java SE

Dobroslav Pelc

Bakalářská práce
2015



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2014/2015

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Dobroslav Pelc**
Osobní číslo: **A12189**
Studijní program: **B3902 Inženýrská informatika**
Studijní obor: **Informační a řídicí technologie**
Forma studia: **kombinovaná**

Téma práce: **Knihovna pro implementaci vyhledávacích filtrů v Java SE**
Téma anglicky: **A Library for the Implementation of the Search Filter in Java SE**

Zásady pro vypracování:

- Analyzujte možnosti implementace knihovny pro uživatelsky přívětivou definici filtrů pro vyhledávání v relační databázi. Zohledněte napojení výsledné podmnožiny relační databáze na konkrétní akci.
- Prostudujte dostupné platformy a navrhněte nejvhodnější variantu pro implementaci knihovny.
- Navrhněte specifikaci API knihovny s ohledem na snadnou integraci do budoucích web aplikací.
- Dle výše uvedené specifikace knihovnu implementujte.
- Proveďte ukázkovou integraci nové knihovny do vybrané web aplikace. Zhodnoťte přínos nové knihovny z hlediska programátora a uživatele.

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: tištěná/elektronická

Seznam odborné literatury:

- ECKEL, Bruce. Thinking in JAVA. Prentice Hall Professional, 2003.
- WALLS, Craig. Spring in action. Fourth edition. Texas: Manning, 2013. ISBN 978-161-7291-203.
- HARROP, Rob; MACHACEK, Jan. Pro Spring. John Wiley & Sons, 2007.
- HUNT, Charlie; JOHN, Binu. Java performance. Prentice Hall Press, 2011.
- PONEC, Pavel. Ujorm [online]. 2011 [cit. 2015-02-04]. Dostupné z: <http://ujorm.org/>

Vedoucí bakalářské práce:

Ing. Radomír Sohlich

Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce:

6. března 2015

Termín odevzdání bakalářské práce:

22. května 2015

Ve Zlíně dne 6. března 2015



L.S.


doc. Mgr. Milan Adámek, Ph.D.
děkan


prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu


Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl jsem seznámen s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 26. 05. 2016


.....
podpis diplomanta

ABSTRAKT

Tato práce se zabývá analýzou a implementací knihovny pro uživatelsky přívětivou definici filtrů vyhledávání v relační databázi s možností napojení výsledné podmnožiny na konkrétní akci. Výsledné řešení je optimalizované pro webové aplikace na bázi springu a použití AOP.

Klíčová slova: Java SE, Spring, Ujorm, AOP, ORM

Abstrakt ve světovém jazyce

This work deals with analysis and implementation of libraries for user-friendly definition of filters. Applications user filters on a relational database. The ability to link the resulting subset of a particular action or process. Final solution is optimized for web applications based on Spring using AOP.

Keywords: Java SE, Spring, Ujorm, AOP, ORM

PODĚKOVÁNÍ

Rád bych poděkoval především vedoucímu mé bakalářské práce Ing. Tomáši Dulíkovi, Ph.D. za poskytnutí cenných rad, za jeho trpělivost a čas, za odborné vedení a poskytnutí osobních konzultací.

Všem, kteří mě při studiu podporovali, upřímně děkuji.

OBSAH

ÚVOD	8
I. TEORETICKÁ ČÁST	9
1 Požadované vlastnosti nové knihovny	10
2 Použité technologie	11
2.1 Ujorm.....	11
2.1.1 Základní vlastnosti	11
2.1.2 Srovnání s Hibernate	12
Komfort vývoje	12
Rychlost a propustnost ORM.....	12
2.1.3 Kde použít Ujorm.....	12
2.1.4 Shrnutí.....	13
2.2 AOP.....	13
2.2.1 Práce s aspekty	13
2.3 Spring Framework.....	14
2.3.1 Vlastnosti	14
2.3.2 Výhody.....	14
2.4 GWT (Google Web Toolkit)	14
2.4.1 Výhody.....	15
2.4.2 Nevýhody.....	15
2.5 Sencha GXT	16
2.5.1 Inicializace	17
2.5.2 Vykreslování	17
2.5.3 Zobrazení	17
II. Praktická část.....	18
3 Analýza	19
4 Návrh implementace	20

4.1	Databáze	20
4.1.1	Struktura.....	20
4.1.2	Relační vazby.....	20
4.1.3	Datové typy sloupců.....	21
4.1.4	Indexy.....	21
4.1.5	Unikátní indexy.....	21
4.2	DAO	21
4.2.1	Mapování doménových objektů.....	21
4.2.2	UJO	22
4.2.3	Database.java	24
4.2.4	Zavedení Ujormu do aplikace	25
4.3	Aplikační logika	25
4.3.1	Servisní služba	26
4.3.2	Transakce	26
4.3.3	Session	27
4.4	CUJO.....	28
4.4.1	UjoTranslator	29
4.5	Controller.....	29
5	Nasazení do produkce	31
5.1	Databázová struktura.....	31
5.2	GUI.....	32
ZÁVĚR		36
SEZNAM POUŽITÉ LITERATURY		37
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK		39
SEZNAM OBRÁZKŮ.....		41
SEZNAM ZDROJOVÉHO KÓDU		42

ÚVOD

Téměř každý produkční web prezentuje uživateli nějaká data, se kterými ho nechá různým způsobem pracovat. Často však tyto weby postrádají možnost data personalizovat. Tato práce se zaměřuje právě na možnosti personalizace dat

Cílem práce je analyzovat a následně implementovat knihovnu, která bude v maximální možné míře umožňovat různé typově kontrolované průchody databázovou strukturou na úrovni klientské části aplikace (dále jen frontend (1)) bez nutnosti strukturálního zásahu do současného systému.

Práce je omezená a optimalizovaná na relační databáze. Hlavní myšlenkou této práce je uzavřít odpovědnost za validní prezentování dat a typové kontroly dat přímo do relační databáze. V závislosti na ORM mapování a následné přísné modulární architektuře je výsledná aplikace validovaná nejen pro uživatele, který s ní reálně pracuje, ale i pro programátora, který ji tvoří, rozvíjí a udržuje.

Textová část práce má následující strukturu: v první kapitole si představíme vybrané technologie, jako je např. Ujorm, Spring, GWT nebo GXT. Další kapitola naváže analýzou potřeb všech zainteresovaných stran. Po analýze bude následovat návrh modulární architektury aplikace. Zde se zaměříme především na DAO v podání knihovny Ujorm. V poslední kapitole zhodnotíme výsledek práce spolu s jeho nasazením do produkčního prostředí. Podíváme se také na náhled grafického uživatelského rozhraní (dále jen GUI) při tvorbě vlastních filtrů.

I. TEORETICKÁ ČÁST

1 POŽADOVANÉ VLASTNOSTI NOVÉ KNIHOVNY

Navrhovaná knihovna by měla mít dva režimy. První je nutnou podmínkou. Uživatel vyplní filtr, odešle požadavek na server. Server zpracuje požadavek a vrátí výsledek. Aplikace nějakou vhodnou formou prezentuje výsledek uživateli (dále jen vyhledávací režim).

Druhý režim je nadstavbou vyhledávacího režimu. Uživatel stejně jako v prvním případě vidí živá data. Přidanou hodnotou je uložení uživatelského požadavku do databáze. To nám umožní např. reagovat na požadavek se zpožděním, opakovaně, případně v nějakém plánovači (dále jen obslužný režim).

V okamžiku, kdy uživatel nečeká na data, jen si nastaví filtr, je mnohem snazší napojovat jednotlivé akce na daný filtr. Kdybychom chtěli mít pouze vyhledávací režim, který bude podporovat více typů akcí nad výsledkem vyhledaných dat, museli bychom udržovat pro každou akci vlastní šablonu včetně aplikační obsluhy.

Možnosti reálné aplikace jsou značné. Jako základní aplikace může být jednoduché vyhledávání na každém webu. Po drobných úpravách i fulltextové. Když půjdeme dále, můžeme napojit vyhledávání na uživatelskou identitu, např. „naposledy jste hledal“, nebo „pokračovat kde jsem skončil“.

Další možností je napojit na uživatelskou identitu a výsledek vyhledávání nějaký informační kanál (e-mail, rss) a odeslat uživateli výsledek na zvolený kanál. Není žádný problém tyto akce opakovat či plánovat.

Velmi zajímavou aplikací je sdílení jednotlivých filtrů mezi uživatelskými identitami. Nechme uživatele pojmenovat si svoje filtry. Databázový návrh to umožňuje. Pomocí jednoduché relační tabulky mezi filtry a uživatelskými identitami zajistíme sdílení. Např. „odeslat příteli“.

V neposlední řadě je můžeme tyto filtry využívat na servisní úkony. Představte si, že máte v nějaké databázové tabulce data s určitou platností a po expiraci je nutné data stáhnout z webu. Pomocí naplánované úlohy s využitím filtru je to velmi snadné.

2 POUŽITÉ TECHNOLOGIE

Základem této práce je technická adaptace relačních vazeb (2) z databáze do aplikační logiky a jejich následná prezentace uživateli nenásilnou formou. Důraz bude kladen na zachování integrity i typové kontroly při práci s relačními vazbami na všech úrovních aplikace. Tyto základní rysy do značné míry ovlivnily výběr všech klíčových frameworků, které jsou použity pro výslednou realizaci. Postupně projdeme jednotlivé části od databáze až po GUI.

2.1 Ujorm

Nejnižší vrstvou aplikace, kterou využijí ze zdrojů třetích stran, je Ujo framework (3) nebo zkráceně Ujorm. Jak napovídá název, jedná se o ORM (4) framework. Poslouží nám jako plnohodnotná DAO (5) vrstva. Je lehce odlišný od tradičních kandidátů jako například Hibernate (6) apod.

