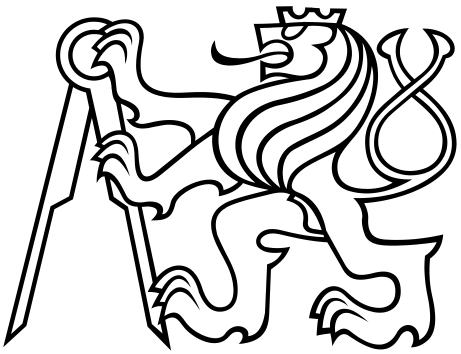
České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra počítačů

BAKALÁŘSKÁ PRÁCE



Webový komunitní systém pro evidenci a sdílení dárků a přání

Vypracovala: Kateřina Ostrihoňová

Vedoucí bakalářské práce: Ing. Miroslav Bureš, Ph.D.

Studijní program: Softwarové technologie a management

Studijní obor: Softwarové inženýrství

Praha 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Kateřina Ostrihoňová**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Webový komunitní systém pro evidenci a sdílení dárků a přání**

Pokyny pro vypracování:

Navrhněte a implementujte systém pro evidenci a sdílení dárků a přání. Systém bude uživatelům umožňovat vytváření přání a dárků, jejich sdílení v rámci komunity uživatelů, rezervaci přání nebo dárku a sledování jejich plnění.
Součástí systému bude administrátorské rozhraní pro správu uživatelů a datových objektů ukládaných systémem.
Systém implementujte jako webovou aplikaci v jazyce Java s použitím vhodných frameworků a otestujte sadou uživatelských testů.

Seznam odborné literatury:

Arlow, J., Neustadt, I. UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky. Brno: Computer Press; 2007.
Fowler, M., Rice, D. Patterns of enterprise application architecture. Boston: Addison-Wesley; 2003.
Anderson, S.P. Přitažlivý interaktivní design: Jak vytvářet uživatelsky přívětivé produkty. Brno: Computer Press; 2012.

Vedoucí: Ing. Miroslav Bureš, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

[redacted]

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S.

[redacted]

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 31. 10. 2014

Poděkování

Děkuji panu doktoru Miroslavu Burešovi za odborné vedení mé bakalářské práce.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vyhotovila samostatně a – v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací – v předložené zprávě uvádím použité informační zdroje.

V Praze ke dni odevzdání 27. května 2016 .....

Anotace

Název práce:

Webový komunitní systém pro evidenci a sdílení dárků a přání

Abstrakt:

Cílem bakalářské práce byla analýza, návrh a implementace webového komunitního systému pro evidenci a sdílení dárků a přání. Text provází čtenáře od analýzy přes výběr nástrojů pro vytvoření aplikace, až k popisu jejich použití. Systém je založen na použití rámce Spring a k vystavění programu je využit nástroj Maven. Aplikace umožňuje přihlášeným uživatelům spravovat přání, třídit je do skupin, zamlouvat a plnit v rámci přátelství. Taktéž přátelé mohou být organizováni ve skupinách. Administrátorům je umožněna správa vkládaných objektů. Funkčnosti ověřila sada uživatelských testů.

Klíčová slova:

Spring, JSF, DAO Dispatcher, JPA, Maven

Annotation

Title:

Community Web System for Tracking and Sharing of Gifts and Wishes

Abstract:

Goal of the work was analysis, design and implementation of a community web system for tracking and sharing of gifts and wishes. The text leads a reader from analysis to choosing the right tools for implementation of the application, to their usage. System is Spring based and uses Maven for its build. Logged-in users can manage their wishes, sort them into groups, booked them or fulfilled them among friends. Friends can be also organized into groups. Administrator can manage saved objects. Functionalities were tested by a set of user tests.

Keywords:

Spring, JSF, DAO Dispatcher, JPA, Maven

Obsah

1 Úvod do textu

Dobrý dárek by měl způsobit na obou stranách upřímnou radost a možná svému příjemci splnit toužebně očekávané přání. Bez možnosti nápovědy se ale i dárek pořizovaný s těmi nejlepšími úmysly může stát nevítaným, nebo dokonce příjemce svou nevhodností urazit. Ale i setkání dvou či více sice žádaných, ale identických dárků, může způsobit jak dárci, tak příjemci rozpaky.

V českém prostředí zatím neexistuje dostatečně komplexní nástroj pro dlouhodobou evidenci dárků, sdílení přání mezi uživateli a vzájemné zamlouvání a plnění v okruhu přátel. Proto jsem zvolila toto téma jako předmět své bakalářské práce, přičemž analyzovaný, navrhovaný a implementovaný systém je koncipován jako webová aplikace.

Text zprávy není členěn na čistě teoretickou část a její praktické užití, ale podobá se svým sledem dokumentaci provedené práce. Následující kapitoly zaznamenají postup plnění cíle mého úsilí – vytvoření úvodní vize projektu včetně komplexního katalogu požadavků, provedení analýzy, návrhu a implementace aplikace.

Kapitola 2 je věnována vizi projektu. Obsahuje informace o klíčových vlastnostech a konkurenceschopnosti mého řešení, určení všech uživatelů systému, rešerši podobných systémů, úvod do obsáhlého katalogu požadavků a diskuzi rozsahu projektu.

Zvolený objem požadavků je podroben analýze v kapitole 3. Model systému zde ilustrují vybraná schémata případů užití, procesní a stavové diagramy. Výsledný analytický doménový model poskytuje „lidský“ pohled na řešený problém.

Protože návrhové vzory jako obecné nástroje považuji za *zavedené* abstraktní řešení, zatímco konstrukty zvolených programových rámců se mohou měnit, rozdělila jsem další text na dvě části.

Kapitola 4 provází návrhem architektury aplikace a rozhraní výhradně z pohledu výběru teoretických konceptů a standardů. Doplňuje ji návrhový doménový model.

Výběr technologických řešení, rámců a popis použitých konstruktů jsou z důvodu možných rozdílů mezi konkrétními verzemi a jejich vývoji umístěny do kapitoly 5. Zde také uvedu výsledný diagram balíčků aplikace a popis vybraných implementačních postupů.

Popisu způsobu ověření realizovaných funkčností aplikace jsem věnovala kapitolu 6. Obsahuje jak prvotní stanovení testovacích úrovní částí aplikace, tak samotné výsledky uživatelských testů.

Závěrečné zhodnocení dosažení mnou nastoleného cíle a rozbor možností případných rozšíření jsou obsahem poslední kapitoly 7.

2 Vize projektu a modelování požadavků

Následující vize projektu v souladu s její charakteristikou uvedenou [Conallen, 2003, str. 162] určí problémovou doménu, stanoví budoucí uživatele, klíčové vlastnosti a zhodnotí konkurenceschopnost vyvíjené aplikace. Přiřazením priorit výsledným požadavkům bude určena rozsáhlost analyzovaného, navrhovaného a implementovaného projektu.

2.1 Motivace pro vznik

Jak jsem již nastínila v úvodu: darování i přijímání dárků se může ve vybraných případech stát společensky náročnou záležitostí. Problém výběru vhodného dárku by vyřešil systém umožňující vysněná přání sdílet a mezi přáteli rezervovat pro následující splnění.

2.2 Stávající systémy v českém prostředí

Jednou z podobně zaměřených webových aplikací je SeznamDarku.cz, který nabízí uživateli tvorbu několika seznamů přání a jejich splnění s konkrétním datem. Přání lze po registraci vybírat z nabídky serveru Darek.cz, nebo přidat ručně. Jako průvodce pro výběr dárků slouží taktéž externí zdroj na zmiňovaném webu. Systém inzeruje funkci připomenutí narozenin.

Druhým a dle mého názoru graficky a funkčně vydařenějším projektem je DejMiDarek.cz. Projekt využívá lokálních blogerů[1] a majitelů obchodů pro spolupráci formou vytvoření takzvaných kolekcí. Kolekce obsahuje tipy na dárky z různých internetových obchodů. Projekt uživatelům umožňuje komentovat kolekce a přání, je přímo spojen s blogem[2] a nabízí možnost soutěžit o dárky.

2.3 Budoucí uživatelé systému

Webová aplikace by měla být ve své kompletní formě přístupná široké veřejnosti, ale pro využívání některých funkčností bude vyžadováno přihlášení do jedné ze vzájemně výlučných rolí. Systém bude rozlišovat následující tři typy uživatelů:

- **nepřihlášený** uživatel,
- uživatel systému **přihlášený v roli uživatele**
- a uživatel systému **přihlášený v roli administrátora**.

Uvedenému rozdělení návštěvníků webové aplikace je podřízeno jak členění skupin katalogu funkčních a nefunkčních požadavků, tak diagramů případů užití systému.

[1] bloger – autor internetového deníku
[2] blog – internetový deník

2.4 Klíčové vlastnosti a konkurenceschopnost mého řešení

Aplikace si klade za cíl poskytnout návštěvníkům uživatelsky přívětivý komunitní web, mezi jehož výhody budou patřit například funkčnosti uvedené v následujícím výčtu.

- Touhu po daném dárku určí pro každé přání jeho hodnota daná počtem hvězdiček.
- Přání přátel bude možné zamluvit, ale také jeho rezervaci zrušit.
- Přání bude řazeno do skupiny s možností určit příležitost k darování, což umožní tvorbu specializovaných skupin – například pro suvenýry či sběratelské předměty.
- Přání bude mít své místo původu pro umožnění sbírky předmětů z celého světa.
- Sám vlastník přání nebude mít možnost zjistit, která přání již přátelé zamluvili. Nepřijde proto o překvapení.
- Vlastník přání bude moci zamluvit své vlastní přání a tím zrušit zamluvení přáteli.
- Zamilované přání mohou uživatelé (od)označit a tím určovat jeho oblíbenost.

2.5 Požadavky na systém správy dárků a přání

Katalog se skládá z funkčních a nefunkčních požadavků. Funkční požadavky jsou rozděleny do skupin v závislosti na typu uživatele. Ve skupině obecných funkčních požadavků jsou zahrnuty budoucí funkčnosti přístupné taktéž nepřihlášeným návštěvníkům.

Každou položku seznamu jsem opatřila identifikační značkou pro snadnější mapování na případy užití. Protože je ale výčet požadavků detailní, odpovídá každému z nich právě jeden případ užití.

Priority požadavků označené příslušnými písmeny metodou MoSCoW [Arlow a Neustadt, 2008, str. 82] jsou vztaženy k termínu odevzdání dokumentu bakalářské práce:

- *Must have* požadavky na funkční jádro, bez kterého nelze aplikaci využívat,
- důležité *Should have* požadavky, bez kterých ale bude aplikace stále použitelná,
- *Could have* požadavky, na jejichž implementaci v termínu nemusí zbýt čas
- a *Want to have/Won't have* požadavky, které budou splněny v příští verzi.

Kompletní katalog požadavků je rozsáhlý, proto jej připojuji k textu jako přílohu A.

2.6 Stanovení rozsahu realizace k datu odevzdání

Jak jsem již zmínila – důležitou částí práce je stanovení úrovně zpracování požadavků. Dle informací uvedených [Paul a kol., 2010, str. 215] jsem se rozhodla postupovat po iteracích a podrobit v první fázi kompletní péči analýzou, návrhem, implementací a testováním pouze vybrané požadavky označené kategorií M, S, a C. Požadavky W nejsou zahrnuty v následujících kapitolách a budou případně předmětem příští iterace.

3 Analýza domény správy dárků a přání

Analýza poskytne srozumitelný popis *obchodních* procesů a jednotlivých entitních typů doménového modelu. V souladu s iteračním přístupem k vývoji, zvoleným v sekci 2.6, budou analýze podrobeny pouze požadavky vybrané dle svých priorit.

Podle pravidla uvedeného [Arlow a Neustadt, 2008, str. 330] zachytí představené diagramy analytického modelu funkce, které musí systém poskytovat pro uspokojení požadavků vize. Způsob, jakým budou do problémové domény zanesena architektonická a technická řešení, bude obsahem kapitol příštích.

3.1 Popis procesů aplikace správy dárků a přání

V následujících částech popíši pomocí vhodných diagramů procesy problémové domény. Vybrala jsem funkčně nejdůležitější entitní typy a jejich definované vlastnosti.

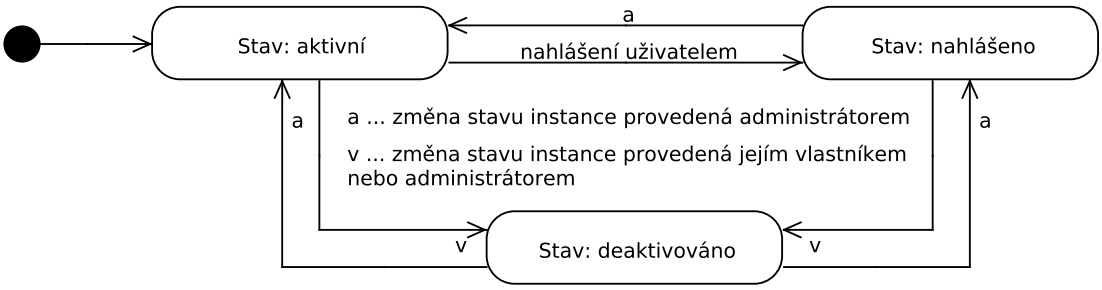
3.1.1 Popis změny stavu vybraných entitních typů

V případě entitních typů hrajících klíčovou roli bude aplikace využívat možnosti nesmazat jeho instance úplně, ale změnit jejich stav na deaktivovaný. Protože obsah aplikace vytvoří primárně uživatelé, zvolila jsem tento způsob manipulace s daty pro uchování historických nebo neúmyslně odstraněných údajů.

Obrázek 3.1 ukazuje, že administrátor může měnit stav v administračním rozhraní spravovaných instancí dle libosti, zatímco uživatel může:

- instanci nahlásit a zařadit ji tím mezi obsah určený k moderaci administrátorem
- nebo – je-li jí vlastníkem – instanci deaktivovat, čímž ztratí možnost ji spravovat.

Aby *spořádané* uživatele neobtěžovala nebo dokonce neomezovala zlomyslnost jiných registrovaných návštěvníků, neovlivní případné nahlášení jimi vlastněných instancí jakkoliv přístup k poskytovaným funkcím. Kupříkladu přání smí být stále zamluveno.



Obrázek 3.1: Přechody mezi stavy vybraných entitních typů

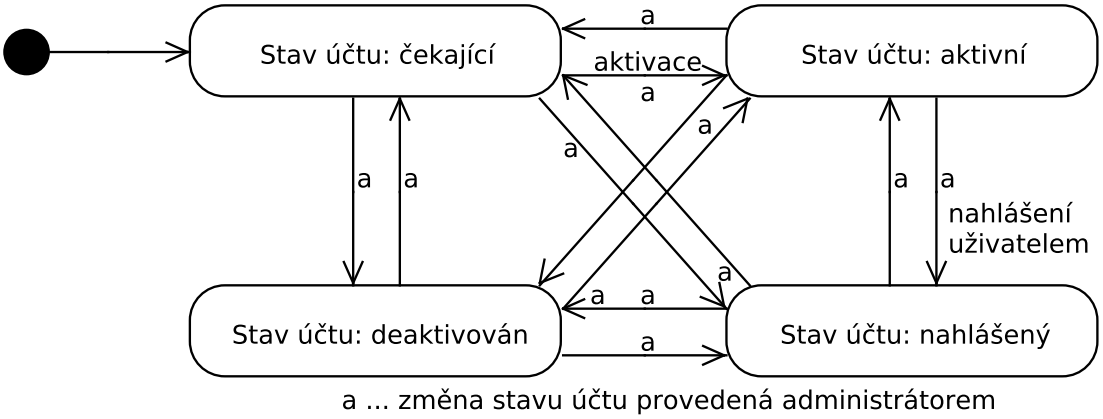
3.1.2 Popis změny stavu účtu uživatele systému

Také v případě účtu uživatele systému jsem zvolila možnost nastavit stav na deaktivovaný, čehož využiji pro zabránění již deaktivovaným uživatelům v opětovné registraci.

Stavem účtu uživatele systému může od okamžiku registrace manipulovat výhradně administrátor s výjimkou dvou operací:

- aktivace účtu po vytvoření nebo registraci nastavením hesla jeho vlastníkem
- a nahlášení *nevhodného* profilu uživatele jiným registrovaným uživatelem.

Registrovaný či vytvořený uživatel systému vzniká ve stavu čekání na aktivaci. Ověřením přiděleného tokenu[1] si jeho majitel nastaví vlastní heslo a účet se stává aktivním.



Obrázek 3.2: Přechody mezi stavy účtu uživatele systému

3.1.3 Popis změn stavu plnění přání

Uživatelům systému v roli *běžného* uživatele je umožněno přáním manipulovat pouze není-li deaktivováno. Samotný stav plnění přání vznikne jako výsledek existence nebo neexistence odpovídajících vztahů – vazeb na uživatele zamlouvajícího či plnícího přání.

Každé přání vytvořené výhradně osobou vlastníka je prvotně volné a určené ke splnění. Již během tvorby může být označeno za splněné. Další stavy při vzniku zvolit nelze.

Jak je patrno ze schématu 3.3: vlastník přání má – co se týče rezervace přání – „navrch“. Mohlo by se zdát zvláštní, že může své přání zamluvit i přesto, že je již zamluveno přítelem, ale důvod je prostý: uživateli nesmí být zkaženo překvapení a proto pro něj zamluvení přání cizí osobou není viditelné.

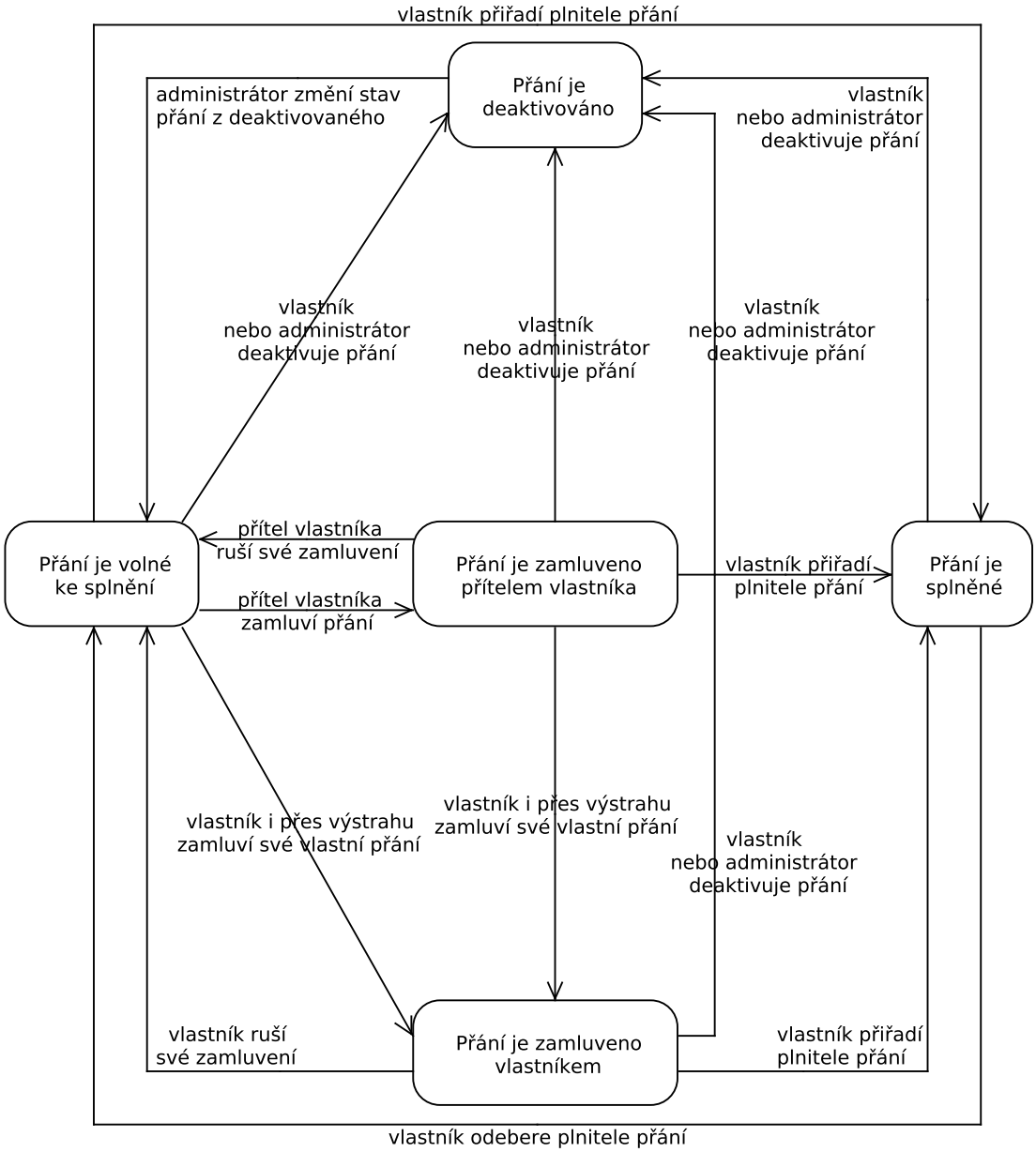
Z důvodu utajení informace o zamluvení přání před jeho vlastníkem, není taktéž ostatním návštěvníkům profilu, kteří nejsou přáteli vlastníka profilu, tato informace zobrazena. Případná veřejnost profilu toto specifikum neovlivní.

