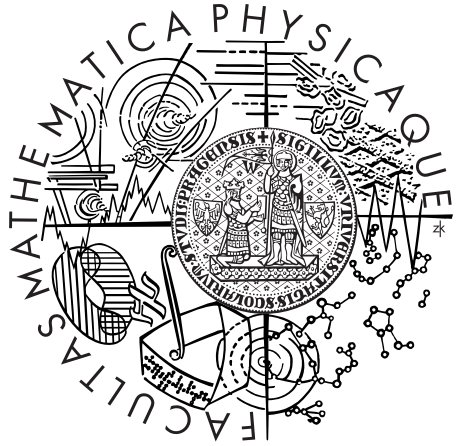
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁRSKA PRÁCA



Radoslav Krivák

Issue tracker s webovým rozhraním

Katedra softwarového inženýrství

Vedúci bakalárskej práce: RNDr. Tomáš Poch
Štúdijný program: Informatika, Obecná informatika

2009

Rád by som sa predovšetkým poďakoval RNDr. Tomášovi Pochovi za vedenie bakalárskej práce, za jeho cenné pripomienky, rady a konzultácie. Ďakujem svojim rodičom a starým rodičom za ich neustálu podporu.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe dňa Radoslav Krivák

Obsah

Názov práce: Issue tracker s webovým rozhraním
Autor: Radoslav Krivák
Katedra (ústav): Katedra softwarového inženýrství
Vedúci bakalárskej práce: RNDr. Tomáš Poch
e-mail vedúceho: Tomas.Poch@mff.cuni.cz

Abstrakt: Predložená práca sa zaoberá analýzou, návrhom a implementáciou issue tracking systému s webovým rozhraním. V úvode je vysvetlené, čo je to issue tracker a aké sú prínosy jeho nasadenia v organizácii alebo firme. Následne je navrhnuté nové riešenie, ktoré je popísané pomocou usecases. Práca prináša prehľad existujúcich technológií a postupov používaných pri vývoji webových aplikácií v jazyku Java. Je popísaná architektúra aplikácie a spôsob použitia jednotlivých technológií. Hlavnou časťou práce je implementácia samotnej aplikácie v jazyku Java za použitia technológií Struts2, Spring, Hibernate a Lucene.

Kľúčové slová: issue tracker, Java, Spring, Struts2, Hibernate

Title: Issue tracking system with web interface
Author: Radoslav Krivák
Department: Department of Software Engineering
Supervisor: RNDr. Tomáš Poch
Supervisor‘ s e-mail address: Tomas.Poch@mff.cuni.cz

Abstract: The present work deals with the analysis, design and implementation of an issue tracking system with web interface. The introduction is to explain what an issue tracker is and what are the benefits of its deployment in the organization or company. Consequently, new solution is designed and described with the help of usecases. The work brings an overview of technologies and patterns used in Java web application development. Architecture of the application and the way selected technologies are used is described. The main part of the work is the implementation of the application itself in the Java language with the help of Struts2, Spring, Hibernate and Lucene technologies.

Keywords: issue tracker, Java, Spring, Struts2, Hibernate

Kapitola 1

Úvod

1.1 Čo je to issue tracker?

Issue tracker je softvérová aplikácia, ktorá slúži na správu a sledovanie životného cyklu issues v rámci nejakého projektu alebo celej organizácie. Issue[1] môže reprezentovať problém, ktorý je potrebné riešiť, úlohu, ktorú je potrebné splniť alebo požiadavok na ktorý je potrebné odpovedať. Medzi základné atribúty issue patria *stav*, *riešiteľ* a *priorita*. Stav udáva polohu issue v životnom cykle, od vzniku issue až po jeho vyriešenie. Riešiteľ je konkrétna osoba zodpovedná vyriešenie issue, prípadne za vykonanie istej práce smerujúcej k jeho vyriešeniu. Priorita určuje dôležitosť issue vo vzťahu k ostatným issues. Jedna osoba je zvyčajne v konkrétnom časovom okamihu zodpovedná za riešenie viacerých úloh. Na základe priority sa rozhodne ktoré issue je treba riešiť skôr. Issues sú obvykle rozdelené do projektov alebo kategórií.

Tento typ aplikácie sa najčastejšie používa v technicky zameraných firmách zaoberajúcich sa vývojom software alebo zákazníckou podporou. Hlavným prínosom použitia issue trackeru je zlepšenie organizácie práce. Manažér môže prostredníctvom systému prideliť svojím podriadeným úlohy a sledovať priebeh ich riešenia. K issue sa postupne pridávajú ďalšie informácie alebo v ňom môže prebiehať komunikácia medzi zadávateľom a riešiteľom prostredníctvom komentárov. Issue tracker pomáha udržovať prehľad o tom aké problémy sa momentálne riešia, kto je zodpovedný za ich riešenie a o tom čo je ešte potrebné urobiť. Systém zároveň slúži ako preukázateľný archív pracovnej komunikácie a zodpovednosti za vykonanú prácu.

[1] V mnohých systémoch sa používa aj názov *ticket* podľa papierových kartičiek pomocou ktorých sa zvykli zadávať úlohy v niektorých tímoch.

1.2 Issue tracker vs. Bug tracker vs. Helpdesk

Na tomto mieste je vhodné zamyslieť sa nad vzťahom issue trackera k iným aplikáciám podobného typu. *Bug tracker* je aplikácia ktorej hlavným zameraním je zber a evidencia chýb v software počas jeho vývoja. *Helpdesk* systém slúži na zadávanie chybových hlásení a zber požiadavkov od klientov. Hranica medzi týmito aplikáciami nie je jasne stanovená. Aj keď majú rozdielne ciele ich funkcionalita sa z veľkej časti prekrýva.

Issue tracker nie je presne definovaný pojem. V tomto texte používame tento pojem v najvšeobecnejšom zmysle, tak ako sme ho popísali vyššie. V tomto zmysle sú bug tracker a helpdesk systém špeciálnym prípadom issue trackera. Toto zovšeobecnenie aplikácií podobného typu do pojmu issue tracker však nezostáva iba v abstraktnej rovine. V rámci zjednodušenia a sprehľadnenia vnútorných procesov alebo konsolidácie sa mnohé firmy snažia využívať len jednu centrálnu aplikáciu – issue tracker – ktorá plní všetky tieto úlohy.

1.3 Prehľad existujúcich riešení

Zorientovať sa vo všetkých existujúcich systémoch nie je jednoduché. Tu predstavíme niekoľko reprezentatívnych populárnych riešení. Vo všetkých prípadoch sa jedná o systémy s ktorými sa dá pracovať cez webové rozhranie. Vyčerpávajúci výčet je možné nájsť v [1].

Bugzilla Je primárne systém na evidenciu softvérových chýb. Vyznačuje sa rýchlosťou a z užívateľského hľadiska nie príliš prívetivým rozhraním. Poskytuje pomerne široké možnosti nastavenia. Jedná sa o open-source aplikáciu napísanú v jazyku Perl.

JIRA Medzi najpoužívanejšie issue tracking systémy patrí JIRA od austrálskej firmy Atlassian. Jedná sa o komerčný produkt s jednorazovým poplatkom za licenciu na jeden server. Atlassian však zadarmo poskytuje licenciu pre open-source projekty, čo značne prispelo k celkovému rozšíreniu tohto produktu. Systém JIRA začal svoju existenciu ako bug tracker, neskôr sa však vyvinul v univerzálny issue tracker. Je to veľmi flexibilný systém ktorý poskytuje bohaté možnosti konfigurácie a prispôsobenia: možnosť pridania nových dátových položiek k issue, systém právomocí založený na projektových rolách a konfigurovateľné workflow (možnosť definovať vlastné stavy issue a ich prechody). Systém JIRA je naprogramovaný v jazyku Java.

zendesk Filozofia tohto produktu je odlišná od predchádzajúcich. Jedná sa o aplikáciu, ktorá beží na serveroch jej tvorcov a je poskytovaná ako služba za mesačný poplatok. Ako názov napovedá, je to primárne helpdesk určený na správu klientskej podpory. Možnosti systému z hľadiska konfigurovateľnosti

sú obmedzené. Zendesk je ale dobrým príkladom systému s jednoduchým používaeľským rozhraním zameraným na bežných používateľov. Zendesk je naprogramovaný v jazyku Ruby a založený na frameworku Ruby on Rails.

Kapitola 2

Analýza a návrh

2.1 Vymedzenie zadania

Vytvoriť issue tracker ktorý by vo funkcionalite a množstve vlastností konkuroval vyššie spomenutým existujúcim riešeniam by si vyžadovalo úsilie ktoré je mimo rozsah školskej práce. Pri návrhu sme sa snažili preto zamerať na niektoré aspekty issue tracikngu ktoré sú podľa nášho názoru riešené v existujúcich aplikáciách nedostatočne.

Ako bolo spomenuté v úvode, mnoho organizácií využívajúcich issue tracker interne má snahu využívať tú istú aplikáciu aj na komunikáciu so svojimi klientami[1]. Nemusí sa jednať len o zber chybových hlásení a technickú podporu ale môže ísť aj o obchodnú komunikáciu a správu nových požiadavkov. To má mnohé výhody oproti tradičnej komunikácii založenej na e-maile. Komunikácia s klientom je tak centrálne evidovaná a preukázateľná. Cesta od požiadavku klienta k jeho vyriešeniu je kratšia keďže klient môže priamo komunikovať s pracovníkom zodpovedným za riešenie.