Vlastní princip mapování databáze (schématu, tabulek a sloupců) na doménové objekty je velmi rychlý, intuitivní a jednoduchý. Základní myšlenkou je generické mapování klíč-hodnota. Nepracujeme s klasickými POJO (7) objekty, jak jsme zvyklí, ale se speciální mapou označovanou jako UJO.

2.1.1 Základní vlastnosti

Ujorm je postavený na objektech typu OrmTable (dále jen UJO), kde je každá vlastnost popsána statickou konstantou typu Key (dále jen Key). Key se v rámci UJO chová jako klíč k hodnotě proměnné. Přesto, že UJO na první pohled připomíná spíše klasickou mapu (klíč-hodnota), je možné s ním pracovat jako s plnohodnotnou JavaBean (8). Úpravy v UJO s tím spojené budou lépe představeny v praktické části. Mezi základní myšlenky Ujormu patří nahrazení častých a pomalých operací, na které se běžně užívá reflexe (9). UJO nabízí rozhraní (dále jen API) pro získání výčtu všech Key.

Key má široké využití. Lze ho použít pro získávání datového typu proměnné, pro čtení a zápis hodnot, pro validování hodnot, pro určení defaultní hodnoty. Lze je také řetězit do relačních vazeb, které v ORM metajazyku fungují jako JOIN na úrovni SQL (10).

Při reálné obsluze nad ORM vrstvou je situace dost podobná Hibernate. Existuje metajazyk (dále jen Query), který je zastupuje roli SQL. Jeho klíčovou částí je ekvivalent pro celou sekci WHERE podmínky z SQL jazyka (dále jen Criterion).

2.1.2 Srovnání s Hibernate

Srovnání dlouhodobě prověřené praxí jednoznačně mluví ve prospěch Ujormu. Rozhodně se nejedná o druhořadý ORM framework pro malé aplikace, jak může jeho skromný vzhled zprvu napovídat. Během srovnání byly sledovány následující body.

Komfort vývoje

Vývoj aplikace je velmi podobný. S oběma frameworky se pracuje dobře. Hibernate má díky velké robustnosti lehce vyšší vstupní odpor (na naučení se). Je tu však jedno velké ale. Oba frameworky nabízí typovou kontrolu kontextových funkcí po úvodním namapování. Jenže zatím co Hibernate umí validovat formou syntaktické chyby, kterou vypíše kompilátor až při překladu na byte code (dále jen kompilace), Ujorm díky generikám v UJO a Key dokáže validovat nativně psaný kód bez nutnosti kompilace.

Př.: Právě tvoříme ekvivalent k SQL pomocí metajazyka daného zvoleným ORM. Typicky ve WHERE podmínce SQL dotazu porovnááme sloupec a hodnotu. Zatímco Hibernate porovná klidně číslo s textem, Ujorm se v IDE (11) okamžitě rozsvítí červeně bez nutnosti cokoli buildovat.

Rychlost a propustnost ORM

Zde již výsledky nejsou tak podobné. Za předpokladu, že jsou oba ORM správně a vhodně nastavené, jsou časy srovnatelné (lehce ve prospěch Ujormu). Zásadní odlišnost je v přístupu k programátorovi.

Pro dosažení stejných výsledků (rychlosti a propustnosti) Hibernate předpokládá, že programátor je naprosto geniální tvor a dělá vše a vždy nejlepším možným způsobem. Ujorm předpokládá přesný opak, především že je programátor líný, dělá chyby a nečte dokumentaci. Proto je na Hibernate potřeba odborník s mnohaletou praxí, zatím co na Ujorm postačí první programátor po ruce.

Na číselné srovnání vychází přibližně o 5% rychlejší Ujorm. Nicméně stačí např. v Hibernate pracovat s dědičností, a v ten okamžik je zbytečná jakákoliv další snaha optimalizovat. Hibernate začne zaostávat za Ujormem o desítky procent.

Hibernate je dobrý sluha, ale špatný pán.

2.1.3 Kde použít Ujorm

Obecně ORM je vhodný nástroj na identifikaci a mapování objektů. Nad každým objektem (tabulkou) zprostředkuje základní operace – vytvořit, načíst, upravit, odstranit (v originále create, read, update, delete, dále jen CRUD) a to včetně relačních vazeb. Pokud situace vyžaduje něco složitějšího, je vhodné Ujorm kombinovat s jinými nástroji.

Za pomoci dalších modulů, které má Ujorm k dispozici, je po namapování doménových objektů a implementaci servisních služeb (aplikační logiky) téměř kompletní administrace včetně GUI (12). Dopisuje se pouze GUI pro speciální případy.

2.1.4 Shrnutí

- Ujorm je jednoduchý a snadno modifikovatelný ORM s typovou kontrolou v reálném čase.
- Ujorm je jeden z nejrychlejších dostupných ORM.
- Ujorm se nesnaží obsáhnout 100% potřeb DAO, řeší jen tu část, na kterou jsou principy ORM vhodné.
- Ujorm se snadno kombinuje s dalšími frameworky (např. MyBatis (13)).
- Ujorm lze s výhodou využít především v začátcích projektu, stejně jako při technologických přestavbách.

2.2 AOP

Většinu serverové části aplikace (dále jen backend (1)) tvoří servisní služby s implementací aplikační logiky. Nad každým doménovým objektem je definována právě jedna servisní služba CRUD API. Proto je nezbytné každou servisní službu chápat jako samostatnou jednotku, které je třeba definovat transakci a také spojení s databází (dále jen session). Využijeme modulárního návrhu ORM a s výhodou navážeme tuto vrstvu aplikace pomocí AOP (14).

Na výběr máme ze dvou typů:

- statického AOP
- dynamického AOP

2.2.1 Práce s aspekty

Jednotlivé přístupy (statický, dynamický) mají rozdílný způsob, jakým vkládají aspekt do definovaných částí aplikace (dále jen weaving). Statický přístup, jak napovídá název, provádí weaving během kompilace. Po dokončení kompilace jsou již aspekty uzavřeny a nelze je měnit za běhu aplikace. Díky tomu je výsledek nepatrně rychlejší, než jeho konkurenční varianta. Typickým příkladem je AspectJ (15).

Dynamický přístup vykoná weaving až v běhovém prostředí (16). Logicky je pomalejší, ale otevírá nám velkou škálu doposud nevídaných možností – především měnit aspekty za běhu aplikace. Kvůli jednotlivým změnám aspektů není nutné znovu kompilovat celou aplikaci. Vhodným použitím můžeme ve výsledku dosáhnout rychlejšího řešení než se statickým přístupem, který musíme neustále

dokola kompilovat. Bohužel s tím také narůstají nároky jak na implementaci, tak na uvědomění si konkrétního kódu jako celku. Tento přístup je nejčastěji používán v aplikacích vystavených na Spring frameworku, jako je i tato práce.

Abychom opravdu využili plnou sílu AOP, je nezbytné identifikovat a ostře oddělit funkcionalitu (odpovědnost) jednotlivých částí kódů – nalézt funkční celky, kterým následně definujeme společné aspekty.

2.3 Spring Framework

Dalším frameworkem, který zvýrazňuje modularitu aplikace je Spring Framework (17). Není ho nutné příliš představovat. Je to dnes již standartní nástroj pro vývoj J2EE aplikací. Má obrovskou komunitou příznivců po celém světě. Velmi příjemně překvapí fakt, že přes mnoholetý živelný vývoj si stále zachoval původní pružnost a tvárnost.

2.3.1 Vlastnosti

- Usnadňuje vývoj především webových aplikací, můžeme říci obecně – korporátních aplikací.
- Pracuje na návrhovém vzoru, který předává odpovědnost za vytvoření instance a jejich vazeb z aplikace do frameworku (dále jen IoC).
 - Vnitřně pracuje s namapovanými Beans.
- Existuje několik způsobů, jakými v IoC předáváme závislosti (Dependency Injection):
 - Setter Injection (vsazení pomocí metod set)
 - Constructor Injection (vsazení pomocí konstruktora)
 - Interface Injection (vsazení pomocí rozhraní)

2.3.2 Výhody

- kód se svobodnou licencí (Apache Licence (18))
- zdůraznění modularity
- integrace s ORM
- zprostředkovatel AOP

2.4 GWT (Google Web Toolkit)

Dalším povedeným frameworkem, který stojí za zmínku, je GWT (19). Jelikož se Google brání označení framework, bude nadále označován jako toolkit, jak byl poprvé veřejně prezentován na Google developer day (20) (dále jen GDD) v roce 2007.

Tento toolkit umožňuje v pohodlí prostředí programovacího jazyku Java vyvíjet klientskou část aplikace – Ajax application. Na komfortu neztrácí ani samotný programátor. K dispozici zůstává plnohodnotné vývojové a ladící prostředí vašeho oblíbeného IDE.

Během kompilace dochází k překladu z Java tříd na samotné JavaScript soubory. Výsledný JavaScript je optimalizovaný pro vybraná jádra prohlížečů (dle konfiguračního souboru) a také (volitelně) obfuskovaný (21).

Obfuskace (z anglického slova obfuscate) je proces, který upravuje původní zápis zdrojového kódu do lidsky nečitelného zápisu. Tímto lze například zabránit v podvrhnutí upravené výkonné části klientské webové aplikace, kde si může kdokoliv upravit některé JavaScript třídy. S takto upravenými JavaScript třídami je to téměř nemožné.

Dostaneme tedy přeložený JavaScript, nicméně při jeho implementaci v syntaxi jazyku Java to musíme mít neustále na paměti.. Java kód se musí psát tak, aby se co nejvíce přiblížil optimálnímu JavaScript. Přeložení Java kódu na JavaScript chápeme spíše jako překlad syntaktických rozdílů. Nestačí tedy umět programovat v jazyce Java. Je nutné znát syntaxi jazyku Java a stejně dobře chápat logiku vnitřních procesů jazyka JavaScript. S výslednou aplikací pracuje uživatel, který každé naše pochybení pocítí na odezvě GUI (rychlosti práce výsledné JavaScript aplikace).