[1] token – ve významu „žetonu“ s unikátní informací přidělenou a poskytnutou uživateli systému

Deaktivaci přání může provést pouze jeho vlastník nebo administrátor, pokud usoudí, že je díky svému obsahu nevhodné.

Přání nenabývá, co se týče zamluvení, konečného stavu, ale pokud je deaktivováno administrátorem nebo vlastníkem, všechny vazby zamluvení a splnění jsou zrušeny. Ze stavu deaktivace může být přání vyvedeno pouze administrátorem a ztrátou vazeb se tak stane opět volným ke splnění.

Kýženým stavem přání je ale jeho splnění, jehož dosažení je zcela závislé na uživatelově vůli udat v detailu přání přítele, který jej obdaroval. Domnívám se, že bez provázanosti s webovými obchody, které by nákup dárku – hmatatelné formy přání – uživatele potvrdily, není snadnějšího způsobu.



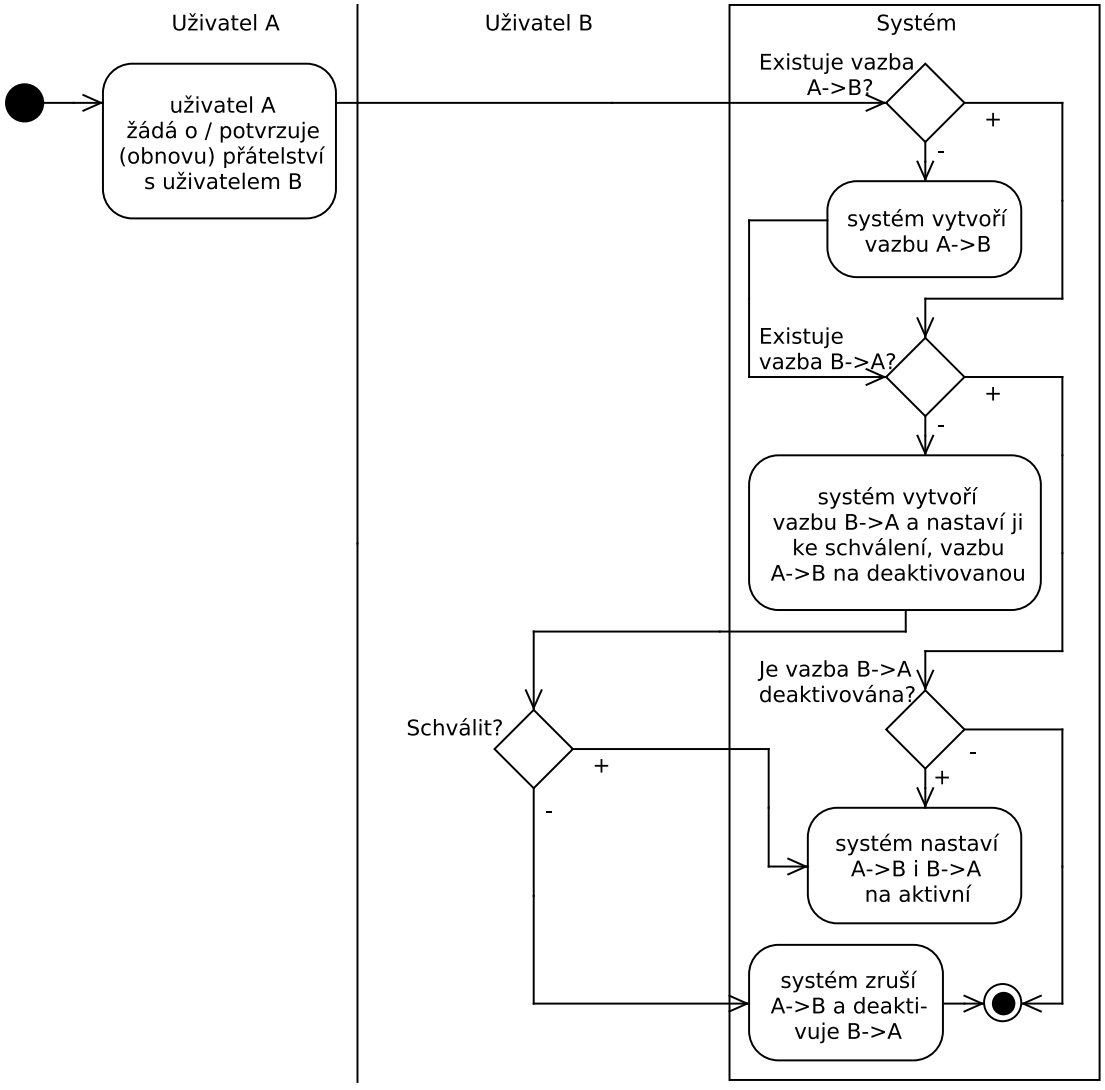
Obrázek 3.3: Přechody mezi stavy plnění přání

3.1.4 Proces žádosti, vyhovění, rušení a obnovy přátelství uživatelů

Následující část textu věnuji důležitému rysu doménovému modelu – přátelství – a pomocí diagramů popíši proces žádosti, potvrzení a obnovy přátelství mezi uživateli.

Abych uchovala informaci o aktuálním stavu přátelství, je ve dvojici přátel vždy tvořeno dvěma vzájemnými vazbami. Označení A→B vybírá relaci přátelství směrem od jejího *vlastníka* A k příteli B a naopak. Přátelství ve dvojici uživatelů je z pohledu funkčnosti použitelné pouze ve chvíli, kdy jsou oba směry v aktivním stavu schválení.

Jak jsem již naznačila – zajímavostí je, že uživatel, jenž přátelství deaktivoval, jej může znovu obnovit. Potíže tedy nebude činit přátelství omylem zrušené. Zároveň bude na uživatele kladena větší zodpovědnosti za výběr přítele v aplikaci, protože obnovení přátelství bude výhradně na něm. Smazání přátelství na straně rušitele má také důvod v zajištění zakázání všech vazeb záměrně deaktivovaného uživatele. Po případné opětovné aktivaci účtu administrátorem může uživatel svá přátelství obnovit.



Obrázek 3.4: Proces žádosti/potvrzení/obnovení přátelství uživatelem A k B

Stavy schválení přátelství reprezentují pohled od vlastníka vazby k opačnému konci:

- neexistující vazba umožní odeslat *cílovému* uživateli žádost o přátelství,
- deaktivovaná vazba symbolizuje odebranou možnost odeslat žádost,
- vazba ke schválení vybízí ke stejné akci jako její označení
- a aktivní vazba umožňuje přátelství pouze zrušit.

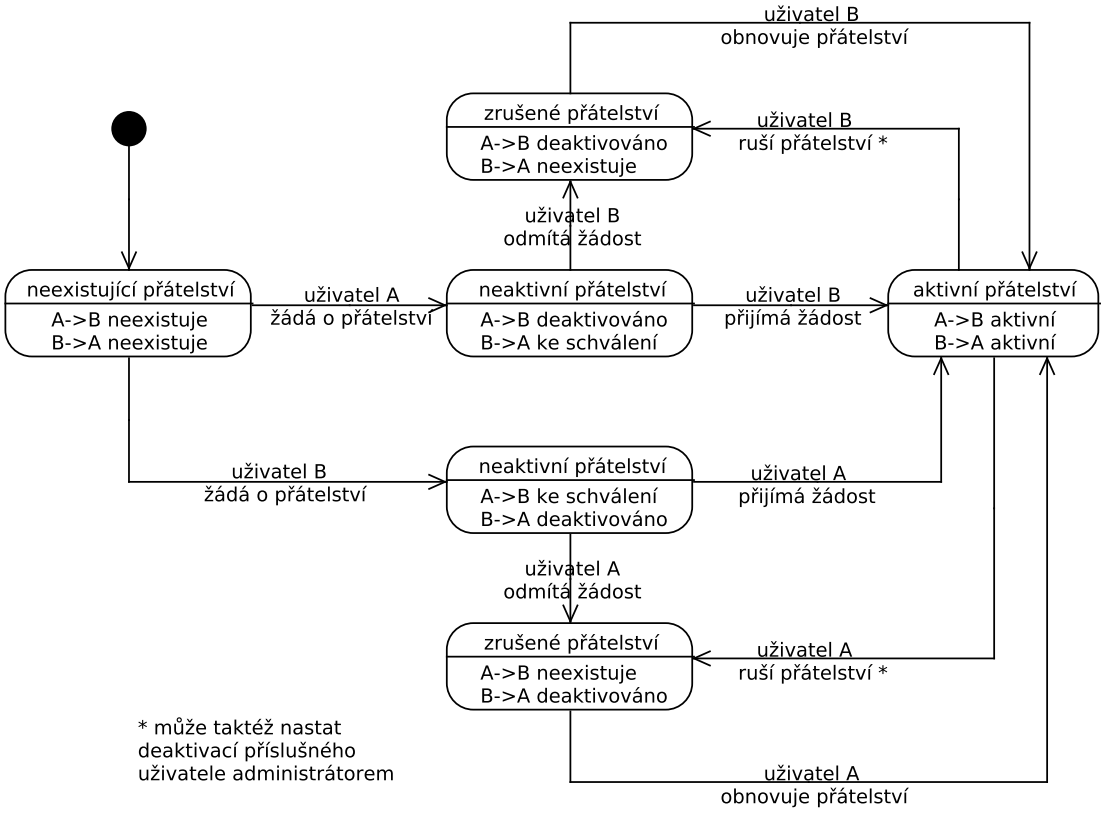
Z pohledu systému se žádost o přátelství, o jeho obnovu nebo přijetí vyhodnotí stejně. Analogicky také odmítnutí žádosti či zrušení přátelství reprezentuje jediná operace.

Kompletní obrázek o stavech přátelství ve dvojici uživatelů nabízí schéma 3.5. Diagram 3.4 je iniciován náročnější ze dvou procesů: uživatel A „žádá“ o přátelství uživatele B.

Existuje-li již v obou směrech vazba a je-li B→A deaktivována, přijímá uživatel A žádost o přátelství uživatele B a oba směry přátelství jsou nastaveny na aktivní.

V případě, že vazba B→A nalezena není, je vytvořena a nastavena ke schválení, ve směru od vlastníka – iniciátora akce – k cíli je deaktivována. Uživatel A požádal o přátelství.

Uživatel B do diagramu vstupuje ve chvíli, kdy další postup závisí na jeho *vyjádření.* Schválením žádosti je přátelství aktivováno, jinak jej uživatel B odmítnutím zruší a je pouze na jeho úvaze, zda přátelství v budoucnu obnoví odesláním vlastní žádosti.



Obrázek 3.5: Změny stavu schválení přátelství

3.2 Případy užití aplikace správy dárků a přání

V souladu s postupem uvedeným [Arlow a Neustadt, 2008, str. 99] následuje specifikaci požadavků, kterou jsem provedla v kapitole 2.5, model případů užití.

Případy užití jsem rozdělila v závislosti na uživatelské roli anebo oblasti použití. Jak jsem již zmínila v sekci požadavků, neuvádím zde mapování případů užití na požadavky, neboť jsem požadavky uvedla tak podrobně, že jednomu případu užití zpravidla odpovídá alespoň jeden požadavek. Pokrytím požadavků by neměl být žádný z nich opomenut.

Protože již byly některé provázanosti a způsoby použití popsány během procesů, vybírám v následujících částech pouze zajímavější či složitější diagramy případů užití.

3.2.1 Případy užití aplikace uživatelem

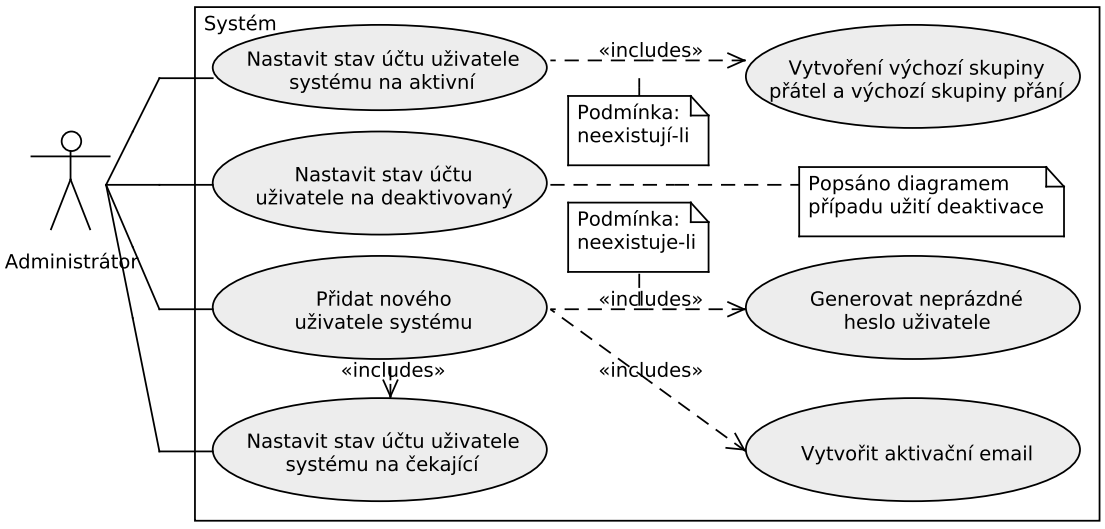
Uživatel je nejdůležitějším účastníkem případů užití a tedy celé aplikace, neboť vytváří její obsah. Uživatel je vládce svých skupin přání, přátel a správy přátelství.

3.2.2 Případy užití aplikace libovolným návštěvníkem

Nepřihlášenému návštěvníkovi je umožněno prohlížení výhradně těch přání a profilů, které jsou svými majiteli vedeny jako veřejné. Kromě toho se může samozřejmě registrovat nebo přihlásit. Seznamy přání, které může nepřihlášený uživatel prohlížet, neobsahují informace o stavu zamluvení. V opačném případě by se mohl registrovaný uživatel lehce připravit o překvapení zobrazením svého profilu před přihlášením.

3.2.3 Případy užití aplikace administrátorem

Nejdůležitější pravomocí a úkolem administrátora je posuzování nahlášených uživatelů, přání a komentářů. Výběrem stavu instance může ovlivnit následně provedenou akci. Diagram 3.6 demonstruje moc, která je administrátorovi svěřena při výběru stavu účtu.



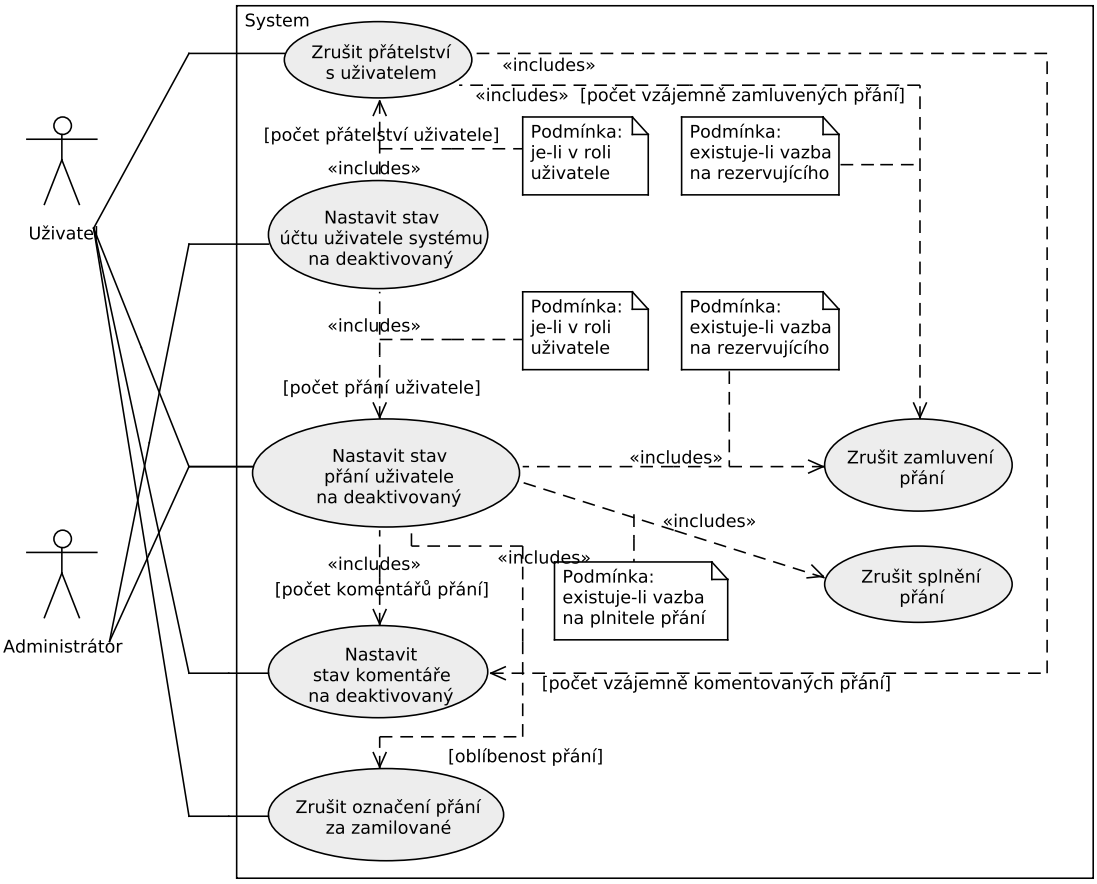
Obrázek 3.6: Vliv změny stavu účtu uživatele provedené administrátorem

3.2.4 Deaktivace instancí entitních typů

Případy užití deaktivace instancí klíčových entitních typů se silně ovlivňují. Následující diagram 3.7 demonstruje jejich vzájemnou provázanost při volbě stavu *deaktivovaný*.

Nejvíce entitních typů ovlivní změna stavu účtu *běžného* uživatele způsobující kaskádovitě deaktivaci veškerých přání, jimž je vlastníkem, zamluvení, komentářů i přátelství.

Úprava postupuje kaskádovitě, dokud není dílo dokončeno. Pouze v případě, že uživatel splnil přání cizí osobě, zůstává tato informace neměnná. Instance, jejichž vlastníkem je přímo uživatel, jím mohou být deaktivovány také samostatně.



Obrázek 3.7: Případy užití deaktivací instancí klíčových entitních typů

3.3 Analytický doménový model

Na základě popisů procesů aplikace uvedeného v sekci 3.1 a případů užití v předchozí části textu, jsem mohla vytvořit analytické třídy doménového modelu.

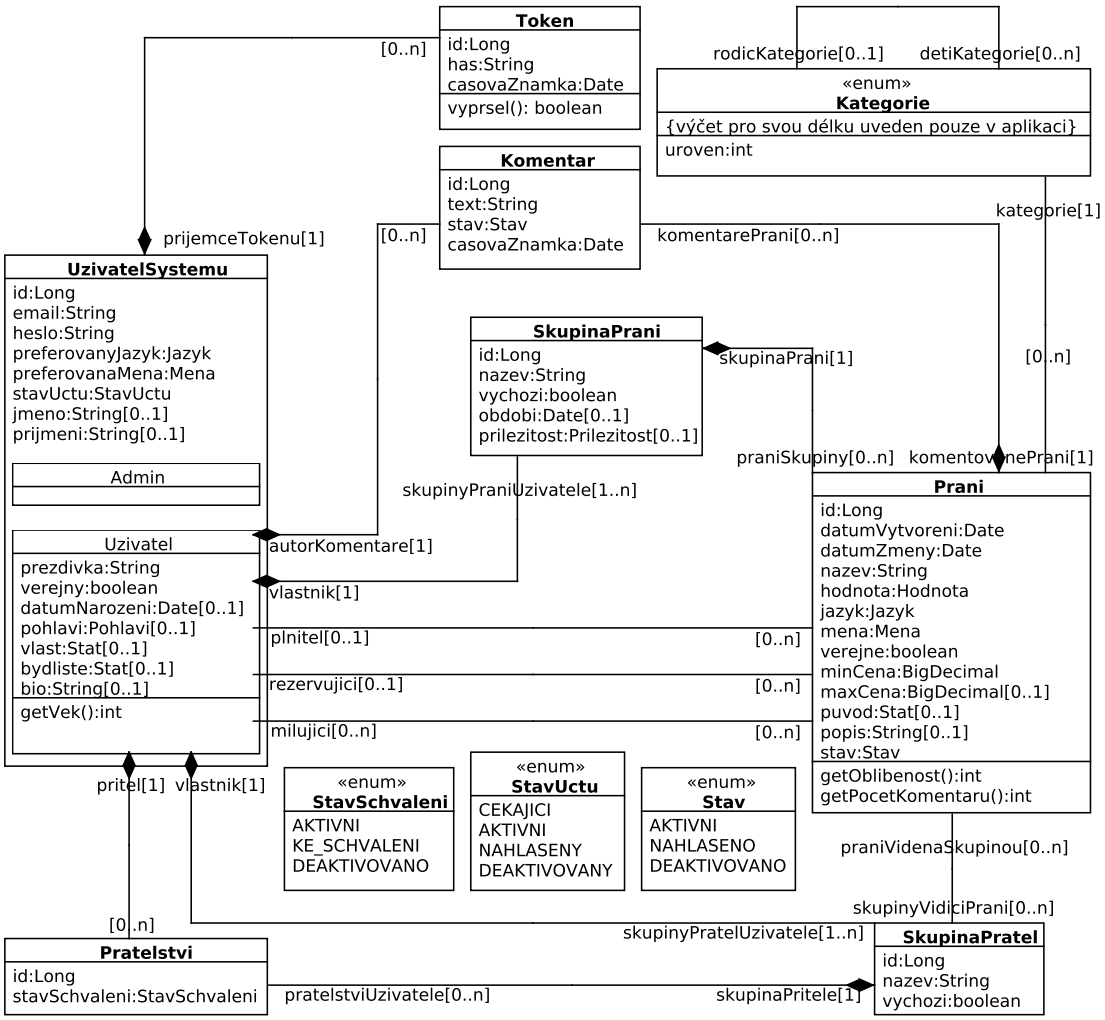
Analytický doménový model dle [Arlow a Neustadt, 2008, str. 173] poskytuje pohled na entitní typy mapující pojmy skutečného světa. Vynechány jsou zde všechny identifikátory a počáteční hodnoty. Specifika, které jsem připojila doménovým třídám ve fázi návrhu, budou uvedena v sekci 4.3.