Snaha použiť issue tracker aj na externú komunikáciu však naráža na problémy, pretože väčšina issue trackerov takýto prípad použitia neberie do úvahy. Modelovým prípadom použitia nášho issue trackera bude preto nasadenie v malej firme, ktorá ho bude využívať interne a zároveň ako bránu na komunikáciu s viacerými klientmi. Existujúce systémy majú v tomto ohľade nedostatky. Systém JIRA napr. nepodporuje zabezpečenie na úrovni jednotlivých dátových položiek issue[2]. Iné systémy (napr. zendesk) neposkytujú dostatočne flexibilnú konfiguráciu na to, aby mohli byť súčasne využívané ako interný issue tracker.

[1] keď hovoríme o klientoch máme na mysli najmä iné firmy, nie koncových užívateľov

[2] tzv. *Field level security*. Bol to historicky najžiadanejší požiadavok na novú funkcionalitu v systéme JIRA. Atlassian ho napokon definitívne zamietol s tým, že implementácia by si vyžadovala obrovské náklady a v podstate prepísanie sytému od základu. `http://jira.atlassian.com/browse/JRA-1330`

2.2 Základné princípy a ciele

- **Prispôsobiteľnosť** Systém by mal byť prispôsobiteľný rôznym spôsobom využitia a špecifickým potrebám rôznych organizácií. Jedným zo spôsobov ako to dosiahnuť je umožniť definovať vlastné typy issue s rôznymi dátovými položkami. (Pri komunikácii s klientami sa hodí napríklad issue typu „Faktúra“ s dátovou položkou cena.)

- **Dôraz na zabezpečenie dát a právomoci** Issue tracker bude obsahovať citlivé údaje. Dôležité je napr. aby jeden klient nemal prístup k informáciám ktoré do systému zadávajú iný klienti. Zabezpečenie a obmedzenie prístupu by malo byť detailne konfigurovateľné z ohľadom na rôznych používateľov. Implementujeme tiež zabezpečenie na úrovni jednotlivých dátových položiek issue.

- **Prehľadnosť a jednoduchosť používania** Problémom mnohých existujúcich issue trackerov je prílišná komplexnosť ich užívateľského rozhrania. To nemusí prekážať technickému personálu firmy, ale mnohí klienti s tým môžu mať problém. To často vedie k tomu že na komunikáciu s firmou naďalej používajú tradičné prostriedky ako e-mail a firma tým prichádza o výhody popísané vyššie. Používateľské rozhranie by malo preto byť jednoduché, prehľadné a intuitívne.

- **Transakcionalita a vyhľadávanie** Transakcionalita jednotlivých akcií v systéme zabezpečí, že stav systému (teda dáta a konfigurácia) bude v každom okamihu konzistentný. Keďže jednou z funkcií issue trackera je to, že slúži ako archív komunikácie, malo by v ňom byť možné kvalitné fulltextové vyhľadávanie.

- **Rozšíriteľnosť** Výsledný systém by mal byť ľahko rozšíriteľný o novú funkcionalitu. Nedali sme si za cieľ vytvoriť aplikáciu s (nevyhnutne) obmedzenou funkcionalitou čo najrýchlejšie ako to len pôjde, ale vytvoriť systém ktorý bude dobrým základom pre ďalšie rozširovanie.

2.3 Definícia entít

Táto podkapitola obsahuje definíciu a popis základných entít v systéme (sú použité anglické názvy kvôli konzistencii s UML diagramami a jazykovým rozhraním aplikácie).

User reprezentuje používateľa systému so svojím prihlasovacím menom a heslom. Systém bude mať viac užívateľov a budeme predpokladať, že užívateľom bude vždy jedna konkrétna osoba. Špeciálny typ užívateľa – administrátor – bude mať prístup k administračnému rozhraniu.

Group združuje používateľov s rovnakým vzťahom k aplikácii resp. k organizácii používajúcej aplikáciu (napr. administrátori, zamestnanci, klienti, zamestnanci konkrétnej firmy). Jeden užívateľ môže patriť do viacerých skupín. Hlavným dôvodom existencie skupín je zjednodušenie a sprehľadnenie správy užívateľov.

Issue je základnou entitou v systéme. Reprezentuje nejaký problém alebo úlohu ktorá je riešená jedným zamestnancom. Každé issue má niekoľko systémových atribútov (dátových položiek) ktoré sú spoločné pre všetky issues. Patria medzi ne *zhrnutie*, *detaily*, *typ*, *priorita*, *dátum vytvorenia*, *dátum poslednej zmeny*, *zadávateľ*, *riešiteľ* a *stav* (status). Issue môže byť v danom okamihu v jednom z týchto štyroch stavov:

New	vyjadruje že issue ešte nebolo evidované zodpovednou osobou (napr. po tom čo klient zadá issue do systému)
Open	vyjadruje že issue je nevyriešené
Resolved	vyjadruje že práca na issue je hotová ale vyžaduje kontrolu
Closed	issue je vyriešené a nevyžaduje ďalšiu pozornosť

Project združuje súvisiace issues. Každé issue patrí práve do jedného projektu. Môže reprezentovať konkrétny softwarový projekt alebo združovať všetky issues súvisiace s jedným klientom.

Project role reprezentuje vzťah užívateľa k projektu (napr. vedúci projektu, vývojár, tester, klient). Jeden užívateľ môže byť v rámci jedného projektu vo viacerých rolách. Členovia rolí (na rolu sa môžeme dívať ako na skupinu užívateľov v rámci projektu) budú nastaviteľný pre každý projekt.

Comment Užívatelia budú môcť k issue pridávať komentára. Viditeľnosť kometára môže byť obmedzená na konkrétnu projektovú rolu.

Issue type Každému issue bude priradený jeho typ (napr. „Bug“, „Support Requets“, „Task“). Zoznam typov issue bude konfigurovateľný.

Custom field Issue môže mať okrem štandardných (systémových) atribútov aj voliteľné dátové položky. Každý custom field bude istého typu (text/číslo/dátum...). Zoznam custom fieldov bude konfigurovateľný administrátorom s ohľadom na typ issue. Viditeľnosť a možnosť editácie konkrétneho custom fieldu bude nastaviteľná s ohľadom na projektové role.

Permission reprezentuje právo užívateľa vykonať istú akciu alebo pristúpiť k dátovej položke. Ku každej permission je možné nastaviť množinu projek-

tových rolí ktoré majú právo vykonať danú akciu. Zoznam permissions v systéme[3]:

- Vidieť projekt
- Vytvoriť issue
- Upraviť issue
- Zmeniť stav issue
- Prideliť issue užívateľovi
- Byť pridelený k issue
- Komentovať issue
- Obmedziť viditeľnosť komentára

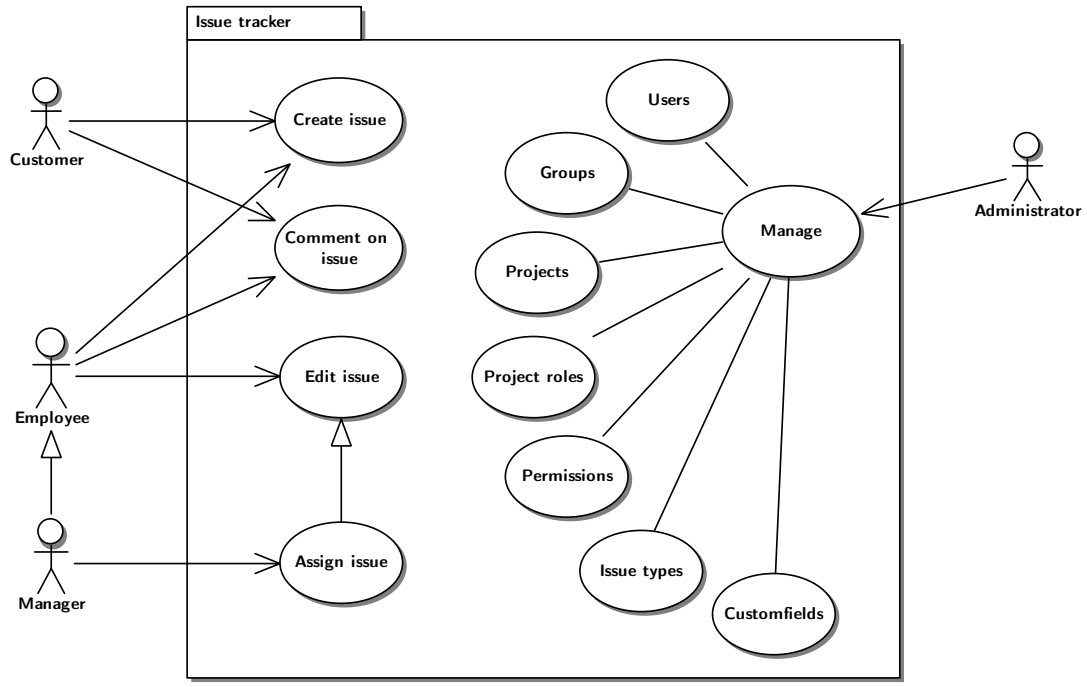
2.4 Usecases

2.4.1 Význam usecases

Usecase zachycuje kontrakt medzi aplikáciou a jej užívateľmi. Najčastejšie má formu scenára – sekvencie akcií ktoré sú vykonávané *aktérmi*. Aktérom môže byť užívateľ (nejakého typu) alebo samotná aplikácia, prípadne jej subsystém. Tvorba usecasov pomáha ujasniť správanie a podobu software ešte pred začatím samotného vývoja.