Pro začátek postačí dodržovat tři základní principy:

- nepoužívat globální proměnné
- nepoužívat v konstruktoru žádnou funkci, ani výkonovou logiku
- nepoužívat dědičnost (jen v nezbytných případech)

2.4.1 Výhody

- Jedná se o otevřený zdrojový kód.
- Fyzicky programujeme v jazyku Java (používáme syntaxi jazyku Java).
- Klientskou část aplikace přeložená do jazyka JavaScript lze ladit i profilovat v reálném čase z IDE.
- Máme k dispozici komponenty po vzoru JComponent (22), které se také překládají do jazyka JavaScript (v tomto případě kombinace HTML a JavaScript).
- Obdobně jsou na tom možnosti rozvržení komponent, kontejnerů a práce s událostmi.

2.4.2 Nevýhody

- JavaScript nedokáže plnohodnotně nahradit jazyk Java.

- Některé třídy či metody kompilátor neumí přeložit na JavaScript (např. reflexe v jazyce Java).
- Jiné třídy JavaScript sice podporuje, ale chovají se jinak než v nativním prostředí jazyku Java (např. regulární výrazy).
- Nelze vyloučit ani třídy, které ještě nejsou v GWT implementované.
- Kompilace i časová náročnost stoupá exponenciálně s rostoucím počtem klientských tříd.
- Vývoj s sebou nese časté hledání a mazání různých keší na úrovni OS.

2.5 Sencha GXT

GXT (23) můžeme ve zkratce označit jako grafickou nadstavbu pro GWT. Obsahuje komponenty, které vychází z původních GWT, stejně tak Event, Layout apod. Mimo precizně nastýlovaných komponent přináší také grafickou uniformitu nezávisle na prohlížeči:

- Druhá řada (tj. verze 2.2.x) podporuje generaci prohlížečů od Internet Explorer 6 až po současnost.
- Třetí řada (tj. verze 3.x.x) podporuje generaci prohlížečů v závislosti na podpoře HTML5 (obecně od Internet Explorer 8).

Práce s tímto frameworkem bohužel mění některé principy práce s GWT. Je namístě si objasnit, jakým způsobem probíhá vykreslování komponent a jejich zařazení do HTML struktury v podání GXT. Každá komponenta má mnoho stavů, které jsou řízeny pomocí událostí přes centrální řadič. Jednotlivé události jsou vždy potomky třídy Event (dále jen Event). Centrální řadič je také implementovaný a vychází ze základního rozhraní EventBus (dále jen EventBus). Posloupnosti událostí se liší v závislosti na:

- typu komponenty
- typu prohlížeče
- charakteru dat prezentovaného obsahu
 - synchronní zdroj dat
 - asynchronní zdroj dat

Obecně však musíme předpokládat, že každá komponenta projde stavy, které jsou blíže popsány níže a po celou dobu proces řídí EventBus, kam jsou vystavovány jednotlivé události (např. požadavek na vytvoření nové instance objektu).

2.5.1 Inicializace

Hlavní pracovní vlákno klientské části aplikace naslouchá centrálnímu EventBusu. Zde odposlechne, že vznikl požadavek na vytvoření nové instance. Zamrazí hlavní pracovní vlákno, tím pádem přestane reagovat GUI na uživatelské vstupy. Zavolá se konstruktor a vykoná jeho implementace. V tuto chvíli komponenta na úrovni HTML ještě neexistuje.

Pokud se v konstruktoru volá časově náročná funkcionalita, zamrzne hlavní vlákno GUI. Po dobu vykonávání čeká celá klientská aplikace.

2.5.2 Vykreslování

Po dokončení inicializace se spustí vzájemně synchronní posloupnost událostí „beforeRender“, „onRender“ a „afterRender“. Na úrovni HTML vzniká značka (tag), který obsahuje celou komponentu. Ta se ale uměle skrývá mimo pozorovatelnou uživatelskou zónu a hibernuje se, dokud není EventBusem povolána od akce.

Tento proces bývá často označován jako parkování či dokování komponenty. Taková komponenta je zdeformovaná na minimální možnou velikost (pokud si v kódu nenadefinujeme jinak, je to zpravidla 0×0 px). Účelem je pouze přenést to daného elementu všechny jeho požadované části jako je např. style, id, class a čekat na další instrukce, které ale také nemusí nikdy přijít.

2.5.3 Zobrazení

Opět na vyzvání EventBusu přechází komponenta do dalšího stavu. Komponenty, které mají charakter okna či dialogu, se aktivují událostí onResize (požadavek na uvedení komponenty do finální velikosti) vzájemně synchronní posloupnost událostí beforeShow, onShow a afterShow.

U komponent, které nemají charakter okna ani dialogu, jsou události nahrazeny analogicky-místo Show se použije Layout.

Teprve nyní dojde k vyhodnocení HTML značky jako celku. Následuje jeho zařazení na správné místo ve výsledném HTML kódu stránky. Na závěr se mu nastaví všechny požadované vlastnosti.

II. PRAKTICKÁ ČÁST

3 ANALÝZA

Nejprve si musíme uvědomit celý proces, a co má být jeho výsledkem. Hledáme optimální řešení, jak uživateli umožnit vytvářet různé filtry nad databází, a na základě těchto filtrů definovat další operace, které uživatel může nějak zhodnotit.

Jako příklad si představme funkci „Hlídací pes“. Uživatel vytvoří filtr např. nad tabulkou nabídky práce. S určitou periodou budeme tabulku prohledávat zadaným filtrem. Jakmile přibude nová položka, kterou filtr dříve nevracel, uvědomíme uživatele.

Identifikace jednotlivých aplikačních celků (dále jen modulů), které přímo přijdou, případně mohou přijít do styku s nově vzniklou knihovnou:

- databáze + DAO
- část aplikační logiky
- administrace projektu
- prezenční část projektu

Požadavky ze strany uživatele:

- snadná a rychlá tvorba definic pohledů na omezenou množinu dat

Požadavky ze strany programátora:

- snadná integrace do systému
- maximum otevřeného kódu
- generický kód s typovou kontrolou

4 NÁVRH IMPLEMENTACE

4.1 Databáze

Jako první přichází na řadu databáze. Zde je nutné prověřit, zda množina dat, kterou chceme zpřístupnit pro tvorbu uživatelských filtrů, má:

- požadovanou strukturu
- správně provázané relační vazby mezi jednotlivými tabulkami
- vhodně zvolené datové typy sloupců
- indexy nad klíčovými sloupci
- unikátní indexy nad vhodnými datovými sloupci

Nedá se říci, že by některý z bodů byl více či méně důležitý než jiný. Všechny mají přímý a nekompromisní dopad na výsledek.

V dalším kroku vznikne aplikační mapování ušité přímo na míru tomuto databázovému výseku a pozdější zásahy do namapované databázové struktury znamenají návrat na začátek.

4.1.1 Struktura

Reálná databázová struktura se velmi často liší od struktury, kterou se uživateli snažíme naservírovat formou různých aplikačních komponent. Pokud nám databázová struktura neumožňuje přímo zobrazit uživateli tabulky (případně části tabulek jako jednotlivých celků, máme možnost např. připravit databázové pohledy (materializované či nikoliv).

Pohledy znamenají nárůst časové i výpočetní složitosti. Tento ústupek je vhodný spíše pro vyzkoušení nových možností bez nutnosti dělat invazivní změny v databázi. Nakonec až finální konfrontace s potřebami uživatelů (případně chování trhu) napoví, zda strukturu upravit. Pokud ano, nevyhneme se invazivním změnám databáze.

4.1.2 Relační vazby

Nejhorší případ je chybně provázaná relační vazba – tzn. do jiné tabulky. V takovém případě bude aplikace prezentovat nesmyslné hodnoty pro očekávaná pole. Dalším stejně častým pochybením je úplná absence cizího klíče. V důsledku se zobrazí uživateli místo vybraného sloupce z relační tabulky pouze surová hodnota (povětšinou primární klíč, tedy číslo) ze zdrojové tabulky.

4.1.3 Datové typy sloupců

Musíme předpokládat, že uživatel neví, co chce, a už vůbec to neumí správně předat do systému. Proto je nezbytné každý jeho krok v maximální míře validovat. K tomu nám poslouží do značné míry také datové typy jednotlivých sloupců. S jejich pomocí můžeme kontrolovat, zda při filtrování např. na sloupec s hodnotami typu datum, uživatel zadal platné datum splňující podporovanou masku pro datum. Obdobně je tomu u číselných sloupců (celých i s desetinným rozvojem), u textových sloupců omezených na určitá slova (např. pohlaví může být pouze muž nebo žena).

4.1.4 Indexy

Bohužel musíme také zvážit i méně příjemné důsledky našeho počínání. Nevzniká pouze uživatelem definovaný filtr, který vrací hledané výsledky. Vzniká také zátěž na databázový stroj. Těmto nepříjemným důsledkům můžeme do značné míry předcházet. Vždy se musíme ujistit, že sloupce, nad kterými necháváme uživatele filtrovat, mají indexy. Stejně tak jsou důležité sloupce, pomocí kterých jsou provázané tabulky. V opačném případě hrozí dočasné přetížení databázového stroje a v závislosti na počtu aktivních uživatelů a uživatelských filtrů také celkový kolaps.