Konceptuální model je tvořen devíti entitními typy. Obsahuje hierarchii pro rozlišení rolí registrovaných uživatelů. Instance typů `UzivatelSystemu`, `Uzivatel`, `Admin` a `Prani` jsou identifikovány výhradně svým vlastním klíčem. Ostatní entitní typy jsou s nimi v relacích existenční – a tedy i identifikační – závislosti.

Diagram obsahuje také typy reprezentující výčet konstant, které neuvažuji mezi ukládané entitní typy a slouží pro svou neměnnost jako jejich případné instanční parametry. Některé z enumerací jsem ovšem uvedla mimo těla entitních typů – například role instancí určujících stav byla jako klíčová popsána již v sekci věnované procesům 3.1. Příklad rekurzivní vazby *rodič-potomek* ukazuje výčtový typ `Kategorie`.

Model doplňuji výčtem omezení, která musí být dodržena pro zachování integrity dat.

- Přítel uživatele je zařazen pouze v jedné jeho skupině přátel.
- Právě jedna skupina přání resp. skupina přátel uživatele je výchozí.
- Přání smí mít nejvýše jednu *ne*nulovou vazbu z dvojice zamluvení a splnění.



Obrázek 3.8: Entitní typy a relace analytického doménového modelu

4 Návrh architektury a uživatelského rozhraní

Následující text věnuji návrhu architektury, uživatelského rozhraní a transformaci analytického doménového modelu do návrhového. Tato kapitola poskytuje takovou úroveň detailu, aby bylo možné v další fázi zvolit problémové doméně vyhovující implementaci.

4.1 Návrh architektury komunitního webového systému

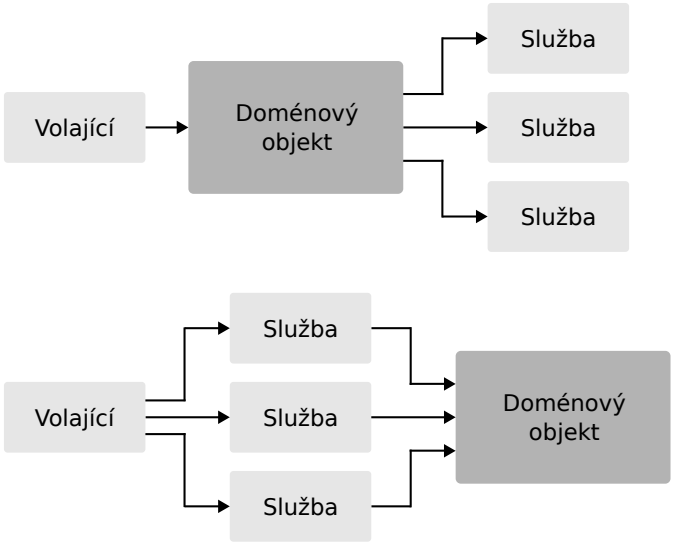
Jak jsem již vysvětlila v úvodu: protože návrhové vzory považuji za *zavedené* abstraktní řešení, zatímco konstrukty zvolených programových rámců se mohou měnit i z pohledu verzí, budu architekturu aplikace modelovat nejprve výhradně výběrem vhodných návrhových vzorů. Volba specifických nástrojů a jejich použití obsáhne kapitola 5.

Struktura aplikace vychází z konceptu popsaném [Fowler, 2003, str. 20] a dodržuje členění na tři klíčové části:

- **prezentační vrstva** obsluhující požadavky uživatelů webového rozhraní,
- **vrstva doménové/*business* logiky** jako funkcionální *srdce* aplikace
- a **zdroj dat**.

4.1.1 Vrstva doménové logiky

[Fowler, 2003] poskytuje několik pohledů na organizaci doménové logiky – v protipólu zde leží především anemický doménový model a tzv. *Rich Domain Model*[1]. Rozdíl v použití každého z nich demonstruje dle [Gupta, 2009] následující schématický obrázek.



Obrázek 4.1: *Bohatý* versus anemický doménový model

[1]Rich Domain Model – *bohatý* doménový model (překlad autorky)

Zatímco instance doménových objektů *bohatého* doménového modelu poskytují kromě *getrů* a *setrů*[2] další metody nebo funkce v duchu konceptu OOP[3], instance objektů anemického doménového modelu je manipulována přes volajícímu poskytované služby.

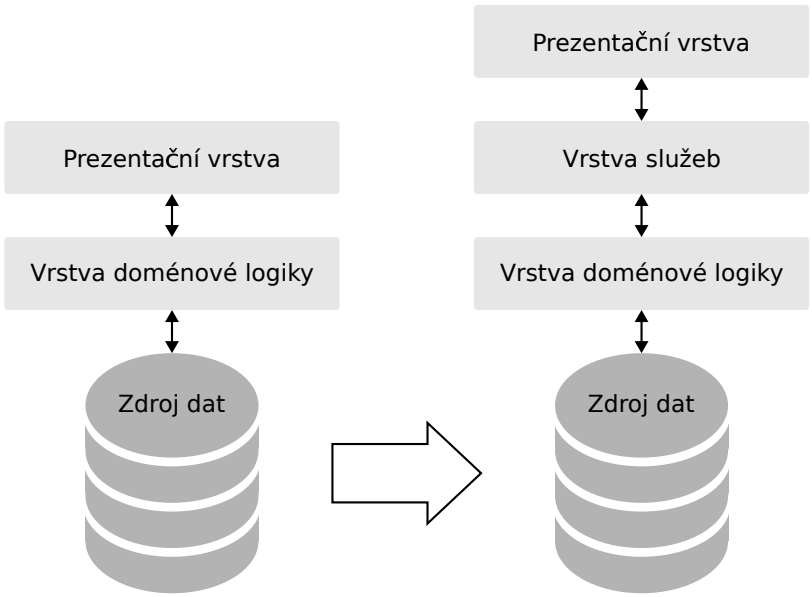
Ve svém návrhu architektury doménové logiky jsem se řídila praxí popsanou [Fowler, 2003, str. 134] a vytyčila si rozdělení zodpovědností za funkčnosti na dvě části:

- **logika doménových objektů** – metody, které mohou atomicky v rámci svého doménového objektu a jeho vazebních vztahů poskytnout požadovanou funkčnost – kupříkladu výpočet věku uživatele, počtu komentářů přání, určení oblíbenosti, stavu splnění či viditelnosti v závislosti na poskytnutých vazbách,
- a **aplikační logika** – komplexní *řídící* metody zahrnující použití více – v doménovém modelu leckdy *vzdálených* nebo přímou relací *ne*propojených – objektů.

Ve chvíli, kdy položím větší důraz na poskytovanou logiku aplikační a budu předpokládat, že logika doménových objektů bude postupována dle potřeby právě skrze ní, stává se tato dle [Cockburn, cit. dle Fowler, 2003, str. 134] v podstatě deklarací rámce poskytovaných služeb aplikace a je vhodné ji použít jako návrhový vzor *Service Layer*.

Z původní třívrstvé architektury je tak vyčleněna vrstva, kterou použiji k zapouzdření implementace aplikační doménové logiky a kontrolu transakcí.

Obrázek 4.2 ukazuje schéma rozdělení vrstvy aplikační doménové logiky na dvě části. Množina aplikací poskytovaných operací může být díky servisní vrstvě přístupná různým typům klientů a uživatelských nebo systémových rozhraní.



Obrázek 4.2: Vyčlenění servisní vrstvy neboli vrstvy služeb z vrstvy doménové logiky

[2]funkce na vyzdvižení resp. nastavení třídních nebo instančních atributů
[3]OOP – objektově orientované programování

4.1.2 Zdroj dat

Vrstva zdroje dat definuje způsob, jakým jsou informace ve vybrané formě persistovány – tj. *trvale* uchovány do svého úložiště – tak, aby bylo uživateli báze dat umožněno k nim spolehlivě přistupovat a provádět nad uloženými daty dotazy.

Návrhem musím nutně vyřešit tři zásadní aspekty architektury zdroje dat:

- způsob persistování dat,
- umístění samotného kódu této interakce s úložištěm v rámci architektury
- a výběr vhodného návrhového vzoru pro organizaci metod persistujících data.

4.1.2.1 Způsob persistování dat

Doménový model mé aplikace reprezentují objekty, zatímco báze dat, do které mají být ukládány jejich jednotlivé instance, zastupuje relační databáze.

Jak uvádí [Keith a Schincariol, 2009, str. 3-4] na nejjednodušším příkladu: již jen atribut, který je ve světě objektů jednoznačně definován svým datovým typem, může být převeden na fyzické médium několika rozdílnými způsoby. S každou vazbou nebo volitelnými atributy lze situaci ještě zkomplikovat. Doménový model může ve výsledku nabývat rozličných fyzických reprezentací v různých úložištích.

Použiji-li implementaci přístupu specifickou výhradně pro jednu z vybraných variant, vznikne nutně závislost kódu na databázi. Změna úložiště by poté mohla způsobit nefunkčnost celé mé aplikace. Z tohoto důvodu musím zvolit dostatečně flexibilní způsob převodu mezi světem objektů a informací v databázi.

Řešením je objektově-relační mapování (ORM), pro které je dle [Fowler, 2003, str. 38] vhodné zužitkovat existující nástroje, oproti vlastní implementaci například ve formě realizace návrhového vzoru *Data Mapper*[4]. Ten si dle [Fowler, 2003, str. 165] taktéž klade za cíl odstiňovat doménový model od implementace dotazů nad databází.

Protože zadání stanovilo použití jazyka Java, zvolila jsem standard *Java Persistence API* (JPA), který – jak uvádí [Keith a Schincariol, 2009, str. 14] – umožní mapování jakéhokoliv *Plain Old Java Object* (POJO) – tj. objektu prostého požadavků jiných než dodržování syntaxe jazyka Java – pomocí anotací, či konfiguračního XML[5] souboru.

Převodu návrhového doménového modelu do fyzického se věnuji v kapitole zaobírající se realizací návrhu 5, konkrétně v sekci 5.3.1. Tamtéž uvedu konkrétní konstrukty standardu JPA, které jsem zužitkovala, a podrobně popíši způsob jejich použití na entitních typech, jejich vzájemných relacích, hierarchii a atributech.

[4] *Data Mapper* – *mapovač* dat (překlad autorky) mezi úložištěm a pamětí aplikace
[5] XML – *Extensible Markup Language* – rozšiřitelný značkovací jazyk pro výměnu a zpracování dat

4.1.2.2 Umístění kódu pro persistování dat v rámci architektury

Nyní, když vím, že využiji pro objektově-relační mapování standard JPA, by bylo vhodné určit, kde bude umístěn samotný kód.

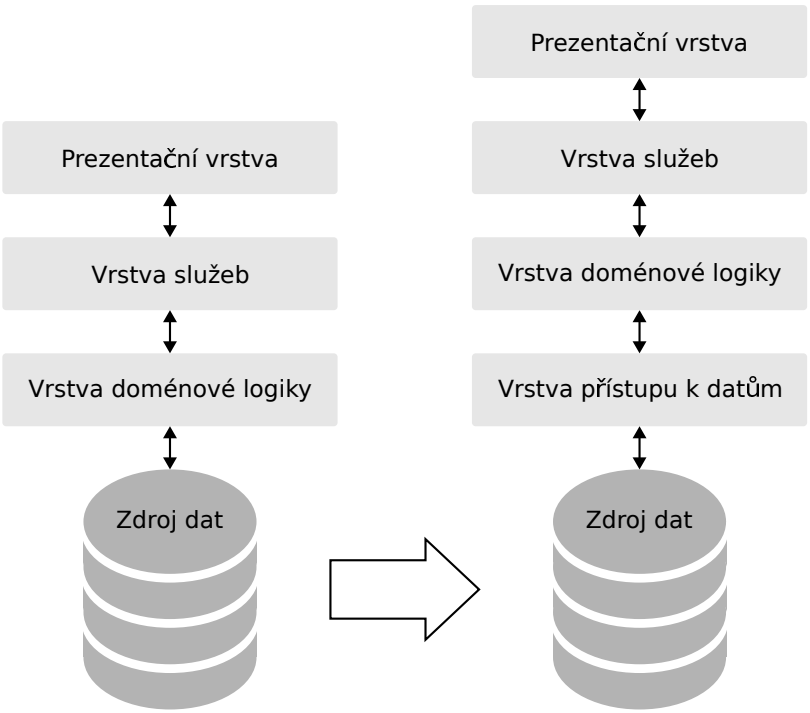
JPA umožňuje umístění přímo na doménových objektech v podobě *Named Queries* – pojmenovaných dotazů (překlad autorky), přičemž je dle [Keith a Schincariol, 2009, str. 185] vybírán takový entitní typ, který co nejvíce odpovídá navracenému výsledku dotazu.

Používání pojmenovaných dotazů může být přístupem rychlým pro vývoj funkčního prototypu aplikace, ale neumožňuje například znovupoužití v jiném projektu, ani zapouzdření kódu pro persistenci dat.

Nejjednodušší formou vyčlenění všech záležitostí týkajících se vyzdvihování a ukládání instancí každého doménového objektu mimo něj je dle [Mička a Kouba, 2013] vytvoření specializovaného objektu pro přístup k datům – *Data Access Object* – zkráceně DAO.

Každý takový DAO by měl příslušnému objektu doménového modelu poskytovat potřebné CRUD[6] operace.

Existence armády rozhraní a implementací DAO objektů pro entitní typy doménového modelu, napovídá logickému oddělení vrstvy přístupu k datům – podobně jako jsem využila oddělení servisní vrstvy pro zapouzdření doménových objektů. Změnu v architektuře v souladu s doporučením zmíněného zdroje demonstruje obrázek 4.3.



Obrázek 4.3: Vyčlenění vrstvy přístupu k datům

[6]CRUD – operace ***C**reate*, ***R**ead*, ***U**pdate*, ***D**elete* neboli vytvoření, čtení, aktualizace a smazání

4.1.2.3 Výběr návrhového vzoru pro organizaci metod persistujících data

[Mička a Kouba, 2013, str. 1-6] poskytují přehlednou analýzu možností návrhu architektury vrstvy přístupu k datům s ohledem na dodržování základních principů návrhu objektově orientovaných softwarových systémů:

- **zapouzdření** pro skrytí implementačních detailů,
- zamezení duplikaci kódu – **princip DRY**[7],
- zamezení požadavků na implementaci, která nebude použita – **princip YAGNI**[8]
- **jedna odpovědnost pro každou třídu**,
- **znovupoužitelnost** vytvořeného kódu pro další projekt,
- a co nejsnazší[9] **testovatelnost**.

Aby byly všechny zmiňované principy dodrženy, představují [Mička a Kouba, 2013, str. 3-6] jako výstup výsledek hledaní – návrhový vzor *DAO Dispatcher* – dispečer objektů pro přístup k datům (překlad autorky).

Použitím *DAO Dispatcher* získám několik způsobů přístupu k datům:

- generický přístup pro všechny entitní typy doménového modelu pomocí dispečera,
- volitelné třídně specifické přístupy dispečerem registrovaných entitních typů,
- volitelné třídně specifické implementace generického přístupu registrovaných tříd.

Konkrétní implementaci vrstvy přístupu k datům pomocí návrhového vzoru *DAO Dispatcher* a s využitím standardu JPA popisuji a blíže vysvětluji v sekci 5.3.6.

4.1.3 Prezentační vrstva

Návrhový vzor, nalézající uplatnění v poslední ze tří klíčových vrstev aplikace – prezentační, je *Model View Controller* (MVC) – model pohled kontrolér (překlad autorky).

Jak uvádí [Fowler, 2003, str. 330]: model reprezentuje informace o doméně, pohled zobrazuje model v uživatelském rozhraní a kontrolér manipuluje model podle změn provedených v pohledu a navrací příslušnou odezvu.

Pohledy budou v navrhované aplikaci tvořeny jednotlivými stránkami a roli kontroléru převezmou třídy obsluhující požadavky uživatele. Model by měly představovat instance doménového modelu. [Mička, a, sn. 10/12-12/12] ovšem upozorňuje na jistá úskalí.

[7] DRY – *Do not repeat yourself* – „neopakuj se“, pojmenováno [Hunt a Thomas, 1999, str. 30]

[8] YAGNI – *You ain't gonna need it* – "nebudeš to potřebovat", dle [Fowler, 2015] autorem K. Beck

[9] uvažuji testovatelnost co nejsnazší z pohledu dostatečného *atomického ohraničení* testovaných subjektů, co nejmenšího počtu potřebných testovacích nástrojů a jednoznačného určení výsledku testování

Používáním entitních typů za hranicí zabezpečené servisní vrstvy mohou být v nejhorším případě odhalena citlivá data získaná z úložiště jakými jsou například šifrovaná hesla, rodná nebo jinak důvěrná čísla a podobně.

[Mička, a, sn. 10/12-12/12] poskytuje řešení v podobě použití instancí *Data Transfer Object* (DTO) – přepravky na data, která:

- umožňuje přenos mezi servisní vrstvou a uživatelským rozhraním
- a obsahuje pouze data a reference na jiná DTO do zvolené hloubky.

Vznikající duplicitu mezi třídami entitních typů a odpovídajících DTO lze dle [Mička, a, sn. 11/12] vyřešit použitím mapování mezi doménovými objekty a DTO.

Ve svém návrhu jsem raději mapování nevyužila z obavy o ztrátu kontroly nad určením hloubky a úrovně provázanosti navracených DTO. Taktéž jsem uvažovala možné odlišnosti DTO a entitních typů, týkajících se jejich atributů nebo vazeb.

[Fowler (2003), str. 401] uvádí vzor DTO jako kombinaci atributů více entitních typů.

Podobný druh přepravek ovšem nepoužiji, neboť má problémová doména neposkytuje možnost vytvořit jednoduchou a na více místech použitelnou kompozici instancí. DTO budou vytvářeny podle požadované úrovně detailu a s vybranými referencemi.

4.2 Návrh webového uživatelského rozhraní

Následujícím textem se budu věnovat aspektům, které ovlivnily návrh uživatelského rozhraní aplikace. Klíčovou roli zaujímá kýžená dvojice použitelnosti a přístupnosti. Prvky, které nejsou nutně technického charakteru a jejichž cílem je spíše zvyšování přívětivosti uživatelského webového rozhraní psychologicky vybírám a popíši v sekci 4.2.2.

4.2.1 Použitelnost a přístupnost navrhované aplikace

Vysoká přístupnost aplikace zaručuje, že i uživatelé s možnými – také technickými – omezeními budou moci využívat obsah aplikace. Použitelnost určuje míru, s jakou lze aplikaci bez velké námahy využívat.

Přístupnost považuji za základ, který bude v navrhované aplikaci dodržen především:

- použitím textových symbolů namísto grafických ikon,
- čitelností písma a jeho dostatečným kontrastem s pozadím,
- označením a vhodným členěním polí tabulek a formulářů
- a alternativními nebo vysvětlujícími texty obrázků nebo složitějších akcí.

Z důvodů přístupnosti a použitelnosti jsem se rozhodla co nejvíce omezit použití skriptů na straně klienta, nepřinášejí-li významnou výhodu. Aktivní formulářová kontrola poskytovaná HTML5[10] nesmí být v prezentační vrstvě jedinou validací vstupu uživatele.

Aby byla webová stránka použitelná i na zmenšených obrazovkách a napříč prohlížeči, zvolila jsem systém *Grid View* – mřížky, popsaný [w3schools.com].

Každá stránka bude pomyslně rozdělena na řadu sloupců stejné velikosti s elementy odpovídajícími šířce vybraného počtu sloupců mřížky. Velikosti jednotlivých částí stránky určí CSS[11] styl. Umístění elementů webového dokumentu v sekvenci *krabiček* mřížky zaručí správné poskládání prezentace i na zmenšených obrazovkách. Pomocí CSS bude možné také podle potřeb cílového zařízení nastavit šířku všech elementů na maximum a tím webovou stránkou simulovat aplikaci určenou přímo pro telefon.

Abych zamezila deformaci ovládacích prvků zmenšováním nebo zvětšováním webové prezentace vyvarovala jsem se v návrhu uživatelského rozhraní použití obrázků jako pozadí a nahradila například grafiku tlačítek CSS styly.

