SLOVENSKÁ TECHNICKÁ UNIVERZITA

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

ILKOVIČOVA 3, 812 19 BRATISLAVA

HIERARCHICKÁ WIKI S PRÁVAMI

(TÍMOVÝ PROJEKT)

Autori: Bc. Miroslav Kacera
Bc. Petra Majzúnová
Bc. Martin Repta
Bc. Miroslav Soha
Bc. Peter Študent
Bc. Ján Tóth

Vedúci tímu:	Ing. Ján Suchal
Tím:	AttackHere (č.10)
Študijný program:	Softvérové inžinierstvo a Informačné systémy
Školský rok:	2009/2010

OBSAH

ÚVOD

Tento dokument predstavuje dokumentáciu riadenia k tímovému projektu „Hierarchická wiki s právami“ realizovaného tímom číslo 10. Projekt je vyvíjaný pomocou agilnej metodiky SCRUM, pričom vývoj je rozdelený do dvojtýždňových šprintov. Obsah predkladaného dokumentu tvorí ponuka k realizovanému projektu, informácie o členoch tímu a ich úlohách v rámci tímu a štábna kultúru. Prílohu dokumentu tvoria zápisnice zo stretnutí tímu.

PONUKA

ČLENOVIA TÍMU

BC. MIROSLAV KACERA

Absolvoval som bakalárske štúdium na FIIT, STU v Bratislave v odbore Informatika. Moja bakalárska téma bola zameraná na rozhranie človek-stroj, kde som pracoval s vnorenými systémami. Momentálne pracujem už vyše dvoch rokov vo firme Orange, kde som vyvíjal podpornú aplikáciu na testovanie dátových služieb mobilných zariadení. Znalosti a technológie: C/C++, MFC, Java, Assembler, HTML, CSS, SQL, Grafické nástroje a nástroje na spracovanie videa, Windows Mobile SDK

BC. PETRA MAJZÚNOVÁ

Bakalárske štúdium som absolvovala na FIIT, STU v Bratislave v odbore Informatika. Mám ročnú skúsenosť s vývojom webových aplikácií pod platformou .NET (ASP .NET, C#, MSSQL, AJAX). Druhým rokom sa venujem databázovým technológiám, konkrétne vývoju aplikácií v jazyku PL SQL, v rámci projektov pre Orange. Ďalej mám skúsenosti s databázou Mysql a Postgree SQL. Zaoberám sa tvorbou webových aplikácií. Napr. Vývoj stránky tt.elsa.sk a mnoho iných projektov, kde uplatňujem svoje znalosti s technológiami PHP, HTML, CSS, Javascript, AJAX a XML.

BC. MARTIN REPTA

Bakalárske štúdium som absolvoval na FIT VUT v Brne, kde som mal možnosť, za tri roky štúdia, sa stretnúť a programovať v jazykoch Assembler, C, C++, Lisp, Prolog, Java, OpenGL, VHDL, paralelné programovanie v jazyku pm2 a vyskúšať si prácu s databázovými systémami MySql, MSSQL a vo väčšej miere Oracle. Vo voľnom čase sa zaoberám webovými technológiami, s ktorými som začal už na strednej škole, od HTML, CSS, XTHML, PHP, Javascript a ASP.NET v ktorom som vytvoril taktiež svoju bakalársku prácu - Genealogický informáčný systém pre generovanie a správu rodokmeňov. Počas štúdia v Brne som externe pracoval pre firmu CZ Online Webdesign.

BC. MIROSLAV SOHA

Som absolventom bakalárskeho programu Informatika na FIIT, STU v Bratislave. Svoju bakalársku tému som vypracoval na tému „Rozdeľovanie práce pomocou multiagentových systémov“. Mám skúsenosti s organizovaním väčších skupín ľudí nakoľko som sa podieľal na príprave viacerých stretnutí, programov a výročí v spolupráci s LKS (Liptovské kultúrne stredisko). Momentálne pracujem pre firmu Alcatel-Lucent Bratislava. Znalosti a technológie: C, C++, Java, HTML, CSS, PHP, MySQL, Dizajn (Photoshop, Corel)

BC. PETER ŠTUDENT

Bakalárske štúdium som úspešne absolvoval na FIIT, STU v Bratislave. V minulosti som svojimi článkami z oblasti IT pravidelne prispieval do viacerých médií ako sú napríklad denník SME, ITnews.sk alebo disk.sk. Už viac ako rok pôsobím popri škole ako vedúci vývojového tímu v rádiu Impuls, kde spoločne so svojim tímom a s využitím agilnej metodiky SCRUM vyvíjam moderné webové aplikácie (osobné financie, CMS, sociálna sieť a iné). Z oblasti technológií sa v súčasnosti zameriavam predovšetkým na jazyk Ruby v spojitosti s frameworkom Ruby on Rails a databázou MySQL. Mám prax s vývojom aplikácií v jazykoch Java, C#, PHP, Prolog a LISP. Momentálne sa vo svojom voľnom čase zamýšľam nad problematikou crowdsourcingu a lokálneho vyhľadávania.

BC. JÁN TÓTH

Absolvent bakalárskeho štúdia v odbore Informatika na FIIT, STU v Bratislave. Počas svojho štúdia získal dobré znalosti programovacích jazykov Java, C/C++ a prehĺbil svoje vedomosti o XML, XPath, XSLT, HTML, CSS a SQL. Dobre ovláda prácu s vývojovými prostrediami MS Visual Studio a Eclipse. Vo svojej bakalárskej práci - Aplikácia pre vytváranie elektronického podpisu vo formáte XAdES sa venoval tejto problematike. V zamestnaní, kde pracuje od leta 2009, sa zoznámil s platformou .NET a programovacím jazykom C#.

MOTIVÁCIA

Publikačné systémy Wiki sa veľmi rýchlo stali fenoménom súčasnej doby najmä kvôli ich jednoduchosti pri publikovaní a slobode autorského prejavu. Táto technológia je ešte mladá, a preto je priestor pre jej vylepšenie takmer nevyčerpateľný. Akademické využitie je obmedzené práve koncepciou voľného prístupu k editovaniu článkov, nakoľko takéto príspevky nemôžu byť považované za akademicky relevantné. Preto je nutné evidovať tvorbu článkov a zaznamenávať ich zmeny.

Zadanie tohto projektu nás prinútilo sa zamyslieť nad správou a obmedzením prístupových práv užívateľov tvoriacich obsah. Inšpiráciou pri koncipovaní tohto riešenia je reálna organizácia vysokoškolskej výučby, na ktorej sa okrem pedagógov a ich znalostí podieľajú aj študenti svojimi prácami. Správne nastavenie a koncipovanie takýchto pravidiel v konečnom dôsledku povedie k rapídnemu skvalitneniu výučby nielen na našej fakulte.

Vysporiadať sa s meniacimi sa požiadavkami na realizáciu tohto projektu nebude jednoduché, avšak jednoduché riešenie nemusí mať želaný efekt pre náš rozvoj a benefit pre fakultu. Nebojíme sa prijať náročné výzvy, ktoré nás dovedú ku kvalitnejšiemu riešeniu. Zvolená kombinácia technológií SCRUM a RUBY je pre nás veľkým lákadlom a taktiež možnosťou ako uplatniť a rozšíriť naše vedomosti.

Nadviazanie na predchádzajúcu prácu na tomto projekte je dobrým odrazovým mostíkom pre vytvorenie riešenia, ktoré má potenciál stať sa celo-fakultným systémom využívaným ako študentmi, tak aj pedagógmi. Účasť na tvorbe tohto systému môže byť veľmi cennou skúsenosťou prispievajúcou k nášmu odbornému rozvoju.

KONCEPCIA RIEŠENIA

Funkcionalitu a koncepciu jej rozdelenia na menšie celky môžeme zhrnúť do nasledujúcich bodov:

1. umožniť autorovi spravovať prístupové práva k svojho príspevku a to nielen na
2. najvrchnejšej úrovni
3. umožniť administrátorovi modifikovať prístupové práva
4. vytvorenie notifikačného systému novo pridaných príspevkov pre používateľom
5. definované skupiny
6. rozšírené fulltextové vyhľadávanie vo všetkých článkoch
7. emailový konferenčný systém s prepojením na notifikačný systém
8. možnosť jednoduchého zálohovania systému pre administrátora portálu
9. vytvorenie sekcie „Moje články“
10. možnosť sledovania histórie zmien v článkoch
11. automatické ukladanie článkov s možnosťou obnovenia predchádzajúcich verzií
12. možnosť exportu článkov do zvoleného formátu
13. zobrazovanie novovytvorených alebo upravených článkov na hlavnej stránke

ZORADENIE VŠETKÝCH TÉM PODĽA PRIORITY

Poradové číslo	Názov témy
1	Portál pre časopis (Časopis)
2	Mobilný cestovný poriadok pre iPhone (Mobilný Poriadok)
3	Hierarchická wiki s právami (Wiki)
4	Digitálne mapy (Digmapy)
5	Informačný systém stredných škôl (SS IS)
6	Web 2.0 v knižniciach alebo od OPACu k portálu (DLPortál)
7	Dizajn s použitím obohatenej reality (ARDizajn)
8	Webový portál pre zdravotne postihnutých občanov (ZŤP Portál)
9	Webové stránky pre cestovnú kanceláriu (Cestovka)
10	Podpora kontroly plagiarizmu (Plagiarizmus)
11	Evidencia publikačnej činnosti (EPCA) (EPCA)
12	Textový editor obohatený o grafické prvky (Editor)
13	RoboCup tretí rozmer (RoboCup 3D)
14	Využitie sociálnych sietí pri vytváraní pracovných tímov - druhý pokus :) (Sociálne siete)
15	Elastické komunikačné centrum (EKCentrum)
16	Knižnica (Knižnica)
17	Automatizovaná podpora predmetu z oblasti programovania (DSAPodpora)
18	Virtuálna FIIT (VFIIT)
19	Grafická podpora vyhľadávania znalostí v dokumentoch (Dokumenty)
20	Vizualizácia softvérových artefaktov v 3D priestore (3DVizual)
21	Tvorba rozvrhov (Rozvrhy)
22	Imagine Cup 2010: Game Design (IC Game Design)