4.1.5 Unikátní indexy

V neposlední řadě se musíme přesvědčit o kvalitě dat, kterou předkládáme uživateli, jako výsledek filtru. Předáváme mu nejen data, která splňují podmínky jeho filtru, ale také data, které se do databáze dostávají z rukou jiných uživatelů. Opět musíme předpokládat, že uživatel je člověk, tudíž dělá chyby. Při absenci unikátních indexů nad datovými sloupci může docházet k duplicitám v datech. To je nejčastější chyba, s kterou se uživatel v praxi musí potýkat - duplicitní záznamy (v tomto případě duplicitní výsledky).

4.2 DAO

Tato vrstva na první pohled nepřináší nic nového. Vzniká jedna k jedné vůči databázové struktuře. Nicméně zde zůstane zapouzdřená zodpovědnost za validitu vznikajících uživatelských filtrů. Díky tomu je možné se opřít o sílu zvoleného ORM frameworku. Připomínám velmi důležitý fakt, že k úspěšnému generátoru uživatelských filtrů je potřebná především tato vrstva. Každý doménový objekt, nad kterým jsou plánované uživatelské filtry, musí být namapován včetně všech relačních vazeb. Nyní si ukážeme, jak se s takovým frameworkem pracuje.

4.2.1 Mapování doménových objektů

Nejprve si připravíme dvojici objektů, sloužících pro mapování doménového objektu:

- UJO slouží pro serverovou část aplikace,
- CUJO je odlehčená verze z původního UJO, slouží pro klientskou část aplikace
 - Jsou serializovatelné (připravené pro REST api ke komunikaci klient server).

4.2.2 UJO

Třída UJO je obrazem databázové tabulky, se kterou pracuje aplikace. Současně pokud dojde k načtení dat z databáze, instance UJO prezentuje ze jmenované tabulky vždy právě jeden řádek. Klíčem k úspěchu je správně namapovat databázovou strukturu.

Mapování objektů je vhodné udělat ručně (alespoň zpočátku). Pomůže nám to lépe pochopit, kde a jak dochází k předávání metadat k jednotlivým databázovým tabulkám. Je také možné objekty vytvořit generátorem.

V obou případech je průběh i princip stejný. Klíčové informace obsahuje SQL pro vytvoření databázové tabulky. Každý databázový sloupeček se namapuje jako Key. Metadata se definují anotací Column nad daným Key. Metadata si Ujorm předává i do klientské části aplikace. S výhodou je používá nejen na validování uživatelských filtrů, ale také na validaci administračních dialogů jednotlivých tabulek.

Anotace Column nabízí následující proměnné k uchování metadat:

- name název databázového sloupce
- mandatory je nad sloupcem not null constraint
- length maximální povolená délka dat ve sloupci
- pk je nad sloupcem primární klíč
- index je nad sloupcem index
- uniqueIndex je nad sloupcem unikátní index

Podobných anotací Ujorm nabízí více. Vytváří tak pomocná metadata, které se využívají do značné míry na klientské části aplikace. Anotace GuiIgnored umožňuje definovat, které sloupce se během procesu generování administračního GUI mají přeskočit. Nabízí tři základní modifikace:

- display zobrazit sloupec v klientské části aplikace
- edit je tento sloupec editovatelný v klientské části aplikace
- validation má se tento sloupec validovat před zápisovými operacemi

Poslední zajímavou anotací je Sortable. Každá relační vazba mezi dvěma databázovými tabulkami je v aplikaci prezentována uživatelsky příjemnější formou – nezobrazuje se přímo cizí klíč (pověštinou id z druhé relační tabulky), ale právě sloupec označený v relační tabulce anotací Sortable. Zbývá zajistit statickou inicializaci třídy pomocí rodičovské metody init.

Mapování databázové tabulky jako UJO objektu je hotové. Aby se s UJO dalo pracovat jako s klasickým POJO, zbývá zajistit několik náležitostí. Je vhodné vytvořit prázdný konstruktor. Každému Key je nutné vytvořit get a set metody. Na jejich vytvoření lze využít připravený modul pro IDE UjoGenerator.

Je vhodné (nikoliv nezbytné) přetížít metodu toString a zajistit vhodnější textovou prezentaci mapovaného UJO objektu (nabízí se např. použít jako návratovou hodnotu get metodu nad Key označeným anotací Sortable).

Cílem mapování tedy je vytvořit z SQL definiční třídu UJO (viz náhled níže).

```
CREATE TABLE "public"."discount_condition" (  
  "id" int8 NOT NULL,  
  "description" varchar(255) COLLATE "default" NOT NULL,  
  "condition" text COLLATE "default" NOT NULL,  
  "serialized_condition" text COLLATE "default" NOT NULL,  
  "user_created" int8 NOT NULL,  
  "date_created" timestamptz(6) NOT NULL,  
  "user_modified" int8,  
  "date_modified" timestamptz(6),  
  "user_deleted" int8,  
  "date_deleted" timestamptz(6),  
  "active" int2 NOT NULL,  
  "condition_version" int4 NOT NULL  
);
```

Zdrojový kód 1: SQL struktura databázové tabulky

```

public class DiscountCondition
    extends DovolenaUjo
    implements UserVirtualModifiable {

    /** Primary Key ...3 lines */
    @GuiIgnored(validation = true)
    @Column(pk = true)
    public static final Key<DiscountCondition, Long> id = newKey();

    /** ...3 lines */
    @Sortable
    @Column(
        name = "condition",
        mandatory = true,
        length = CoreSettings.DB_TEXT_MAX_LENGTH)
    public static final Key<DiscountCondition, String> condition = newKey();

    /**
     *
     */
    @Column(
        name = "description",
        mandatory = true,
        length = CoreSettings.DB_STRING_MAX_INDEX_LENGTH,
        index = "AUTO")
    public static final Key<DiscountCondition, String> description = newKey();

    /** ...3 lines */
    @GuiIgnored(edit = true, display = true)
    @Column(
        name = "serialized_condition",
        mandatory = true,
        length = CoreSettings.DB_TEXT_MAX_LENGTH)
    public static final Key<DiscountCondition, String> serialCondition = newKey();

    ...

    /** Property initialization ...3 lines */
    static {
        init(DiscountCondition.class);
    }

    public DiscountCondition() {...2 lines }

    @Override
    public String toString() {
        return getCondition();
    }

    GET / SET
}

```

Obrázek 1: Příklad mapování UJO objektů (třída DiscountCondition)

4.2.3 Database.java

Jakmile je hotová UJO třída (mapování sloupců tabulky), nadešel čas namapovat i samotnou tabulku. K těmto účelům slouží třída Database.java. Jedná se o velmi podobnou strukturu, jakou má UJO. Také dědí z OrmTable a je popsána pomocí statických konstant typu RelationToMany. V genericce uvádíme jako nosnou třídu Database a namísto mapované tabulky připravenou UJO třídu. Konstanta

se inicializuje metodou v rodiči `newRelation`. Pomocí anotace `Table` z ujo frameworku se definuje vazba mezi vytvořenou konstantou a konkrétní databázovou tabulkou. Máme na výběr ze dvou proměnných v rámci anotace:

- `name` povinná, udává název databázové tabulky
- `sequence` (nepovinná, je-li v databázi sekvence, přijde sem její název).

```
public class Database extends OrmTable<Database> {  
  
    @Table(  
        name = "discount_condition",  
        sequence = "sq_discount_condition_id"  
    )  
    public static final RelationToMany<Database, DiscountCondition>  
        DISCOUNT_CONDITION = newRelation();  
}
```

Zdrojový kód 2: Definice anotace `Table`

4.2.4 Zavedení Ujormu do aplikace

Ze spring kontextu využijeme bean `OrmHandler` a předáme jí připravené mapování databázových tabulek na UJO třídy. Zbytek zařídí Ujorm.

```
import org.springframework.web.context.WebApplicationContext;  
import org.ujorm.orm.OrmHandler;  
  
private WebApplicationContext ctx;  
private OrmHandler handler;  
  
protected void initUjorm() throws  
    BeansException,  
    IllegalArgumentException {  
    log.info("Dovolena ujorm loading...");  
    // Spusteni UJORM  
    handler = (OrmHandler) ctx.getBean("ormHandler");  
    handler.config(getMetaRoot());  
    handler.loadDatabase(Database.class);  
}
```

Zdrojový kód 3: Spuštění Ujormu

4.3 Aplikační logika

Aplikační logiku tvoří vrstva servisních služeb. Vzhledem k připraveným UJO třídám, se bude snadno škálovat aplikační logika napříč doménovými objekty. Bude tím docíleno zapouzdření odpovědnosti za jeden doménový objekt na jednom místě a to v servisní službě.

Základním předpokladem je existence spring kontextu a zavedení všech servisních služeb do něj. Dále se jednotlivé metody servisních služeb pomocí AOP obalí nejprve session a následně transakcí.

Samozřejmostí je následný Autowired jednotlivých servisních služeb po horizontální úrovni (tedy tak, aby neopustili servisní vrstvu – aplikační logiku).

4.3.1 Servisní služba

Nejprve se definuje rozhraní servisní služby k dané UJO třídě DiscountCondition.

```
public interface DiscountConditionService {  
  
    /**  
     * Metoda přijme Criterion a pomocí Base64Coderu ho vrátí  
     * serializovaný.  
     *  
     * @param criterion objekt určený k serializaci  
     * @return serialized String  
     */  
    String serializeCriterion(Criterion criterion);  
  
    /**  
     * Metoda přijme serializovaný String a pomocí Base64Coderu  
     * ho vrátí jako Criterion.  
     *  
     * @param serializedCriterion objekt určený k serializaci  
     * @return deserialized Criterion  
     */  
    Criterion deserializeCriterion(String serializedCriterion);  
  
}
```

Zdrojový kód 4: Definice rozhraní servisní třídy

Při vlastní implementaci je nutné zacílit čistě na aplikační logiku. Během implementace lze snadno zapojit i další servisní služby.