4.2.2 Přívětivost grafického návrhu uživatelského rozhraní

Při návrhu uživatelského rozhraní webové aplikace jsem čerpala zajímavé rady z publikace [Anderson, 2012] a mezi zohledněné aspekty zařadila následující *doporučení*:

- vysvětlit uživateli výhody nabízených akcí nebo důvody pro poskytnutí informací,
- význam ovládacích prvků podpořit odpovídající barvou, tvarem nebo symbolem,
- pro zobrazení složitější nebo obsáhlé struktury dat či akcí zvolit lehce zapamatovatelný, ale nikoliv přímočarý případně hierarchický systém grafických symbolů,
- nezatěžovat uživatele dlouhými a nestrukturovanými formuláři,
- nevolit výrazné barvy sousedících tlačítek, má-li být jen jedna akce preferovaná
- a umožnit uživateli zobrazení dalších informací jako volbu, nikoliv povinnost.

Uložila jsem si za cíl navrhnout rozhraní tak, aby uživatel nemusel dlouze přemýšlet, kterou akci hledá, ale mohl se spolehnout na její vhodné namapování na barvu nebo grafickou reprezentaci.

Symboly jsem zvolila nejen pro akce, ale také jako rychle čitelné indikátory zamluvení či viditelnosti přáním přátelům jeho majitele. Vybírala jsem ze *základních* a široce podporovaných sad, takže systém, který vznikl není přímočarý, ale tím zajímavější.

Protože považuji za důležitý aspekt *soutěžení* mezi přáteli „kdo splní víc přání“, podrobila jsem zvolenému systému indikátorů také barvy. Kupříkladu uživatel, který splní přání přítele, vidí u takového dárku motivačně zlatou korunku.

[10] HTML5 – značkovací jazyk pro tvorbu struktury a obsahu webové prezentace, verze pět
[11] CSS – *Cascading Style Sheets* – značkovací jazyk pro popis vzhledu webové prezentace

Pro akce, od kterých chci uživatele odradit jsem záměrně zvolila výstražně červenou barvu a agresivní symbol, aby uživatel zvážil, zda je chce skutečně provést. Za *nebezpečné* akce hodné varování považuji například smazání upravovaného objektu nadobro, zrušení přátelství, nebo zrušení zamluvení přání.

Příklady návrhu dosažení mapování akce na barvu a symbol ukazuje následující tabulka.

Symbol	Popis symbolu	Barva	Typ akce, nebo kýžený dojem
✓	Odškrtnutí	Zelená	Vítaná akce uložení nebo odeslání
⚠	Výstraha	Červená	Upozornění před akcí
☠	Lebka s hnáty	Červená	Nebezpečná akce, od které chci odradit
↻	Stočená šipka	Modrá	Akce způsobí úpravu a opětovné zobrazení
✗	Křížek	Šedá	Zrušení úprav bez uložení a navrácení zpět
→	Šipka vpravo	Fialová	Akce způsobí přesun na jinou stránku
✎	Tužka	Odkazů	Úprava předmětu, ke kterému je připojena
▲▼	Trojúhelníky	Oranžová	Rozšíření nebo skrytí částí stránky

Tabulka 4.1: Mapování akcí uživatelského rozhraní na symboly a barvy

4.3 Návrhový doménový model

V souladu s [Arlow a Neustadt, 2008, str. 343] mnou vytvořený návrhový doménový model poskytuje oproti analytickému dostatečný detail, jakým je:

- určení viditelnosti atributů v souladu s popisem [Arlow a Neustadt, 2008, str. 155],
- přesná specifikace názvů a datových typů s ohledem na vybraný jazyk Java,
- definice počátečních hodnot atributů, je-li to vyžadováno,
- převod operací objektů analytického modelu na konkrétní metody
- a oddělení nadřazeného entitního typu `Entita` pro potřebu persistence dat.

Protože byla doména již popsána analytickým doménovým modelem v sekci 3.3 a taktéž z důvodu prostorové náročnosti připojuji diagramy k textu v podobě přílohy B.

Začlenění diagramem poskytnutého pohledu na model problémové domény do celé aplikace bude popsáno v textu kapitoly 5, sekci 5.2.

5 Výběr technologií a implementační detaily

Cílem následujících sekcí je popsat a přiřadit vybraným návrhovým vzorům a standardům z kapitoly 4 konkrétní implementace. První částí 5.1 charakterizuji vybrané technologie a rámce. Obecně definovaná architektura aplikace nabude reprezentace popisem a diagramem balíčků v sekci 5.2. Vybrané implementační detaily využívající konstruktů zvolených technologií uvedu sekcí 5.3.

5.1 Charakteristiky vybraných technologií a rámců

Pro realizaci své aplikace jsem zvolila podle doporučení uvedených [Minter, 2008] následující nástroje. Vybrané konstrukty jako anotace, značky nebo konfigurace těchto nástrojů včetně jejich použití uvedu v sekci 5.3.

5.1.1 Aplikační rámec Spring

Jako vybraný rámec, na kterém bude aplikace založena, jsem zvolila rámec Spring, protože dle [Mička, b, sn. 3/18 a 8/18] poskytuje následující výhody:

- nevyžaduje použití aplikačního serveru,
- je otevřený použití jiných rámců a knihoven,
- spravuje a případně také řídí životnost každého vhodně označeného POJO[1]
- a poskytuje *Dependency Injection* (DI)[2] jako formu *Inversion of Control* (IoC)[3].

Rámec Spring bude prostupovat napříč vrstvami aplikace a kromě zmíněných výhod v nich poskytne především zabezpečení, jehož implementaci je věnována sekce 5.3.4.

5.1.2 Projekt řídící nástroj Maven

Protože rámec Spring umožňuje zakomponování velkého množství knihoven, roste tak i počet souborů potřebných k *postavení* aplikace. Každý další balíček nebo jen změna verze v kódu zakomponovaného zdroje je časově náročná a správa nepřehledná.

Proto [Minter, 2008, str. 11 a 19] doporučuje použití nástroje Maven umožňujícího automatickou kontrolu přítomnosti všech definovaných knihoven a zajišťujícího jejich dostupnost a použití pro postavení aplikace. Sama jsem se při implementaci přesvědčila, že přidání nových knihoven je ve vývojovém prostředí NetBeans s doporučeným nástrojem Maven skutečně rychlé a pohodlné.

[1] POJO –*Plain Old Java Object*; objekt prostý požadavků jiných než dodržování syntaxe jazyka Java

[2] *Dependency Injection* – injektování závislostí – tj. správných implementací instancemi požadovaných zdrojů – a jejich životnost je řízena rámcem Spring

[3] *Inversion of Control* – převrácené řízení – aplikace je v pozici knihovny řízené rámcem v duchu Hollywoodského principu *„Don't call us, we'll call you.“* – „Nevolejte nám, my se vám ozveme.“

5.1.3 JPA/Hibernate

Výhody standardu JPA zahrnula sekce 4.1.2 věnovaná zdroji dat. Rámec Hibernate je implementací specifikace umožňující použití typově bezpečných dotazů – *kritérií*.

5.1.4 AOP – Spring AOP/AspectJ

Ačkoliv doménový model ctí objektově orientovaný návrh, mohou vznikat další požadavky, které nejsou objektového charakteru a/nebo prostupují více vrstev, často se opakují, ale není jim vyhrazeno jedno konkrétní místo. Odpovídajícím problémem může být dle [Minter, 2008, str. 86] například bezpečnost nebo ukládání záznamů o vykonání metod. Řešením se zabývá programování orientované na aspekty (AOP).

Podporu AOP rámcem Spring uplatním při řešení opakujících se úkonů, jakými je kontrola přístupu uživatele k požadovaným informacím a úprava navracené množiny dat v závislosti na vztahu k uživateli. Oddělením těchto aspektů mimo doménovou logiku, je zabráněno redundanci v tělech AOP *zachycovaných* metod servisní vrstvy.

5.1.5 Apache Tomcat

Obsluhu požadavků webových klientů bude podle doporučení [Minter, 2008, str. 14] provádět kontejner Apache Tomcat, jehož existence rovněž umožňuje použití JSF (níže).

5.1.6 Derby

Dle doporučení [Keith a Schincariol, 2009, str. 443] budu pro vývoj systému používat nenáročnou vestavěnou databázi Apache Derby. Informace o nastavení uvádím v 5.3.7.

5.1.7 Java Server Faces

Java Server Faces (JSF) bude nejdůležitějším rámcem prezentační vrstvy. Poskytne validaci na straně serveru, navigaci a tvorbu šablon stránek.

5.1.8 Omnifaces

Omnifaces jsou taktéž užitečným nástrojem prezenční vrstvy a dle titulku [Omnifaces] je jejich cílem „ulehčit život s JSF“ (překlad autorky). Výhodnou vlastností, kterou zakomponuji ve své aplikaci je především poskytnutí automatického převodu mezi objekty a jejich zvolenou textovou reprezentací – přetížením metody tisku – ve výběrech položek formulářů. Bez použití Omnifaces by bylo nutné implementovat vlastní konvertory. Od verze 2 vyžadují instalaci CDI – podpory IoC a DI (popsáno 5.1.1) *Java Enterprise Edition* aplikačních serverů, což je vzhledem k použití Spring nevhodné.

5.1.9 Primefaces

Primefaces podporují vývoj aplikace *bohaté* na straně klienta. V souladu s návrhem uživatelského rozhraní 4.2, bude použit pouze uživatelsky přívětivý textový editor.

5.2 Návrh architektury s použitím zvolených technologií

Vícevrstvou architekturu aplikace, jak jsem ji definovala v sekci 4.1 zde se znalostí konkrétních technologií vytříbím do modelu graficky reprezentovaného diagramem balíčků 5.2.11. Jednotlivé části popíši z pohledu role, kterou zaujímají v rámci celé aplikace, a jejich vzájemné komunikace. Detailní obsah a implementační detaily uvedu pro přehlednost ihned v následující sekci 5.3.

5.2.1 Webové uživatelské rozhraní – balíček `webapp`

Tento balíček vytváří a nastavuje tvář aplikace. Obsahuje především konfigurační soubory rámců, které budou použity pro tvorbu webového systému a celkový obsah prezentační vrstvy. Webové stránky používají rámcem Spring řízené *backing beany* jako nosiče zobrazovaných dat.

5.2.2 Spring *backing beany* – balíček `back`

Beanám jsou přiřazeny různé životnosti a podle nich slouží vybraným účelům. Zpravidla ale uchovávají potřebně dlouho zobrazovaná data a provádějí uživatelem webového rozhraní požadované akce. Informace uskladňují pomocí instancí souvisejících DTO[4], které slouží jako přepravky. *Beany* řízené rámcem Spring využívají servisní vrstvu pro zprostředkovanou úpravu doménového modelu a případné navrácení aktualizovaných dat.

5.2.3 Vrstva služeb – balíček `service`

Servisní vrstva, jejímuž návrhu se věnovala sekce 4.1.1, zapouzdřuje doménový model a vykonává aplikační logiku. Skládá se z rozhraní, které deklaruje poskytované služby a implementace, která je realizuje. Používá vrstvu přístupu k datům pod sebou, aby provedla potřebné úkony. V balíčku nalezne uplatnění rámec Spring.

5.2.4 Vrstva přístupu k datům – balíček `dao`

Zde nalezne uplatnění návrhový vzor *DAO Dispatcher* obecněji popsaný v sekci 4.1.2.3. Třídy balíčku používají doménový model pro manipulaci a vykonávání dotazů nad daty. Aby kód zamezil zneužití záměrně škodlivým příkazem, bude nutno použít implementace *kritérií* standardu JPA. Implementaci vrstvy podrobně popíši v sekci 5.3.6.

5.2.5 Persistované domény modelu – balíček `model`

Balíček obsahuje třídy, které představují entitní typy doménového modelu. Protože jsem v sekci 4.1.3 na základě informačních zdrojů vyloučila použití instancí entitních typů za úrovní servisní vrstvy, jsou potřebné instance doménového modelu transformovány do odpovídajících DTO. Stejně jako ve vrstvě přístupu k datům i zde je využito implementace standardu JPA. Převodu modelu do fyzické reprezentace věnuji sekci 5.3.1.

[4] *Data Transfer Object* – návrhový vzor objektu přepravky na data více popsaný v sekci 4.1.3

5.2.6 Přepravky na data – balíček `dto`

Jak jsem již vysvětlila v návrhu prezentační vrstvy – 4.1.3, instance DTO tříd slouží výhradně pro uchování a přenos dat mezi servisní vrstvou a uživatelským rozhraním. Svou podstatou představují DTO přepravky na data užívané oběma vrstvami. Jejich existence bude vázána na životnost přiřazenou *backing beanám*.

5.2.7 Platnost Spring *bean* na míru – `scope`

Vznik tohoto balíčku podmínila potřeba implementace Springem řízených *backing bean* s životností přesně vymezenou jedním pohledem, tedy webovou stránkou. Vytvořená třída bude používána rámcem Spring při správě *bean* prezenční vrstvy. Podrobný rozbor problematiky a jejího řešení uvádím v sekci 5.3.2.

5.2.8 Třídy AOP – balíček `aop`

Sekce 5.1.4 seznamuje s důvody použití nástrojů AOP. V této části jsou umístěny veškeré třídy, které potřebné funkčnosti zabezpečují. V mém návrhu jsou AOP metody *nasazeny* výhradně na metody servisní vrstvy.

5.2.9 Poskytovatel kódování hesla – balíček `provider`

Balíček obsahuje rozhraní a implementaci funkce kódování hesla. Zapouzdřením třídy, která bude v navrhované aplikaci poskytovat šifrovací algoritmus, lze v budoucnu zvolený BCrypt vyměnit za libovolný jiný způsob kódování.

5.2.10 Výčtové typy – balíček `enums`

Třídy v balíčku reprezentují neměnné výčtové typy, které jsou použity napříč aplikací. Enumerace tvoří atributy entitních typů doménového modelu, DTO i *backing bean*. Některé slouží výhradně prezentační vrstvě.

Literály nejsou přímo prezentovány uživateli, ale obvykle slouží jako klíče pro poskytnutí odpovídajícího textu, případně i vybraného jazyka překladu.

Instance výčtových typů nejsou na rozdíl od instancí modelu, které jsou ukládány spolu s odkazy nebo hodnotami konkrétních literálů enumerace, samostatně persistovány.

Důvodem rozhodnutí byl předpoklad příliš častého dotazování na tyto jinak konstantní hodnoty. Tento způsob může být v jistých případech nebezpečný – dle zvoleného způsobu ukládání atributů typu enumerace, může dojít ke ztrátě konzistence nebo chybě či významové změně interpretace uložených dat.

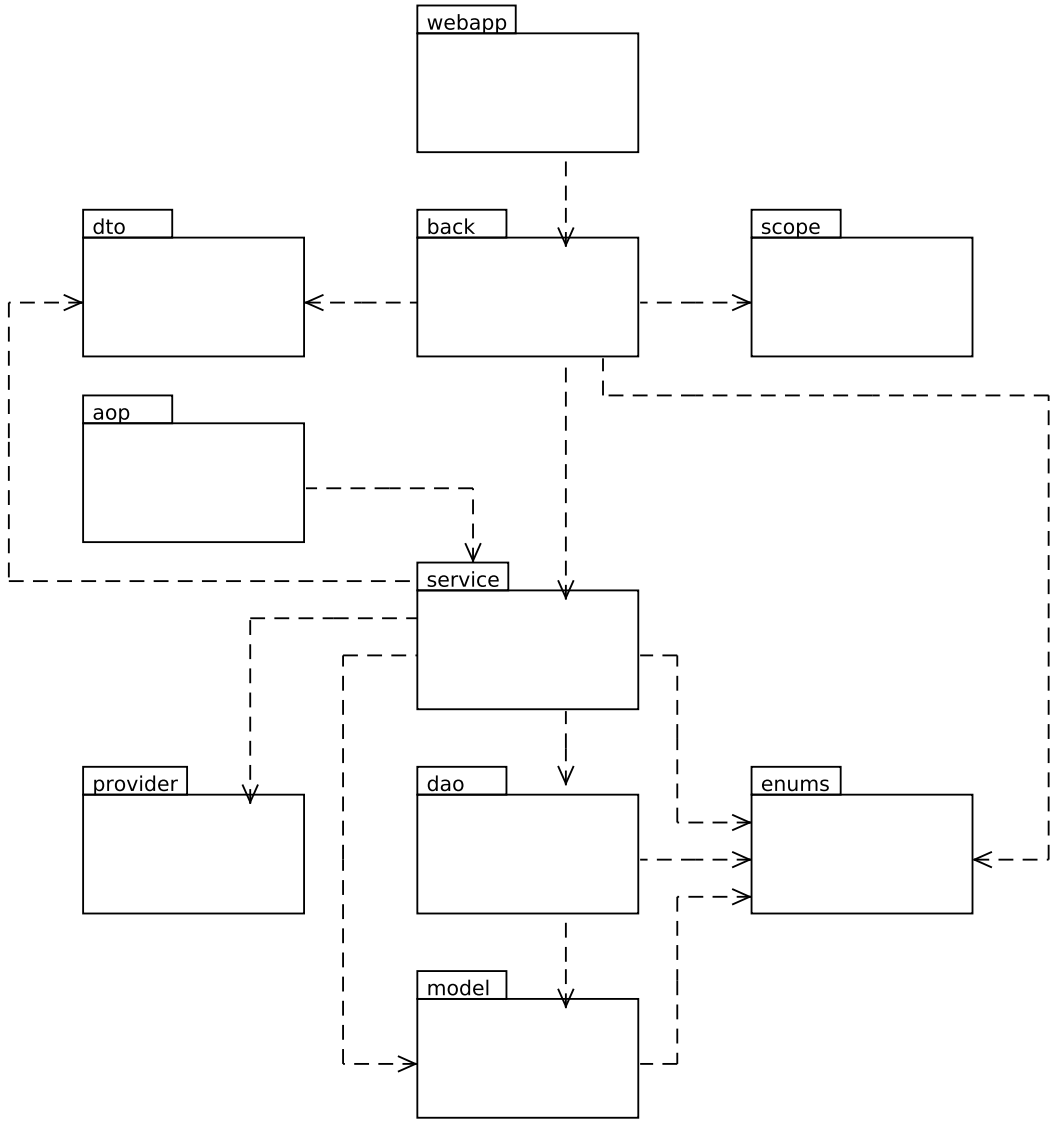
Přesto jsem se rozhodla takto postupovat, neboť jsou v případě mé problémové domény výčtové typy zpravidla představovány zavedenými textovými označeními. V takovém případě lze pohodlně přidávat nové položky a/nebo měnit pořadí stávajících. Přejmenování literálu by jistě mohlo potíže způsobit, proto navrhuji v takové situaci modifikovat pouze odpovídající jazykovou reprezentaci.

V několika případech jsem považovala za vhodné použít persistování pomocí číselných hodnot procházejících po jednotlivých položkách výčtu. Literály zde ovšem v podstatě odpovídají svému obsahu a seřazení je kýžené. Protože by případné rozšíření přidáním nového literálu pouze zvětšilo číselnou škálu, nepovažuji tuto situaci za problém. Podrobný popis převodu do fyzické reprezentace uvádím v sekci 5.3.1.

5.2.11 Návrh architektury reprezentovaný diagramem balíčků

Právě popsanou strukturu a komunikaci mezi balíčky – často reprezentujícími celé vrstvy – demonstruje diagram návrhu architektury 5.1.

Vícevrstvá architektura prosazuje zásadu volání vždy jen přímé nižší vrstvy. Protože jsou ovšem instance modelu použity jak vrstvou přístupu k datům, tak servisní, dodržuje návrh mé aplikace dle [Mička, a, sn. 5/12] *relaxovanou* vícevrstvou architekturu.



Obrázek 5.1: Architektura systému v podobě diagramu balíčků

5.3 Realizace vybraných aspektů a vrstev aplikace

Návrh zakončený popisem architektury tvoří základ pro implementaci. Následujícím textem popíši způsob použití konstruktů, případně nastavení zvolených technologií a vybraná řešení dílčích úkolů, kterými jsem se během realizace návrhu zabývala.

5.3.1 Převod návrhového modelu do fyzického

Jak jsem podrobněji rozvedla v *obecněji* zaměřené části návrhu – sekci 4.1.2 – a po upřesnění zvolené technologické implementace mě čeká úkol použít konkrétní anotace standardu JPA, implementovaného rámcem Hibernate, k určení způsobu převodu návrhového doménového modelu do jeho fyzické reprezentace v úložišti.

Anotacemi určím objektově-relační mapování klíčových prvků návrhu modelu:

- instancí entitních typů a jejich vzájemné hierarchie,
- relací s různými násobnostmi a (ne)povinnostmi účasti objektů
- a instančních atributů s ohledem na jejich (ne)povinnost.

Pro správné mapování vlastních atributů entitních typů jsem jako referenční příručku použila knihu [Keith a Schincariol, 2009] a v zásadě si vystačila s:

- parametrem dovolené nulovosti atributu `nullable` anotace `@Column`,
- parametrem jména `name` sloupce v tabulce anotace `@Column`,
- parametrem jedinečnosti `unique` záznamu v tabulce anotace `@Column`,
- parametrem identifikátoru `id` anotace `@Column`,
- typem odpovídajícím typu atributu nesoucího časový údaj s anotací `@Temporal`
- a výběrem způsobu uložení atributu reprezentovaného instancí výčtového typu `EnumType.STRING` nebo `EnumType.ORDINAL`.