AKTUÁLNY ROZVRH VŠETKÝCH ČLENOV TÍMU

		7:00-7:50	8:00-8:50	9:00-	10:00-	11:00-	12:00-	13:00-	14:00-	15:00-	16:00-	17:00-	18:00-	19:00-	20:00-
	Peter Študent								OOANS		TP		VSS		
	Kacera Miroslav		Preferovany cas				PDbT		Zaklady krypto.		TP		VISS		
Pondel	Majzúnová Petra								OOANS		TP		VSS		
	Repta Martin			Rehabilitacia					Zaklady krypto.		TP		VISS		
	Soha Miroslav								Zaklady krypto.		TP		VISS		
	Tóth Ján										TP		VSS		
	Peter Študent			ZS						MPSIS		MPSIS		MPSIS	
	Kacera Miroslav					Preferovany cas				MPSIS		MPSIS		MPSIS	
Utorok	Majzúnová Petra	Kodovanie								MPSIS		MPSIS		MPSIS	
	Repta Martin			Rehabilitacia						MPSIS		MPSIS		MPSIS	
	Soha Miroslav									MPSIS		MPSIS		MPSIS	
	Tóth Ján									MPSIS		MPSIS		MPSIS	
	Peter Študent														
	Kacera Miroslav					PeWe								Preferovany cas	
Streda	Majzúnová Petra							Tanec							
	Repta Martin			Rehabilitacia											
	Soha Miroslav														
	Tóth Ján									DD		DD			
	Peter Študent								ASS			OANS		ZS	
	Kacera Miroslav	Zaklady krypto.				Preferovany cas			PDbT		Arch. inf. systemov				
Štvrtok	Majzúnová Petra	Kodovanie							ASS			OANS		Tanec	
	Repta Martin	Zaklady krypto.		Rehabilitacia					PDbT		Arch. inf. systemov				
	Soha Miroslav	Zaklady krypto.							PDbT		Arch. inf. systemov				
	Tóth Ján								ASS			OOANS		OOANS	
	Peter Študent														
	Kacera Miroslav			PDbT											
Piatok	Majzúnová Petra														
	Repta Martin			PDbT											
	Soha Miroslav			PDbT						Tanec					
	Tóth Ján														
Legend	Prednáška														
	Cvičenie														
	Mimoškolská aktivita														

ÚLOHY ČLENOV TÍMU

MANAŽÉRSKE ÚLOHY

- Bc. Miroslav Kacera — manažér monitorovania
- Bc. Petra Majzúnová — manažér dokumentácie
- Bc. Martin Repta — manažér testovania a kvality
- Bc. Miroslav Soha — manažér plánovania
- Bc. Peter Študent — projektový manažér
- Bc. Ján Tóth — manažér rizík a bezpečnosti

ROLE V PROCESE SCRUM

- Bc. Miroslav Kacera — člen tímu
- Bc. Petra Majzúnová — člen tímu
- Bc. Martin Repta — člen tímu
- Bc. Miroslav Soha — člen tímu
- Bc. Peter Študent — vedúci SCRUM
- Bc. Ján Tóth — člen tímu
- Ing. Ján Suchal — zadávateľ projektu

DLHODOBÉ ÚLOHY

- Bc. Miroslav Kacera — programátor, tester, návrhár
- Bc. Petra Majzúnová — programátor, správca webového sídla
- Bc. Martin Repta — programátor, návrhár, tester
- Bc. Miroslav Soha — programátor, dokumentarista
- Bc. Peter Študent — programátor, zabezpečenie podporných prostriedov
- Bc. Ján Tóth — programátor, návrhár, tester

ŠTÁBNA KULTÚRA

ADRESÁROVÁ ŠTUKTÚRA

Projekt využívajú štandardnú adresárovú štruktúru definovanú frameworkom Ruby on Rails.

KONVENCIE PRE NÁZVOSLOVIE

- Názvy tried, metód, stĺpcov v DB a premenných musia byť uvedené výhradne v anglickom jazyku
- Všetky názvy by mali byť krátke ale hlavne zmysluplné. Názvy musia byť zvolené tak, aby bol zrejmí obsah pri premenných, resp. činnosť pri metódach. Zmysluplné názvy musia byť bezpodmienečne uvedené
- Lokálne premenné – prvé písmeno malé, viacslovné názvy oddelené podtržítkom, tzv. snake case zápis (server_item)
- Konštanty – všetky písmená veľké, viacslovné názvy oddelené podtžítkom (CONSTANT)
- Názov tabuľky v databáze – podstatné meno v množnom čísle, malými písmenami (orders)
- Názvy stĺpcov v tabuľke– podstatné meno, jednotné číslo, malými písmenami (id)

ŠTRUKTÚRA ZDROJOVÉHO KÓDU

- Odsadenie jednotlivých riadkov musí zodpovedať vnoreniu blokov kódu v rámci logickej štruktúry aplikácie. Jednej úrovni vnorenia zodpovedajú **vždy a len 2 medzery**.
- Na každom riadku by sa mal nachádzať len jeden príkaz, pričom riadok by nemal presahovať obvyklú šírku obrazovky (max. 80 znakov).
- Za čiarkou v kóde nasleduje medzera.
- Taktiež za začiatkom a pred koncom kučeravých zátvoriek patrí medzera
- Jednotlivé **logické celky** kódu je potrebné **oddeľovať** minimálne jedným prázdnym riadkom.
- Príkaz, ktorý obsahuje návratovú hodnotu metódy musí byť oddelený od predošlého kódu prázdnym riadkom
- **Zložitejšie časti** kódu ako sú pokročilé algoritmy alebo dátové štruktúry musia obsahovať **komentár** priamo v zdrojovom kóde - komentár by pritom nemal priamo duplikovať činnosť kódu, ale objasňovať použitý postup.
- Je vhodné komentovať celé logické celky kódu a to pred ich začiatkom

PRÁCA S DATABÁZOU

- Vytváranie ako aj zmeny databázovej štruktúry musia byť realizované prostredníctvom migrácií. Priame úpravy v databáze nie sú prípustné.
- Preferuje sa využívanie štandardných metód obsiahnutých v Acitive Record pred písaním vlastných SQL dotazov.

ARCHITEKTÚRA MVC

MODEL

Model slúži na reprezentáciu aplikačných dát (typicky umiestnených v databáze) a definuje pravidlá na manipuláciu s nimi. Z toho vyplýva, že základná aplikačná logika by mala byť umiestnená iba v modeloch. Operácie s dátami by mali byť definované výhradne v modeloch a nie v controlleroch a už vôbec nie vo views/helperoch. Z prostredia controllera sa volá len čisto metóda s parametrami z príslušného modelu, ktorý vykoná potrebnú manipuláciu s dátami.

VIEW

Views by nemali obsahovať žiadnu zložitú logiku. Ich náplňou je výhradne formátovať výstupné dáta a vytvárať vstupné formuláre.

CONTROLLER

Úlohou controllera je prepojenie medzi modelom a views. Controller by mal byť napísaný tak, aby napĺňal architektonický princíp REST. V prípade používania scaffoldu je nevyhnutné controller očistiť od nabytočných metód.

HELPER

Helper by mal obsahovať len krátke časti kódu, ktoré typicky pomáhajú pri formátovaní vo views.

RDOC

V rámci zdrojového kódu sa komentáre uvádzajú v takej podobe, aby z nich bolo možné následne vygenerovať dokumentáciu. Na generovanie dokumentácie sa používa RDoc.

LOKALIZÁCIA

Na lokalizáciu aplikácií sa využíva integrovaná podpora lokalizácie priamo v Rails. Lokalizácia spočíva vo volaní metódy t vo views s parametrom, ktorý definuje, ktorá reťazec sa má použiť. Samotné lokalizácie sa nachádzajú v adresári config/locales v súbore formátu YML, ktorý nesie ako názov skratku jazyka.

Priamo v zdrojovom kóde aplikácií sa nesmú počas celého vývoja vyskytovať žiadne texty zadané na pevno - všetky texty musia byť vložené cez lokalizačnú metódu.

Lokalizačný súbor by mal mať nasledujúcu štruktúru:

- na najvyššej úrovni je špecifikovaný jazyk lokalizácie. Jazyk je definovaný kódom podľa normy ISO 639-1(sk - slovenčina, cs - čeština, ...)
- na ďalšej úrovni sú špecifikované všeobecné slovné spojenia používané naprieč aplikáciou bez ohľadu na modul (napr. ďalej, späť, dátum, ...)
- na rovnakej úrovni sú zadefinované jednotlivé moduly aplikácie
- pod jednotlivými modulmi sú najprv uvedené preklady textových reťazcov spoločné a špecifické pre celý model (napr. header: Správy)
- na ďalšej úrovni pod modulmi sa nachádzajú názvy controllerov
- ak modul obsahuje len jeden controller, môže sa táto úroveň v preklade vynechať a prejsť priamo o úroveň nižšie
- pod controllerami sa nachádzajú jednotlivé akcie a k ním príslušné preklady
- na konci súboru sa nachádza definícia a špecifikácia lokalizácie číselných, dátumových a časových dát.
- pre konštanty pod ktorými sa daný textový reťazec vyskytuje je potrebné voliť **názvy v angličtine**, pričom musia byť dostatočne **výstižné** .

METODIKA – PREHLIADKY KÓDU

ROLY ZODPOVEDNOSTI

Rola	Zodpovednosť
Programátor	- implementovať zadané úlohy - opraviť kód z prehliadok
Dokumentarista	- dokumentovať priebeh jednotlivých etáp vývoja - spisovať zápisnice zo stretnutí
Projektový vedúci	- riadiť celý proces tvorby softvérového systému - plánovanie termínov pre jednotlivé etapy - vykonávať prehliadky

Tabuľka 1: Roly zodpovednosti v procesoch prehliadky kódu

PROCESY

Proces	Názov
1.	Implementácia úlohy
2.	Vykonávanie prehliadky
3.	Zmeny v kóde
4.	Odovzdanie opraveného kódu

Tabuľka 2: Procesy prehliadok kódu

IMPLEMENTÁCIA ÚLOHY

Vstupy: úlohy

Výstupy: zdrojový kód

Zodpovednosť: programátor

Požadovaný výstup od programátor musí spĺňať nasledovné kritériá:

1. Programátor by mal vždy do najbližšieho stretnutia tímu implementovať aspoň minimálnu časť úlohy,
2. Ak je stretnutie na konci šprintu, mala by byť implementovaná celá úloha,
3. V prípade, že je úloha dokončená, programátor by mal vytvoriť testy, na overenie správnej funkčnosti vytvoreného kódu, v opačnom prípade nemôže byť úloha akceptovaná.

VYKONÁVANIE PREHLIADKY

Vstupy: kód

Výstupy: kód, pripomienky ku kódu, zápisnica zo stretnutia

Zodpovednosť: projektový vedúci

Proces vykonávania prehliadky prebieha na každom stretnutí tímu počas priebehu šprintu. Mimo týchto stretnutí môžu taktiež prebiehať prehliadky kódu napríklad pomocou emailovej komunikácie (záleží na dohode s projektovým vedúcim). V druhom prípade, je potrebné, aby programátor nahral vytvorený kód do spoločného repozitára (git@github.com:hidden/bonsai.git), aby si ho mohol projektový vedúci vykonávajúci prehliadku kódu prezrieť a spísať pripomienky.

Postup pri kontrole kódu:

1. Programátor oboznámi projektového vedúceho o tom, čo naprogramoval, ukáže a vysvetlí kód,
2. Vedúci projektu skontroluje kód, v rámci ktorého:
    a. Vhodnosť zvoleného spôsobu riešenia danej úlohy (pritom hodnotí napr. efektívnosť)
    b. Kontroluje základné programátorské konvencie
    c. Princípy dodržania OOP, MCV,...
    d. Funkčnosť testov,
    e. Môže navrhnúť zmeny v kóde, ktoré sa môžu týkať zmien metód, celkovej filozofie riešenia daného problému.
3. Programátor môže vykonať potrebné zmeny ešte počas stretnutia,
4. Zo stretnutí, kde sa vykonávajú prehliadky kódu dokumentarista spíše zápisnicu,
5. Okrem vytvorenej zápisnice zo stretnutia by si mal programátor zapísať všetky pripomienky, návrhy ku kódu, aby mohol správne opraviť nedostatky.