4.3.2 Transakce

Po zavedení implementace servisní služby do springu, lze aplikovat na servisní službu transakční řízení za pomoci AOP. Pro tyto účely je již přímo v Ujormu připravený aspekt, kterým s výhodou obalíme vlastní servisní služby.

```
<!--
    Objekt, který se stara o obalování transakci
-->
<bean
    id="aroundServiceTransaction"
    class="org.ujorm.orm.support.AroundServiceTransaction" >

    <constructor-arg ref="ujoSessionFactory" />
</bean>

<!--
    TRANSAKCE okolo služeb
-->
<aop:config proxy-target-class="false">
    <aop:pointcut id="serviceAroundTransaction" expression="(
        execution(* cz.sa.dovolena.server.service.*Service+.*(..))
    )" />
    <aop:aspect
        id="aroundTransaction"
        ref="aroundServiceTransaction"
        order="2" >

        <aop:around
            pointcut-ref="serviceAroundTransaction"
            method="aroundFilter" />
        </aop:aspect>
</aop:config>
```

Zdrojový kód 5: Zavedení transakce okolo služby

4.3.3 Session

Servisní službu s transakcí bude navíc obalena ještě session, opět pomocí AOP.

```
<!--
    SESSION okolo služeb
-->
<aop:config proxy-target-class="false">
    <aop:pointcut id="serviceAroundSession" expression="(
        execution(* cz.sa.dovolena.server.service.*Service+.*(..))
    )" />
    <aop:aspect
        id="aroundSession"
        ref="ujoSessionFactory"
        order="1" >
        <aop:around
            pointcut-ref="serviceAroundSession"
            method="aroundSession" />
        </aop:aspect>
</aop:config>
```

Zdrojový kód 6: Zavedení session okolo služby

4.4 CUJO

Klientský protějšek UJO třídy je na implementaci již mnohem snazší. Opět dědí z rodiče s již implementovanou logikou.

Ke každé statické konstantě typu Key v UJO třídě se zde definuje jedna statická konstanta typu CujoProperty. Pro představu uvažujme, že Ujorm pracuje namísto UJO a CUJO tříd se dvěma mapami (klíč-hodnota). Aby bylo možné mezi mapami přenášet hodnoty, musí se klíče shodovat. Proto je nutné při inicializaci CujoProperty vložit do metody newProperty jako první vstupní parametr přesný název Key.

Stejně jako v UJO třídě i zde existují specifika relačních vazeb. CUJO třída slouží především potřebám serializace. Celý proces serializace obstarává GWT. Není to úplně černá skříňka, do které by nebylo možné nahlédnout. Je to spíše Pandořina skříňka, kterou je lépe nechat zavřenou.

Proto je nutné udělat jeden ústupek vůči GwtRpcPolicy a znovu danou třídu (relační CUJO) nadefinovat jako privátní proměnnou. Zmíněná vazba je v CUJO třídě uvedena v generice a navíc v generice jiné třídy (CujoProperty). Zde ji současná implementace GWT serializeru neumí najít.

V tomto duchu budou zavedeny do CUJO třídy všechny ostatní CujoProperty podle předlohy (UJO a Key). Pro účely mapování je to vše.

```
public class CDiscountCondition
    extends DovolenaCujo {

    public static final CujoProperty<CDiscountCondition, CAdminUser>
        userCreated = pl.newProperty(
            "userCreated",
            CAdminUser.class
        );

    private CAdminUser dummyUser;
}
```

Zdrojový kód 7: Příklad mapování CUJO

Při práci s CUJO třídou je vhodné myslet na důsledky použití GWT. Jedná se o třídu, která je implementovaná v jazyce Java, ale následně je převedena pomocí GWT překladače do jazyku JavaScript. V běhovém prostředí je to již čistý JavaScript. Nemáme tudíž spoustu vymožeností, na které jsme zvyklí z jazyku Java, jako např. reflexi.

4.4.1 UjoTranslator

UjoTranslator zajišťuje obousměrný překlad mezi UJO a CUJO. Je to jedna z mnoha šikovných utilit, která je uvnitř Ujormu. Ve své podstatě je to „pár řádek kódu“, ale radikálně mění možnosti, jak lze pracovat s UJO / CUJO objekty.

Nejcitelnější je situace při překladu UJO na CUJO. UJO reprezentuje jeden řádek databáze z namapované tabulky. Zpravidla ho vrací servisní služba, která ho z databáze načte. Každý takto načtený objekt má session. Díky tomu může až při požadavku na relační vazbu dočíst řádek z relační tabulky. Tento proces je (nejen) v Ujormu označován jako „lazy loading“ (24). Při překladu na CUJO objekt ztrácí jak session, tak i jakoukoliv přímou možnost dočítat relační vazby. Vstupním parametrem do procesu překladu lze nastavit (vynutit) hloubku zanoření (kolik relačních vazeb se má přeložit). Výchozí hodnota je nastavena na úroveň 1.

Opačný proces překladu, kdy opět zpravidla přijde CUJO z klientské aplikace je podobný. Příchozí objekt neobsahuje session, ale může obsahovat vnořené relační vazby do libovolné hloubky. Vzniká UJO objekt, který zatím (po překladu) nemá session, ale převezme z CUJO objektu relační vazby právě do nastavené hloubky. Výchozí hodnota je stejná (tedy 1). Ostatní data se zahodí.

4.5 Controller

Další proces, který se dostává na řadu, je delegování načtených dat z databáze v podobě UJO objektů do klientské části aplikace. Controllery tvoří speciální vrstvu aplikace, která zajišťuje pouze obousměrný překlad UJO a CUJO objektů. Dalším specifikem je hybridní režim controllerů.

Na serverové straně (backend) je controller standardní třída v jazyce Java, je standardně mapovaná do springu jako SpringBean (25). Na klientské straně je controller hotová implementace (součást GWT). Vytváří se pouze speciální rozhraní. Jedná se o asynchronní verzi původního rozhraní controlleru, který máme implementovaný na serveru. Vzniká tedy trojice – dvě rozhraní a jedna implementace.

Díky této vrstvě je zajištěno více vláknové (ajaxové) volání z klientské části aplikace na serverovou část aplikace.

Postup je jednoduchý. Nejprve se připraví klasické rozhraní, které ale je umístěno do klientského balíčku (balíčku určeného k přeložení pomocí GWT překladače do jazyku JavaScript). Rozhraní dědí ze třídy „RemoteService“, která je součástí GWT.

```
package cz.sa.dovolena.client.controller;  
  
public interface DiscountConditionController extends RemoteService
```

Zdrojový kód 8: Definice Controlleru

Teď je potřeba obdobně jako při zavedení bean do springu, zavést tento controller do povědomí GWT. K tomu slouží anotace `RemoteServiceRelativePath`.

```
package cz.sa.dovolena.client.controller;  
  
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;  
  
@RemoteServiceRelativePath(  
    "controller/discountConditionController.rpc"  
)  
public interface DiscountConditionController extends RemoteService
```

Zdrojový kód 9: Registrace controlleru do GWT

Nakonec v těle controlleru lze definovat klasickým způsobem hlavičky metod.

```
String serializeCriterion(CCriterion criterion) throws  
    GxruntimeException;  
  
CCriterion deserializeCriterion(String serializedCriterion) throws  
    GxruntimeException;
```

Zdrojový kód 10: Zavedení metod do controlleru

Serverová implementace je již klasická java třída. Jedinou nutností je podědit z `RemoteServiceServlet` (GWT). Controller se mapuje do springu a naopak ze springu se vytáhne servisní službu, předá do controlleru. Servisní službě se předávají přeložené vstupní parametry (případně naopak).

```
public class DiscountConditionControllerImpl  
    extends RemoteServiceServlet  
    implements DiscountConditionController {  
  
    @Autowired  
    private DiscountConditionService discountConditionService;  
  
    @Override  
    public Service<DiscountCondition> getService () {  
        return discountConditionService;  
    }  
}
```

Zdrojový kód 11: Definice serverové implementace controlleru

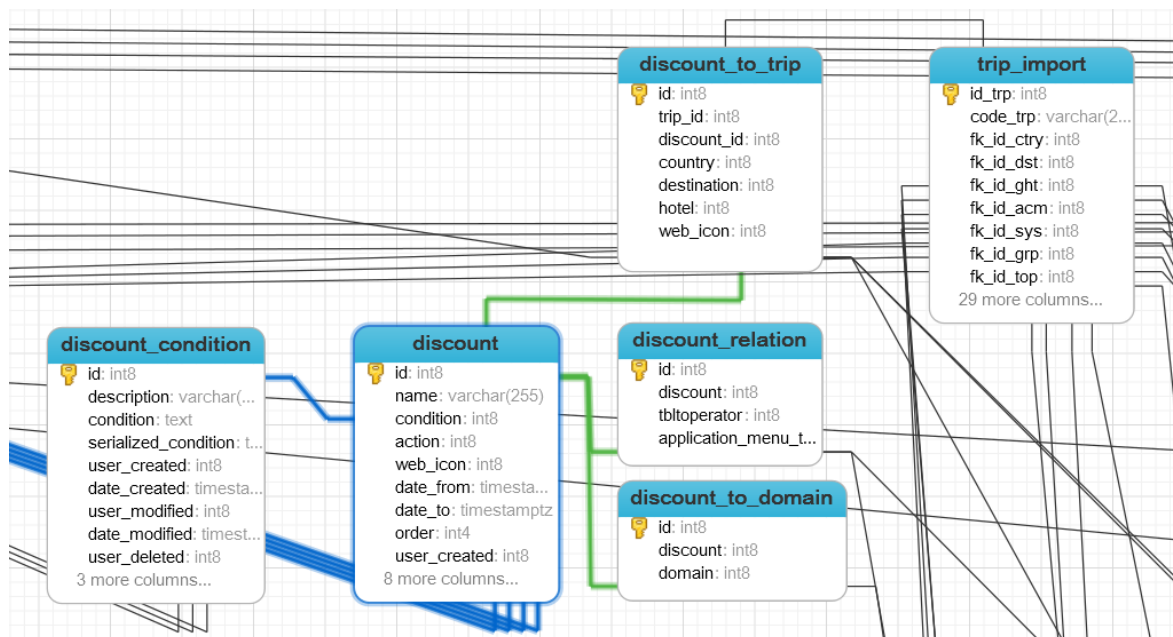
5 NAsAZENÍ DO PRODUKCE

Konkrétní implementace vyhledávací komponenty byla realizovaná na modelu slevového systému ve společnosti Student Agency na produktu dovolena.cz.