U poslední položky seznamu bych se ráda pozastavila. V sekci návrhu architektury vyčleněné výčtovým typům – 5.2.10 – jsem se věnovala způsobu ukládání atributů typu enumerace. Zvolenému přístupu zde odpovídá výběr způsobu mapování. Pro instance, jejichž jméno je důležitější než pořadí a je možné, že budou přibývat, jsem zvolila typ `EnumType.STRING`. V případě mapování seznamu sekvence číselných údajů jsem – v souladu s návrhem – s klidným svědomím vybrala `EnumType.ORDINAL`.

V seznamu taktéž uvedený atribut jedinečnosti jsem použila při převodu pouze jedenkrát – u emailové adresy sloužící jako unikátní přihlašovací údaj.

Mapování integritním omezením, které by zahrnovaly více entitních typů zároveň jsem se z obavy z příliš silného omezení vyhnula a kontrola konzistence ukládaných dat, které by jinak mohly mít složené klíče, je přenechána aplikaci.

Jednalo se například o komplikovanější vazbu přátelství mezi uživateli, zahrnující vzájemnou vazbu několika existenčně závislých entitních typů. Zaručení, že přítel uživatele bude přidán jen jedenkrát a při přesunu bude vždy právě v jedné skupině přátel, závisí na provedení procesů aplikační logikou. Taktéž kontrola existence vždy pouze jedné výchozí skupiny přátel a jedné výchozí skupiny přání pro každého uživatele není provedena objektově-relačním mapováním.

Protože uživatelé systému nabývají dvou rozdílných rolí, musela jsem vyřešit způsob, jakým budou uloženi v databázi. Již ve fázi analýzy bylo jasné, že jak uživatel, tak administrátor budou potomci uživatele systému.

Z pohledu návrhu databázového systému by jistě byla vhodná varianta, kdy každé roli odpovídá jedna obvykle zcela zaplněná tabulka.

Dotazován je ale také uživatel systému bez ohledu na roli, určila jsem tedy, že zaštiťuje jak uživatele, tak administrátora nejen v objektovém návrhu, ale i v úložišti. Ačkoliv jsem pro přístup k datům použila objektového dotazování pomocí *kritérií*, které umožňují snadné slučování tabulek, v případě změny způsobu obstarávání dat by sjednocená tabulka byla jistě navazujícímu programátorovi výhodou.

Způsob mapování, který jsem právě zvolila, využívá ve třídě uživatele systému:

- parametr jména `name` anotace `@DiscriminatorColumn` – sloupec který budu dotazovat pro zjištění role uživatele systému,
- parametr datového typu označení role `discriminatorType` s nastavením textové reprezentace (číselná reprezentace by byla nevhodná)
- a hodnotu parametru `strategy` anotace `@Inheritance` nastavenou na variantu `InheritanceType.SINGLE_TABLE` – mapování potomků do jedné třídy.

Nastavení doplňuji – v případě dřívějšího výběru parametru `@DiscriminatorColum( discriminatorType = DiscriminatorType.STRING)` nepovinným – umístěním anotace `@DiscriminatorValue` s označením role ve sloupcích tabulky uživatelů systému.

Pro způsob, jakým budou mapovány vztahy mezi jednotlivými entitními typy a jak se budou v budoucnu vzájemně ovlivňovat, mi posloužil přehled [Mihalcea, 2015].

Názvy anotací vztahů odpovídají svým vzorům, přičemž nejčastěji jsem použila parametry uvedené v následujícím seznamu:

- parametr `cascadeType`, jehož hodnota určí, do jaké míry akce provedená na instanci ovlivní druhý konec vazby,
- parametr `orphanRemoval`, který zajistí odstranění instance, jakmile ztratí odkaz na druhý konec vazby, a znemožní taktéž změnu reference
- a parametr `mappedBy` určující název instance vlastnící vazbu.

Z důvodu prostorové náročnosti uvádí seznam vztahů mezi entitními typy příloha C.

5.3.2 Určení životnosti rámcem Spring řízených *backing bean*

Důvod a způsob použití *backing bean* popisuji v sekci 5.2.2. V následujícím textu se budu věnovat výběru jejich vhodné životnosti.

Ve své aplikaci jsem vybrala z rámcem Spring – dle [Mook Kim, 2012a] poskytnutého popisu možností řízení životnosti instancí – *backing bean*, následující:

- `request` – nová instance pro každý HTTP požadavek,
- `session` – nová instance pro každou HTTP sešnu[5]
- a `application` – jedna instance pro celou webovou aplikaci – dle [Johnson a kol.].

První druh životnosti jsem použila pro *beany*, od nichž v aplikaci žádám během jejich života pouze jednorázovou akci. Příkladem jsou například funkce pro získání přihlašovacího *řetězce*[6] uživatele.

Životnost `session` jsem zvolila například pro uchování uživatelských preferencí, tedy informací, které jsou specifické každému uživateli zvlášť.

Poslední typ nalezl uplatnění v poskytování neměnných hodnot sdílených mezi uživateli. Kupříkladu seznam výčtu položek enumerací pro výběr položek ve formuláři.

Abych zaručila, že uživatel bude moci například upravovat několik přání současně, nemohla jsem pro uchování zobrazovaných dat použít *beany* s životností `session`. Tento způsob by také mohl snadněji způsobit neaktuálnost zobrazovaných dat například v seznamech. Mým cílem bylo vytvořit uživatelsky přívětivou aplikaci. Přiklonila jsem se tedy k variantě `request`.

Záhy jsem ovšem zjistila, že pro stránky s komplexnějšími formuláři bych ráda uchovala informace mezi jednotlivými kontrolami. Ačkoliv jsem pro validaci vstupních dat formulářů použila kromě JSF také jeho podporu HTML5 s okamžitou kontrolou na polích, chtěla jsem se vyhnout závislosti na validaci HTML5.

Hledaným typem životnosti se po prostudování [Scholtz, 2011] ukázala varianta životnosti `@ViewScoped` – *beana* zůstává naživu, dokud jsou na ní volány metody navracející `null` nebo `void`. Protože jinak JSF využívá navraceného řetězce metod *bean* pro navigaci, znamená to také, že *beana* zůstává naživu, dokud nedojde k přesměrování.

Rámec Spring vybranou variantu neposkytuje, proto jsem využila návodu [Civici, 2010] a vytvořila třídu, která implementuje vlastní životnost *backing bean* na míru.

Anotaci životnosti `@Scope("view")` jsem použila na všech *backing beanách*, které si mají pamatovat parametry zobrazení či data až do změny webové stránky.

[5] objekt uchovávající informace na straně serveru
[6] v mém návrhu je přihlašovací *jméno* reprezentováno emailovou adresou

5.3.3 Validace a konverze formulářových vstupů prezenční vrstvy

Abych zaručila, že vstupy budou odpovídat kýženému formátu, použila jsem vybrané parametry JSF elementů webové stránky a jejich podporu HTML5.

Pro povinná pole formulářů jsem specifikovala parametr `required = true` a určila odpovídající text chybové zprávy v atributu `requiredMessage`. Ten je zobrazován v JSF elementu `message` pro konkrétní pole nebo `messages` pro celý formulář.

Stejné parametry jsem uvedla také ve formě určené pro validaci HTML5. Pomocí takzvaných `passThroughAttributes` jsem nastavila požadované chování a chybovou zprávu. Ta se uživateli zobrazí ještě před odesláním ke zpracování JSF. V případě chybějící podpory HTML5 je zobrazen můj vlastní přepis chybové zprávy rámce JSF.

Aby byla informace o nutnosti vyplnění vždy dobře přístupná, umístila jsem ji také do titulku `title` daného pole. Již při najetí myši se uživateli zobrazí tato jakási nápověda.

Podobně jako parametr povinnosti vyplnění vybraného formulářového pole a určení odpovídající zprávy lze zvolit validátor a text zprávy oznamující chybu validace. Analogicky k atributu pole `required` použiji v takovém případě parametr `validator` a související `validatorMessage`, jedná-li se o vlastní implementaci validátoru. Zpravidla jsem si vystačila použitím definovaných značek JSF:

- na kontrolu délky `validateLength` pro určení maxima pomocí atributu `maximum`
- a určení formátu řetězce `validateRegex` zadáním výrazu předlohy `pattern`.

Specifickou zprávu kontroly formátu čísel jsem nastavila pomocí `converterMessage`.

Aby byla ve většině případů či při rozšiřování systému zachována konzistence zpráv validace nebo automatické konverze napříč aplikací, doplnila jsem dle návodu [Mook Kim, 2012b] soubory klíčů JSF chyb validace a konverze a odpovídajících textových zpráv. Soubory existují stejně jako překlady ve verzích podporovaných jazyků aplikace – v jazyce českém a anglickém.

Konverze instancí doménových objektů na textovou reprezentaci a zpět – jen pro potřebu použití ve formulářových polích výběru – může být náročná.

Vlastní validátory pro jednoduše textově reprezentovatelné entitní typy tvoří poměrně dost kódu. Na každý vlastní implementovaný validátor by muselo být u každého pole správně odkazováno. Mohlo by také platit: kolik entitních typů, tolik konvertorů. Vytváření velkého množství validátorů jsem se chtěla vyhnout.

Řešení mi poskytl rámec Omnifaces, popsaný již dříve v sekci 5.1.8.

Ve své aplikaci jsem tak nakonec využila `SelectItemsConverter` – konvertor pro pole výběru. Jedinou podmínkou pro použití je přetížení metody tisku tříd entitních typů, které budou konvertovány. Následně jsem u všech polí výběru potřebných konvertoru bez ohledu na obsah zapsala `converter="omnifaces.SelectItemsConverter"`.

5.3.4 Zabezpečení aplikace rámcem Spring a AspectJ

Primárním poskytovatelem kontroly přístupu ke zdrojům je bezpečnostní knihovna rámce Spring. Při výběru konkrétních způsobů jsem využívala možnosti uvedené knihou [Minter, 2008] a zabezpečení provedla na více úrovních:

- zabráněním přístupu ke stránkám pomocí nastavení značek `intercept-url`,
- skrytím vybraného obsahu stránek v uživatelském rozhraní pomocí značek bezpečnostní knihovny rámce Spring, případně kontrolou JSF,
- v servisní vrstvě pomocí anotace `@Secured`
- a třídami implementujícími kontrolu programováním zaměřeným na aspekty.

Nejhrubší verzí zabezpečení je omezení přístupu na konkrétní webové stránky pomocí deklarace `intercept-url` s vybraným oprávněním. Zajímavostí ovšem je, že i nepřihlášenému uživateli musí být nastaven přístup ke zvoleným stránkám pomocí `IS_AUTHENTICATED_ANONYMOUSLY`.

Přesnějším způsobem cílení bezpečnosti se jeví zabezpečení jednotlivých metod aplikační logiky. V mém případě se prvotně jedná o metody servisní vrstvy.

V rozhraní deklarující servisní vrstvu jsem podle potřeby a dle určení případů užití (uvádí sekce 3.2) použila nastavení:

- `@Secured("ROLE_admin")`, `@Secured("ROLE_uzivatel")` pro kontrolu přístupu k vybrané metodě servisní vrstvy uživatelem systému s příslušnou rolí, případně `@Secured("ROLE_admin", "ROLE_uzivatel")` pro kohokoliv registrovaného
- a `@Transactional(readOnly = true)` k zabránění zápisu dat do úložiště.

O kontrolu přístupu k metodám servisní vrstvy, na kterou nestačily předešlé konstrukty, protože jsou role příliš obecné, jsem použila dodatečné třídy AOP pomocí anotací rámce AspectJ. Důvody využití jsem již popsala dříve v textu sekce 5.1.4.

Nasazením anotace na vybranou metodu, například upravím výslednou hodnotu navraceného objektu, nebo zkontroluji souhlasící identifikátor přihlášeného uživatele a vlastníka dotazované instance entitního typu.

Bezpečnost servisní vrstvy je doplněna vybranými značkami JSF a bezpečnostní knihovny rámce Spring. Stejně, jako jsem použila výběr rolí pro povolení přístupu k metodám, podmíním v uživatelském rozhraní vykreslení vybraných částí díky:

- uvedení podmínky zobrazení JSF elementu v parametru `rendered`
- nebo obsáhnutí celé sekce ve dvojici bezpečnostních značek rámce Spring – mohu například povolit přístup s libovolnou ze zadaných rolí pomocí `ifAnyGranted roles="ROLE_admin, ROLE_uzivatel"`.

Zabezpečení servisní vrstvy jsem vyhodnotila jako nejdůležitější a tedy jsou i metody pro navracení dat opatřeny anotacemi, případně je doplňují AOP bezpečnostní metody. Použitím `@Secured` bude automaticky zkontrolována každá stránka, která metodu servisní vrstvy zavolá a případně návštěvníka přesměruje na chybovou stránku.

Nastavení bezpečnosti aplikace rámcem Spring umožňuje přesměrování na chybovou stránku libovolně dle zadaného kódu chyby, nebo typu programové výjimky, čehož jsem využila v metodách bezpečnostních AOP při *vyhození* výjimky pro neoprávněný přístup k datům, nebo pro upozornění na nedostupný zdroj.

Pro šifrování hesla jsem zvolila vestavěnou knihovnu rámce Spring implementující algoritmus BCrypt a nastavila jej také jako způsob ověření přihlašovacích údajů podle návodu popsaného [Mook Kim, 2014].

5.3.5 Nastavení konfiguračních souborů webové aplikace

K vytvoření projektu spravované nástrojem Maven, s využitím všech zmiňovaných rámců bylo třeba zadat konfiguraci následujícím souborům:

- `pom.xml` pro nastavení typu projektu a použitých knihoven pro vystavění aplikace nástrojem Maven (popsán v sekci 5.1.2)
- `applicationContext-security.xml` sloužící k určení způsobu ověřování přihlašovacích údajů, rolí pro přístup k webovým stránkám, poskytovatele implementace přihlašovací a odhlašovací funkce a deklaraci knihovny šifrovacího algoritmu,
- `applicationContext.xml` pro potřebu umožnění použití vlastní implementace životnosti *backing beany*, anotací, AspectJ, JPA mapování a přístupu do úložiště.
- `faces-config.xml` pro konfiguraci rámce JSF – konkrétně nastavení možnosti použití *backing bean* řízených rámcem Spring v JSF stránkách, určení podporovaných jazykových verzí aplikace, umístění a přístupu jejich odpovídajících překladů a nastavení navigačních pravidel
- a `web.xml`, kde je uvedena cesta ke konfiguračním souborům rámce Spring, údaje potřebné pro obsluhu HTTP dotazů, nastavení bezpečnostního filtru rámce Spring, hlavní stránky webové aplikace, doby vypršení uživatelského přihlášení a cílů pro zvolené chybové kódy a programové výjimky.

Při konfiguraci jsem vycházela z informací knih [Johnson a kol.] a [Saleh a kol., 2013]. Pro výběr vzájemně kompatibilních verzí knihoven rámců potřebných pro vystavení aplikace nástrojem Maven mi pomohly informace uvedené [Mička a Šmíd, 2013].

Omnifaces záměrně nepoužívám ve verzi 2 a výše, protože vyžadují instalaci CDI[7] do zvoleného kontejneru Apache Tomcat, což bylo vzhledem k užití Spring nevhodné.

[7] CDI – podpora IoC a DI (popsáno 5.1.1) *Java Enterprise Edition* aplikačních serverů

5.3.6 Implementace vzoru *DAO Dispatcher* a CRUD operací

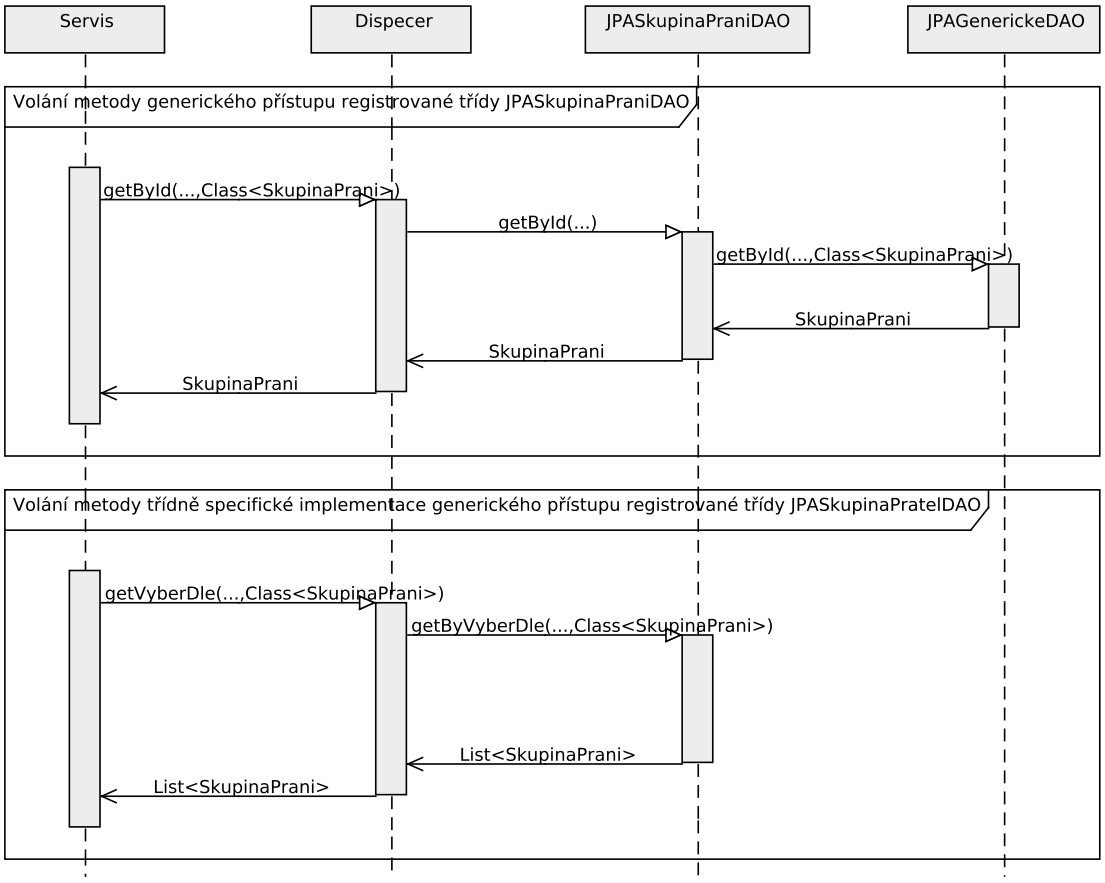
Následující text věnuji realizaci návrhu vrstvy přístupu k datům. Popíši implementaci návrhového vzoru *DAO Dispatcher*, zvoleného v sekci 4.1.2.3, a způsob dotazování.

Při implementaci jsem se řídila především popisem jednotlivých komponent poskytnutým článkem [Mička a Kouba, 2013] a strukturou ukázkové realizace [Mička, 2013].

Diagram 5.3 ilustruje, jak jsem vzor implementovala s mírným odchýlením od popisu [Mička a Kouba, 2013] v podobě závislosti rozhraní `DAODispecer` na rozhraní `GenerickeDAO` pro vynucení obsluhy všech generických metod.

Rozhraní `DAODispecer` je používáno servisní vrstvou pro generický přístup k instancím všech entitních typů. Při volání metody zkontroluje, zda byla pro parametr třídy entitního typu registrována implementace specifického přístupu k instancím.

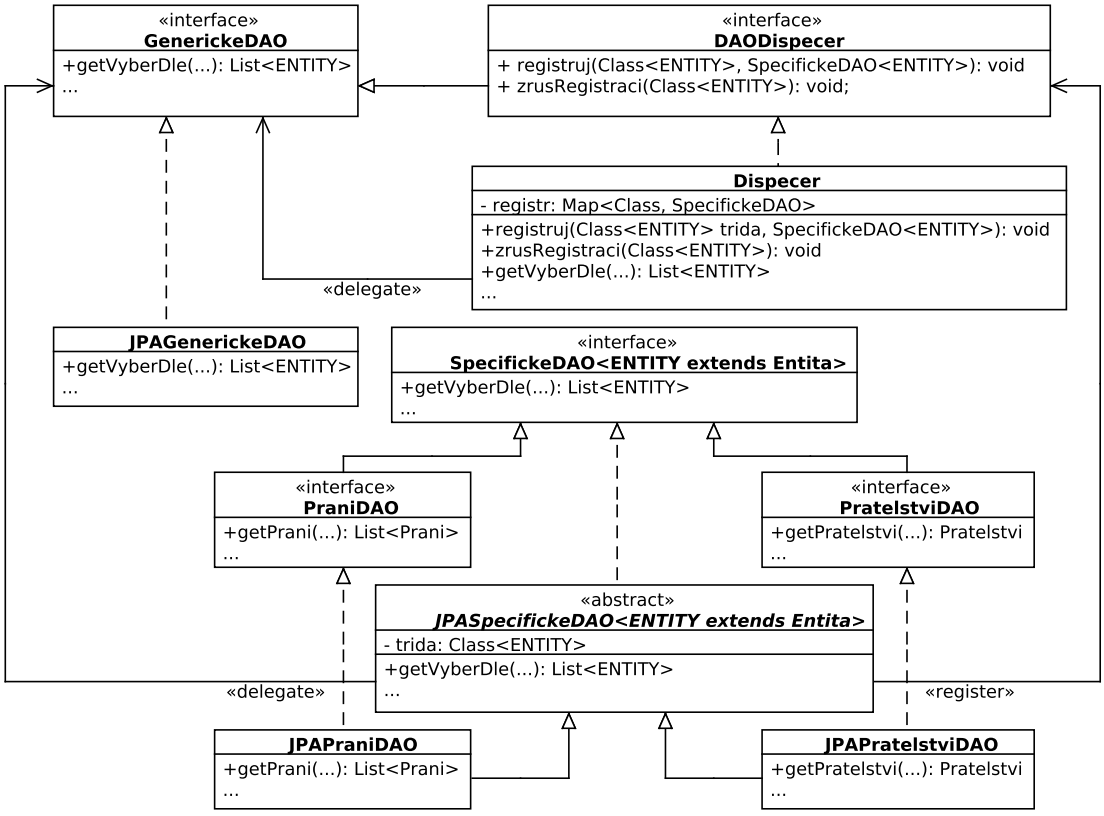
Je-li v tomto případě metoda přepsána implementací, vykoná ji ona, jinak je delegována na rozhraní generických volání. Přesměrování provede třída `JPASpecifickeDAO` – jakási fasáda chránící implementace specifických přístupů před vynucenou implementací metod deklarovaných v rozhraní `SpecifickeDAO`. Volání generické metody nepřetížené resp. přetížené specifickou implementací registrované třídy ukazuje diagram 5.2.