[3]Zvažovali sme aj pridanie permission pre akciu „zmazať issue“. Takáto potreba by však nemala vzniknúť často keďže issue tracker má slúžiť ako archív. Zmazať issue bude mať právo iba administrátor.

2.4.2 Usecase diagram



2.4.3 Zoznam usecases

Usecase by mal na začiatku obsahovať zoznam a popis aktérov. Za účelom zamedzenia duplikácie definujeme aktérov pre všetky usecasy spoločne. V nasledujúcich usecasoch sa budú vyskytovať tieto štyri typy aktérov:

Zamestnanec je z vecného hľadiska zamestnanec firmy (dodávateľa) používajúceho aplikáciu na komunikáciu s klientami. Z hľadiska aplikácie je to užívateľ v projektovej role „ZAMESTNANEC“. Táto rola má priradené všetky permissions.

Klient je z vecného hľadiska zamestnanec klientskej firmy, ktorý má na starosti komunikáciu s dodávateľom. Predpokladáme že má právo vidieť iba jeden projekt, založený za účelom sledovania požiadavkov klientskej firmy. V tomto projekte je tento užívateľ v projektovej role „KLIENT“ s obmedzenými právomocami.

Administrátor je z vecného hľadiska zamestnanec dodávateľa ktorý sa stará o chod a nastavenia aplikácie. Z hľadiska aplikácie je to osobitný typ užívateľa s prístupom k administračnému rozhraniu.

Aplikácia je samotný issue tracking systém.

Usecase 1: Vytvorenie nového issue klientom

Trigger: Klient má požiadavok alebo otázku na firmu.

1. Klient klikne na tlačítko Create Issue v hlavnom menu.
2. Zobrazí sa menu s výberom projeku a typu issue.
3. Klient vyberie typ issue (vidí iba jeden projekt takže projekt si nemôže vybrať).
4. Aplikácia zobrazí formulár na vytvorenie nového issue. Formulár bude obsahovať:
   - systémové polia: Summary, Details, Priority
   - custom fields pre daný typ issue ktoré má projektová rola KLIENT právo editovať
5. Klient vyplní a odošle formulár.
6. Aplikácia vytvorí nové issue v aktívnom projekte. Aplikácia nastaví status issue na *New* a ako zadávateľa (reporter) issue nastaví užívateľa Klient.
7. Aplikácia presmeruje Klienta za stránku s novovytvoreným issue.

Usecase 2: Vytvorenie nového issue zamestnancom

Rozdiel oproti Usecase 1 je iba v kroku 4.
Formulár bude naviac obsahovať:

- pole Status, typicky pre to aby mohol vytvoriť issue rovno v stave *Open*
- pole Assignee, zoznam možných hodnôt bude obsahovať všetkých užívateľov ktorý majú *Assignable* permission v danom projekte a položku *unassigned*

Usecase 3: Komentovanie issue zamestnancom

Trigger: Zamestnanec chce informovať o priebehu práce, potrebuje sa opýtať na dodatočné informácie alebo chce reagovať na predchádzajúcu otázku.

1. Zamestnanec si otvorí stránku s prehľadom issue ktoré chce komentovať.
2. Klikne na tlačítko Add Comment.
3. Aplikácia zobrazí formulár na vytvorenie nového komentára. Formulár bude obsahovať:
   - pole Text
   - pole Visible for Role, pomocou ktorého bude môcť zamestnanec obmedziť viditeľnosť komentára
4. Zamestnanec vyplní a odošle formulár.
5. Aplikácia pridá komentár k danému issue.
6. Aplikácia presmeruje Zamestnanca za stránku a kotvu s pridaným komentárom.

Usecase 4: Editácia issue zamestnancom

Trigger: Zamestnanec chce upraviť niektoré z dátových polí issue alebo zmeniť jeho status

1. Zamestnanec si otvorí stránku s prehľadom issue ktoré chce editovať.
2. Zamestnanec klikne na tlačítko Edit Issue.
3. Aplikácia zobrazí formulár na úpravu existujúceho issue ktorý bude obsahovať‘ :
   - systémové polia (zhrnutie, detaily, assignee(riešiteľ))
   - custom fields ktoré má Zamestnanec právo editovať
   - pole State ak ho má zamestnanec právo meniť v závislosti na nastavených permissions pre rolu ZAMESTNANEC
4. Zamestnanec vyplní a odošle formulár.
5. Aplikácia zmení hodnoty daného issue a presmeruje Zamestnanca sa stránku s prehľadom issue.

Usecase 5: Vytvorenie novej projektovej role administrátorom

Trigger: Vznikne potreba vytvoriť novú projektovú rolu.

1. Administrátor sa prepne do administračného rozhrania a zvolí položku Roles
2. Aplikácia zobrazí zoznam existujúcich projektových rolí.
3. Administrátor klikne na tlačítko Create Role.
4. Aplikácia zobrazí formulár na vytvorenie novej projektovej role.
5. Administrátor vyplní a odošle formulár.
6. Aplikácia vytvorí novú projektovú rolu a presmeruje Administrátora na zoznam existujúcich projektových rolí.
7. Administrátor zvolí položku Permissions z menu administračného rozhrania
8. Administrátor nastaví novovytvorenej role požadovanú podmnožinu právomocí
9. Administrátor zvolí položku Role members z menu administračného rozhrania
10. Administrátor nastaví novovytvorenej role požadovaných členov pre konkrétne projekty

Usecase 6: Vytvorenie a nastavenie nového projektu administrátorom

Trigger: Vznikne potreba vytvoriť nový projekt.

1. Administrátor sa prepne do administračného rozhrania a zvolí položku Projects.
2. Aplikácia zobrazí zoznam existujúcich projektov.
3. Administrátor klikne na tlačítko Create Project.
4. Aplikácia zobrazí formulár na vytvorenie nového projektu ktorý bude obsahovať:
   - polia Name a Description
   - pole Issue Types: výber podmnožiny zo všetkých typov issue ktoré sa budú môcť používať v novom projekte
5. Administrátor vyplní a odošle formulár.
6. Aplikácia vytvorí nový projekt a presmeruje Administrátora na zoznam existujúcich projektov.
7. Administrátor klikne na odkaz Edit Role Members vedľa novovytvoreného projektu
8. Aplikácia zobrazí formulár na editáciu členov projektových rolí.
9. Administrátor priradí projektovým rolám jednotlivých užívateľov a odošle formulár.
10. Aplikácia zmení členov projektových rolí pre daný projekt.

Usecase 7: Vytvorenie nového customfieldu administrátorom

Trigger: Vznikne potreba zaznamenávať dodatočné informácie k issue.

1. Administrátor sa prepne do administračného rozhrania a z menu zvolí položku Custom Fields
2. Aplikácia zobrazí zoznam existujúcich custom fields.
3. Administrátor klikne na tlačítko Add Custom Field.
4. Aplikácia zobrazí formulár na vytvorenie nového custom fieldu. Formulár bude obsahovať:
   - polia Name a Description
   - pole Typ customfieldu s výberom dátového typu (číslo, text)
   - výber rolí ktoré budú mať právo *vidieť* daný custom field
   - výber rolí ktoré budú mať právo *editovať* daný custom field
   - výber typov issue ktorých dátový model sa rozšíri o daný cusom field
5. Administrátor vyplní a odošle formulár.
6. Aplikácia vytvorí nový custom field a pridá ho do typov issue ktoré boli zaškrtnuté.

Usecase 8: Vytvorenie nového typu issue administrátorom

Trigger: Vznikne potreba vytvoriť nový typ issue na evidenciu špecifickej skupiny požiadavkov alebo úloh.

1. Administrátor sa prepne do administračného rozhrania a zvolí položku Issue Types
2. Aplikácia zobrazí zoznam existujúcich typov issue.
3. Administrátor klikne na tlačítko New Issue Type.
4. Aplikácia zobrazí formulár na vytvorenie nového typu issue. Formulár sa bude obsahovať:
   - polia Name a Description
   - zaškrtávací zoznam customfieldov ktoré bude nový typ issue obsahovať
5. Administrátor vyplní a odošle formulár.
6. Aplikácia vytvorí nový typ issue.

Kapitola 3

Prehľad technológií

Táto kapitola obsahuje prehľad technológií a často používaných postupov pri návrhu a vývoji web aplikácií založených na platforme Java.

3.1 Java & Web

Java je objektovo orientovaný jazyk vyvinutý spoločnosťou Sun Microsystems. Od svojho uvedenia v roku 1996 si získal veľkú popularitu najmä pre vývoj internetových aplikácií. Java EE[1] (aktuálne vo verzii 5) je sada štandardov umožňujúcich vývoj serverových aplikácii v jazyku Java. V nasledujúcich riadkoch popíšeme najdôležitejšie z nich a zasadíme ich do historického kontextu. Budeme sa venovať aj technológiám, ktoré nie sú oficiálnou súčasťou Java EE, ale predstavujú doplnenie resp. alternatívu.