VYKONANIE ZMIEN V KÓDE

Vstupy: kód

Výstupy: zmenený kód

Zodpovednosť: programátor

Programátor musí vykonať všetky zmeny a návrhy, ktoré prekonzultoval s projektovým vedúcim. Projektový vedúci akceptuje úlohu len v prípade, že programátor vykoná všetky zmeny, ktoré boli počas stretnutí a prehliadok kódu prediskutované a ich vypracovanie bolo vyžadované.

ODOVZDANIE OPRAVENÉHO KÓDU

Vstupy: opravený kód

Výstupy: uložený kód v repozitári

Zodpovednosť: programátor

Zmenený a opravený kód musí programátor nahrať do repozitára (git@github.com:hidden/bonsai.git) do ukončenia šprintu, aby mohla byť úloha považovaná za úspešne vyriešenú.

METODIKA - MANAŽMENT VERZIÍ ZDROJOVÉHO KÓDU

ÚVOD

Účelom tejto metodiky je definovať postupy práce s repozitárom zdrojových kódov. Čitateľovi poskytuje informácie o najčastejších procesoch súvisiacich so správou verzií. Podrobnejšie je opísaný spôsob práce s jednotlivými vetvami a tvorba commitov.

NAJČASTEJŠIE POUŽITÉ POJMY

- *repozitár* – centrálne úložisko verzií zdrojových kódov
- *branch* – vetva úložiska
- *klon* – lokálna kópia synchronizovaná s adresárom umiestneným v repozitári
- *merge* – proces spájania funkcionality dvoch vetiev
- *commit* - potvrdenie zmien do repozitára

DEDIKÁCIA METODIKY

Táto metodika je primárne určená pre programátorov pracujúcich s repozitárom prípadne ich správcov. Popisuje postupnosť základných operácií pri práci s repozitárom ako i spôsoby vytvárania nových vetiev a pomenovania nových verzií zdrojových kódov.

VYMEDZENIE ROLÍ A ZODPOVEDNOSTÍ PRI PRÁCI S REPOZITÁROM

Rola	Zodpovednosť
programátor	pracuje na pridelenom projekte a zaznamenáva zmeny do repozitára
správca repozitára	vytvára nové a spravuje existujúce repozitáre
manažér kvality	zodpovedá za kontrolu kvality práce a funkčnosti testov

NAJČASTEJŠIE OPERÁCIE PRI PRÁCI S VERZOVACÍM SYSTÉMOM GIT

- *init* – vytvorenie nového repozitára
- *branch* – vytvorenie novej vetvy
- *clone* – vytvorenie lokálnej kópie z existujúcej vetvy
- *commit* – potvrdenie lokálnych zmien do úložiska
- *merge* – spojenie funkcionality rozličných vetiev
- *push* – nahratie zmien do repozitára
- *pull* – stiahnutie zmien z repozitára

TOPOLÓGIA ÚLOŽISKA

Topológia úložiska zdrojových kódov je hierarchicky organizovaná do stromovej adresárovej štruktúry. Vzdialený repozitár je uložený na serveri *github.com*, ku ktorému majú všetci programátori a členovia vývojového tímu prístup. Repozitár obsahuje viacero adresárov, ktoré prislúchajú jednotlivým projektom, prípadne modulom daného projektu, kde adresár modulu je v takomto prípade podradeným adresárom daného projektu.

ORGANIZÁCIA VETIEV PROJEKTU

Každý projekt obsahuje jednu hlavnú vetvu *Master,* ktorá obsahuje aktuálnu a stabilnú verziu zdrojových kódov. Všetci programátori si vytvárajú svoj vlastný *klon,* z ktorého pri vývoji vychádzajú.

Pre každú úlohu *(issue)* je vytvorená nová pracovná vetva pomenovaná typicky názvom riešenej úlohy. Pre jej vytvorenie je potrebné požiadať správcu repozitára, ktorým je *Bc. Peter Študent*. Do tejto vetvy programátor zaznamenáva všetky zmeny viažuce sa k jemu pridelenej úlohe.

ZAZNAMENANIE ZMIEN DO REPOZITÁRA

Všetci vývojári si pred začatím novej časti úlohy vždy spravia *klon* aktuálnej verzie *Master* vetvy daného projektu. Všetky zmeny zaznamenávajú najskôr do svojich lokálnych *klonov*. Je dôležité časté lokálne *commitovanie* kvôli ucelenosti zdrojového kódu. Pre zverejnenie je potrebné zmeny odoslať do repozitára.

SPÔSOB POMENOVANIA COMMITOV

Pomenovanie nových verzií sa riadi internými pravidlami.

NOVÉ VERZIE

Každá nová verzia je pomenovaná v anglickom jazyku, kde prvou časťou je číslo *issue* a ďalšou popis vykonaných zmien vzhľadom k predchádzajúcej verzii. Nové verzie obsahujú opravy chýb predchádzajúcich verzií, alebo vylepšenia v akejkoľvek oblasti, vrátane funkčnosti, výkonu a použiteľnosti, bez možnosti návratu späť.

POMENOVANIE OPRAVNÝCH VERZIÍ

Opravné verzie sú označované podobne ako nové verzie, odlišujú sa tým, že obsahujú popis stavu *fixed.* Za ním nasleduje popis opravy v anglickom jazyku vzťahujúcej sa k predošlej verzii.

ZARADENIE ZAZNAMENÁVANÝCH ZMIEN DO JEDNOTLIVÝCH VETIEV

Pri publikovaní lokálnych zmien sa zaradenie do vetiev daného projektu riadi internými pravidlami. Pre zlúčenie lokálnych zmien s konkrétnou vetvou je potrebné vykonať operáciu *merge*.

MASTER

Master vetva sa používa výhradne k uchovávaniu otestovanej a plne funkčnej verzie zdrojových súborov projektu. Pred vykonaním *merge* pracovnej vetvy s vetvou *Master* je potrebné aby výsledky všetkých testov boli zelené. Overenie funkčnosti testov je možno vykonať na serveri *runcoderun.com,* na ktorý majú všetci programátori prístup.

PRACOVNÉ VETVY

Každá pracovná vetva obsahuje aktuálnu verziu viažucu sa k danej *issue,* ku ktorej bola vytvorená. Je dôležité časté odosielanie lokálnych zmien i kvôli informovanosti ostatných členov tímu. Pracovná vetva sa používa na publikovanie zmien novej funkcionality ako aj opravnej, v prípade že:

- je v dobrom stave a neobsahuje známe chyby
- je vhodná na použitie pri ďalšom vývoji

Každá nová funkcionalita alebo zmena funkcionality sa testuje v tejto vetve. Primárne slúži na odhalenie a odladenie všetkých chýb ešte predtým, ako sa vykoná *merge* s vetvou *Master.*

VYUŽITIE NÁSTROJA GITGUI PRI SPRÁVE REPOZITÁROV ZDROJOVÝCH KÓDOV

V tejto kapitole sú popísané postupy využitia nástroja GitGui pri správe repozitárov verzií zdrojových kódov verzovacieho systému Git z pohľadu programátora alebo správcu repozitára.

PRIDANIE PROJEKTU DO REPOZITÁRA

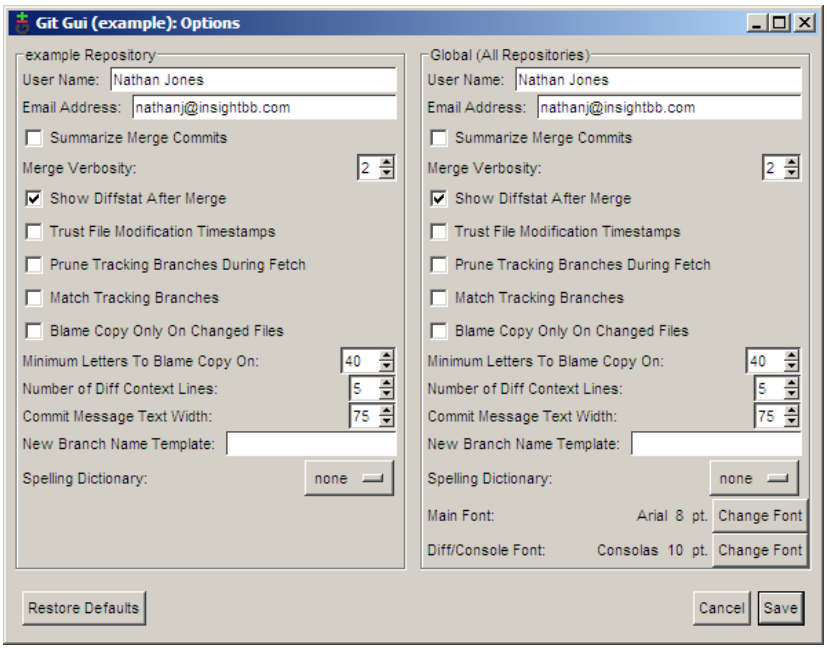
Vstup	**vytvorený repozitár**
Výstup	projekt sprístupnený v repozitári
Zodpovedný	správca repozitára

Pred sprístupnením projektu ostatným používateľom je potrebné nakopírovať zdrojové súbory do priečinka repozitára.

Ešte pred vykonaním operácie *Commit* je potrebné nastaviť konfiguračné nastavenia repozitára v hornom menu, voľbou *Edit → Options.*

V ľavej časti dialógového okna je možné zvoliť nastavenia o mene a emailovej adrese zakladateľa repozitára. Tieto nastavenia budú použité iba pre konkrétne vytváraný repozitár.

V pravej časti sa nachádzajú globálne konfiguračné nastavenia, ktoré budú platiť na všetky repozitáre vytvorené v budúcnosti.



Obr. 1 Konfiguračné nastavenia repozitára

Potvrdením voľby *Save* sú nastavenia uložené a je možné pokračovať v dokončení pridania projektu do repozitára.

Po označení všetkých súborov a zvolením možnosti v hornom menu pod voľbou *Commit -> Stage to Commit* sa vykoná *commit* novo pridaných súborov.

Túto operáciu možno realizovať konzolovým príkazom

git add .