5.1 Databázová struktura

Již z naznačené databázové struktury je patrné, jak celý model pracuje. Ve stručnosti si představíme databázové tabulky:

- discount_condition (tabulka uživatelských filtrů)
- discount (tabulka aktivních slev)
- trip_import (tabulka zájezdů)
- discount_to_trip (vazební tabulka mezi slevami a zájezdy)
- další pomocné tabulky

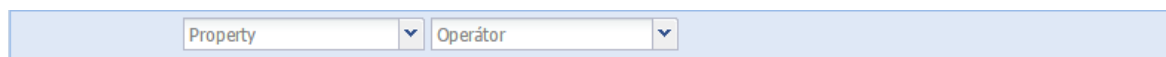


Obrázek 2 Výřez z ER diagramu databáze (pohled na tabulky slevového systému)

Ve stručnosti si představíme také způsob použití. Uživatel (administrátor webu) definuje filtr, který omezuje tabulku se zájezdy. Uživatel si dále nastaví k jednotlivým slevám adekvátní podmínky slev (uživatelský filtr). Aplikační logikou aplikujeme filtr nad zájezdy. Zájezdy splňující podmínku jsou pomocí vazební tabulky provázány se slevami, které tuto podmínku používají. Pomocí vyplněné vazební tabulky mohou následně zákazníci na webu vyhledávat v reálném čase zájezdy podle jednotlivých slev.

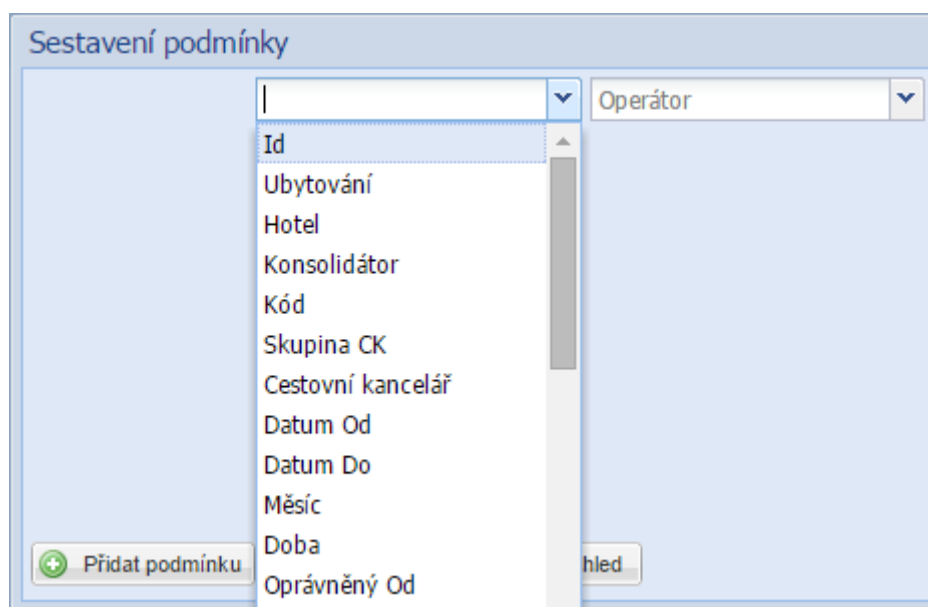
5.2 GUI

Nyní si předvedeme náhled výsledné aplikace s ohledem na výše zmíněný způsob použití a databázovou strukturu. Pomocí jednoduchého průvodce si uživatel navolí průchod přes určitou množinu dat. Z počátku může průvodce působit nenápadně a zakřiknutě, ale již po prvním kliknutí uživatele velmi mile překvapí rychlost a síla GUI.



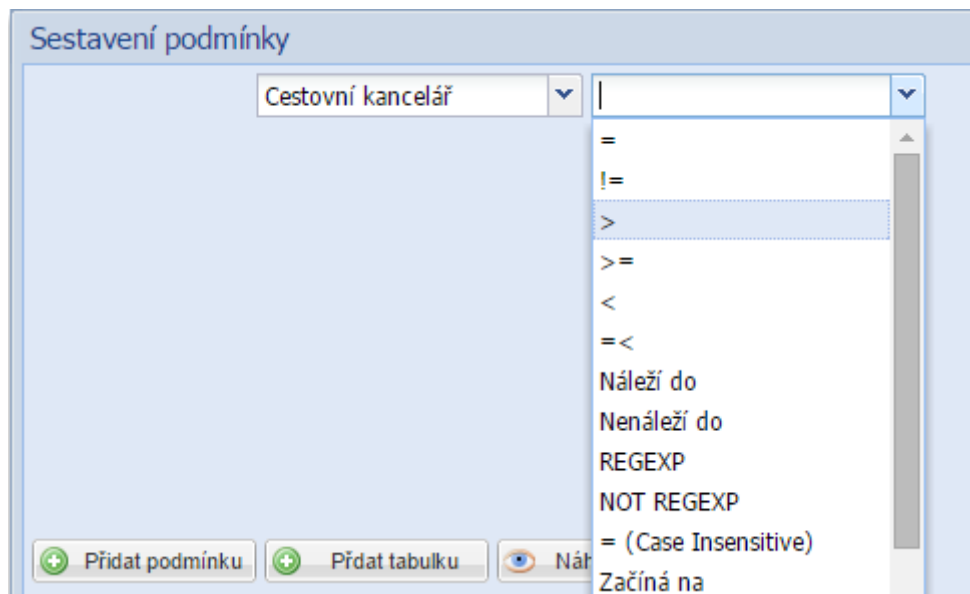
Obrázek 3: Prázdný řádek filtru

Po rozbalení první nabídky je jasné, že generické editační dialogy nepustí uživatele ani o kousek mimo návrh původní databáze. Za předlohu může uživatel volit pouze namapované (databázové) sloupce.



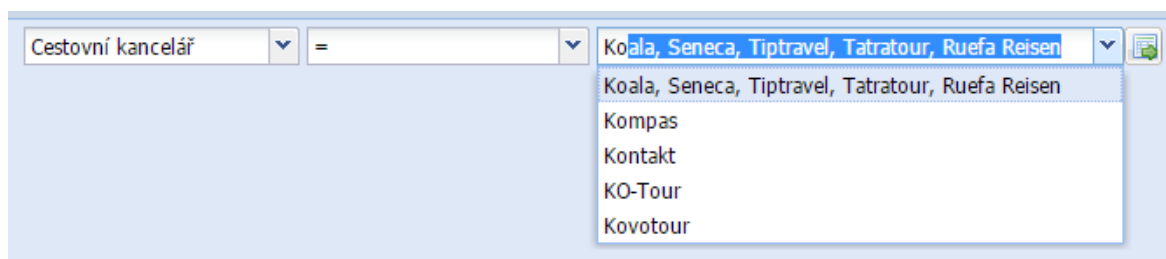
Obrázek 4: Řádek filtru, výběr předlohy

Obdobně je tomu při volbě operátora mezi předlohou a referenční hodnotou. Na výběr má uživatel z celé řady operátorů, přičemž všechny nabízí plnohodnotnou validaci na uživatelský vstup do referenční hodnoty.



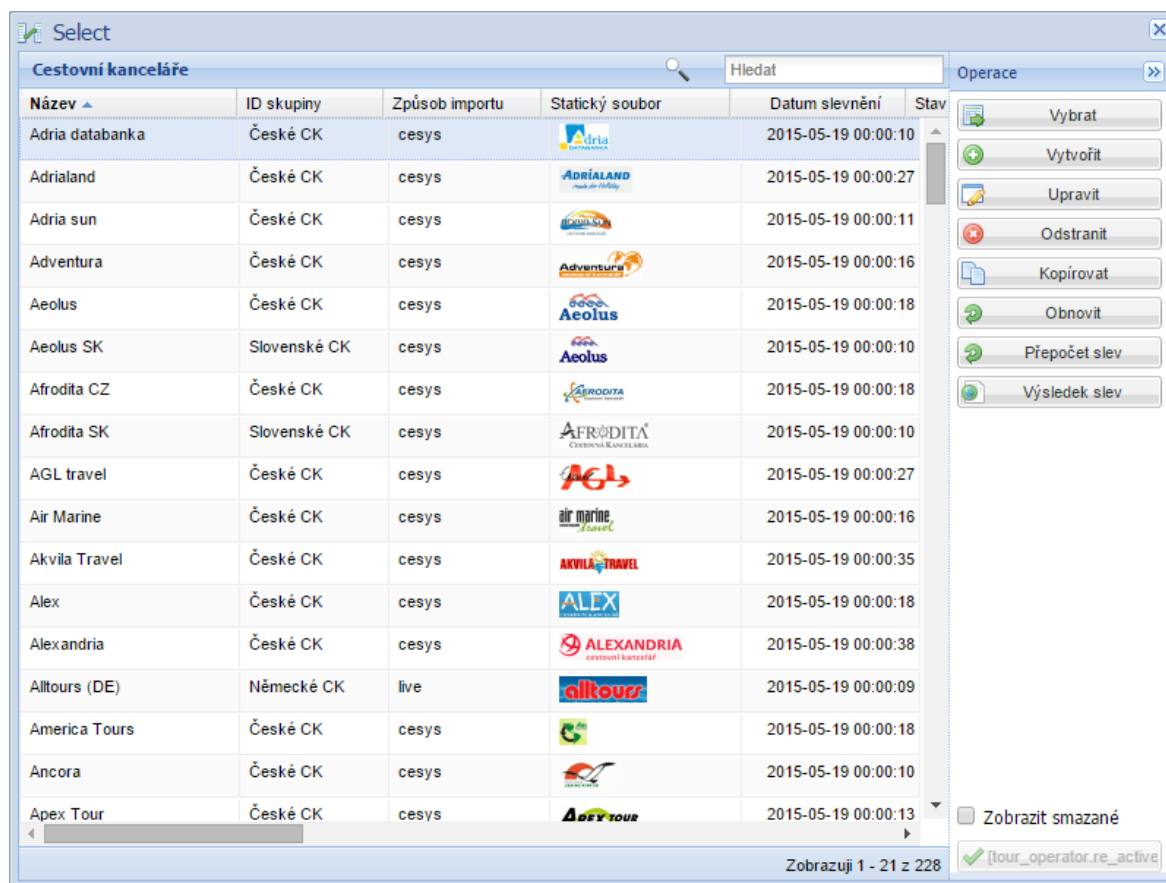
Obrázek 5: Řádek filtru, výběr operátora