3.1.1 Servlet API

Servlet je komponent ktorý prijíma požiadavok (typicky HTTP) a dynamicky generuje odpoveď [2]. Rozhranie `Servlet` je centrálnou abstrakciou Servlet API, avšak servlety väčšinou vznikajú rozšírením abstraktnej triedy `HttpServlet`. Servlet API tvorí tenkú vrstvu nad bezstavovým HTTP protokolom ktorá naviac pridáva možnosť vytvoriť session. Session umožňuje uchovať stav medzi požiadavkami od toho istého klienta, ktoré prichádzajú v krátkom časovom slede za sebou.

Servlet sa načíta do *servlet kontainera* (je časťou web servera), ktorý ho inicializuje a pre každý požiadavok zavolá metódu `service()`. Servlet API je štandard, ktorý je základom všetkých techlológií Java EE postavených na protokole HTTP a od svojho uvedenia v roku 1997 sa výrazne nevyvíjal. Výraznejšie zmeny prinesie až verzia 3.0.

[1] Java Platform, Enterprise Edition

3.1.2 JSP

Vytvorenie štandardu JSP (Java Server Pages) bolo odpoveďou na popularitu iných technológii na vývoj dynamických webových stránok: PHP a Microsoft ASP. Tieto technológie umožňujú mixovať kód vykonávajúci logiku aplikácie spolu s HTML v jednom súbore, čo bolo považované za ich najväčšiu výhodu. Podobne je to aj u JSP stránok ktoré môžu obsahovať HTML, Java kód a sadu pseudo-html tagov interpretovaných prekladačom JSP, ktorý z JSP stránky vygeneruje Java kód servletu. Tento servlet je následne automaticky skompilovaný a načítaný do servlet kontajnera. Tento proces je transparentný pre programátora.

Nasledujúce verzie štandardu JSP pridali možnosť definovať vlastné tagy. Vznikol štandard JSTL (JSP Standard Tag Library), ktorý definuje bohatú knižnicu tagov pre kontrolné štruktúry alebo spojenie s databázou. S pomocou JSTL je možné napísať celú web aplikáciu vrátane jej logiky ako JSP stránky bez použitia jazyka Java. Ukázalo sa však, že takýto prístup (miešanie aplikačnej logiky s prezentáciou) je slepou uličkou, pretože vedie k chabému dizajnu a ťažko spravovateľným aplikáciám. Súčasný prístup je využívať JSP stránky len ako šablóny na generovanie HTML výstupu.

3.1.3 EJB a Spring

Pojem Java EE (alebo J2EE[2]) je najčastejšie spájaný s technológiou Enterprise Java Beans (EJB). Štandard EJB (prvá verzia uvedená r. 1998) popisuje komponentovú architektúru a framework na budovanie objektovo orientovaných serverových aplikácii v jazyku Java [3]. EJB umožňuje vývoj robustných aplikácií s pokročilými vlastnosťami ako transakcionalita, distributívne transakcie, vzdialené volanie procedúr, udalosti a asynchrónne zasielanie správ.

Enterprise JavaBean je komponent zapuzdrujúci aplikačnú logiku. Je načítaný do *EJB kontajnera* (ktorý je obvykle jednou zo súčastí tzv. aplikačného servera) a využíva jeho služby pre dosiahnutie deklaratívnych transakcií, zabezpečenia etc. Technológia EJB (vo verziách 1.0 až 2.1) sa stala notoricky známou pre jej komplexnosť, aj vytvorenie jednoduchej aplikácie prostredníctvom EJB bolo neúmerne zložité.

Reakciou na to bol vznik takzvaných „odľahčených“ kontajnerov, ktorých najprominentnejším zástupcom je Spring framework. Spring bol navrhnutý tak aby lepšie zodpovedal požiadavkám bežných web aplikácií, bol jednoduchší ako EJB a podporoval jeho najdôležitejšie vlastnosti [4]. Poskytuje napríklad podporu pre rozdelenie aplikácie do komponent a deklaratívne transakcie.

V roku 2006 bola vydaná verzia 3.0 štandardu EJB, ktorá priniesla výrazné zjednodušenie vývoja. Spring, na druhej strane, postupom času preberal čoraz viac funkcionality, ktorú poskytuje EJB. V súčasnosti je jeden z hlavných roz-

[2]platforma bola premenovaná z „J2EE“ na „Java EE“ medzi (po sebe nasledujúcimi) verziami 1.4 a 5

dielov v nasadení: aplikácie založené na Springu môžu bežať na obyčajnom web serveri (napr. Tomcat, Jetty), kdežto aplikácie založené na EJB vyžadujú zložitejší aplikačný server (napr. WebLogic, JBoss).

EJB a Spring riešia problémy spojené s vývojom aplikačnej logiky a jej rozdelením do znovu použiteľných komponent. Problémy spojené s vytvorením rozhrania aplikácie prostredníctvom webových technológií (HTTP, HTML,...), ktoré na vývoj aplikácií neboli pôvodne určené, riešia tzv. web frameworky. Web frameworkom sa budeme venovať nižšie.

3.2 Používané postupy

3.2.1 Viacvrstvá architektúra

Každú aplikáciu ktorá nejakým spôsobom uchováva dáta a prezentuje ich vo forme používateľského rozhrania môžeme rozdeliť na niekoľko logických častí. Hovoríme im vrstvy, pretože každá z nich poskytuje abstrakciu pre vrstvu nad ňou.

Prezentačná vrstva je zodpovedná za používateľské rozhranie.

Aplikačná vrstva obsahuje implementáciu „biznis“ logiky a funkcionality.[3]

Perzistenčná vrstva sa stará o dlhodobé uchovávanie dát a prístup k nim.

Každá z vrstiev by mala byť závislá len na vrstve priamo pod ňou. Okrem týchto troch vrstiev môžeme definovať ešte „vertikálnu“ vrstvu doménových objektov. Doménové objekty reprezentujú objektový model skutočného sveta. Tieto objekty sú používané naprieč spomenutými tromi vrstvami.

3.2.2 Model-View-Controller

Model-View-Controller (MVC) je architektonickým návrhovým vzorom používaným na prezentačnej vrstve. Jeho úlohou je oddeliť aplikačnú logiku od používateľského rozhrania tak aby mohli byť implementované (a upravované) nezávisle na sebe. MVC rozdeľuje implementáciu užívateľského rozhrania do troch častí z ktorých každá má iné zodpovednosti. Časti MVC sú na abstraktnejšej úrovni ako povedzme jednotlivé triedy v jazyku Java.

Model zapuzdruje stav (dáta) a aplikačnú logiku.

View je (vizuálnou) reprezentáciou modelu. V kontexte web aplikácií má obvykle podobu vyrenderovanej HTML stránky.

[3] často sa pre túto vrstvu používa aj názov „vrstva biznis logiky“

Controller prijíma vstup od užívateľa a mapuje ho na updaty modelu. Ďalej vyberá View ktorý sa zobrazí ako odpoveď. Controller tak definuje správanie aplikácie z pohľadu užívateľa a obsahuje prezentačnú logiku.

Koncept MVC má svoj pôvod vo frameworku pre budovanie grafického užívateľského rozhrania v jazyku Smalltalk[5]. Jeho použitie pri vývoji web aplikácii si vyžiadalo isté zmeny oproti pôvodnému návrhu. Sada zodpovedností jednotlivých častí zostala, ich interakcie sa však zmenili. Implementácia tohto vzoru (alebo skôr konceptu) sa však aj v rôznych web frameworkoch značne líši.

3.2.3 Web frameworky

Pri implementácii webových aplikácii riešime množstvo problémov ktoré sa opakujú v každom projekte. Ide napríklad o prevod parametrov HTTP požiadavku (ktoré prichádzajú v textovej podobe) na ich logické dátové typy alebo validáciu užívateľského vstupu. Nemá zmysel riešiť ich pri každom novom projekte od základu. Web frameworky pomáhajú riešiť tieto problémy tým, že predpisujú štruktúru projektu v ktorej je potrebné implementovať iba časti špecifické pre daný projekt. Dobrý web framework by mal automatizovať bežné úlohy, nemal by však obmedzovať možnosti programátora pri riešení komplikovanejších problémov.

Za posledné roky sa na platforme Java vyvinul bohatý ekosystém web frameworkov. Väčšina z nich implementuje koncept MVC. Môžeme ich rozdeliť do dvoch základných kategórii:

1. Požiadavkom riadené frameworky, ktoré na jeden HTTP požiadavok odpovedajú zobrazením jednej HTML stránky.
   - Struts
   - WebWork/Struts2 [4]
   - Spring MVC
   - Stripes
2. Komponentovo orientované frameworky, ktoré rozdeľujú prvky užívateľského rozhrania do komponent. Umožňujú tak vývoj ktorý sa bližšie podobá vývoju desktopových aplikácií.
   - JSF (Java Server Faces)
   - Tapestry
   - Wicket
   - GWT (Google Web Toolkit)

[4]Struts2 nevychádza zo Struts ale je rebrandovaním pokročilejšieho frameworku WebWork

Komponentové frameworky poskytujú vyššiu úroveň abstrakcie a uľahčujú vývoj zložitejších používateľských rozhraní. Požiadavkom riadené frameworky na druhej strane lepšie zodpovedajú paradigme webu: požiadavok/odpoveď.