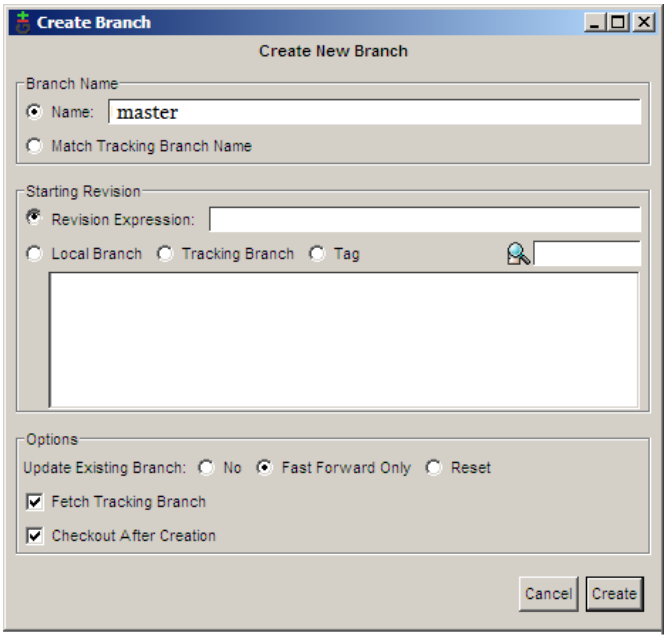
ktorý automaticky zahrnie všetky súbory v danom adresári, alebo

git add <nazov>, ktorý pridá iba súbor s konkrétnym názvom.

VYTVORENIE NOVEJ VETVY

Vstup	**vytvorený repozitár**
Výstup	nová vetva
Zodpovedný	správca repozitára

Nová vetva sa vytvorí kliknutím na voľbu *Branch -> Create* v hornom menu. Je potrebné zadať názov novej vetvy (*napr. Master,* ak bude hlavnou vetvou).



Obr. 2 Vytvorenie novej vetvy

Vetva sa vytvorí po kliknutí na tlačidlo *Create.*

Alternatívou postupnosti týchto krokov je príkaz git branch <nazov>, ktorým je možné založiť novú vetvu s požadovaným názvom.

SYNCHRONIZÁCIA PRACOVNEJ VETVY S MASTER VETVOU

Vstup	**pracovná vetva a vetva master umiestnená na vzdialenom serveri**
Výstup	pracovná vetva zosynchronizovaná s vetvou master
Zodpovedný	programátor

Pred vykonaním zlúčenia (merge) pracovnej vetvy s master vetvou je potrebné vykonať synchronizáciu (rebase) všetkých *commitov.* Túto operáciu je možné vykonať v pracovnej vetve priamo z príkazovej riadky použitím príkazu:

```
git rebase master
```

ZMENY A ZLUČOVANIE COMMITOV

Vstup	**pracovná vetva obsahujúca niekoľko commitov**
Výstup	odstránené alebo zlúčené commity
Zodpovedný	programátor

V prípade potreby zlúčenia niekoľkých commitov prípadne zmazania je nutné tak vykonať ešte pred vykonaním merge. Túto operáciu je možné vykonať v danej vetve použitím príkazu:

```
git rebase -i HEAD^<pocet poslednych commitov, ktore zlučujeme>
```

Po zadaní tohoto príkazu sa otvorí editor s výpisom *commitov*, kde je u každého možno špecifikovať či sa má:

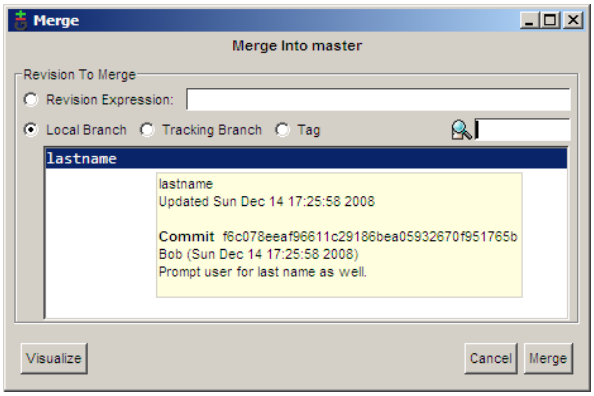
- ponechať - **pick** (nie je potrebné vykonať žiadne zmeny)
- zmazať - (v tomto prípade je potrebné zmazať daný riadok)
- zlúčiť - **squash** (v tomto prípade sa daný commit zlúči s commitom uvedeným o riadok vyššie)

Po vykonaní potrebných zmien a uložení, je možné vykonať operáciu merge.

ZLÚČENIE DVOCH VETIEV

Vstup	**dve vetvy**
Výstup	spojenie funkcionality dvoch vetiev do jednej
Zodpovedný	správca repozitára

Spojenie funkcionality dvoch vetiev do jednej sa vykoná kliknutím na voľbu *Merge -> Local Merge* v hornom panely menu. V dialógovom okne je potrebné zo zoznamu vetiev vybrať tú, z ktorej sa budú zmeny aplikovať a kliknúť na tlačidlo *Merge*.



Obr. 3 Zlúčenie vetvy lastname s vetvou master

Túto operáciu možno alternatívne realizovať zadaním príkazov:

git checkout <cielova vetva> a git merge <zdrojova vetva>.

NAHRANIE (PUSH) VETVY REPOZITÁRA NA VZDIALENÝ SERVER

Vstup	**repozitár umiestnený na lokálnom pc**
Výstup	kópia vetvy daného repozitára na vzdialenom serveri zmien
Zodpovedný	programátor

Pred vykonaním operácie **push** je potrebné mať na niektorom serveri typu *github.com* založený účet. Autentifikácia prebieha pomocou *ssh kľúča*. Pri problémoch s autentifikáciou, alebo založením účta sa obráťte na vedúceho vývojového tímu, ktorým je *Ing. Ján Suchal*.

Operácia **push** sa vyvolá kliknutím na voľbu *Remote -> Push* z ponuky horného panela nástroja GitGui. Po zvolení vetvy z ponuky *Source Branches* a vyplnení adresy cieľového umiestnenia v časti *Arbitrary URL* bude po kliknutí na tlačidlo *Push* operácia dokončená.

Pre ušetrenie času a zrýchlenie procesu odosielania na vzdialený server, je možné jeho adresu pridať do konfiguračného súboru. Táto adresa bude potom automaticky predvyplnená v časti *Arbitrary URL*.

Táto konfigurácia sa nastavuje priamo z príkazovej riadky spustením príkazu:

git remote add github <adresa vzdialeneho repozitara>

STIAHNUTIE ZMIEN (PULL) ZO VZDIALENÉHO REPOZITÁRA

Vstup	**repozitár umiestnený na vzdialenom serveri**
Výstup	kópia repozitára na lokálnom pc serveri zmien
Zodpovedný	programátor

Podmienkou pre vykonanie operácie **pull** je pridanie správnej cesty do konfiguračného súboru. To je možné vykonať priamo z príkazovej riadky:

git remote add github <adresa vzdialeneho repozitara>

Vytvorenie lokálnej kópie zdrojových súborov umiestnených v repozitári na vzdialenom servery sa vykoná z menu v časti *Remote → Fetch.*

METODIKA – SPRÁVA ÚLOH V PROJEKTE

ÚVOD

Tento dokument sa zaoberá problematikou riadenia a evidencie úloh. V nasledujúcich kapitolách sú špecifikované procesy riadenia úloh ako aj s nimi súvisiace roly osôb zúčastnených na tvorbe softvérového projektu za použitia agilnej metódy Scrum. Cieľom tejto metodiky je navrhnúť množinu podmienok, postupov a obmedzení, ktoré majú za úlohu zvýšiť efektívnosť a transparentnosť práce. Tento dokument je určený programátorom, vedúcim a spolupracovníkom pracujúcim v rámci tímového projektu v akademiskom roku 2009/2010.

DEFINÍCIA POJMOV

BACKLOG

Jeden zo základným prvkov používaných pri metóde Scrum. Slúži ako zásobník pre úlohy, ktoré sú zahrnuté do projektu ale ešte neboli priradené do spracovania (Product backlog) alebo úlohy spracovávané a aktuálnom šprinte (Sprint backlog).

ŠPRINT

Pevne stanovené časové obdobie, počas ktorého by mal vývojový tím dokončiť všetky úlohy definované v aktuálnom *sprint backlog-u*.

ZADÁVATEĽ

Zadávateľ (eng. Product owner) je členom tímu zodpovedným za tlmočenie požiadaviek samotného zákazníka. Jeho úlohy v projektu sú popísané nižšie (viď *Roly procesu návrhu*)

ROLY PROCESU SPRÁVY ÚLOH

Rola		Popis
Zadávateľ	***Požiadavky***	• Znalosť metódy Scrum • Znalosť dekompozičných princípov
	Úlohy	• Pochopenie a formulácia požiadaviek zákazníka • Písanie užívateľských prípadov použitia (user story) • Prioritizácia existujúcich prípadov použitia • Priebežná kontrola smerovania vývojového tímu
Scrum master	***Požiadavky***	• Znalosť metódy Scrum • Výborná znalosť použitého programovacieho jazyka a realizovanej problematiky
	Úlohy	• Odstraňovanie konceptuálnych alebo implementačných prekážok pri realizácii jednotlivých úloh v šprinte • Sprostredkovanie informácií medzi tímom a okolím • Dohľad nad dodržiavaním pripraveného plánu
Programátor	***Požiadavky***	• Znalosť metódy Scrum • Znalosť používaného programovacieho jazyka.
	Úlohy	• Realizácia úloh pridelených v rámci šprintu • Prezentovanie priebežných výsledkov práce

PROCES SPRÁVY ÚLOH

DEFINOVANIE UŽÍVATEĽSKÝCH PRÍPADOV POUŽITIA

Vstup: **POŽIADAVKY UŽÍVATEĽA NA APLIKÁCIU**

Výstup: **ŠTRUKTÚROVANÉ PRÍPADY POUŽITIA APLIKÁCIE**

Zodpovedná osoba: **ZADÁVATEĽ**

Zadávateľ na základe konzultácie s používateľmi aplikácie špecifikuje jednotlivé prípady použitia. Takto špecifikované prípady použitia musia vytvárať ucelenú koncepciu funkcionality, ktorú má vyvíjaná aplikácia realizovať. Každý prípad použitia musí byť koncipovaný vo forme uceleného prípadu, ktorý vychádza z realizovaných častí systému a preto s nimi nemôže byť v rozpore. Takto navrhnuté prípady použitia sú následne pripojené do *product backlog-u*, odkiaľ sa neskôr vyberajú na implementáciu do jednotlivých šprintov.

VÝBER UŽÍVATEĽSKÝCH PRÍPADOV POUŽITIA PRE REALIZÁCIU V ŠPRINTE

Vstup: **ŠTRUKTÚROVANÉ PRÍPADY POUŽITIA APLIKÁCIE (PRODUCT BACKLOG)**

Výstup: **VYBRANÉ PRÍPADY POUŽITIA PRE REALIZÁCIU**

Zodpovedná osoba: **ZADÁVATEĽ , SCRUM MASTER**

Z *product backlog-u* sa vyberajú tie prípady použitia, ktoré majú najvyššiu prioritu od zadávateľa. Počet užívateľských prípadov použitia je limitovaný veľkosťou a skúsenosťami vývojového tímu. Faktorom pre výber prípadov použitia je okrem priority aj ich náväznosť na realizovanú funkcionalitu tak, aby sa v obmedzenom čase realizovala čo najväčšia časť požadovanej funkcionality.