Obrázek 5.2: Volání metod dispečerem registrované třídy entitního typu Uzivatel

Metody pro entitní typy bez registrované implementace jsou dispečerem delegovány na rozhraní `GenerickeDAO` přímo. Naopak volání třídně specifických přístupů nepřetěžujících žádnou z generických metod je realizováno mimo dispečer – přes registrovanou implementaci rozhraní specifického přístupu pro daný entitní typ.

Aby mohla být rozhraní a třídy implementující specifický přístup registrována, musí jejich deklarace odpovídat parametrem vybranému entitnímu typu – například dvojice: `public interface PraniDAO extends SpecifickeDAO<Prani>` a `public class JPAPraniDAO extends JPASpecifickeDAO<Prani> implements PraniDAO`, a k vykonání registrace již při spuštění použitím knihovny `InitializingBean` musí být třída implementace specifického přístupu řízena rámcem Spring pomocí anotace `@Component`.



Obrázek 5.3: Implementace návrhového vzoru *DAO Dispatcher*

Třídy přistupující k datům jsou implementovány s použitím *kritérií* standardu JPA, protože mi poskytly hned několik zásadních výhod. Při tvorbě dotazů jsem:

- jednoduše slučovala podmínky výběru funkcemi `and`, `or`, `equal`, `notEqual`,
- využila jednoduchosti poskytnutého spojování tabulek pomocí `join`, například: `equal(prani.join("skupinaPrani").get("vlastnik"), vlastnik.getId())`,
- vyhnula se používání textových řetězců v dotazech, nutnosti pamatování si pojmenování názvů tabulek a sloupců a závislosti kódu dotazů na těchto jménech
- a využila k výběru do seznamů podpory stránkování záznamů na úrovni databáze.

5.3.7 Nastavení připojení k databázi Apache Derby a vložení dat

Jako úložiště jsem použila Apache Derby, kterou lze snadno vytvořit ve vývojovém prostředí. Rámci Spring je definován zdroj dat, který používá atributy uložené v souboru klíčů a hodnot `jdbc.properties`. Zde je třeba nastavit patřičné údaje:

- ke klíči `jdbc.url` adresu databáze,
- při `jdbc.username` její jméno
- a konečně pro klíč `jdbc.password` zadat heslo.

Pro vytvoření struktury úložiště je třeba ve vývojovém prostředí projekt poprvé spustit.

Ke smazání všech tabulek databáze lze použít u projektu připojený soubor `drop.sql`, naopak k naplnění úložiště daty poslouží spuštění příkazů souboru `data.sql`.

5.3.8 Použití *backing bean* `@Scope("view")` v uživatelském rozhraní

Většina stránek aplikace obsahuje nějaký formulář nebo tabulku dat, zároveň je ale třeba, aby mohl uživatel zobrazit více různých stránek stejného názvu a měl vždy k dispozici aktuální údaje. Proto jsem pro většinu *backing bean* – v souladu s předchozím rozborem v sekci 5.3.2 – zvolila životnost `view`.

Stránce příslušná *beana* je nejprve vytvořena konstruktorem za zjištění aktuálně nastavených parametrů adresy. Žádné další akce zde neprovedu, neboť veškeré injektované[8] prostředky budou nejdříve použitelné v metodě anotované `@PostConstruct`. Zde mohu poprvé z úložiště získat požadovaná data převedená do formy odpovídajících DTO[9].

Jak jsem již vysvětlila v sekci 5.3.2: ze jména životnosti `view` vyplývá, že přežívá mezi akcemi uživatele na jedné a té samé stránce, dokud není provedena volba navigace nebo *refresh*. Protože jsem nenalezla jednoznačnou odpověď, jakým způsobem data obnovovat, snažila jsem se dodržet následující:

- nemění-li akce uživatele zásadně parametry zobrazení stránky, nebo nemá-li vliv na sousední zobrazovaná data, provedu po vykonání akce pouze úpravu v pohledu,
- má-li provedená změna zásadní vliv na zobrazení parametrů stránky a mohlo-li by se stát, že ani při navazující nebo související akci by si uživatel nevyžádal nejaktuálnější data, je jednodušší vytvořit *beanu* novou.

Příkladem prvního typu může být například metoda hledaní uživatele podle přezdívky. Byť opakované zobrazení výsledku nemá vliv na ostatní prvky stránky. V případě, že by uživatel chtěl ostatní části aktualizovat, stačí mu provést obnovení stránky. Naopak zásadní akce jako smazání jedné celé skupiny přátel způsobí aktualizaci stránky.

[8] injektované pomocí *Dependency Injection* (DI) rámce Spring – zmíněno v sekci 5.1.1
[9] *Data Transfer Object* – návrhový vzor objektu přepravky na data více popsaný v sekci 4.1.3

6 Testování implementovaného systému

Vyvinutý systém jsem v souladu se zadáním cíle práce otestovala sadou uživatelských testů. Následující text je věnován stanovení priorit pro testování jednotlivých částí aplikace a zhodnocení výsledků.

6.1 Stanovení priorit pro testování částem aplikace

Všechny části aplikace si z hlediska potřeby testování nejsou rovny. Často používané nebo komplexní funkcionality webového systému mohou být jak náchylnější k chybám, tak způsobit vyšší ztrátu (uživatelů či finanční) při jejich selhání.

Do procesu určení priorit pro testování tedy dle [Spillner a kol., 2014, str. 177 a 196] vstupují dva aspekty:

- **pravděpodobnost** selhání vybrané části aplikace, jak jsem ji již popsala,
- a vážnost **dopadu** selhání na použitelnost systému.

Pro potřebu určení priorit testování uživatelskými testy jsem stanovila tři stupně každému z obou aspektů a výslednou prioritu *zaokrouhlila směrem nahoru*.

Č.	Část aplikace	Pravděpodobnost	Dopad	Priorita
1.	Zobrazení a správa profilu uživatele	Vysoká	Vážný	Vysoká
2.	Zobrazení a správa přání	Vysoká	Vážný	Vysoká
3.	Správa přátelství uživatele	Střední	Střední	Střední
4.	Správa skupin přátel	Střední	Nízký	Nízká
5.	Správa skupin přání	Střední	Střední	Střední
6.	Administrace uživatelů systému	Vysoká	Střední	Vysoká
7.	Administrace přání	Vysoká	Střední	Vysoká
8.	Administrace komentářů	Střední	Střední	Střední
9.	Registrace účtu	Vysoká	Střední	Vysoká
10.	Přihlášení do systému	Nízká	Vážný	Střední

Tabulka 6.1: Určení priorit pro testování dle pravděpodobnosti a dopadu selhání

6.2 Uživatelské testy dodaného webového systému

Sadu uživatelských testů jsem se snažila koncipovat tak, aby pokryla části aplikace s četností či náročností danou stanovenými prioritami pro testování. Webové stránky s obsáhlými formuláři vyžadují složitější testovací scénáře, zatímco části aplikace s nízkou prioritou byly otestovány jednodušeji. Pomocí testů administrace byla průchodem ověřena konzistence dat během změn stavů.

Následujícím výčtem uvádím pro každou část aplikace výčet uživatelských testů, kterými jsem funkčnost aplikace ověřila.

Č.	Část aplikace	Pokrývající uživatelské testy
1.	Zobrazení a správa profilu uživatele	Vyplnění uživatelských údajů po registraci, Úprava uživatelských údajů
2.	Zobrazení a správa přání	Vytvoření nového přání uživatelem, Úprava údajů existujícího přání, Zamluvení a zrušení zamluvení přání
3.	Správa přátelství uživatele	Žádost o přátelství uživatele vyhledaného přezdívkou, Zrušení stávajícího přátelství
4.	Správa skupin přátel	Přesun přítele do nově vytvořené skupiny přátel, Přesun přátel mazané skupiny do výchozí
5.	Správa skupin přání	Přesun přání do nově vytvořené skupiny přání, Přesun přání mazané skupiny do výchozí
6.	Administrace uživatelů systému	Administrace uživatelů systému typu uživatel, Administrace uživatelů systému typu administrátor
7.	Administrace přání	Administrace přání
8.	Administrace komentářů	Administrace komentářů
9.	Registrace účtu	Registrace návštěvníka v roli uživatele
10.	Přihlášení do systému	Přihlášení uživatele systému

Tabulka 6.2: Pokrytí částí aplikace zvolenými uživatelskými testy

Testování proběhlo úspěšně. Pomocí testů jsem ověřila funkčnost aplikace pomocí průchodů a testů životnosti dat. Podrobné testovací uvádím ve formě přílohy D.

7 Závěrečné zhodnocení

Cílem mé bakalářské práce bylo projít všemi fázemi vývoje projektu webového komunitního systému pro správu přání a dárků – tedy vytvoření vize, analýzy, návrhu a implementace – sadou uživatelských testů otestované – aplikace.

Domnívám se, že uvedený cíl byl naplněn a zadaní bakalářské práce jsem splnila.

V uplynulých kapitolách jsem ilustrovala svůj postup během celé činnosti, počínaje určením klíčových vlastností, udáním důvodů k zahájení projektu a stanovením objemu požadavků, které budou – na základě priorit stanovených metodou MoSCow vůči termínu odevzdání – podrobeny následnému procesu vývoje.

Analýzou jsem popsala všechny důležité aspekty týkající se problémové domény systému pro správu dárků a přání. Ověřila jsem si užitečnost UML diagramů pro názorné zachycení procesů, doménových objektů i případů, jakými bude systém používán.

Návrhu jsem věnovala nejprve část obecnější, ve které jsem zvolila pro jednotlivé vrstvy architektury aplikace užitečné návrhové vzory jako například *DAO Dispatcher*. S péčí jsem se věnovala také vybraným aspektům návrhu webového uživatelského rozhraní. Kromě použitelnosti a přístupnosti jsem se snažila navrhnout aplikaci s ohledem na uživatelskou přívětivost použitím zjednodušené formy mapování akcí na barvy a symboly.

Technologicky zaměřenou částí návrhu jsem obecnému konceptu architektury přiřadila zvolené programové rámce – především Spring – a výsledným diagramem balíčků opatřeným vysvětlením doplnila v dostatečném detailu poklady pro implementaci.

Poslední téměř třetinu textu jsem věnovala popisu zdolávání vybraných aspektů implementace navrženého systému. Vybrala jsem témata taková, která by případně dalším studentům mohla být užitečným přehledem či pomůckou. Zaměřila jsem se kupříkladu na způsob, jakým jsem volila životnost rámcem Spring řízených *bean* a jak byly použity. Dále také na popis mapování entitních typů s atributy položkami výčtu, či v hierarchii. Zajímavé pro mne bylo použití návrhového vzoru *DAO Dispatcher*. Příjemně mě taktéž potěšila *elegance* JPA *kritérií* pro dotazy nad daty v porovnání s tvorbou textové reprezentace. Při implementaci prezentační vrstvy jsem si uvědomila, jak rámce dokáží zásadně ulehčit práci například v podobě podpory konverze, zabezpečení nebo kontroly na formulářových polích.

Testování bylo poslední částí mé práce. Uživatelskými testy jsem s ohledem na stanovené priority jednotlivých částí aplikace ověřila očekávané funkčnosti.

Ačkoliv mé řešení poskytuje mnoho prostoru pro vylepšení, které si ráda ponechám pro další vývoj, doufám, že postupy, které jsem během práce využila a zprávou popsala, poslouží dalším studentům, kteří se budou potýkat s podobným problémem analýzy, návrhu a implementace komplexního projektu s využitím rámce Spring.

Použité informační zdroje

Anderson, S. P. (2012). *Přitažlivý interaktivní design: Jak vytvářet uživatelsky přívětivé produkty.* Computer Press, Brno. ISBN 978-80-251-3722-2.

Arlow, J. a Neustadt, I. (2008). *UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky.* Computer Press, Brno. ISBN 978-80-251-1503-9.

Civici, C. (2010). Porting jsf 2.0's viewscope to spring 3.0. *Cagatay Civici's Weblog on WordPress.com.* [online]. February 17, 2010 [cit. 2016-04-04]. URL `http://cagataycivici.wordpress.com/2010/02/17/port-jsf-2-0s-viewscope-to-spring-3-0/`.

Conallen, J. (2003). *Building Web Applications with UML: Second Edition.* Addison-Wesley. ISBN 0-201-73038-3.

Fowler, M. (2003). *Patterns of Enterprise Application Architecture.* Addison-Wesley, Boston. ISBN 0-321-12742-0.

Fowler, M. (2015). Yagni. *MartinFowler.com.* [online]. May 26, 2015 [cit. 2016-04-04]. URL `http://martinfowler.com/bliki/Yagni.html`.

Gupta, V. (2009). Organizing domain logic. *Vikas Gupta: Java architect, developer, Speaker, writer.* [online]. November 24, 2009 [cit. 2016-04-02]. URL `http://guptavikas.wordpress.com/2009/11/24/organizing-domain-logic`.

Hunt, A. a Thomas, D. (1999). *The Pragmatic Programmer, From Journeyman to Master.* Addison-Wesley. ISBN 0-201-61622-X.

Johnson, R., Hoeller, J., Donald, K. a Sampaleanu, C. Spring framework reference documentation: 4.2.6.release. [online]. [cit. 2016-04-23]. URL `http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/`.

Keith, M. a Schincariol, M. (2009). *Pro JPA 2: Mastering the Java persistence API.* Apress, New York. ISBN 978-1-4302-1956-9.

Mihalcea, V. (2015). A beginner's guide to jpa and hibernate cascade types. *Vlad Mihalcea's Blog: Teaching is my way of learning.* [online]. March 5, 2015 [cit. 2016-04-04]. URL `http://vladmihalcea.com/2015/03/05/a-beginners-guide-to-jpa-and-hibernate-cascade-types/`.

Minter, D. (2008). *Beginning Spring 2: From novice to professional.* Apress, Berkeley. ISBN 978-1-59059-685-2.

Mička, P. Architektura aplikací. [online]. [cit. 2016-04-04]. URL `http://cw.fel.cvut.cz/wiki/_media/courses/a7b39wpa/architektura.pdf`.

Mička, P. Spring framework. [online]. [cit. 2016-04-04]. URL http://cw.fel.cvut.cz/wiki/_media/courses/a7b39wpa/spring1.pdf.

Mička, P. (2013). flexidd (v. 1.0). [code]. [cit. 2016-04-04]. URL http://kbss.felk.cvut.cz/web/portal/dao-dispatcher.

Mička, P. a Kouba, Z. (2013). Dao dispatcher pattern: A robust design of the data access layer. *PATTERNS 2013: The Fifth International Conferences on Pervasive Patterns and Applications, May 27 - June 1, 2013. Valencia.* [online]. May 27, 2013 [cit. 2016-04-03]. ISSN: 2308-3557. 1–6. URL http://www.thinkmind.org/index.php?view=article&articleid=patterns_2013_1_10_70030.

Mička, P. a Šmíd, M. (2013). wpa-jpa-spring-jsf. [code]. [cit. 2016-04-04]. URL https://cw.fel.cvut.cz/wiki/_media/courses/ad7b39wpa/wpa-jpa-spring-jsf.zip.

Mook Kim, Y. (2012a). Spring bean scopes example. *Mkyong.com.* [online]. January 6, 2012 [cit. 2016-04-23]. URL http://www.mkyong.com/spring/spring-bean-scopes-examples/.

Mook Kim, Y. (2012b). Customize validation error message in jsf 2.0. *Mkyong.com.* [online]. August 29, 2012 [cit. 2016-04-23]. URL http://www.mkyong.com/jsf2/customize-validation-error-message-in-jsf-2-0/.

Mook Kim, Y. (2014). Spring security password hashing example. *Mkyong.com.* [online]. May 21, 2014 [cit. 2016-04-04]. URL http://www.mkyong.com/spring-security/spring-security-password-hashing-example/.

Omnifaces. Selectitemsconverter. *OmniFaces Showcase: To make JSF life easier.* [online]. [cit. 2016-04-05]. URL http://showcase.omnifaces.org/converters/SelectItemsConverter.

Paul, D., Yeates, D. a Cadle, J. (2010). *Business Analysis: Second Edition.* British Comp Society Series. BCS Learning & Development Limited. ISBN 1-90612-461-2.

Saleh, H., Christensen, A. L. a Wadia, Z. (2013). *Pro JSF and HTML5: Building rich internet components.* Apress, New York. ISBN 978-14-302-5010-4.

Scholtz, B. (2011). Communication in jsf 2.0. *The BalusC Code: Code depot of a Java EE developer.* [online]. September 17, 2011 [cit. 2016-04-04]. URL http://balusc.omnifaces.org/2011/09/communication-in-jsf-20.html.

Spillner, A., Linz, T. a Schaefer, H. (2014). *Software Testing Foundations: A Study Guide for the Certified Tester Exam, 4th Edition.* Spillner, Linz, Schaefer. ISBN 978-1-937538-42-2.

w3schools.com. Responsive web design - grid-view. *w3schools.com.* [online]. [cit. 2016-04-03]. URL http://www.w3schools.com/css/css_rwd_grid.asp.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

AOP *Aspect-Oriented Programming* – programování orientované na aspekty

API *Application Programming Interface* – sada funkčností pro vývoj

CDI *Contexts and Dependency Injection* – podpora IoC a DI *Java Enterprise Edition* aplikačních serverů

CRUD *operace **C**reate, **R**ead, **U**pdate, **D**elete neboli vytvoření, čtení, aktualizace a smazání*

CSS *Cascading Style Sheets* – značkovací jazyk pro popis vzhledu webové prezentace

DAO *Data Access Object* – objekt přístupu k datům

DI *Dependency Injection* – injektování závislostí – tj. správných implementací instancemi požadovaných zdrojů – a jejich životnost je řízena rámcem

DTO *Data Transfer Object* – objekt přepravky na data

HTML5 značkovací jazyk tvorby struktury a obsahu webové prezentace, verze pět

IoC *Inversion of Control* – převrácené řízení – aplikace je v pozici knihovny řízené rámcem v duchu Hollywoodského principu *„Don't call us, we'll call you.“* [Mička, b, sn. 8/18] – „Nevolejte nám, my se vám ozveme.“

JPA *Java Persistence API* – objekt přístupu k datům

MDA *Model Driven Architecture* – modelem řízená architektura

MoSCoW *„must have, should have, could have, want to have but won't have this time“* [Paul a kol., 2010] – označení priorit požadavků vůči datu dodání projektu

OMG technologické konsorcium *Object Management Group*

OOP Objektově orientované programování

ORM Objektově-relační mapování

POJO *Plain Old Java Object* – objekt, na který nejsou kladeny žádné požadavky kromě dodržování syntaxe jazyka Java

XML *Extensible Markup Language* – rozšiřitelný značkovací jazyk pro výměnu a zpracování dat

A Katalog funkčních a nefunkčních požadavků

A.1 Funkční požadavky

A.1.1 Požadavky na rozhraní pro administrátory

M	FA1	Přihlásit se
M	FA2	Změnit vlastní heslo
S	FA3	Zobrazit seznam nahlášených uživatelů systému
S	FA4	Zobrazit seznam deaktivovaných uživatelů systému
S	FA5	Zobrazit seznam aktivních uživatelů systému
C	FA6	Zobrazit seznam čekajících uživatelů systému
S	FA7	Nastavit stav účtu uživatele systému na aktivní
S	FA8	Nastavit stav účtu uživatele systému na deaktivovaný
C	FA9	Nastavit stav účtu uživatele systému na čekající
C	FA10	Nastavit stav účtu uživatele systému na nahlášený
S	FA11	Zobrazit profil uživatele
M	FA12	Upravit osobní údaje uživatele kromě hesla
C	FA13	Upravit osobní údaje administrátora kromě hesla
S	FA14	Přidat nového uživatele systému v roli uživatele
C	FA15	Přidat nového uživatele systému v roli administrátora
C	FA16	Zobrazit seznam nahlášených komentářů
C	FA17	Zobrazit seznam deaktivovaných komentářů
C	FA18	Zobrazit seznam aktivních komentářů
C	FA19	Nastavit stav komentáře na aktivní
C	FA20	Nastavit stav komentáře na deaktivovaný
C	FA21	Nastavit stav komentáře na nahlášený
C	FA22	Nastavit text komentáře
S	FA23	Zobrazit seznam nahlášených přání
S	FA24	Zobrazit seznam deaktivovaných přání
S	FA25	Zobrazit seznam aktivních přání
S	FA26	Nastavit stav přání na aktivní
S	FA27	Nastavit stav přání na deaktivováno
C	FA28	Nastavit stav přání na nahlášeno
C	FA29	Nastavit jazyk přání
M	FA30	Upravit přání
C	FA31	Zobrazit detail nedeaktivovaného přání