3.2.4 Dependency injection

Dependency injection je proces kedy do komponentu vložíme jeho závislosti (teda. komponenty resp. jednou z aplikácií obecnejšieho princípu nazývaného *Inversion of Control* (IoC) [6].

Tradičný prístup je, že si komponent sám zadováži svoje závislosti. To môže urobiť vytvorením nových objektov alebo nejakou inou formou napr. získaním objektu od zodpovedajúcej Factory. V prípade použitia dependency injection sa táto zodpovednosť prenesie na IoC kontajner. IoC kontajner spravuje životný cyklus komponentov a pri ich vytvorení im poskytne zodpovedajúce závislosti na základe konfigurácie pomocou metadát (vo forme XML alebo anotácie) alebo konvencie.

Použitie dependency injection vedie k čistejšiemu kódu pretože závislosti stačí len deklarovať. Vývojár s tak nemusí starať o vzájomné poprepájanie komponent a môže sa sústrediť na implementáciu logiky daného komponentu. Ďalšou výhodou je (kvalitatívne) zníženie závislosti medzi jednotlivými komponentmi. Umožňuje deklarovať závislosť iba na rozhraní a konkrétnu implementáciu zvoliť pomocou konfigurácie bez zásahu do kódu daného komponentu.

Dependency injection je tradične podporovaná vo frameworku Spring a od verzie 3.0 aj v EJB.

3.2.5 Objektovo-relačné mapovanie

Relačné databáze sú v dnešnej dobe stále najpoužívanejším riešením prezistencie dát. Jedným z dôvodov je existencia vysoko kvalitných open-source produktov ako PostgreSQL a MySQL. V objektovo orientovaných jazykoch pracujeme s objektovým modelom dát. Medzi relačným a objektovým modelom dát sú však fundamentálne rozdiely. V objektovom modeli sú vzťahy vyadrené pomocou priamych referencií, v reačnom modei prostredníctvom primárnych a cudzích kľúčov. V relačnom modeli ďalej neexistujú koncepty zapúzdrenia, dedičnosti a polymorfizmu.

Úlohou ORM nástroja je preklenúť tieto rozdiely a umožniť objektovo orientovaný vývoj. Použitia ORM takisto prináša zrýchlenie vývoja. Na implementáciu perzistencie je potrebných výrazne menej riadkov kódu ako pri priamom prístupe k databáze pomocou JDBC. Ďalšou výhodou použitia ORM je, že sa aplikácia stane nezávislou na konkrétnom RDBMS[5]. SQL príkazy sú ORM nástrojom generované v SQL dialekte zodpovedajúcom použitej databáze. Nevýhodou je zvýšenie komplexnosti aplikácie. ORM nástroje sú pomerne komplikované a v konečnom dôsledku nikdy neposkytujú dokonalú abstrakciu nad relačnou databázou.

[5]Relational DataBase Management System

Medzi najrozšírenejšie ORM nástroje patria Hibernate, TopLink, OpenJPA a iBatis[6]. Ako súčasť štandardu EJB 3.0 bolo vydané rozhranie na prácu s ORM nástrojmi JPA (Java Persistence API). Spomenuté ORM nástroje (okrem iBatis) implementujú toto rozhranie avšak pridávajú aj neštandardné možnosti nad jeho rámec.

[6]iBatis je jednoduchším nástrojom, ktorý sa sústreďuje najmä na mapovanie relácia/objekt a neplatí preňho všetko čo bolo povedané vyššie

Kapitola 4

Architektúra aplikácie

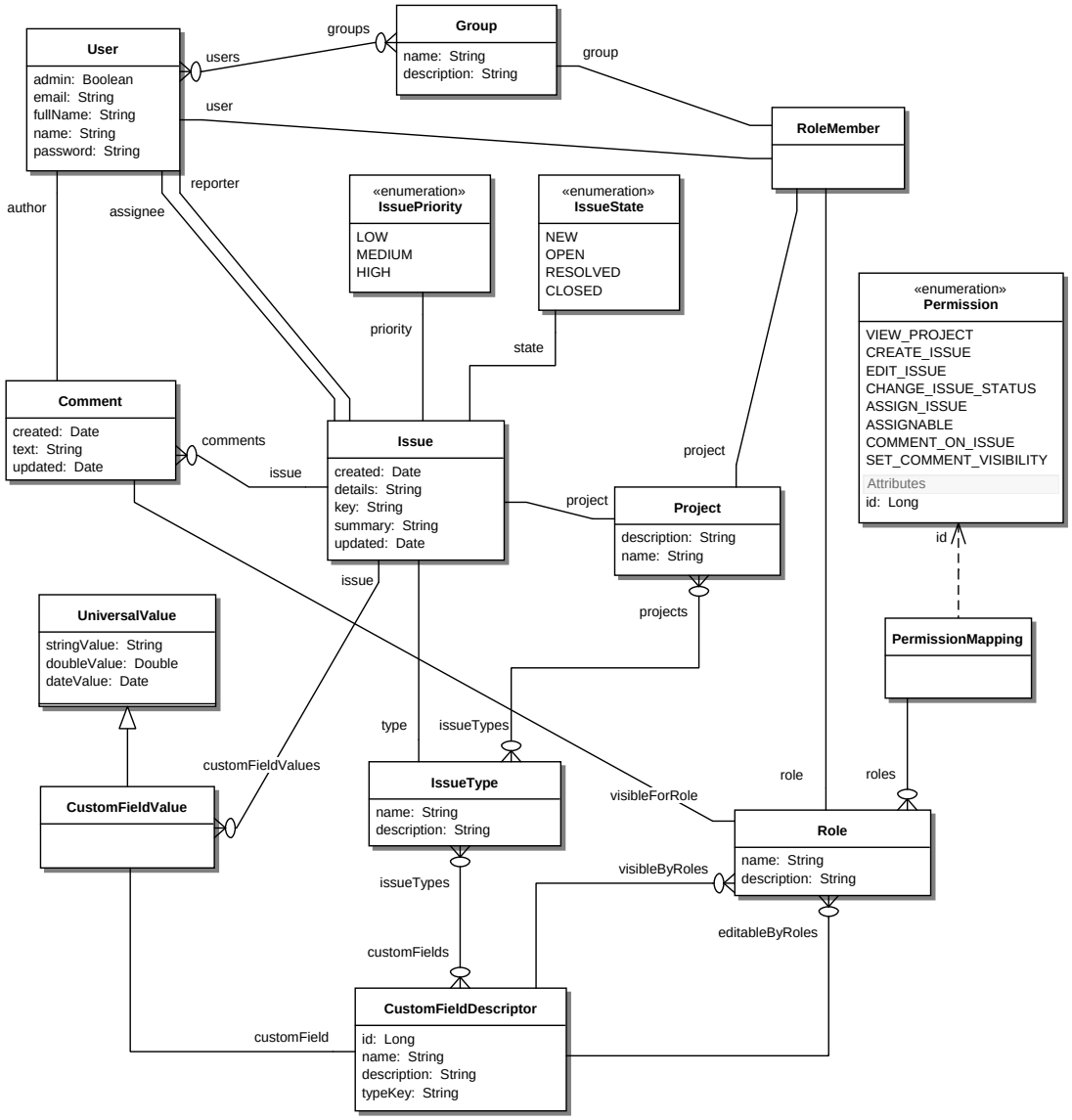
Táto kapitola sa zaoberá popisom architektúry aplikácie.

4.1 Doménový model

Keďže sme sa rozhodli použiť ORM nástroj, návrh aplikácie sme mohli začať definovaním objektového doménového modelu (a nie databázového modelu ako je pri web aplikáciách zvykom). Doménové objekty – entity – a ich vzťahy znázorňuje Obr. 4.1.

Centrálnou entitou celého systému je Issue. Issue patrí práve do jedného projektu a má zadávateľa(*reporter*) a riešiteľa(*assignee*) ktorými sú jednotlivý užívatelia. Issue nemusí mať vždy riešiteľa a v takom prípade je považované za nepridelené (unassigned). Issue je nejakého typu (*IssueType*) a podľa toho má určený zoznam custom fieldov (*CustomFieldDescriptor*) v ktorých môže, ale nemusí mať vyplnené hodnoty (*CustomFieldValue*).

Ďalšou zásadnou entitou je projektová rola (*Role*). Pomocou rolí je určené právo vykonávať jednotlivé akcie (permissions), viditeľnosť/editovateľnosť jednotlivých custom fieldov a môže byť obmedzená viditeľnosť komentára. Príslušnosť k danej role v danom projekte je vyjadrená pomocou mapovacej entity (a v konečnom dôsledku databázovej tabuľky) *RoleMember*, ktorá vyjadruje trojitý vzťah User-Role-Project resp. Group-Role-Project. V prípade, že v tejto trojici chýba Project (hodnota je `null`), je daný užívateľ resp. skupina považovaná za člena danej role v každom projekte.