PLÁNOVACIE STRETNUTIE ŠPRINTU

Vstup: **VYBRANÉ PRÍPADY POUŽITIA PRE REALIZÁCIU**

Výstup: **MNOŽINY PRIDELENÝCH ÚLOH PRE JEDNOTLIVÝCH ČLENOV TÍMU**

Zodpovedná osoba: **SCRUM MASTER, TÍM**

Náplňou plánovacieho stretnutia pred začatím šprintu je rozdelenie a ohodnotenie jednotlivých vybraných prípadov použitia určených pre daný šprint. Každý prípad použitia sa rozdelí na úlohy a následne členovia vývojového tímu ohodnotia každú úlohu samostatne časom a úsilím potrebným na jej realizáciu. Toto hodnotenie sa premietne do počtu *scrum bodov*, ktoré sú formálnym zobrazením náročnosti úlohy. Ohodnotené úlohy sa burzovým princípom prerozdelia medzi členov vývojového tímu a následne sa tieto množiny zaznačia do systému správy úloh.

DENNÝ SCRUM MEETING

Vstup: **MNOŽINY AKTUÁLNE PRIDELENÝCH ÚLOH PRE JEDNOTLIVÝCH ČLENOV TÍMU**

Výstup: **UPRAVENÝ STAV PRIDELENÝCH ÚLOH**

Zodpovedná osoba: **SCRUM MASTER, TÍM**

Toto stretnutie slúži na priebežné informovanie členov vývojového tímu o postupe pri realizácii zadaných úloh. Náplňou tohto stretnutia je aby členovia tímu podali základné informácie o svojom postupe, problémoch a plánoch na najbližšie obdobie. Postup v realizácii ako aj vzniknuté problémy je nutné zaznačiť do systému správy úloh. Toto stretnutie nezahŕňa kontrolu kódu ani iné prvky softvérového procesu, má čisto informačný význam. Po skočení tohto stretnutia v prípade potreby sú realizované konzultácie medzi členmi tímu avšak už bez formálneho vedenia(*Scrum of scrums*).

HODNOTIACE STRETNUTIE ŠPRINTU

> *Vstup:* **ÚLOHY PRIDELENÉ NA REALIZÁCIU V AKTUÁLNOM ŠPRINTE**
>
> *Výstup:* **HODNOTIACA SPRÁVA**
>
> *Zodpovedná osoba:* **SCRUM MASTER, TÍM, ZADÁVATEĽ**

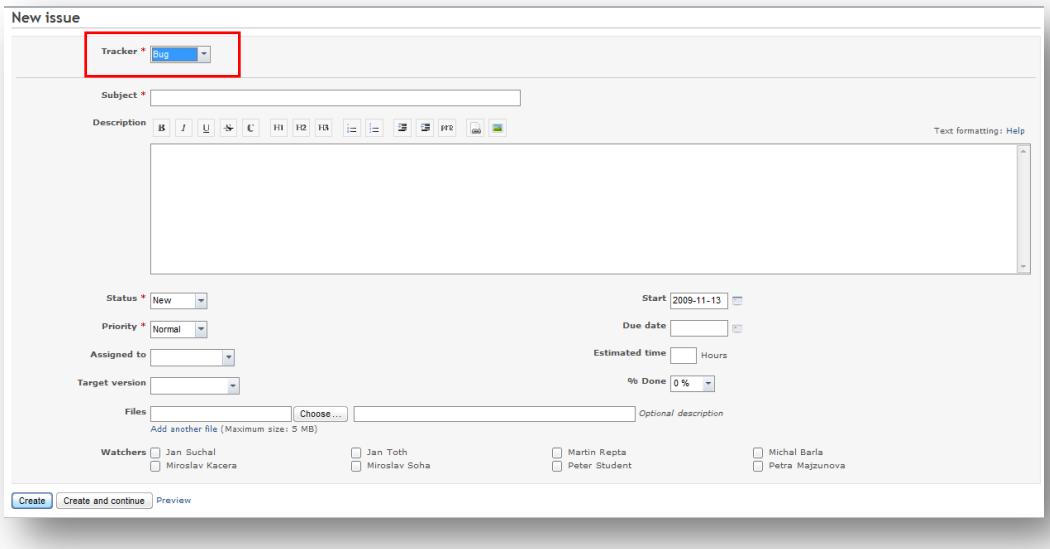
Úlohou tohto stretnutia je zhodnotiť výsledky dosiahnuté v aktuálnom šprinte. Podkladom sú úlohy pridelené k aktuálnemu šprintu. O ich splnení alebo nesplnení rozhoduje zadávateľ. Úlohy, ktoré splnili definované podmienky sú prezentované užívateľom a po ich akceptácii sú priradené do systému správy úloh ako uzavreté. Nesplnené úlohy automaticky prechádzajú do nasledujúceho šprintu. Po ukončení hodnotiaceho stretnutia je realizované *scrum retrospective* stretnutie.

EVIDENCIA ÚLOH V SYSTÉME REDMINE

V tejto časti sú zadefinované záväzné postupy pre evidenciu úloh v systéme Redmine. V prípade nejasností alebo problémov pri spracovaní, kontaktuje administrátora systému (*Ján Suchal, jan.suchal@stuba.sk*)

DEFINOVANIE UŽÍVATEĽSKÝCH PRÍPADOV POUŽITIA

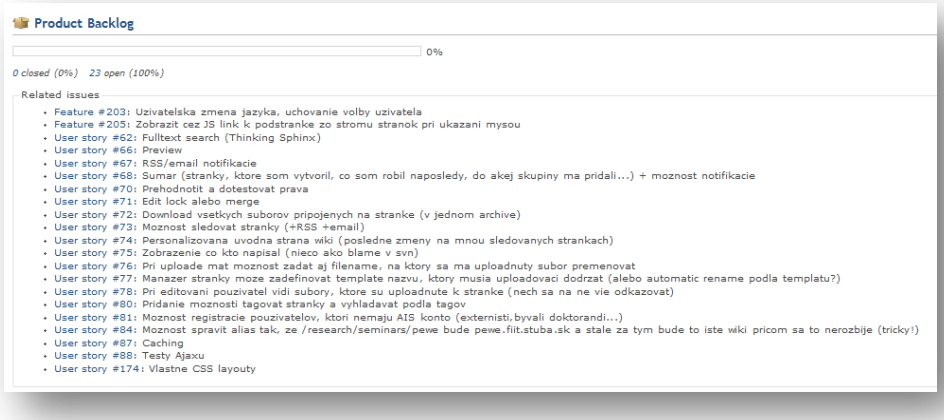
Užívateľský prípad použitia definovaný podľa pravidiel popísaných vyššie je nutné zaznačiť do systému, aby bolo možné spravovať tento scenár v ďalších etapách. Pre jeho pridanie so systému sa použije formulár *new isue* nasledovne:



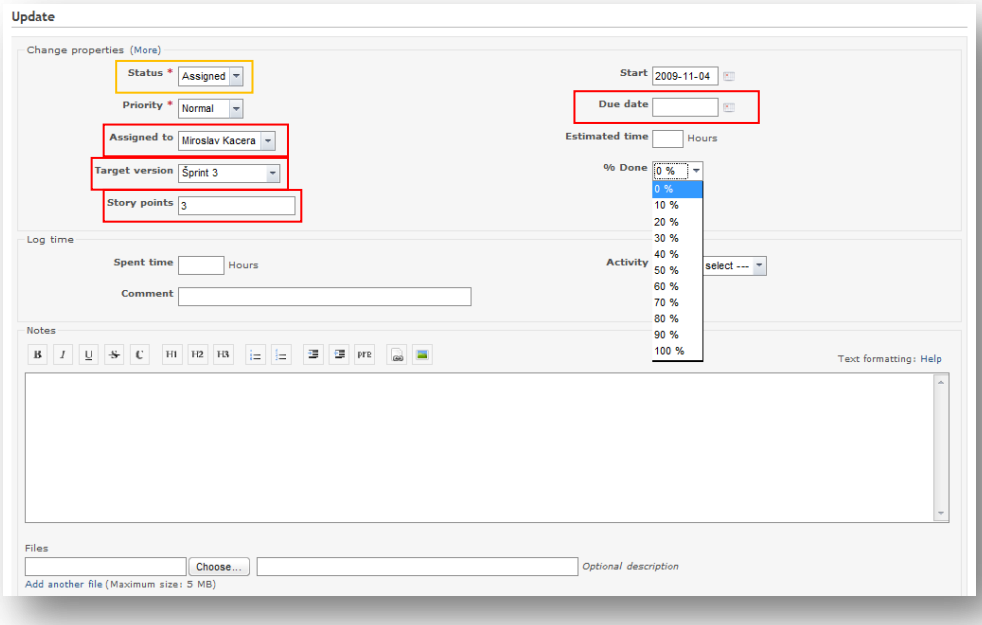
1. *Tracker* sa vyplní hodnotou *user story (resp. bug pre popis chyby systému)*
2. *Subject* a *Description* sa vyplní názvom a popisom v tak ako bolo špecifikované pri analýze problémov a možností systému (intuiitívny popis pre programátorov)
3. *Status* sa nastaví na *New* keďže definovaný prípad použitia ešte nebol priradený k šprintu
4. Ostatné polia ostanú nevyplnené, tieto polia sa budú postupne vypĺňať v priebehu realizácie

PLÁNOVACIE STRETNUTIE ŠPRINTU

Prvým krokom plánovacieho stretnutia je výber prípadov použitia, ktoré budú rozpracované v plánovanom šprinte. Tieto prípady sa vyberajú z *product backlog-u*.



Vybraným prípadom použitia je potrebné aktualizovať informačné údaje nasledovne:



1. *Status* sa zmení na *Assigned*
2. Priradený šprint sa nastaví na aktuálne rozvrhovaný šprint
3. Dátum ukončenia je nastavený na dátum ukončenia priradeného šprintu
4. Priradený programátor ako aj počet SCRUM bodov sú priradené na základe záverov plánovacieho stretnutia
5. Percentuálna hodnota ukončenia je nastavená na 0%

DENNÝ SCRUM MEETING

Počas denného *scrum meeting-u* každý z členov vývojového tímu okrem slovného vyjadrenia postupu a problémov, s ktorými sa stretol vyjadrí svoj postup aj percentuálne a to podľa nasledovného kľúča:

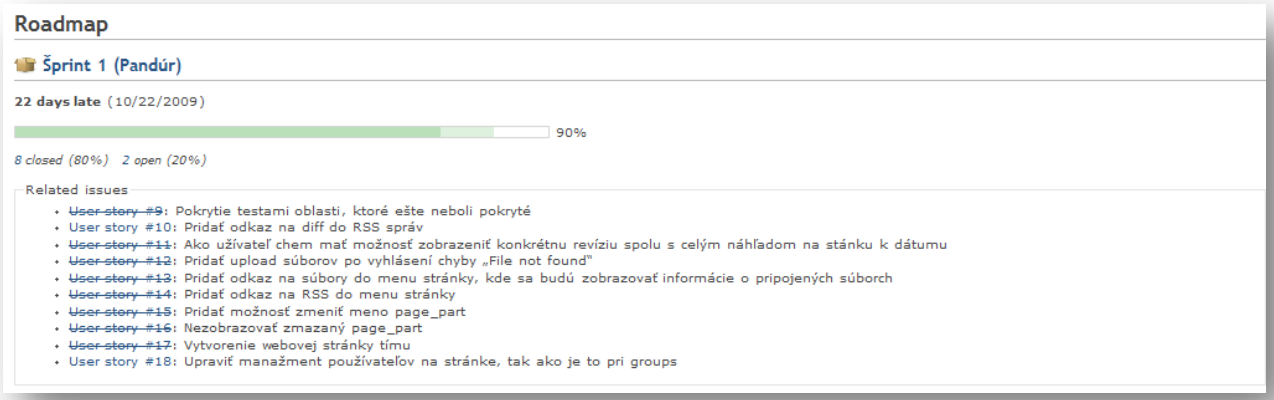
1. 0-20% za naštudovanie danej problematiky riešenia
2. 20-40% za kvalitu pokrytia danej úlohy testovacími scenármi
3. 40-85% za funkcionalitu po úplnom pokrytí úlohy testovacími scenármi (percentuálny nárast sa vyhodnocuje ako pomerná časť realizovanej a celkovej funkcionality danej úlohy)
4. 85-100% za integráciu a záverečné otestovanie vytvorenej funkcionality v kontexte celého systému

Tieto hodnoty musia byť po skončení denného *scrum meeting-u* zanesené do systému (zodpovedný programátor). Po dosiahnutí 100% sa *status* realizovanej úlohy zmení na hodnotu resolved (zodpovedný programátor).