Referenční hodnota se vybírá datově omezená vzhledem k levé části (předloze) před operátorem. Řekněme, že si uživatel vybere Cestovní kancelář (případně jinou relační vazbu, která je definovaná v DB). V ten okamžik má na výběr pouze z hodnot dané databázové tabulky. Ve zrychleném výběru nabízí speciální komponenta – CujoBox (obdoba propracovaného combo boxu) s našeptávačem právě sloupec označený anotací Sortable v relační tabulce. Výběr je podle něj také seřazený.



Obrázek 6: CujoBox

Pokud je relační tabulka složitější a jeden sloupec (označený anotací Sortable) neumožňuje odpovědně vybrat jednu hodnotu, uživatel může využít také tlačítko na konci řádku k otevření plnohodnotné tabulky (speciální komponenty LiveGrid) se všemi sloupci (mimo zakázaných anotací GuiIgnored) a vybrat hledanou hodnotu z této tabulky.

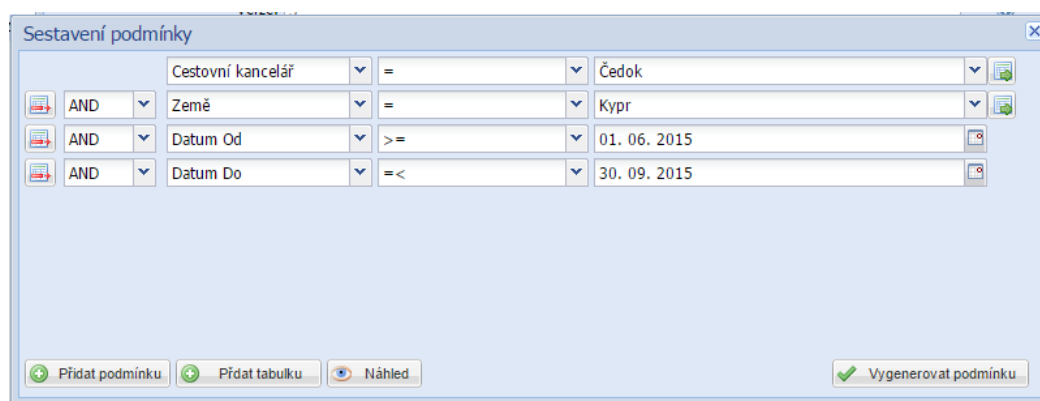


The screenshot shows a software interface titled "Select" with a search bar and a table of travel agencies. The table has columns for Name, Group ID, Import Method, Static File, and Date. A sidebar on the right contains various action buttons like "Vybrat", "Vytvořit", "Upravit", etc.

Název	ID skupiny	Způsob importu	Statický soubor	Datum slevnění	Stav
Adria databanka	České CK	cesys	Adria	2015-05-19 00:00:10	
Adrialand	České CK	cesys	ADRIALAND	2015-05-19 00:00:27	
Adria sun	České CK	cesys	Adria sun	2015-05-19 00:00:11	
Adventura	České CK	cesys	Adventura	2015-05-19 00:00:16	
Aeolus	České CK	cesys	Aeolus	2015-05-19 00:00:18	
Aeolus SK	Slovenské CK	cesys	Aeolus	2015-05-19 00:00:10	
Afrodita CZ	České CK	cesys	AFRODITA	2015-05-19 00:00:18	
Afrodita SK	Slovenské CK	cesys	AFRODITA	2015-05-19 00:00:10	
AGL travel	České CK	cesys	AGL	2015-05-19 00:00:27	
Air Marine	České CK	cesys	air marine	2015-05-19 00:00:16	
Akvila Travel	České CK	cesys	AKVILA TRAVEL	2015-05-19 00:00:35	
Alex	České CK	cesys	ALEX	2015-05-19 00:00:18	
Alexandria	České CK	cesys	ALEXANDRIA	2015-05-19 00:00:38	
Alltours (DE)	Německé CK	live	alltours	2015-05-19 00:00:09	
America Tours	České CK	cesys	America Tours	2015-05-19 00:00:18	
Ancora	České CK	cesys	Ancora	2015-05-19 00:00:10	
Apex Tour	České CK	cesys	APEX TOUR	2015-05-19 00:00:13	

Obrázek 7: LiveGrid

Obdobným způsobem si uživatel navolí i zbytek filtru. Výsledek se po stisknutí tlačítka „Vygenerovat podmínku“ převede na klientský Criterion, který je následným procesem převeden do serializovatelné podoby.

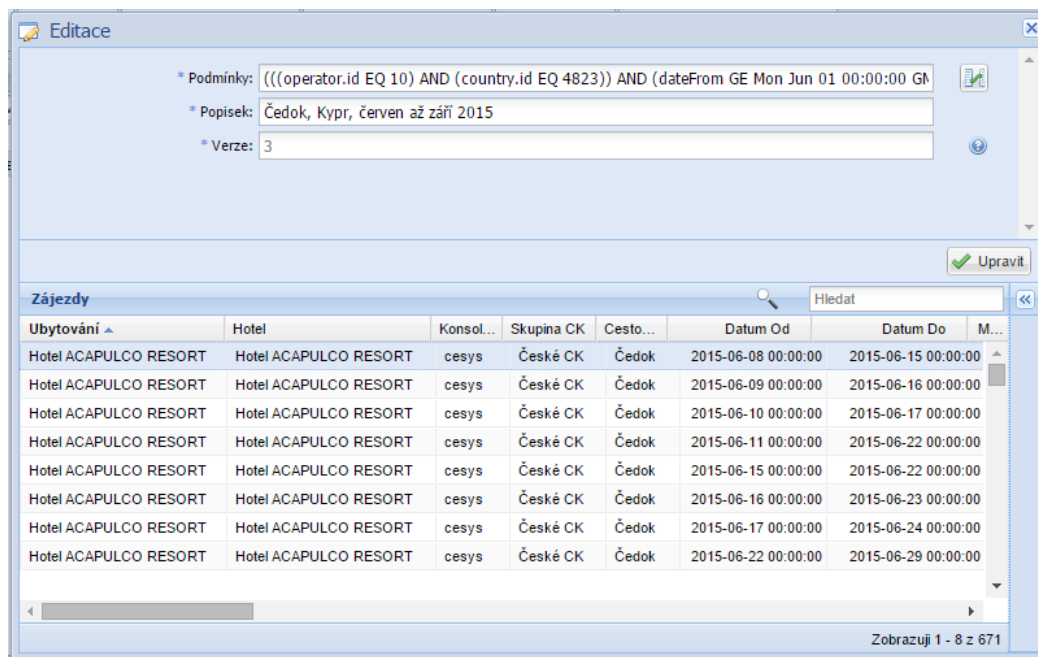


The dialog box shows a visual query builder with four rows of conditions connected by AND operators. The conditions are: Cestovní kancelář = Čedok, Země = Kypr, Datum Od >= 01. 06. 2015, and Datum Do <= 30. 09. 2015. Buttons at the bottom allow adding conditions, tables, and generating the filter.

AND	Cestovní kancelář	=	Čedok
AND	Země	=	Kypr
AND	Datum Od	>=	01. 06. 2015
AND	Datum Do	<=	30. 09. 2015

Obrázek 8: Průvodce sestavením filtru

Z textové podoby filtru si uživatel může kdykoliv přepnout zpět do grafického průvodce, provést dodatečné změny a opět nechat vygenerovat Criterion. Samozřejmostí je živý pohled na data, která odpovídají navolenému filtru.



Obrázek 9: Uložení filtru a živý náhled na data

V posledním kroku uživatel pouze zvolí akci, kterou chce provázat s výslednou množinou dat vytvořeného filtru. V tomto případě vybere jednu slevu, kterou si přeje označit data (zájezdy).