W	FA32	Zobrazit nedeaktivované fotografie přání
S	FA33	Zobrazit nedeaktivované komentáře přání
C	FA34	Deaktivovat komentář na detailu nedeaktivovaného přání
W	FA35	Zobrazit seznam nahlášených fotografií
W	FA36	Zobrazit seznam deaktivovaných fotografií
W	FA37	Zobrazit seznam aktivních fotografií
W	FA38	Nastavit stav fotografie na aktivní
W	FA39	Nastavit stav fotografie na deaktivovaný
W	FA40	Nastavit stav fotografie na nahlášený
W	FA41	Nastavit popisek fotografie
W	FA42	Zobrazit mapu stránek administrace

A.1.2 Obecné požadavky na rozhraní

M	FS1	Zobrazit hlavní stranu
S	FS2	Nastavit jazyk rozhraní pro danou relaci
W	FS3	Nastavit měnu rozhraní pro danou relaci
M	FS4	Registrovat se v roli uživatele
W	FS5	Registrovat se pomocí nástrojů třetích stran
S	FS6	Aktivovat účet
C	FS7	Odeslat žádost o nastavení nového hesla
W	FS8	Odeslat uživateli systému emailovou zprávu pro nastavení hesla
W	FS9	Zkontrolovat viditelnost přání v rámci skupiny přání uživatele
M	FS10	Zobrazit veřejný profil nedeaktivovaného uživatele
M	FS11	Zobrazit veřejná nedeaktivovaná přání nedeaktivovaného uživatele s veřejným profilem bez informace o zamluvení
M	FS12	Zobrazit detail veřejného nedeaktivovaného přání
W	FS13	Zobrazit veřejné nedeaktivované fotografie veřejného nedeaktivovaného přání
S	FS14	Zobrazit nedeaktivované komentáře veřejného nedeaktivovaného přání
S	FS15	Zobrazit podmínky užívání aplikace
W	FS16	Zobrazit mapu stránek pro nepřihlášeného uživatele

A.1.3 Požadavky na rozhraní pro uživatele

M	FU1	Přihlásit se
M	FU2	Upravit vlastní uživatelské údaje

M	FU3	Změnit vlastní heslo
M	FU4	Přidat nové přání
M	FU5	Upravit vlastní přání
M	FU6	Nastavit stav vlastního přání na deaktivovaný
M	FU7	Zobrazit stránku správy stávajících a budoucích přátel
M	FU8	Zobrazit seznam vlastních přátel
M	FU9	Zrušit přátelství s uživatelem
M	FU10	Zobrazit seznam žádostí o přátelství
M	FU11	Přijmout žádost o přátelství
M	FU12	Odmítnout žádost o přátelství
S	FU13	Zobrazit uživatele vyhledané podle přezdívky
M	FU14	Odeslat uživateli žádost o přátelství
C	FU15	Zobrazit stránku úpravy vlastních skupin přátel
C	FU16	Přidat novou skupinu přátel
C	FU17	Přesunout přítele do vybrané skupiny přátel
C	FU18	Smazat vlastní nevýchozí skupinu přátel
C	FU19	Upravit vlastní skupinu přátel
S	FU20	Zobrazit stránku úpravy vlastních skupin přání
S	FU21	Přidat novou skupinu přání
S	FU22	Přesunout vlastní přání do vybrané skupiny
S	FU23	Smazat vlastní nevýchozí skupinu přání
S	FU24	Upravit vlastní skupinu přání
C	FU25	Zobrazit seznam uživatelem zamluvených přání
C	FU26	Zobrazit seznam uživatelem splněných přání
W	FU27	Zobrazit seznam uživatelem zamilovaných přání
M	FU28	Zobrazit profil přítele
M	FU29	Zobrazit přání přítele
M	FU30	Zobrazit detail přání přítele
W	FU31	Zobrazit nedeaktivované fotografie přání přítele
S	FU32	Zobrazit nedeaktivované komentáře přání přítele
C	FU33	Označit přání za zamilované
C	FU34	Zrušit označení přání za zamilované
S	FU35	Komentovat vlastní přání
S	FU36	Komentovat veřejné přání
S	FU37	Komentovat přání přítele
C	FU38	Deaktivovat svůj vlastní komentář
C	FU39	Deaktivovat komentář u vlastního přání
M	FU40	Zamluvit přání

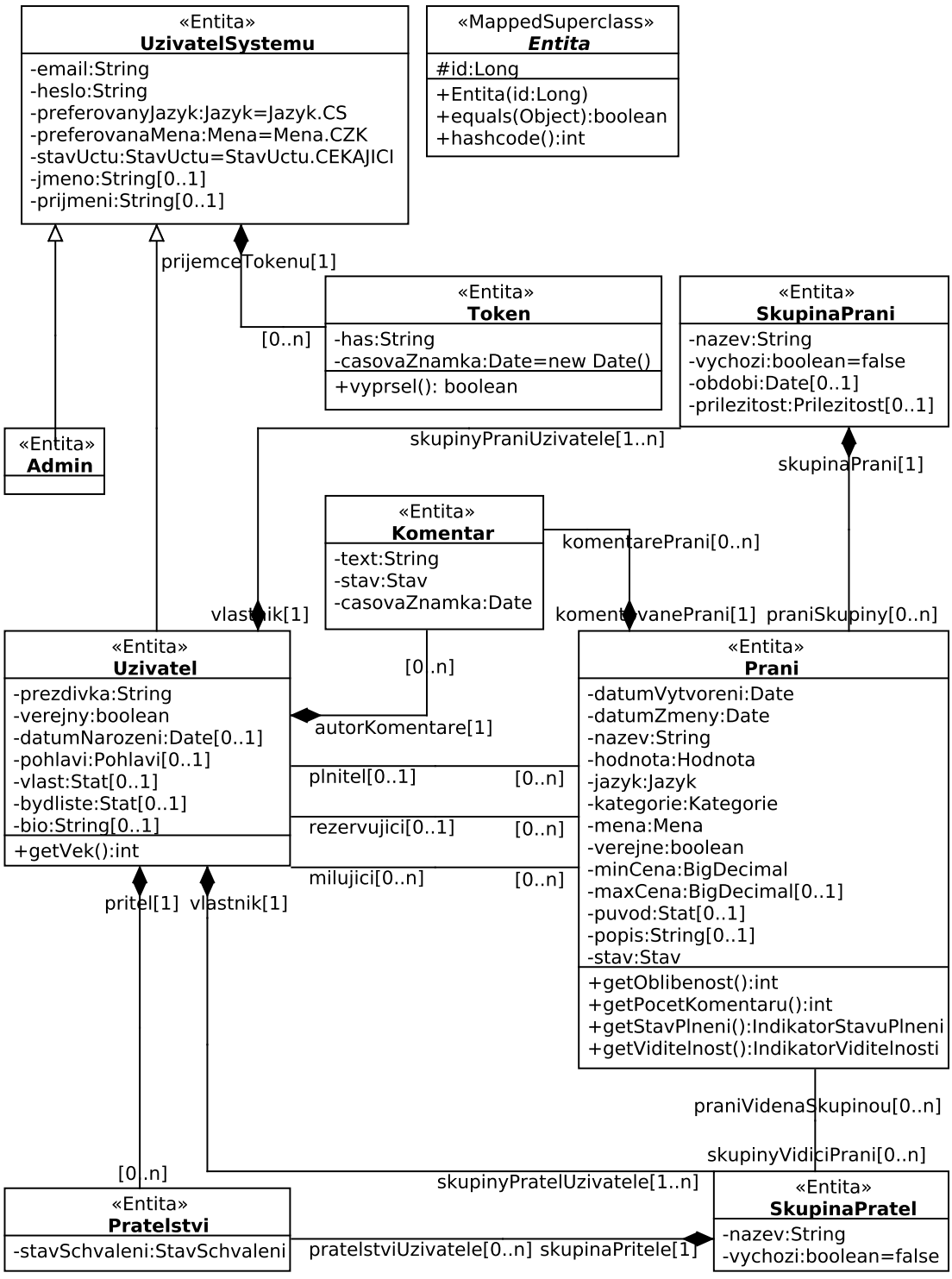
M	FU41	Zrušit zamluvení přání
C	FU42	Kopírovat přání jako nové
C	FU43	Nahlásit profil uživatele
C	FU44	Nahlásit přání
C	FU45	Nahlásit komentář
W	FU46	Nahlásit fotografii
W	FU47	Zobrazit mapu stránek pro přihlášeného běžného uživatele
M	FU48	Odhlásit se

A.2 Nefunkční požadavky

◇ Systém bude využívat principů objektového modelování a vrstevnaté architektury pro snadné rozšíření.

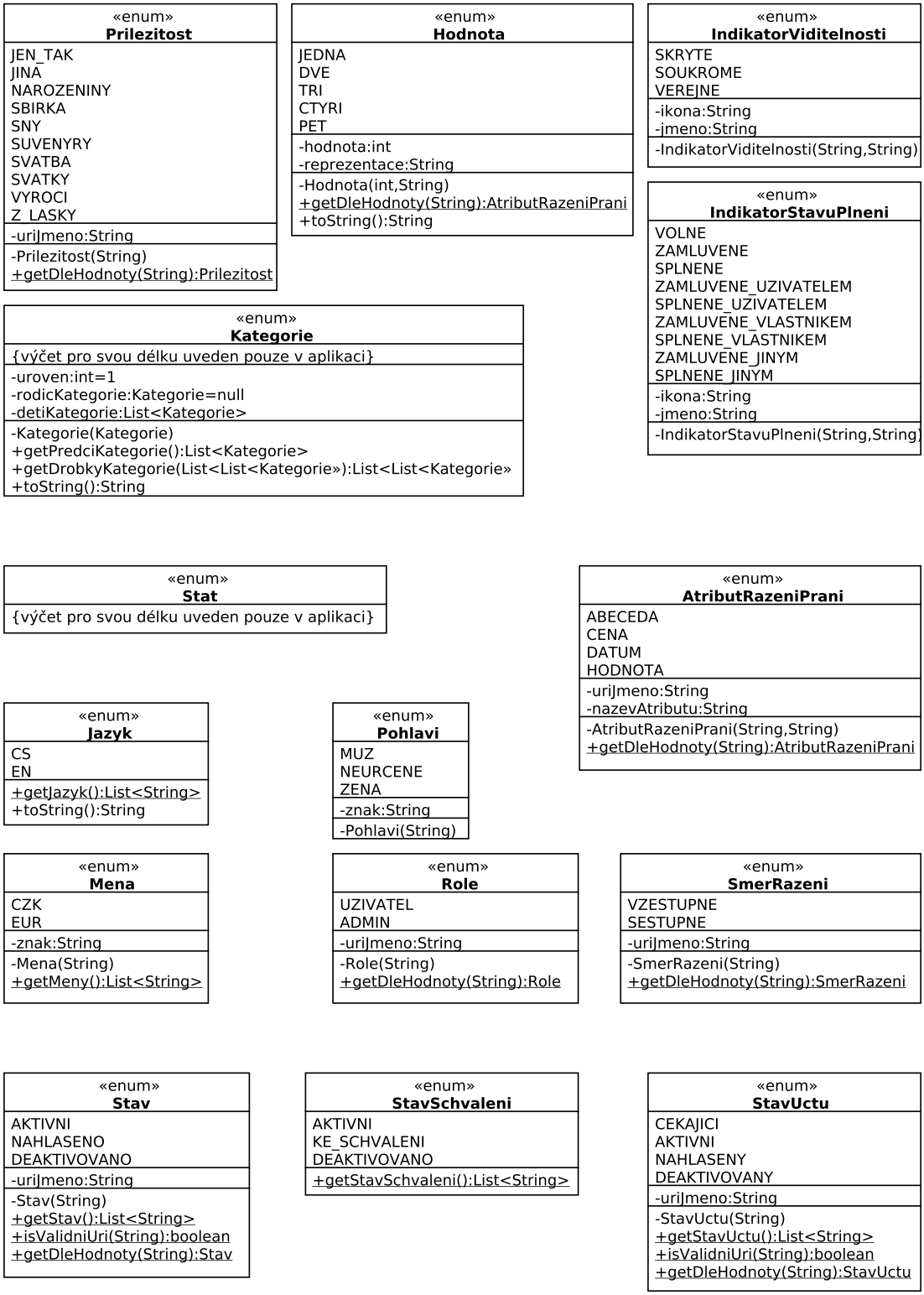
B Diagramy návrhového doménového modelu

B.1 Entitní typy a relace návrhového doménového modelu



Obrázek B.1: Entitní typy a relace návrhového doménového modelu

B.2 Výčtové typy návrhového doménového modelu



Obrázek B.2: Výčtové typy návrhového doménového modelu

C Anotace JPA vztahů entitních typů

C.1 Anotace JPA unidirekcionálních 1:N vztahů

Vlastnící strana vazby	Typ vazby
`@JoinColumn(nullable = false, name = "prijemce_tokenu")` `@ManyToOne` `UzivatelSystemu prijemceTokenu`	Povinně nenulová unidirekcionální vazba **tokenu** na **uživatele systému**, který je příjemcem haše tokenu.
`@JoinColumn(nullable = false, name = "pritel")` `@ManyToOne` `Uzivatel pritel`	Povinně nenulová unidirekcionální vazba N:1 **přátelství** na parametr vztahové entity přítele – **uživatele** ve skupině.
`@JoinColumn(nullable = true, name = "rezervujici")` `@ManyToOne` `Uzivatel rezervujici`	Nepovinná unidirekcionální vazba N:1 **přání** na **uživatele**, který přání zamluvil.
`@JoinColumn(nullable = true, name = "plnitel")` `@ManyToOne` `Uzivatel plnitel`	Nepovinná unidirekcionální vazba N:1 **přání** na **uživatele**, který přání splnil.
`@JoinColumn(nullable = false, name = "autor_komentare", referencedColumnName = "id")` `@ManyToOne` `Uzivatel autorKomentare`	Povinně nenulová unidirekcionální vazba N:1 **komentáře** na **uživatele** – autora.

Tabulka C.1: Anotace JPA unidirekcionálních 1:N vztahů entitních typů

C.2 Anotace JPA bidirekcionálních 1:N vztahů

Vlastnící strana vazby	Typ vazby	Vlastněná strana vazby
`@JoinColumn( nullable = false, name = "vlastnik") @ManyToOne Uzivatel vlastnik`	Povinně nenulová bidirekcionální vazba 1:N **uživatele** a jeho existenčně závislých **skupin přání**. Skupina nemůže být přiřazena jinému uživateli.	`@OneToMany(cascade = CascadeType.ALL, mappedBy = "vlastnik", orphanRemoval = true) List<SkupinaPrani> skupinyPraniUzivatele`
`@JoinColumn( nullable = false, name = "vlastnik") @ManyToOne Uzivatel vlastnik`	Povinně nenulová bidirekcionální vazba 1:N **uživatele** a jeho existenčně závislých **skupin přátel**. Skupina nemůže být přiřazena jinému uživateli.	`@OneToMany(cascade = CascadeType.ALL, mappedBy = "vlastnik", orphanRemoval = true) List<SkupinaPratel> skupinyPratelUzivatele`
`@JoinColumn(nullable = false, name = "skupina_pritele") @ManyToOne SkupinaPratel skupinaPritele`	Povinně nenulová bidirekcionální vazba 1:N určení, která **přátelství** a potažmo přátelé jsou ve **skupině přátel** zařazena; přátelství je možno přesunout do jiné skupiny přátel.	`@OneToMany(cascade = CascadeType.ALL, mappedBy = "skupinaPritele", orphanRemoval = false) List<Pratelstvi> pratelstviUzivatele`
`@JoinColumn(nullable = false, name = "skupina_prani") @ManyToOne SkupinaPrani skupinaPrani`	Povinně nenulová bidirekcionální vazba 1:N mezi **přáním** a jeho zařazením do **skupiny přání**; přání může být přesouváno mezi skupinami přání.	`@OneToMany(cascade = CascadeType.ALL, mappedBy = "skupinaPrani", orphanRemoval = false) List<Prani> praniSkupiny`
`@JoinColumn(nullable = false, name = "komentovane_prani", referencedColumnName = "id") @ManyToOne Prani komentovanePrani`	Povinně nenulová bidirekcionální vazba 1:N **přání** a jeho **komentářů**; komentáře nelze přesunout k jinému přání.	`@OneToMany(cascade = CascadeType.ALL, mappedBy = "komentovanePrani", orphanRemoval = true) List<Komentar> komentarePrani`

Tabulka C.2: Anotace JPA bidirekcionálních 1:N vztahů entitních typů

C.3 Anotace JPA bidirekcionálních M:N vztahů

Vlastnící strana vazby	Typ vazby	Vlastněná strana vazby
`@JoinTable(name = "viditelnost", joinColumns = @JoinColumn(nullable = false, name = "prani", referencedColumnName = "id"), inverseJoinColumns = @JoinColumn(nullable = false, = "skupina_pratel", referencedColumnName = "id")) @ManyToMany(cascade = CascadeType.PERSIST, CascadeType.MERGE) List<SkupinaPratel> skupinyVidiciPrani`	Povinně nenulová bidirekcionální vazba N:M mezi **přáním** a **skupinami přátel**, které jej vidí; mazání kteréhokoliv účastníka vazby nevede k odstranění instancí na opačném konci.	`@ManyToMany(cascade = CascadeType.PERSIST, CascadeType.MERGE, mappedBy = "skupinyVidiciPrani") List<Prani> praniVidenaSkupinou`

Tabulka C.3: Anotace JPA bidirekcionálních M:N vztahů entitních typů

C.4 Anotace JPA unidirekcionálních M:N vztahů

Vlastnící strana vazby	Typ vazby
`@JoinTable(name = "oblibenost", joinColumns = @JoinColumn(nullable = false, name = "prani", referencedColumnName = "id"), inverseJoinColumns = @JoinColumn(nullable = false, name = "milujici", referencedColumnName = "id")) @ManyToMany(cascade = CascadeType.PERSIST, CascadeType.MERGE) List<Uzivatel> milujici`	Povinně nenulová unidirekcionální vazba N:M **přání** na **uživatele**, kteří si přání zamilovali.