Obr. 4.1: *Vzťahy medzi doménovými objektami*

Poznámky k diagramu:

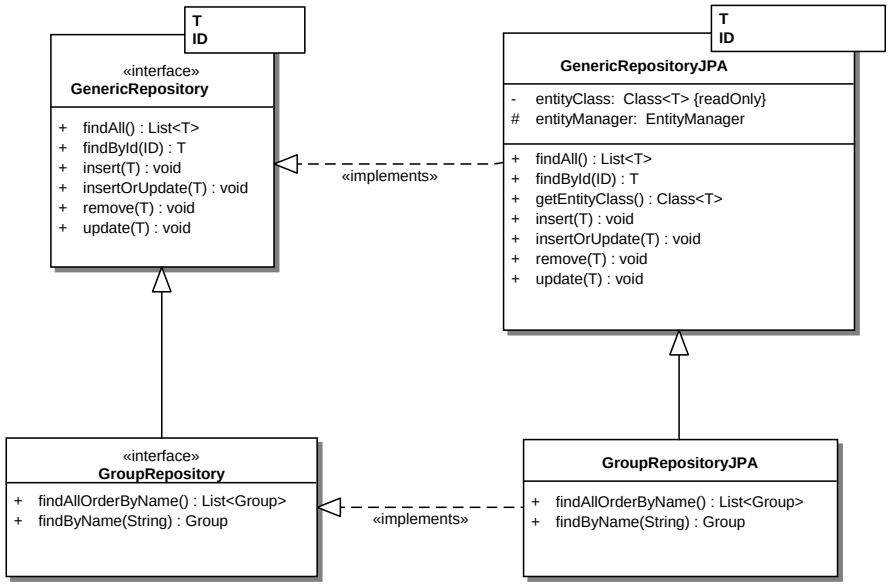
- **CustomFieldDescriptor:** reprezentuje custom field nejakého typu. Typ je určený pomocou `typeKey`.
- **RoleMember:** reprezentuje člena role v danom projekte. Členom môže byť konkrétny užívateľ alebo skupina (exkluzívne).
- **PermissionMapping:** odkazuje na všetky role ktoré majú pridelenú danú permission. Na permission odkazuje nepriamo pomocou svojho `id`.

4.2 Perzistenčná vrstva

Ako úložište dát sme použili relačnú databázu s ktorou komunikujeme prostredníctvom ORM nástroja Hibernate. Pri vývoji sme používali RDBMS PostgreSQL, avšak naša aplikácia môže bežať nad ktoroukoľvek z databáz ktoré Hibernate podporuje. S Hibernate sme pracovali pomocou rozhrania JPA, pokiaľ to bolo možné. Jednotlivé doménové objekty – entity sú namapované do databázových tabuliek pomocou JPA anotácii.

Základnou abstrakciou na perzostenčneej vrstve je *repository*. Tento návrhový vzor je často nazývaný aj DAO – Data Access Object [9]. Repository poskytuje objektovo orientované rozhranie pre prácu s perzistenciou jedného doménového objektu – entity. Všetky repositories dokopy tvoria rozhranie perzistenčnej vrstvy s ktorým pracuje aplikačná vrstva. Teoreticky by bolo možné vytvoriť implementáciu repositories aj nad iným dátovým úložišťom ako relačnou databázou prípadne, avšak v praxi s tým nepočítame. Hlavný prínos použitia tohto vzoru (Repository-/DAO) vidíme v jasnom oddelení kódu súvisiaceho s perzistenciou od aplikačnej logiky.

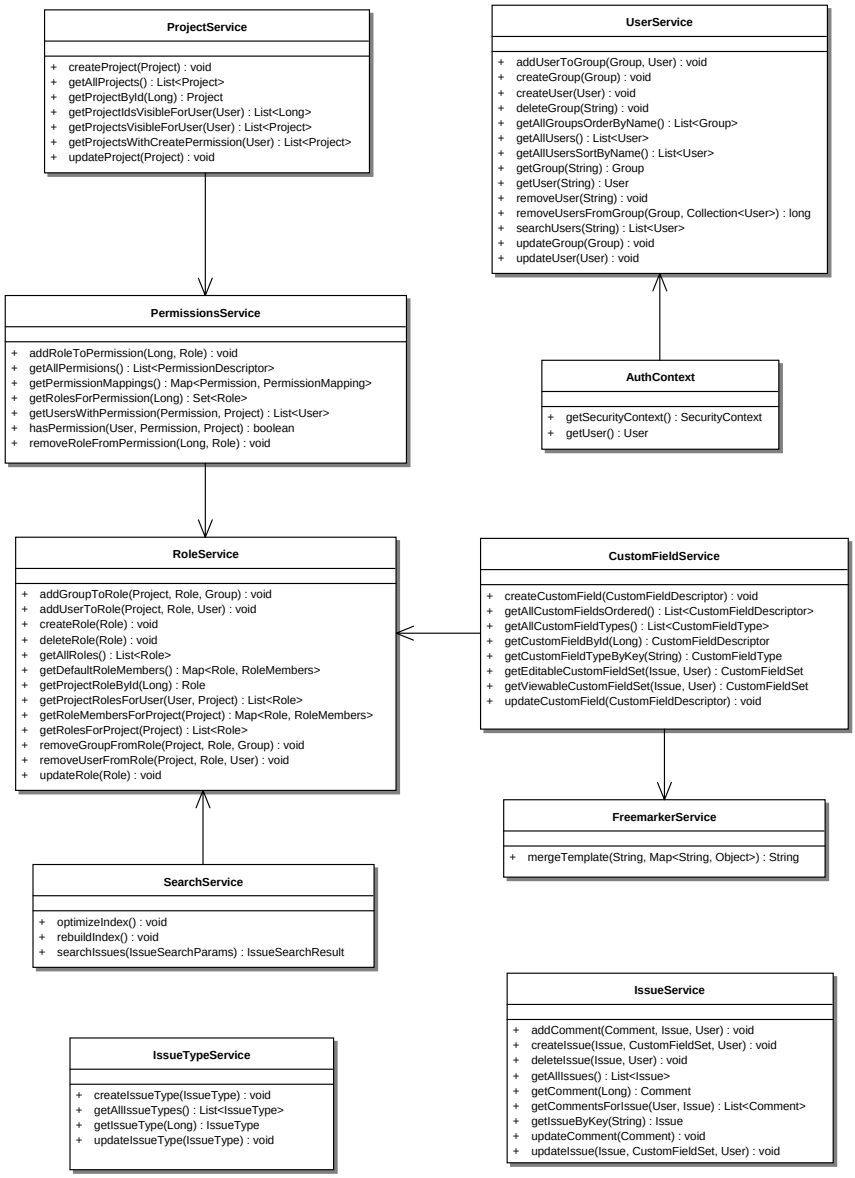
Všetky repositories obsahujú rovnaké základné metódy (insert/update/remove/...). Aby sme sa vyhli ich opakovanej implementáci, vytvorili sme generickú implementáciu `GenericRepositoryJPA`. Od tejto generickej implementáce dedia jednotlivé repositories a pridávajú metódy špecifické pre dané entity. Ako to vyzerá v prípade repository pre entitu Group znázorňuje diagram. Vďaka Java generics bolo možné dosiahnuť typovo bezpečné znovupoužitie.



Obr. 4.2: *Vzor implementácie repositories (`T`: typ entity, `ID`: typ primárneho kľúča)*

4.3 Aplikačná vrstva

Aplikačná logika a funkcionalita je rozdelená do komponentov nazývaných Services. Jedna service prezentuje funkcionalitu súvisiacu s jednou entitou alebo blízko súvisiacimi entitami. Na tejto vrstve je dosiahnutá transakcionalita jednotlivých metód za pomoci deklaratívnych transakcií frameworku Spring.



Obr. 4.3: *Rozhrania jednotlivých services*

Súčasťou aplikačnej vrstvy je aj implementácia funkcionality fulltextového vyhľadávania.

4.3.1 Fulltextové vyhľadávanie

Na implementáciu fulltextového vyhľadávania sme sa rozhodli použiť knižnicu *Apache Lucene*. Alternatívne sme mohli použiť možnosti fulltextového vyhľadávania, ktoré poskytuje niektorý z RDBMS. Možnosti ktoré poskytujú tieto integrované nástroje sú však obvykle rádovo slabšie a prišli by sme takto o nezávislosť na použitej databázovej platforme. Ďalšou alternatívou bolo použitie search enginu *Egothor*, ktorý je vyvýjaný na MFF UK pod vedením RNDr. Lea Galamboša, Ph.D. Dôvody prečo sme sa rozhodli pre Lucene sú existencia (čiastočnej) integrácie s Hibernate v podobe *Hibernate Srarch* a dostupnosť publikácií [7][8].

Lucene spravuje svoj vlastný vyhľadávací index. Vždy keď sa v nejakom dokumente zmenia dáta je potrebné vykonať update indexu, aby vyhľadávanie vracalo aktuálne informácie. V tomto pomáha rozšírenie frameworku Hibernate nazývané *Hibernate Search*, ktoré sa stará o automatický update indexu Lucene, vždy keď sa dáta zmenia v databáze.

Za použitia Lucene sme implementovali vyhľadávanie s pokročilou funkcionalitou. Vo vyhľadávacom rťazci je možné použitie operátorov (AND, OR) a wildcards (?,*). Výsledky vyhľadávania sú zoradené podľa relevancie. Pomocou Lucene bolo možné nakonfigurovať váhu jednotlivým poliam issue. Predpokladajme že hľadáme jedno konkrétne slovo, issues v ktorých sa hľadané slovo vyskytuje v poli *summary* budú obrazené vo vyhľadávacích výsledkoch vyššie ako issues ktoré obsahujú slovo iba v poli *details*.