Operace

- Vytvořit
- Upravit
- Odstranit
- Kopírovat
- Obnovit
- [F2] Dohledej CK
- [F3] Uprav podmínku
- [F4] Upravit článek

Zobrazit smazané

Obnovit

Zobrazuji 58 - 73 z 276

Id	Název	Podmínka
928	Hotely s aquaparkem	(operator EQ 278) AND (hotel IN [1033, 5071, 12471, 1383, 1385])
929	Hotely s aquaparkem	(operator EQ 486) AND (hotel IN [152713, 151012])
930	Hotely s aquaparkem	(operator EQ 296) AND (hotel IN [31398, 12558, 134176, 148, 43])
931	Hotely s aquaparkem	(operator EQ 258) AND (hotel IN [86215, 154720, 5985, 71670, 6])
954	Kapesné na dovolenou 66 E...	(((operator.id EQ 14) AND (type.id EQ 80)) AND (transport.id EQ 1))
969	Kyperské dny - sleva 15%	(((operator.id EQ 10) AND (country.id EQ 4823)) AND (dateFrom GE Mon Jun 01 00:00:00 G...))
-2	Lastminute	
610	Letní slevy až 13 % na letec...	(((operator.id EQ 10) AND (dateFrom GE Fri May 01 00:00:00 G...))
614	Letní slevy až 4 % na zájez...	(((transport.id EQ 36) OR (transport.id EQ 37)) AND (operator.id EQ 1))
164	Narozeninová sleva	operator EQ 490
414	Narozeninová sleva	(((operator EQ 3) AND (country EQ 4847)) AND (dateFrom GE Mon Jun 01 00:00:00 G...))
122	Noci zadarmo	(((operator EQ 38) AND (country IN [4873, 4813, 4826, 4866])) AND (dateFrom GE Mon Jun 01 00:00:00 G...))
6	Novomanželská sleva	operator EQ 9
46	Novomanželská sleva	operator EQ 78
66	Novomanželská sleva	(operator EQ 20) AND (destination NOT_EQ 8481)
124	Novomanželská sleva	(((operator EQ 38) AND (country IN [4873, 4813, 4826, 4866])) AND (dateFrom GE Mon Jun 01 00:00:00 G...))

Obrázek 10: Tabulka slev

ZÁVĚR

Cílem bylo analyzovat a implementovat možnosti vytváření uživatelských filtrů. Cíle bylo dosaženo. Vznikla knihovna, která je implementovaná v jazyku Java (Java SE + Spring). Využívá maximum moderních technologií, které jsou zároveň ověřené praxí.

Hlavní přínos je genericky kontrolovaný kód během vývoje a uživatelské formuláře s asynchronní validací na serverové části aplikace. Tato kombinace ulehčuje práci jak budoucímu programátorovi během implementace na další projekt, tak kontrolu před nevalidním uživatelským vstupem.

Během reálné aplikace této knihovny do produkčního prostředí, došlo k drobným ústupkům v původní architektuře. Je to důsledek rozptýleného kódu, na který musí knihovna různým způsobem reagovat. Zde je prostor pro další vylepšení.

V současné architektuře je naznačená možnost, jak definovat dynamicky akci nad výsledky uživatelských filtrů za běhu aplikace. Nabízí se několik možností, od zadání požadavku na jednorázovou implementaci různých tříd a přepínat mezi cílovými třídami, až po možnost vytvoření dynamicky kompilované třídy v běhovém prostředí. Tento model se však nesetkal s očekáváním produkčního provozu a byl nahrazen statickou obsluhou, která je pevně implementovaná. Myšlenka vychází z předpokladu, že je výhodnější zkopírovat podmínku a vytvořit pro ni novou (jinou) akci. Vždy tedy existuje právě jedna dvojice podmínka-akce.

Tento model v praxi nemusí být pravidlem. Možnosti jak povýšit současné řešení byly již nastíněny výše.

SEZNAM POUŽITÉ LITERATURY

1. Front and back ends. *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] https://en.wikipedia.org/wiki/Front_and_back_ends.
2. Relational database. *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] https://en.wikipedia.org/wiki/Relational_database.
3. Ponec, Pavel. The Ujorm. *Ujorm, the unique open source object-relation mapping*. [Online] 2014. [Citace: 19. 05 2015.] <http://ujorm.org/>.
4. Object-relational mapping. *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] https://en.wikipedia.org/wiki/Object-relational_mapping.
5. Data access object. *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] https://en.wikipedia.org/wiki/Data_access_object.
6. Hibernate ORM, Idiomatic persistence for Java and relational databases. *Hibernate*. [Online] [Citace: 19. 05 2015.] <http://hibernate.org/orm/>.
7. Plain Old Java Object. *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] https://en.wikipedia.org/wiki/Plain_Old_Java_Object.
8. JavaBeans. *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] <https://en.wikipedia.org/wiki/JavaBeans>.
9. Reflection (computer programming). *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] [https://en.wikipedia.org/wiki/Reflection_\(computer_programming\)](https://en.wikipedia.org/wiki/Reflection_(computer_programming)).
10. SQL. *Wikipedia*. [Online] 2016. [Cited: 05 27, 2016.] <https://en.wikipedia.org/wiki/SQL>.
11. Integrated development environment. *Wikipedia*. [Online] 2016. [Cited: 05 27, 2016.] https://en.wikipedia.org/wiki/Integrated_development_environment.
12. Graphical user interface. *Wikipedia*. [Online] 2016. [Cited: 05 27, 2016.] https://en.wikipedia.org/wiki/Graphical_user_interface.
13. Introduction. *mybatis*. [Online] 2016. [Cited: 05 27, 2016.] <http://www.mybatis.org/mybatis-3/>.
14. Aspect-oriented programming. *Wikipedia*. [Online] 2016. [Cited: 05 27, 2016.] https://en.wikipedia.org/wiki/Aspect-oriented_programming.
15. aspectj. *eclipse*. [Online] 2016. [Cited: 05 27, 2016.] <https://eclipse.org/aspectj/>.

16. Java (software platform). *Wikipedia*. [Online] 2016. [Cited: 05 27, 2016.] [https://en.wikipedia.org/wiki/Java_\(software_platform\)](https://en.wikipedia.org/wiki/Java_(software_platform)).
17. Spring Framework. *Spring*. [Online] 2015. [Citace: 20. 05 2015.] <http://projects.spring.io/spring-framework/>.
18. Apache License. *Wikipedia*. [Online] 2016. [Cited: 5 26, 2016.] https://en.wikipedia.org/wiki/Apache_License.
19. GWT. *GWT*. [Online] Google Inc., 2015. [Citace: 20. 05 2015.] <http://www.gwtproject.org/>.
20. Google Developer Day. *Wikipedia*. [Online] 2016. [Cited: 05 26, 2016.] https://en.wikipedia.org/wiki/Google_Developer_Day.
21. Obfuscation (software). *Wikipedia*. [Online] 2016. [Cited: 05 27, 2016.] [https://en.wikipedia.org/wiki/Obfuscation_\(software\)](https://en.wikipedia.org/wiki/Obfuscation_(software)).
22. The JComponent Class. *The Java™ Tutorials*. [Online] Oracle, 2016. [Cited: 05 27, 2016.] <https://docs.oracle.com/javase/tutorial/uiswing/components/jcomponent.html>.
23. Sencha GXT. *Sencha*. [Online] Sencha Inc., 2015. [Citace: 20. 05 2015.] <http://www.sencha.com/products/gxt/#overview>.
24. Lazy loading. *Wikipedia*. [Online] 09. 04 2016. [Citace: 24. 05 2016.] https://cs.wikipedia.org/wiki/Lazy_loading.
25. Spring - Bean Definition. *tutorialspoint*. [Online] 2016. [Cited: 5 26, 2016.] http://www.tutorialspoint.com/spring/spring_bean_definition.htm.
26. ECKEL, Bruce. *Thinking in JAVA*. 2003.
27. WALLS, Craig. *Spring in action*. Fourth edition. 2013. ISBN 978-161-7291-203.
28. HARROP, Rob a MACHACEK, Jan. *Pro Spring*. 2007.
29. HUNT, Charlie a JOHN, Binu. *Java performance*. 2011.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GUI	Graphical User Interface
ORM	Object Relation Mapping
DB	DataBase
SQL	Structured Query Language
DAO	Database Access Object
POJO	Plain Old Java Object.
UJO	Unified Java Object
CUJO	Client UJO
CRUD	Create Read Update Delete
ORM	Object Relation Mapping
AOP	Aspect Oriented Programming
API	Application Programming Interface
SOAP	Simple Object Access Protocol
REST	REpresentational State Transfer
J2EE	Java 2 Enterprise Edition
IoC	Inversion of Control
XML	eXtensible Markup Language
HTTP	Hypertext Transfer Protocol
GWT	Google Web Toolkit
GDD	Google Developer Day
GXT	Ext GWT
IE	Internet Explorer
HTML	HyperText Markup Language
px	pixel
IDE	Integrated Development Environment

JMX Java Management Extensions

OS Operating systém

SEZNAM OBRÁZKŮ

Obrázek 1: Příklad mapování UJO objektů (třída DiscountCondition).....	24
Obrázek 2 Výřez z ER diagramu databáze (pohled na tabulky slevového systému).....	31
Obrázek 3: Prázdný řádek filtru	32
Obrázek 4: Řádek filtru, výběr předlohy	32
Obrázek 5: Řádek filtru, výběr operátora.....	33
Obrázek 6: CujoBox.....	33
Obrázek 7: LiveGrid	34
Obrázek 8: Průvodce sestavením filtru	34
Obrázek 9: Uložení filtru a živý náhled na data.....	35
Obrázek 10: Tabulka slev	35

SEZNAM ZDROJOVÉHO KÓDU

Zdrojový kód 1: SQL struktura databázové tabulky	23
Zdrojový kód 2: Definice anotace Table.....	25
Zdrojový kód 3: Spuštění Ujormu	25
Zdrojový kód 4: Definice interface servisní třídy	26
Zdrojový kód 5: Zavedení transakce okolo služby	27
Zdrojový kód 6: Zavedení session okolo služby.....	27
Zdrojový kód 7: Příklad mapování CUJO	28
Zdrojový kód 8: Definice Controlleru	30
Zdrojový kód 9: Registrace controlleru do GWT	30
Zdrojový kód 10: Zavedení metod do controlleru	30
Zdrojový kód 11: Definice serverové implementace controlleru	30