Tabulka C.4: Anotace JPA unidirekcionálních M:N vztahů entitních typů

D Scénáře uživatelských testů

D.1 Vytvoření nového přání uživatelem

Název testu	Vytvoření nového přání uživatelem
Cíl testu	Ověření správnosti vytváření nového přání a správného uložení všech zadaných údajů
Očekávaný výsledek	Vytvoří se nové přání se zadanými údaji
Podmínky testu	Přihlášený uživatel systému v roli uživatele
Akce	Stisknutí položky menu „Přidat nové přání“
Očekávaný výstup	Zobrazí se stránka „Úprava přání“ bez zadaných údajů
Akce	Vyplnění textu v políčkách „Název popisující přání“ a „Cena (minimální)“
Očekávaný výstup	Ve všech políčkách je možné vyplnit textový řetězec
Akce	Výběr hodnoty v rozbalovacích políčkách „Hodnota“, „Jazyk“, „Kategorie přání“, „Skupina přání“ a „Měna“
Očekávaný výstup	V každém z políček je možné vybrat jakoukoli hodnotu ze seznamu
Akce	Výběr viditelnosti v přepínacím políčku „Viditelnost přání“
Očekávaný výstup	Je možné vybrat jakoukoli hodnotu v přepínacím políčku
Akce	V políčku „Viditelné a plnitelné přáteli ze skupin“ vybrat skupinu přátel
Očekávaný výstup	Je možné vybrat jakoukoli hodnotu v políčku
Akce	Stisknutí tlačítka „Zobrazit více“
Očekávaný výstup	Zobrazí se sekce „Volitelné detaily přání“
Akce	Stisknutí tlačítka „Uložit základní informace“
Očekávaný výstup	Všechny zadané údaje se uložily
Akce	Vyplnění textu v políčkách „Popis či příběh motivující k splnění přání“ a „Maximální cena přání“
Očekávaný výstup	V políčkách je možné vyplnit textový řetězec
Akce	Výběr hodnoty v rozbalovacím políčku „Původ“
Očekávaný výstup	V políčku je možné vybrat jakoukoli hodnotu ze seznamu

Akce	Stisknutí tlačítka „Uložit detaily“
Očekávaný výstup	Všechny zadané údaje se uložily
Akce	Stisknutí tlačítka „Uložit přání“
Očekávaný výstup	Všechny zadané údaje se uložily a uživatel je přesměrován ze stránky pryč

Tabulka D.1: Vytvoření nového přání uživatelem

D.2 Úprava údajů existujícího přání

Název testu	Úprava údajů existujícího přání
Cíl testu	Ověření správnosti zadávání a ukládání změněných údajů přání
Očekávaný výsledek	Všechny změněné údaje se uloží
Podmínky testu	Přihlášený uživatel a vytvořené přání s vyplněnými údaji
Akce	Stisknutí ikonky editace na detailu přání
Očekávaný výstup	Zobrazí se stránka „Úprava přání“ a načítají se uložené údaje přání
Akce	Změna textu v políčkách „Název popisující přání“ a „Cena (minimální)“
Očekávaný výstup	Ve všech políčkách je možné změnit textový řetězec
Akce	Výběr jiné hodnoty v rozbalovacích políčkách „Hodnota“, „Jazyk“, „Kategorie přání“, „Skupina přání“ a „Měna“
Očekávaný výstup	V každém z políček je možné vybrat jinou hodnotu ze seznamu
Akce	Výběr viditelnosti v přepínacím tlačítku „Viditelnost přání“
Očekávaný výstup	Je možné vybrat jinou hodnotu v přepínacím tlačítku
Akce	V políčku „Viditelné a plnitelné přáteli ze skupin“ vybrat skupinu přátel
Očekávaný výstup	Je možné vybrat jinou hodnotu v políčku
Akce	Stisknutí tlačítka „Zobrazit více“
Očekávaný výstup	Zobrazí se sekce „Volitelné detaily přání“ s vyplněnými uloženými údaji
Akce	Stisknutí tlačítka „Uložit základní informace“

Očekávaný výstup	Všechny zadané údaje se uloží
Akce	Vyplnění textu v políčkách „Popis či příběh motivující k splnění přání“ a „Maximální cena přání“
Očekávaný výstup	Ve všech políčkách je možné změnit textový řetězec
Akce	Změna hodnoty v rozbalovacím políčku „Původ“
Očekávaný výstup	V políčku je možné vybrat jinou hodnotu ze seznamu
Akce	Stisknutí tlačítka „Uložit detaily“
Očekávaný výstup	Všechny zadané údaje se uložily
Akce	Změna hodnoty v rozbalovacím políčku „Uživatel, který splnil, došlo-li k tomu“
Očekávaný výstup	V políčku je možné vybrat jakéhokoli uživatele ze seznamu
Akce	Stisknutí tlačítka „Uložit stav plnění“
Očekávaný výstup	Údaje o plnění se uložily
Akce	Stisknutí tlačítka „Uložit přání“
Očekávaný výstup	Všechny zadané údaje se uložily

Tabulka D.2: Úprava údajů existujícího přání

D.3 Vyplnění uživatelských údajů po registraci

Název testu	Vyplnění uživatelských údajů po registraci
Cíl testu	Ověření správnosti zadávání a ukládání uživatelských údajů
Očekávaný výsledek	Všechny zadané uživatelské údaje se uloží
Podmínky testu	Přihlášený uživatel s nevyplněnými uživatelskými údaji
Akce	Stisknutí ikonky editace na profilu uživatele
Očekávaný výstup	Zobrazí se stránka „Editace uživatelských údajů“
Akce	Vyplnění údajů v sekci „Identifikační údaje“
Očekávaný výstup	Ve všech políčkách je možné zadat údaje
Akce	Stisknutí tlačítka „Uložit identifikační údaje“
Očekávaný výstup	Všechny zadané údaje se uložily
Akce	Vyplnění textu v políču „Nové heslo“
Očekávaný výstup	Je možné vyplnit textový řetězec
Akce	Stisknutí tlačítka „Uložit heslo“

Očekávaný výstup	Nové heslo se uloží
Akce	Vyplnění údajů v sekci „Volitelné detailní osobní údaje“
Očekávaný výstup	Ve všech políčkách je možné zadat údaje
Akce	Stisknutí tlačítka „Uložit detailní osobní údaje“
Očekávaný výstup	Všechny zadané údaje se uložily
Akce	Stisknutí tlačítka „Uložit uživatelské údaje“
Očekávaný výstup	Všechny zadané údaje se uložily

Tabulka D.3: Vyplnění uživatelských údajů po registraci

D.4 Přihlášení uživatele systému

Název testu	Přihlášení uživatele systému
Cíl testu	Ověření funkce přihlášení registrovaného uživatele systému do systému
Očekávaný výsledek	Uživatel systému se přihlásí do systému
Podmínky testu	Nepřihlášený registrovaný uživatel systému
Akce	Na hlavní stránce kliknout na odkaz na přihlášení
Očekávaný výstup	Zobrazí se stránka „Přihlášení do uživatelské sekce“
Akce	Vyplnit email a heslo a stisknout tlačítko „Přihlásit se“
Očekávaný výstup	Uživatel systému je přihlášen do systému a zobrazí se mu informace, ke které má přístup na základě role

Tabulka D.4: Přihlášení uživatele systému

D.5 Registrace návštěvníka v roli uživatele

Název testu	Registrace návštěvníka v roli uživatele
Cíl testu	Ověření funkce registrace návštěvníka do systému v roli uživatele
Očekávaný výsledek	Návštěvník stránky se zaregistruje do systému a vytvoří se mu uživatel
Podmínky testu	Návštěvník stránky s dosud neregistrovanou emailovou adresou

Akce	Na hlavní stránce kliknout na odkaz na registraci
Očekávaný výstup	Zobrazí se stránka „Registrace“
Akce	Vyplnit přezdívku, email, zaškrtnout tlačítko „Souhlasím s podmínkami užití“ a stisknout tlačítko „Registrovat se“
Očekávaný výstup	Uživateli se zobrazí zpráva o zaslání emailu s dalšími pokyny k dokončení registrace a v systému se vytvoří uživatel ve stavu „Čekající“

Tabulka D.5: Registrace návštěvníka v roli uživatele

D.6 Úprava uživatelských údajů

Název testu	Úprava uživatelských údajů
Cíl testu	Ověření správnosti změny a ukládání uživatelských údajů
Očekávaný výsledek	Všechny změněné uživatelské údaje se uloží
Podmínky testu	Přihlášený uživatel s vyplněnými uživatelskými údaji
Akce	Stisknutí ikonky editace na profilu uživatele
Očekávaný výstup	Zobrazí se stránka „Editace uživatelských údajů“
Akce	Změna údajů v sekci „Identifikační údaje“
Očekávaný výstup	Ve všech políčkách je možné změnit údaje
Akce	Stisknutí tlačítka „Uložit identifikační údaje“
Očekávaný výstup	Všechny zadané údaje se uložily
Akce	Vyplnění textu v políčku „Nové heslo“
Očekávaný výstup	Je možné vyplnit textový řetězec
Akce	Stisknutí tlačítka „Uložit heslo“
Očekávaný výstup	Nové heslo se uloží
Akce	Změna údajů v sekci „Volitelné detailní osobní údaje“
Očekávaný výstup	Ve všech políčkách je možné změnit údaje
Akce	Stisknutí tlačítka „Uložit detailní osobní údaje“
Očekávaný výstup	Všechny zadané údaje se uložily
Akce	Stisknutí tlačítka „Uložit uživatelské údaje“
Očekávaný výstup	Všechny zadané údaje se uložily

Tabulka D.6: Úprava uživatelských údajů

D.7 Zamluvení a zrušení zamluvení přání

Název testu	Zamluvení a zrušení zamluvení přání
Cíl testu	Ověření správnosti zamluvení a zrušení zamluvení přání
Očekávaný výsledek	Přání bude zamluveno a zamluvení zrušeno
Podmínky testu	Přihlášený uživatel s existujícími nezamluvenými přáními vlastními i přátel
Akce	Na stránce „Můj wishlist“ a stránce profilu stisknutí tlačítka „Zamluvit přání“
Očekávaný výstup	Přání se změní na zamluveno (ikonka klíče) a ve sloupci „Zamluvení“ se zobrazí tlačítko „Zrušit zamluvení“
Akce	Na stránce „Můj wishlist“ a stránce profilu stisknutí tlačítka „Zrušit zamluvení“
Očekávaný výstup	Přání se změní na nezamluvené (ikonka klíče se změní) a ve sloupci „Zamluvení“ se zobrazí tlačítko „Zamluvit přání“

Tabulka D.7: Zamluvení a zrušení zamluvení přání

D.8 Žádost o přátelství uživatele vyhledaného přezdívkou

Název testu	Žádost o přátelství uživatele vyhledaného přezdívkou
Cíl testu	Ověření správnosti funkce hledání nového přítele dle přezdívky a odeslání žádosti o přátelství
Očekávaný výsledek	Jinému uživateli bude zaslána žádost o přátelství
Podmínky testu	Přihlášený uživatel a ještě alespoň jeden další aktivní nebo nahlášený uživatel s hledanou přezdívkou
Akce	Na stránce „Přátelé stávající a budoucí“ stisknout tlačítko „Hledat nového přítele“
Očekávaný výstup	Zobrazí se formulář pro hledání nového přítele
Akce	Zadání přesné přezdívky existujícího aktivního nebo nahlášeného uživatele do políčka „Přezdívka hledaného uživatele“ a stisknutí „Hledat nového přítele“
Očekávaný výstup	V sekci „Výsledek hledání“ se zobrazí hledaný uživatel
Akce	Stisknutí tlačítka „Odeslat žádost o přátelství“

Očekávaný výstup	Žádanému uživateli se zobrazí žádost o přátelství v sekci „Žádosti o přátelství“

Tabulka D.8: Žádost o přátelství uživatele vyhledaného přezdívkou

D.9 Zrušení stávajícího přátelství

Název testu	Zrušení stávajícího přátelství
Cíl testu	Ověření správnosti funkce zrušení stávajícího přátelství
Očekávaný výsledek	U obou uživatelů se odstraní přítel ze seznamu přátel
Podmínky testu	Přihlášený uživatel a ještě alespoň jeden další aktivní nebo nahlášený uživatel v seznamu přátel
Akce	Na stránce „Přátelé stávající a budoucí“ stisknout tlačítko „Zrušit přátelství“
Očekávaný výstup	U obou uživatelů se odstraní přítel ze seznamu přátel

Tabulka D.9: Zrušení stávajícího přátelství

D.10 Přesun přítele do nově vytvořené skupiny přátel

Název testu	Přesun přítele do nově vytvořené skupiny přátel
Cíl testu	Ověření funkce vytvoření nové skupiny přátel a přesunu přítele
Očekávaný výsledek	Vytvoří se nová skupina přátel a je do ní možné přítele přesunout
Podmínky testu	Přihlášený uživatel a ještě alespoň jeden další existující aktivní nebo nahlášený uživatel v seznamu přátel
Akce	Na stránce „Přátelé stávající a budoucí“ stisknout tlačítko „Upravit skupiny přátel“
Očekávaný výstup	Zobrazí se stránka „Úprava skupin přátel“
Akce	Stisknout tlačítko „Přidat novou skupinu přátel“ v sekci „Přidat novou skupinu přátel“ zadat do políčka „Název skupiny přátel“ název skupiny a stisknutí tlačítka „Uložit“
Očekávaný výstup	Vytvoří se nová skupina přátel

Akce	U přítele v jiné skupině přátel změnit v rozbalovacím políčku „Přesunout přítele do skupiny“ zvolit nově vytvořenou skupinu a stisknout tlačítko „Přesunout“
Očekávaný výstup	Přítel se odstraní ze seznamu přátel ve zvolené skupině a zobrazí se v seznamu přátel v nově vytvořené skupině přátel

Tabulka D.10: Přesun přítele do nově vytvořené skupiny přátel

D.11 Přesun přátel mazané skupiny do výchozí

Název testu	Přesun přátel mazané skupiny do výchozí
Cíl testu	Ověření funkce smazání existující skupiny přátel s přesunem obsažených přátel do výchozí skupiny přátel uživatele
Očekávaný výsledek	Smaže se existující skupina přátel a obsažení přátelé budou přesunuti do výchozí skupiny přátel uživatele
Podmínky testu	Přihlášený uživatel a ještě alespoň jeden další existující aktivní nebo nahlášený uživatel v seznamu přátel
Akce	Na stránce „Přátelé stávající a budoucí“ stisknout tlačítko „Upravit skupiny přátel“
Očekávaný výstup	Zobrazí se stránka „Úprava skupin přátel“
Akce	Vybrat skupinu přátel a stisknout tlačítko „Odstranit skupinu“
Očekávaný výstup	Zobrazí se sekce „Odstranit skupinu:[název skupiny přátel]“
Akce	V sekci „Odstranit skupinu:[název skupiny přátel]“ vybrat v políčku „Nenávratně odstranit stávající skupinu“ vybrat položku „Odstranit“ a stisknout tlačítko „Nadobro smazat“
Očekávaný výstup	Vybraná skupina přátel se odstraní a přátelé zařazeni ve skupině budou přesunuti do výchozí skupiny přátel uživatele

Tabulka D.11: Přesun přátel mazané skupiny do výchozí

D.12 Přesun přání do nově vytvořené skupiny přání

Název testu	Přesun přání do nově vytvořené skupiny přání
Cíl testu	Ověření funkce vytvoření nové skupiny přání a přesunu přání

Očekávaný výsledek	Vytvoří se nová skupina přání a je do ní možné přání přesunout
Podmínky testu	Přihlášený uživatel s existujícím přáním
Akce	Na stránce „Úprava přání“ stisknout tlačítko „Editovat skupiny přání“
Očekávaný výstup	Zobrazí se stránka „Úprava skupin přání“
Akce	Stisknout tlačítko „Přidat novou skupinu přání“ a v sekci „Přidat novou skupinu přání“ vypnit políčka a stisknout tlačítko „Uložit“
Očekávaný výstup	Vytvoří se nová skupina přání
Akce	U přání v jiné skupině přání změnit v rozbalovacím políčku „Přesunout přání do skupiny“ zvolit nově vytvořenou skupinu a stisknout tlačítko „Přesunout“
Očekávaný výstup	Přání se odstraní ze seznamu přání ve zvolené skupině a zobrazí se v seznamu přání v nově vytvořené skupině přání

Tabulka D.12: Přesun přání do nově vytvořené skupiny přání

D.13 Přesun přání mazané skupiny do výchozí

Název testu	Přesun přání mazané skupiny do výchozí
Cíl testu	Ověření funkce smazání existující skupiny přání
Očekávaný výsledek	Smaže se existující skupina přání
Podmínky testu	Přihlášený uživatel s existujícím přáním a existující skupinou přání
Akce	Na stránce „Úprava přání“ stisknout tlačítko „Editovat skupiny přání“
Očekávaný výstup	Zobrazí se stránka „Úprava skupin přání“
Akce	Vybrat skupinu přání a stisknout tlačítko „Odstranit skupinu“
Očekávaný výstup	Zobrazí se sekce „Odstranit skupinu:[název skupiny přání]“
Akce	V sekci „Odstranit skupinu:[název skupiny přání]“ vybrat v políčku „Nenávratně odstranit stávající skupinu“ vybrat položku „Odstranit“ a stisknout tlačítko „Nadobro smazat“

Očekávaný výstup	Vybraná skupina přání se odstraní a přání zařazená ve skupině se přesunou do výchozí skupiny přání

Tabulka D.13: Přesun přání mazané skupiny do výchozí

D.14 Administrace přání

Název testu	Administrace přání
Cíl testu	Ověření funkce změny stavu a údajů přání administrátorem
Očekávaný výsledek	Změní se stav a údaje přání
Podmínky testu	Přihlášený administrátor a alespoň jedno existující přání
Akce	Zvolení položky menu „Přání“
Očekávaný výstup	Zobrazí se stránka „Přání“ se seznamem přání
Akce	V seznamu aktivních přání změnit u vybraného přání stav na „Nahlášeno“ a jazyk na „EN“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Přání se přesune do seznamu nahlášených přání a jeho jazyk se změní na „EN“
Akce	V seznamu nahlášených přání změnit u vybraného přání stav na „Deaktivováno“ a jazyk na „CS“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Přání se přesune do seznamu deaktivovaných přání a jeho jazyk se změní na „CS“
Akce	V seznamu deaktivovaných přání změnit u vybraného přání stav na „Aktivní“ a jazyk na „EN“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Přání se přesune do seznamu aktivních přání a jeho jazyk se změní na „EN“

Tabulka D.14: Administrace přání

D.15 Administrace uživatelů systému typu uživatel

Název testu	Administrace uživatelů systému typu uživatel
Cíl testu	Ověření funkce změny stavu účtu uživatelů systému typu uživatel administrátorem

Očekávaný výsledek	Stav účtu uživatele systému typu uživatel projde přes všechny stavy účtu až do počátečního bodu
Podmínky testu	Přihlášený administrátor a existující uživatel systému typu uživatel ve stavu účtu „Aktivní“
Akce	Zvolení položky menu „Uživatelé systému“
Očekávaný výstup	Zobrazí se stránka „Systémoví uživatelé“ se seznamem uživatelů
Akce	V seznamu aktivních uživatelů změnit u vybraného uživatele stav účtu na „Nahlášený“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Uživatel se přesune do seznamu nahlášených uživatelů systému typu uživatel
Akce	V seznamu nahlášených uživatelů typu uživatel změnit u vybraného uživatele systému stav účtu na „Deaktivovaný“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Uživatel se přesune do seznamu deaktivovaných uživatelů systému typu uživatel
Akce	V seznamu deaktivovaných uživatelů systému typu uživatel změnit u vybraného uživatele stav účtu na „Čekající“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Uživatel se přesune do seznamu čekajících uživatelů systému typu uživatel
Akce	V seznamu čekajících uživatelů systému typu uživatel změnit u vybraného uživatele stav účtu na „Aktivní“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Uživatel se přesune do seznamu aktivních uživatelů systému typu uživatel

Tabulka D.15: Administrace uživatelů systému typu uživatel

D.16 Administrace uživatelů systému typu administrátor

Název testu	Administrace uživatelů systému typu administrátor
Cíl testu	Ověření funkce změny stavu účtu uživatele systému typu administrátor administrátorem
Očekávaný výsledek	Stav účtu uživatele systému typu uživatel projde přes všechny stavy účtu až do počátečního bodu

Podmínky testu	Přihlášený administrátor a jiný existující uživatel systému typu administrátor ve stavu účtu „Aktivní“ v systému
Akce	Zvolení položky menu „Uživatelé systému“
Očekávaný výstup	Zobrazí se stránka „Systémoví uživatelé“ se seznamem uživatelů
Akce	Zvolení položky „Administrátor“
Očekávaný výstup	Zobrazí se stránka „Systémoví uživatelé“ se seznamem administrátorů
Akce	V seznamu aktivních uživatelů systému typu administrátor změnit u vybraného uživatele stav účtu na „Nahlášený“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Administrátor se přesune do seznamu nahlášených uživatelů systému typu administrátor
Akce	V seznamu nahlášených uživatelů systému typu administrátor změnit u vybraného administrátora stav účtu na „Deaktivovaný“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Administrátor se přesune do seznamu deaktivovaných uživatelů systému typu administrátor
Akce	V seznamu deaktivovaných uživatelů systému typu administrátor změnit u vybraného administrátora stav účtu na „Čekající“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Administrátor se přesune do seznamu čekajících uživatelů systému typu administrátor
Akce	V seznamu čekajících uživatelů typu administrátor změnit u vybraného administrátora stav účtu na „Aktivní“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Administrátor se přesune do seznamu aktivních uživatelů systému typu administrátor

Tabulka D.16: Administrace uživatelů systému typu administrátor

D.17 Administrace komentářů

Název testu	Administrace komentářů
Cíl testu	Ověření funkce změny stavu a údajů komentářů administrátorem

Očekávaný výsledek	Stav komentáře projde přes všechny stavy až do počátečního bodu
Podmínky testu	Přihlášený administrátor a existující aktivní komentář
Akce	Zvolení položky menu „Komentáře“
Očekávaný výstup	Zobrazí se stránka „Komentáře“ se seznamem komentářů
Akce	V seznamu aktivních komentářů změnit u vybraného komentáře stav na „Nahlášeno“ a text komentáře na „Změna textu 1“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Komentář se přesune do seznamu nahlášených komentářů a jeho text se změní na „Změna textu 1“
Akce	V seznamu nahlášených komentářů změnit u vybraného komentáře stav na „Deaktivováno“ a text komentáře na „Změna textu 2“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Komentář se přesune do seznamu deaktivovaných komentářů a jeho text se změní na „Změna textu 3“
Akce	V seznamu deaktivovaných komentářů změnit u vybraného komentáře stav na „Aktivní“ a text komentáře na „Změna textu 3“ a stisknout tlačítko „Uložit změny“
Očekávaný výstup	Komentář se přesune do seznamu aktivních komentářů a jeho text se změní na „Změna textu 3“

Tabulka D.17: Administrace komentářů

E Obsah přiloženého disku

Přiložený disk obsahuje následující položky:

- `Wishlist` – Maven projekt aplikace,
  - `src` – zdrojové kódy aplikace,
  - `target` – přeložené zdrojové kódy aplikace,
  - `data.sql` – soubor příkazů pro naplnění úložiště základní sadou dat,
  - `drop.sql` – soubor příkazů pro smazání vytvořeného úložiště,
  - `README.txt` – postup nastavení a spuštění aplikace ve vývojovém prostředí,
- `ostrikat_2016bach.pdf` – zpráva bakalářské práce.