Pri implementácii vyhľadávania sme narazili na zásadný problém. Chceli sme, aby bolo možné vyhľadávať nie len v zhrnutí a detailoch issue, ale aj v komentároch. Nie všetky issues sú však viditeľné pre všetkých užívateľov a nie všetky komentáre sú viditeľné pre všetkých užívateľov, ktorý vidia dané issue. Vyhľadávanie by malo rešpektovať tieto bezpečnostné pravidlá. V prípade issues to nie je problém, výsledky vyhľadávania stačí zredukovať iba na issues ktoré sú vidteľné pre hľadajúceho. Je však neprípustné, aby sa do výsledkov vyhľadávania dostalo issue na záklde toho, že obsahuje hľadané slovo v komentári, ktorý hľadajúci nemá právo vidieť. Riešením bolo nakoniec vytvorenie samostatného vyhľadávacieho indexu pre komentáre. V komentároch sa vyhľadáva osobitne (a tak môžu byť rešpektované bezpečnostné pravidlá pre komentáre) a výsledky hľadania sa zlúčia s hľadaním v issues na základe issue id.

4.4 Prezentačná vrstva

Na prezentačnej vrstve sme použili web framework *Struts2* s pomocou ktorého sme vytvorili webové rozhranie aplikácie. Struts2 implementuje variantu MVC

nazývanú *front controller*. Všetky HTTP požiadavky sú na začiatku spracované centrálnym Controllerom, ktorý je súčasťou frameworku. Tento front controller prevedie každý HTTP požiadavok na základe URL adrsy na invokáciu zodpovedajúcej *akcie*. Akcia z hľadiska MVC triády reprezentuje model a je tvorená objektom s (minimálne) jednou metódou vracajúcou `String` (akčná metóda). Akčná metóda zapuzdruje volania metód z aplikačnej vrstvy a na základe jej návratovej hodnoty sa určí JSP stránka resp. iný druh odpovede (napr. presmerovanie). Akcia tak plní z časti aj úlohu Controllra.

Struts2 ďalej umožňuje, aby bolo v jednej akcii (resp. triede definujúcej akciu) definovaných viac akčných metód. To sme využili na vytvorenie generickej CRUD (Create/Update/Delete) akcie (trieda `CrudActionSupport`). Jej rozšírením vznikajú CRUD akcie pre jednotlivé doménové objekty. Takto sme sa vyhli duplikácii kódu keďže mnoho akcií má CRUD charakter.

4.5 Zhrnutie

Aplikácia je rozdelená do troch vrstiev ktoré definujú tri základné typy komponentov: Repository (perzistenčná vrstva), Service (aplikačná vrstva) a Action (prezentačná vrstva). O životný cyklus všetkých komponentov a dependency injection sa stará IoC kontainer frameworku Spring. Perzistencia dát je riešená za pomoci ORM frameworku Hibernate a na vytvorenie webového rozhrania používame MVC framework Struts2. Fulltextové vyhľadávanie je riešené za pomoci knižice Lucene a Hibernate Search.

Je na mieste položiť si otázku, či použitie všetkých týchto frameworkov je naozaj potrebné pre implementáciu takejto relatívne malej aplikácie. Odpoveď je, samozrejme, nie. Celú aplikáciu so všetkou jej funckionalitou (snáď okrem fulltextového vyhľadávania) by sme mohli implementovať len za použitia Servletov, JSP stránok a JDBC. Bolo by to však pracnejšie.

Použitie frameworkov a knižníc nám prináša ďalšie výhody, ktoré sme spomenuli v tejto a predošlej kapitole. Problémy, ktoré tieto frameworky riešia by sme v konečnom dôsledku museli riešiť sami. Frameworky to však robia overeným a stabilným spôsobom ktorý bol časom zdokonaľovaný (vo všetkých prípadoch sa jedná o open source produkty).

Treba však povedať, že výber frameworkov (a rozhodnutie ich použiť) nebol nutne riadený iba pragmatickými pohnútkami. Išlo nám aj o získanie skúseností s týmito v praxi používanými nástrojmi.

Kapitola 5

Záver

Práca sa zaoberala návrhom a implementáciou issue tracking systému. Ciele ktoré sme si stanovili na začiatku sa nám podarilo splniť. Aplikácia je prispôsobiteľná rôznym prípadom použitia k čomu prispieva najmä možnosť definovať vlastné typy issues s rôznymi dátovými poliami (custom fields). V systéme je implementovaný pokročilý systém právomocí (permissions) a obmedzenia prístupu založený na projektoch a projektových rolách. Tieto právomoci sú plne konfigurovateľné pre jednotlivých používateľov a projekty. Je možné obmedziť viditeľnosť jednotlivých custom fieldov (field level security) a komentárov. Najnáročnejšou bola implementácia fulltextového vyhľadávania a to tak, aby rešpektovalo viditeľnosť jednotlivých issues a komentárov pre hľadajúceho.

Na začiatku zme si taktiež dali za cieľ aby bol systém ľahko rozšíriteľný o novú funkcionalitu. To sme dosiahli vytvorením prehľadnej trojvrstvej komponentovej architektúry. Rýchlejšej implementácii novej funkcionality pomôže aj to že sme sa snažili vyhnúť duplikácii kódu a vytvorili sme znovupoužiteľné generické komponenty (`GenericRepository` a `CrudActionSupport`).

Na systéme je stále čo zlepšovať. Väčšia prispôsobiteľnosť by sa mohla dosiahnuť pridaním možnosti definovať vlastné stavy issue a úrovne priority. Ďalším užitočným zlepšením by bola možnosť pridáať k issue súborové prílohy. Za najväčší funkčný nedostatok systému však považujeme neexistujúcu integráciu s e-mailom. V prípade ďalšej práce na systéme by vývoj pokračoval implementáciou e-mailových notifikácií a ovládaním systému (zakladanie issues, pridávanie komentárov) pomocou zasielania e-mailov.

Literatúra

[1] priebežne aktualizovaný zoznam dostupný online: `http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems`.

[2] Rajiv Mordani: *Java^{TM} Servlet Specification 3.0, Proposed Final Draft*, dostupné online: `http://java.sun.com/products/servlet/`, 15.4.2009.

[3] EJB 3.0 Expert Group: *Enterprise JavaBeans 3.0 Specification, Final Release*, dostupné online: `http://java.sun.com/products/ejb/`, 2006.

[4] Juergen Hoeller, Rod Johnson: *Expert One-on-One^{TM} J2EE^{TM} Development without EJB^{TM}*, Wiley Publishing, Inc., Indianapolis, 2004.

[5] Trygve Reenskaug: *Models-Views-Controllers*, dostupné online: `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`, 10.12.1979.

[6] Martin Fowler: *Inversion of Control Containers and the Dependency Injection pattern*, dostupné online: `http://www.martinfowler.com/articles/injection.html`, 2004.

[7] Erik Hatcher, Otis Gospodnetić: *Lucene in Action*, Manning Publications Co., Greenwich, 2005.

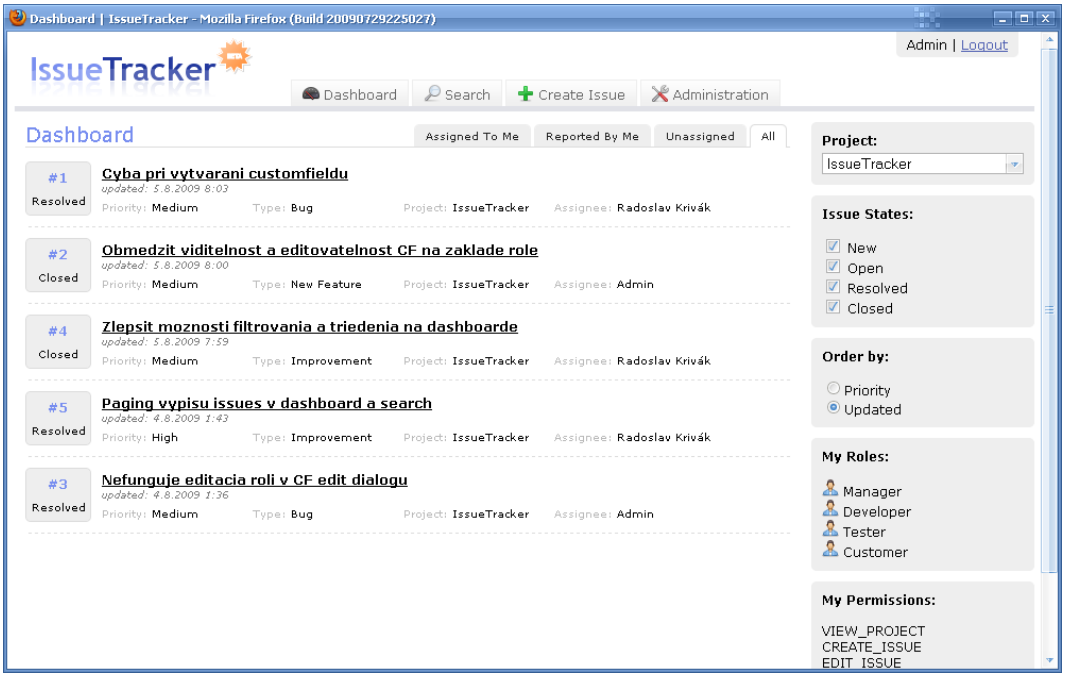
[8] Emmanuel Bernard, John Griffin: *Hibernate Search in Action*, Manning Publications Co., Greenwich, 2008.