Zistené problémy spôsobené predchádzajúcou implementáciou systému sa pridajú do systému ako nové úlohy s hodnotou *tracker* nastavenou ako *bug* a najvyššou prioritou.

HODNOTIACE STRETNUTIE ŠPRINTU

Hodnotiace stretnutie uzatvára každý šprint. Odovzdané úlohy musia prejsť schválením zadávateľa. Na prezentáciu úloh sú automaticky zaradené úlohy so 100% splnením. Na prezentáciu je možné zaradiť aj úlohy a mierou splnenia nad 85% po predchádzajúcom zhodnotení *scrum master-om* a *zadávateľom*.



Zadávateľom schválená úloha je zapísaná do systému ako *closed* a následne je odstránená z aktívnych úloh.

V prípade, že úloha bola schválené zadávateľom a má menej ako 100%, tak sa do poznámky zapíše dôvod jej prijatia a zdokumentujú sa časti, ktoré neboli implementované alebo dotestované pre prípad neskoršieho výskytu problémov. Potom už s touto úlohou pracujeme ako so 100% naplnenou v stave *closed*.

ANALÝZA A MODEL PRIEBEŽNÉHO RIEŠENIA

ÚVOD

Formát wikipédie, ako voľne šíriteľnej internetovej encyklopédie, je už vo svete celkom známy. Dovoľuje komukoľvek vytvoriť vlastnú encyklopédiu, ktorej články môžu všetci návštevníci bez obmedzenia čítať, upravovať, či vymazávať. Navyše umožňuje pojmové prepojenie článkov formou fulltextu, čím prispieva k sprehľadneniu vyhľadávania v aplikácií.

Kvôli svojej flexibilite a bez potreby manažmentu používateľov, je využívaný v mnohých oblastiach. Čo je na druhej strane výhodou, nemusí platiť za každej situácie. Mnohé články, dotýkajúce sa odborných tém, nie sú uznané ako relevantné, práve preto, že nie je možné určiť ich autorov, prípadne editorov. Navyše fakt, že nechceme aby niektoré články, boli zdieľané určitými skupinami ľudí, robí klasický formát wikipédie nepoužiteľným nie len v akademickom prostredí.

Medzi nedostatky, ktoré sme identifikovali pri vytváraní súčasnej verzie wiki riešenia patria:

1. **Nedostatočný manažment práv** – v klasickej wikipédii nie sú vôbec riešené. Všetky vložené informácie sú kolektívnym vlastníctvom návštevníkov, defacto používateľov internetu
2. **Prepojenie článkov** – okrem fulltextu, neexistuje žiadne kontextové, či inak organizované prepojenie článkov.
3. **Podpora členenia obsahu príspevkov** - možnosť rozčleňovať stránky na samostatné oddelene editovateľné časti
4. **Podpora verziovania** - zobrazenie histórie a správa zmien jednotlivých článkov

Cieľom bolo vytvoriť formát encyklopédie, použiteľnej v akademickom prostredí, ktorý bude rešpektovať autorstvo, a dovolí vytvárať každému používateľovi nielen verejne prístupné články, ale aj články s obmedzením prístupu pre skupiny používateľov.

Na základe toho sú v systéme zahrnuté :

1. Stromová schéma stránok, umožňujúca dedenie práv, k jednotlivým položkám
2. Schéma skupín užívateľov a ich práv k prislúchajúcim článkom
3. Zobrazenie histórie a zmien a možnosť vrátiť dokument na pôvodnú verziu
4. Stránky sú rozdelené na jednotlivé časti, page party

KONCEPTUALIZÁCIA

V nasledujúcej kapitole sú popísané prípady použitia, ktoré vznikli na základe analýzy vstupných požiadaviek

POŽIADAVKY NA SYSTÉM

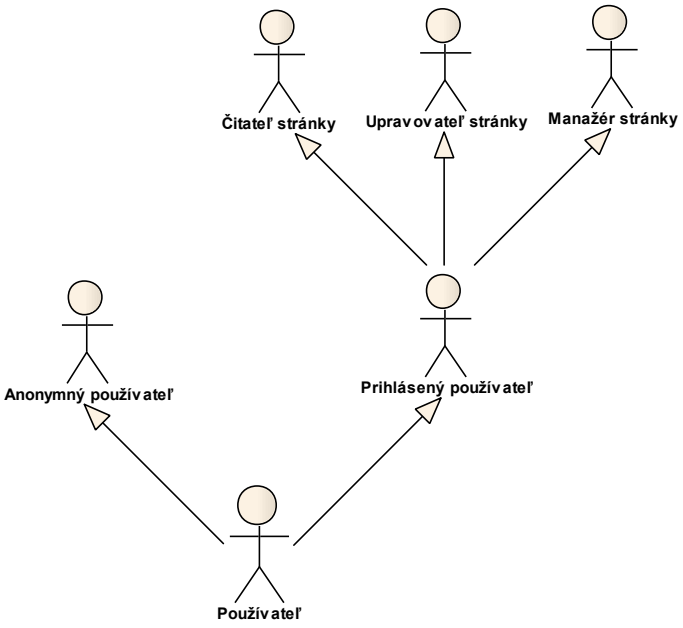
ID požiadavky	Opis požiadavky
RQ001	Umožniť používateľovi vytvárať, editovať a manažovať stránky s jednoduchým obsahom
RQ002	Jednotlivé stránky by mali byť v štruktúrovanej náväznosti, podľa ktorej by sa mal používateľ vedieť zorientovať v spleti všetkých stránok
RQ003	Manažment riadenia prístupov, ktorý by povolil skupinám používateľov vidieť, pridávať, upravovať a manažovať jednotlivé stránky.
RQ004	Možnosť automatického prenášania práv používateľov zo stránky na podstránku

RQ004	Možnosť vytvárania verejných stránok
RQ005	Manažment skupín, možnosť manažovať jednotlivých členov skupín. Vytvárať a rušiť skupiny.
RQ006	Riadenie príloh k stránkam, ako sú obrázky, dokumenty, mediálne súbory,…
RQ006	Rozčlenenie stránok na samostatne pridávané časti.
RQ007	Vhodné zobrazenie údajov na stránke, zvýraznenie syntaxe, zobrazenie príloh, možnosť výberu dizajnu
RQ008	Dashboard – zoznam zmien na stránkach patriacich skupinám, ktorých členom je prihlásený používateľ, prípadne, ktoré má označené za obľúbené
RQ009	Zobrazenie histórie zmien stránky s možnosťou návratu a revertnutia na konkrétnu revíziu

TYPY POUŽÍVATEĽOV

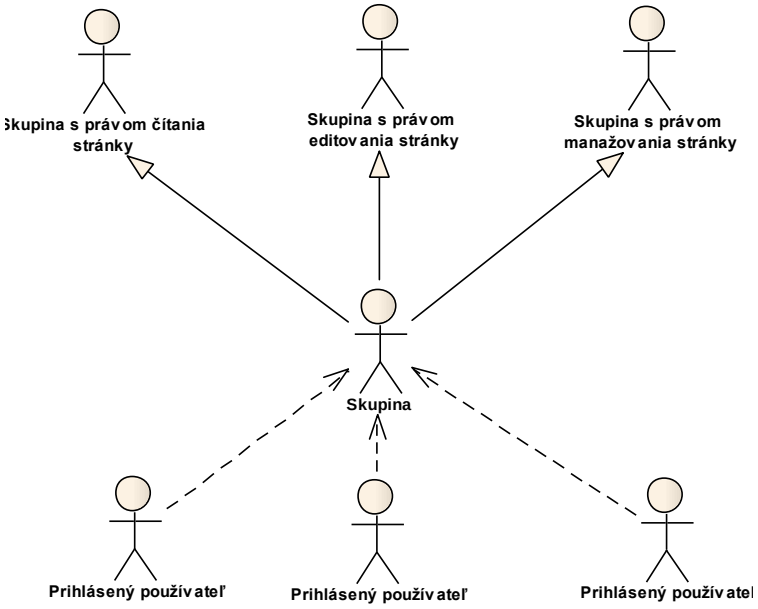
V tabuľke rolí je zoznam základných typov používateľov. Nasledujúce roly sú skôr orientačné. V rámci systému, môže používateľ zastávať aj viacero rolí naraz.

ID používateľa	Názov	Opis
ACT001	Anonymný používateľ	Má univerzálny prístup k wiki, Nemá právo ani vytvárať, ani manažovať a vidí len verejné stránky. Nevidí skupiny, ani používateľov. Nemá právo pridávať prílohy k stránkam.
ACT002	Prihlásený používateľ	Podľa práv ktoré mu boli pridelené, buď jemu samotnému alebo v rámci skupiny, môže vidieť aj editovať neverejné príspevky. Podľa jemu pridelených práv, vidí, edituje, pripadne manažuje skupiny používateľov, ktorým patrí. Má právo pridávať prílohy k stránkam.
ACT003	Manažér stránky	Manažuje viditeľnosť, editovateľnosť stránok jednotlivým používateľom. Manažérom sa môže stať prihlásený používateľ vytvorením samotnej stránky, prípadne tým, že mu, prípadne skupine kam patrí, iný manažér pridelí práva.
ACT004	Editor stránky	Môže upravovať obsah zobrazenej stránky.
ACT005	Čitateľ stránky	Môže prezerať stránku.
ACT006	Skupina	Používateľ môže patriť skupine, pričom práva nastavené skupine, sa vzťahujú automaticky na neho.



Obr. 1 Charakteristika používateľov systému

Pri prvom vstupe na stránku, pred samotným prihlásením vystupuje používateľ v roly anonymného. Po prihlásení, podľa toho, aké práva má na stránku na ktorej sa nachádza, ju môže prezerať, editovať, manažovať.



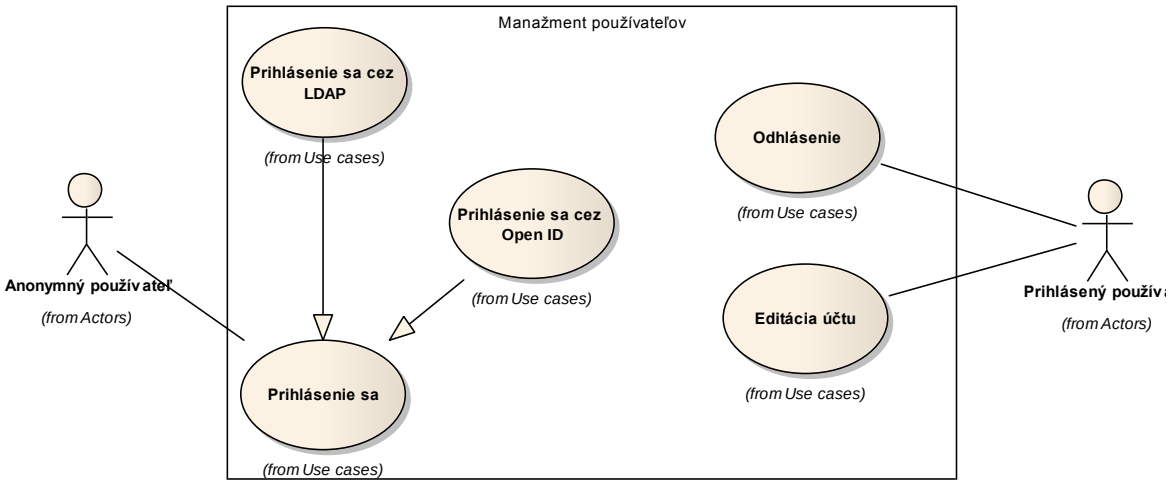
Obr. 2 Charakteristika používateľských skupín systému

Skupina je špeciálny prípad role. Tvorí ju množina používateľov s rovnakými právami pre jednotlivé stránky.

PRÍPADY POUŽITIA

V nasledujúcej kapitole sú rozpísané jednotlivé prípady použitia systému, vytvorené na základe vstupných požiadaviek.

MANAŽMENT PRÍSTUPU POUŽÍVATEĽOV



Obr. 3 Manažment prístupu používateľov

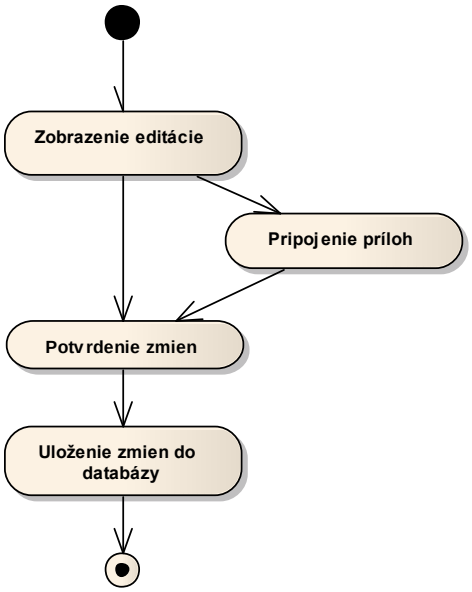
Manažment používateľov popisuje prihlásenie do systému, odhlásenie a editáciu účtu. Registrácia samotných užívateľov nie predmetom požiadaviek z dôvodu získavania dát zo zdrojov, ktoré sú manažované externe.

MANAŽMENT STRÁNOK

Obr. 4 Manažment stránok

Prípad použitia popisuje jednotlivé možnosti správy stránok systému. Bod prezeranie stránok je trochu nejednoznačný. Pokiaľ stránky nie sú verejné - public, ani anonymný užívateľ si ich nemôže prezrieť.

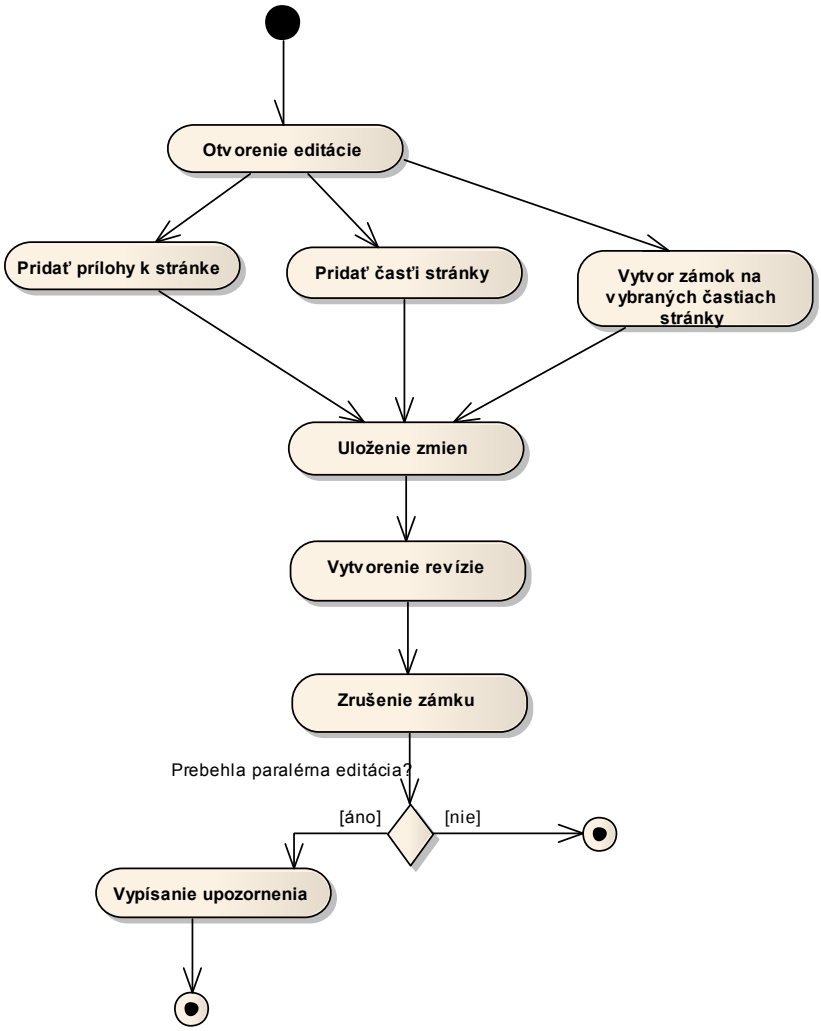
VYTVORENIE STRÁNKY



Obr. 5 Vytvorenie stránky

Princíp vytvorenia stránky spočíva v princípe klasickej wikipédie. Pokiaľ neexistuje stránka pre nami vybraný pojem a používateľ otvorí odkaz s linkou, na ktorej by mala byť v budúcnosti umiestnená stránka, prípadne sa ňu odkáže cez fulltextový odkaz a prebehne načítanie stránky, zobrazí sa odkaz na editor pre vytvorenie novej stránky. Samozrejme k editoru sa dá dostať aj cez linku v hornej lište obrazovky.

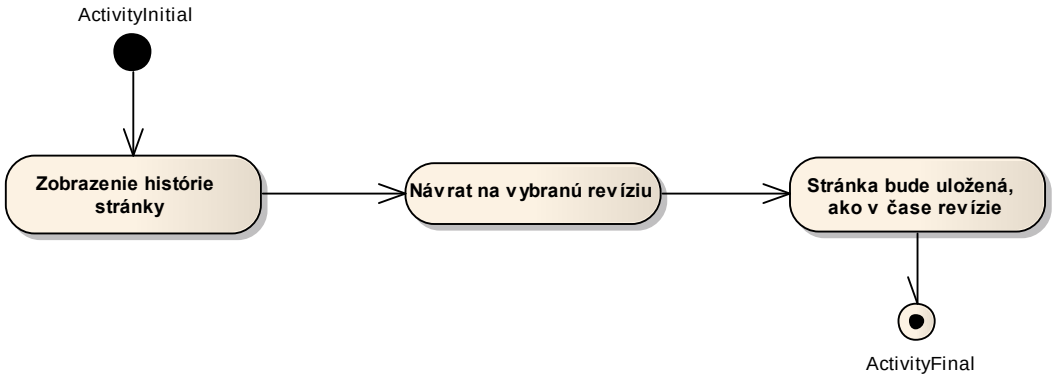
EDITOVANIE STRÁNKY



Obr. 6 Editovanie stránky

Pri editovaní stránky je dôležité si uvedomiť, že jednotlivé upravované časti stránky môžu byť editované naraz dvomi a viac používateľmi. Preto je veľmi dôležitý uzamikací mechanizmus, ktorý upozorní používateľa na to, že zvolenú časť edituje niekto iný.

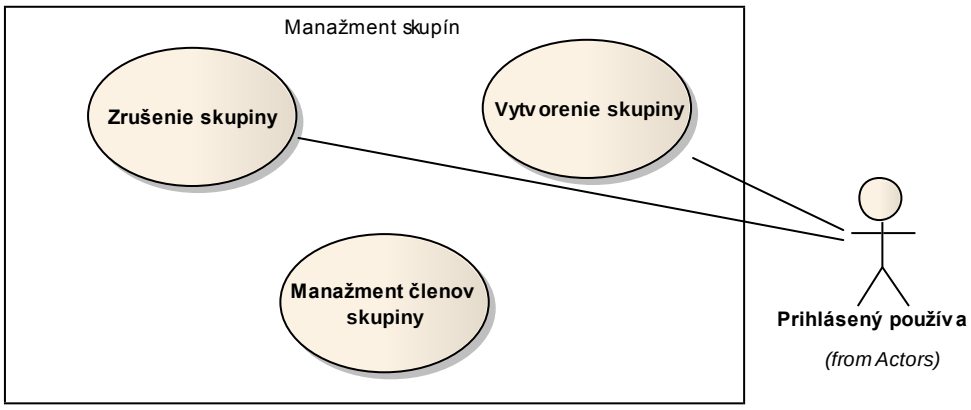
MANAŽMENT REVÍZIÍ STRÁNOK



Obr. 7 Navrátenie revízie stránky

Pre každú stránku je k dispozícií zoznam revízií stránky, jéj história. Pokiaľ je nutné, je dôležité aby bolo možné obnoviť pôvodný stav stránky. Jednotlivé rozdiely vo verziách je možné vidieť vo výstupe obrazovky diff, na ktorú sa taktiež dá dostať z histórie stránky.