[9] William Crawford, Jonathan Kaplan: *J2EE Design Patterns*, O' Reilly & Associates, Inc., Sebastopol, 2003.

Dodatok A

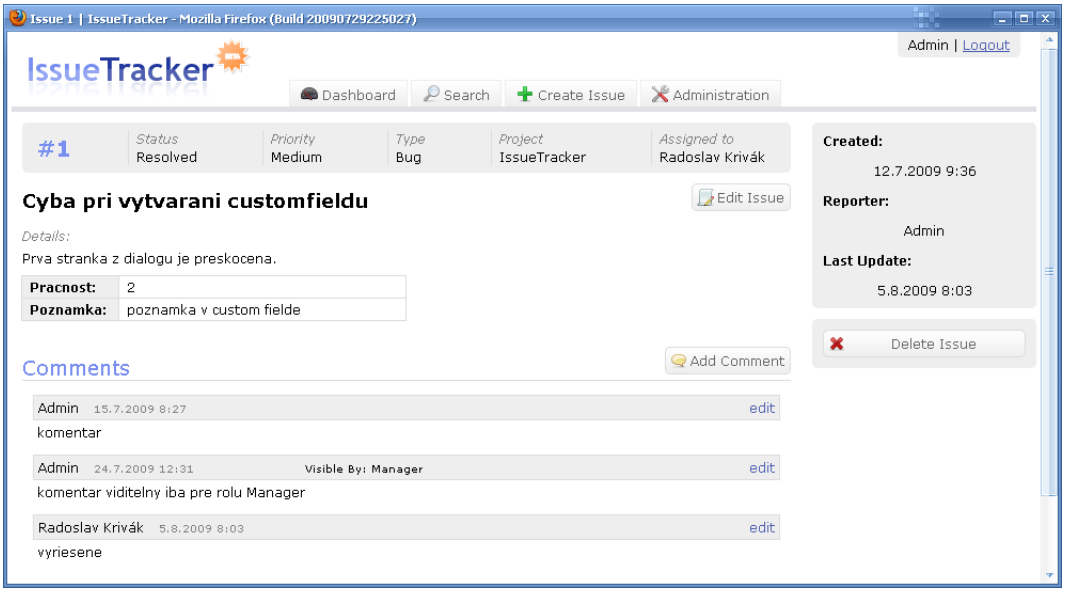
Používateľská dokumentácia

Issue tracker má prehľadné a intuitívne rozhranie, ktoré je do značnej miery samopopisujúce. Tu prinášame popis základných obrazoviek systému a hlavne syntaxe vyhľadávacích reťazcov.



Obr. A.1: *Dashboard*

Dashboard je hlavný panel ktorý poskytuje užívateľovi prehľad o aktuálnych issues ktoré s ním súvisia. Má šryri taby: issues priradené mne (assigned to me), vytvorené mnou(reported by me), nepriradené (unassigned) a všetky (all). Issues je ďalej možné filtrovať na základe projektu alebo stavu a zoradiť podľa priority alebo dátumu poslednej zmeny.



Obr. A.2: *Issue View*

Issue View prezentuje informatívny pohľad na jedno konkrétne issue. V hornej časti sú systémové dátové položky, smerom dolu nasleduje zhrnutie, detaily a tabuľka s hodnotami custom fieldov. Pod tým sú komentáre pridané k issue.

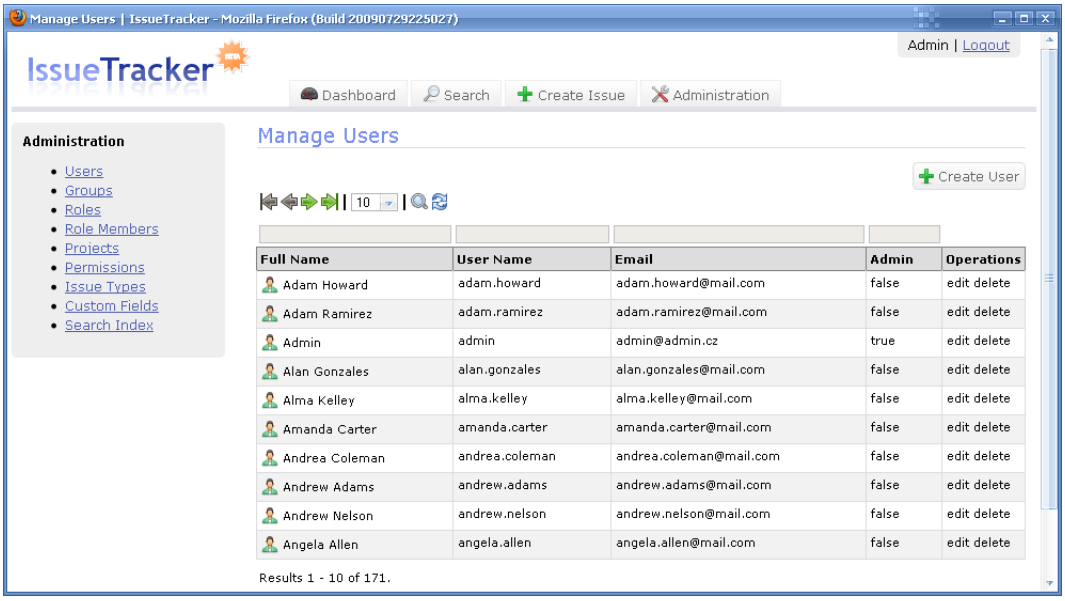
Search Fulltextové vyhľadávanie issues. Vo vyhľadávacom reťazci je možné použiť operátory `AND`, `OR`, `+`, `-`, wildcardy `*` a `?` a uzátvorkovanie. Operátory `AND` a `OR` majú svoj bežný význam. Operátor `+` sa dáva pred reťazce, ktoré sa musia vyskytnúť v texte, operátor `-` naopak pred reťazce ktoré sa nesmú vyskytnúť. Wildcard `?` nahradzuje práve jeden znak a `*` ľubovoľný počet znakov. Nesmú sa vyskytovať na začiatku slova.

Príklady na vyhľadávacie reťazce:

- „`+i*ue +tra?ker`“ je ekvivalentné s „`i*ue AND tra?ker`“
- „`i*ue tra?ker`“ je ekvivalentné s „`i*ue OR tra?ker`“
- „`+(issue AND tracker) -(jira OR bugzilla)`“

Systém vyhľadáva iba v issues a komentároch ktoré má užívateľ právo vidieť.

Create Issue Zobrazí dvojdielny formulár na vytvorenoie nového issue. Najprv umožní vybrať projekt a typ issue a potom vyplniť jednotlivé dátové položky issue.



Obr. A.3: *Administrácia používateľov*

Administration Administračné rozhranie k správe konfigurácie systému. Jednotlivé položky v menu sú samopopisujúce.

Dodatok B

Kompilácia a nasadenie

Na priloženom CD sa zdrojové kódy nachádzajú v adresári `issuetracker/project`. Ako build nátroj je použitý systém Maven[1]. Maven sa stará o automatické sťahovanie závislostí projektu (knžníc) z verejných repozitárov. Niektoré knižnice na ktorých je závislý tento projekt sa nenachádzajú vo verejných repozitároch, preto je na CD v adresári `maven_repositiry` lokálny repozitár so všetkými potrebnými závislosťami. Tento adresár je potrebné pred kompiláciou skopírovať na pevný disk. Týmto príkazom sa projekt skompiluje a zabalí do WAR archívu:

```
mvn package -Dmaven.repo.local=<maven_repository>
```

Na priloženom CD sa nachádza už skompiovaný WAR archív s web aplikáciou v adresári `issuetracker/bin`. Pred nasadením na web server je možné/potrebné nakonfigurovať systém v niekoľkých súboroch:

`WEB-INF/applicationContext.xml`
Nastavenie pripojenia k SQL databáze. Nastavenie predvoleného hesla pre administrátora.[2]

`META-INF/persistence.xml`
Nastavie umiestnenia vyhľadávacieho indexu (nejaký adresár na disku s právom na zápis).

Zmeny v týchto súboroch je možné vykonať pred kompiláciou alebo rovno vo WAR archíve (ktorý je v podstate zip archíiv s príponou war).

[1] `http://maven.apache.org/`

[2] Systém pri nasadení na čistú databázu automaticky vytvorí užívateľa s loginom `admin` a heslom ktoré bude nastavené (predvolene `admin`))

Dodatok C

Demo

Na adrese `http://bedna.pod.cvut.cz:8080/issuetracker` je spustená online demo verzia aplikácie naplnená testovacími dátami. Testovacie dáta reprezentujú modelový prípad použitia vo firme, ktorá issue tracker využíva interne ako aj na komunikáciu s inými firmami.

Prihlasovacie údaje (login/heslo):

`admin/admin`	pre administrátora
`onlio.user/heslo`	pre užívateľa reprezentujúceho klienta s právom vidieť iba jeden projekt