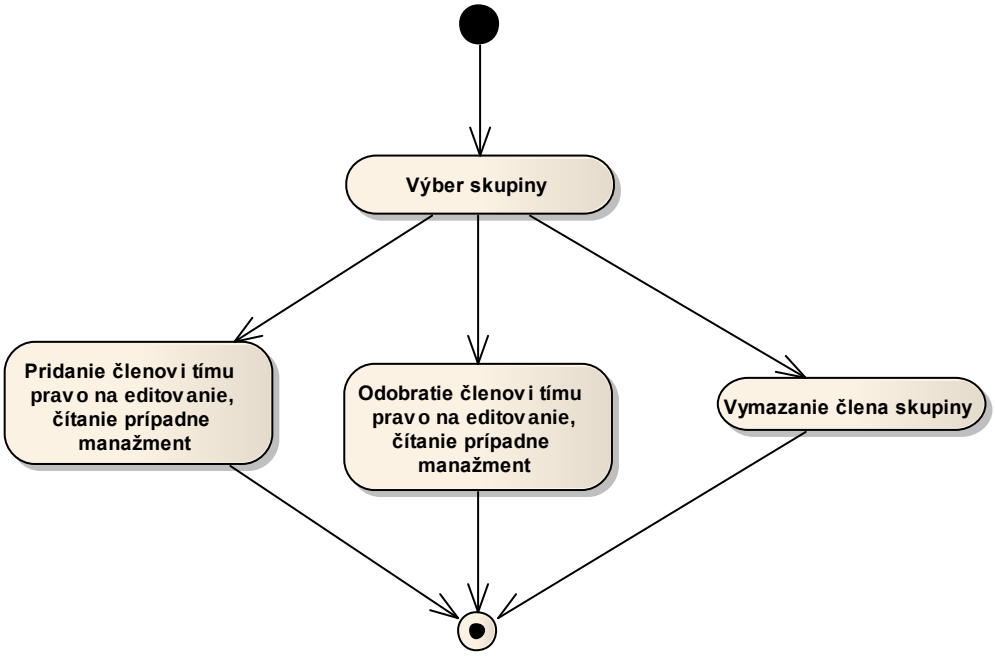
MANAŽMENT SKUPÍN



Obr. 8 Manažment skupín

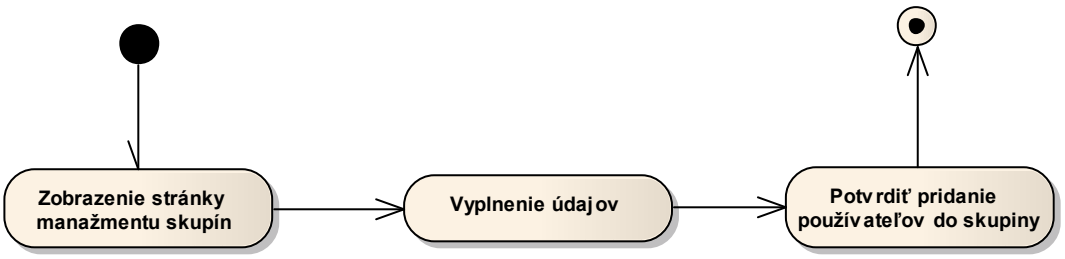
Ako je zrejme z obrázka, užívateľ má právo vytvárať skupiny, rušiť ich, editovať práva, či vidieť jednotlivých členov skupiny podľa jeho práv.

VYTVORENIE SKUPINY A MANAŽMENT ČLENOV SKUPÍN



Obr. 9 Vytvorenie skupiny a manažment členov skupín

Pri zmenách práv členov platí, že nikto si nesmie sám meniť právo na akúkoľvek činnosť v rámci skupiny.



Obr. 10 Vytvorenie skupiny

ARCHITEKTÚRA SYSTÉMU

Na vytvorenie tejto aplikácie je použitý programovací jazyk Ruby a framework Ruby on rails, ktorý je založený na architektúre MVC, čo znamená Model View Controller, v preklade trojvrstvová architektúra. Jeho tri časti tvoria model, view a controller.

Model

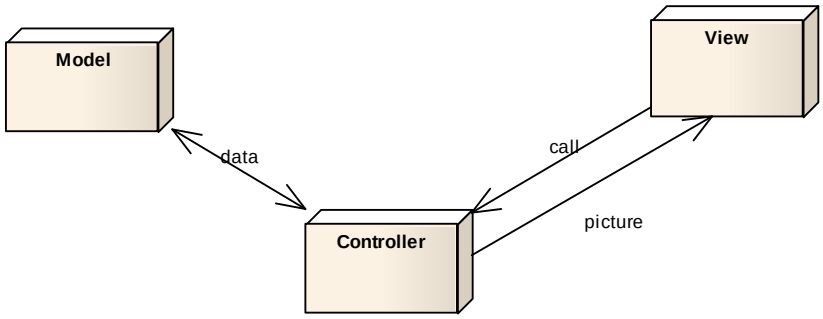
Je doménovo – špecifická reprezentácia dát, nad ktorými pracuje aplikácia. Na sprístupňenie dát z databázy používa návrhový vzor ActiveRecord. Pre každý objekt v modely exituje jeden controller a niekoľko zobrazení - view.

View

Má za úlohu zobraziť dáta, interakcia s používateľom. Posiela požiadavky kontrolnej vrstve.

Controler

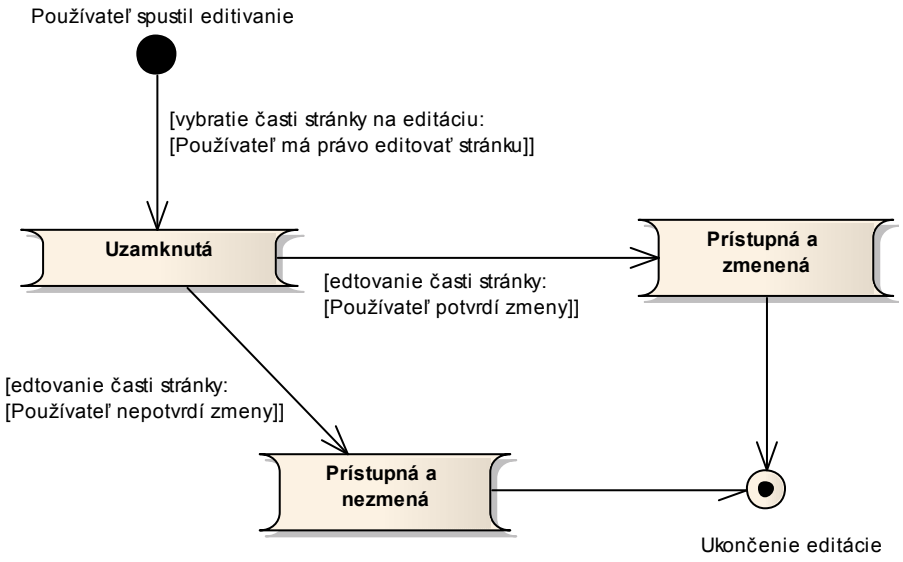
Odpovedá zobrazovacej vrstve, vytvára pre ňu výstup formovaných dát. Komunikuje s Modelom z ktorého získava potrebné dáta.



STAVOVÉ DIAGRAMY

Medzi najdôležitejšie dynamicky sa meniace objekty, ktoré sú opísané v nasledujúcej kapitole sú používateľ, časť stránky a právo na manipuláciu so stránkou.

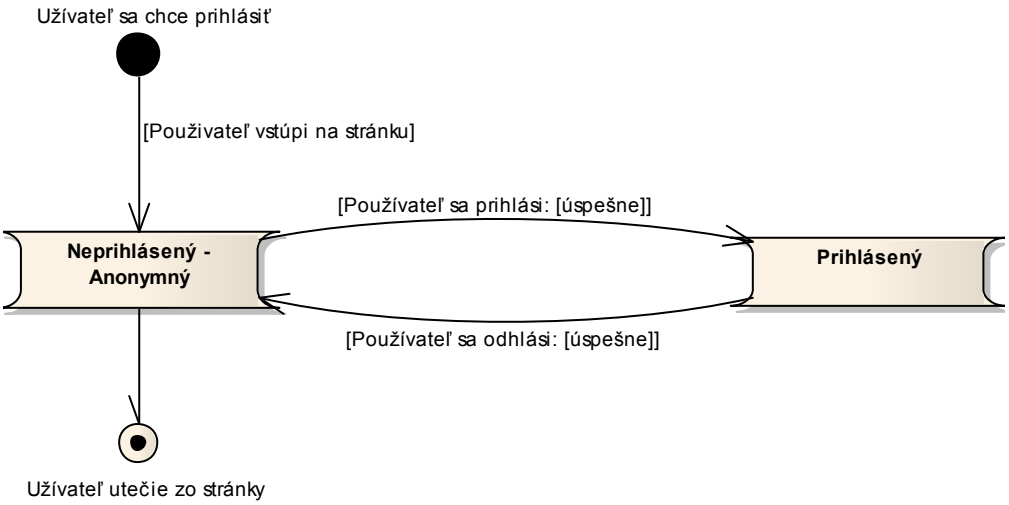
ČASŤ STRÁNKY



Obr. 11 Stavový diagram Časť stránky

Editovanie stránky prebieha vždy len na jej častiach. To umožňuje aby rôzne časti príspevku editovali rôzny používatelia. Pri akejkoľvek úprave je objekt časť stránky uzamknutý, aby používateľ, ktorý ho chce následne editovať, nemohol nevedomky prepísať zmeny pôvodne editujúceho, pred potvrdením jeho zmien. Zámok musí byť vždy len na jednej strane, aby nemohli vzniknúť deadlocky.

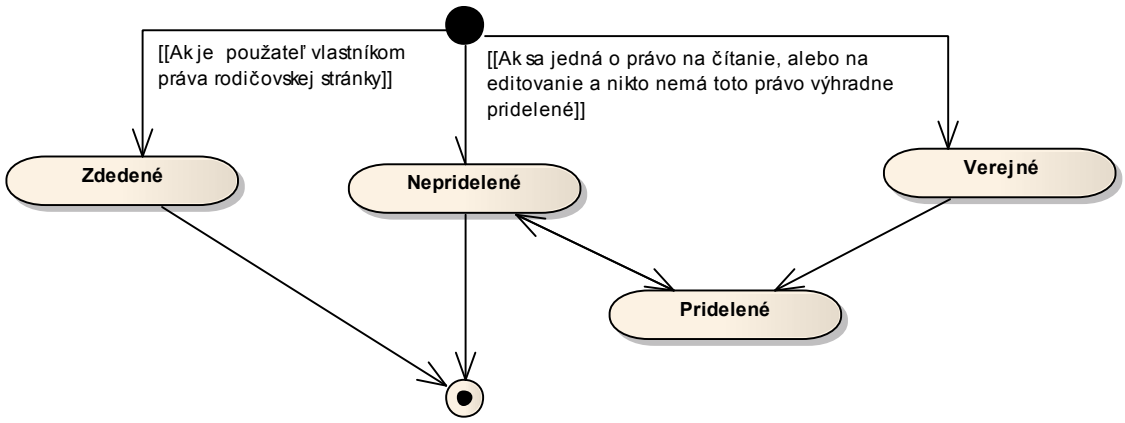
POUŽÍVATEĽ



Obr. 12 Stavový diagram – Používateľ

Každý používateľ začína ako neprihlásený, pokiaľ sa mu podarí prihlásiť na základe LDAP, prípadne Open ID, dostane sa do stavu prihlásený.

POUŽÍVATEĽSKÉ PRÁVA NA STRÁNKU



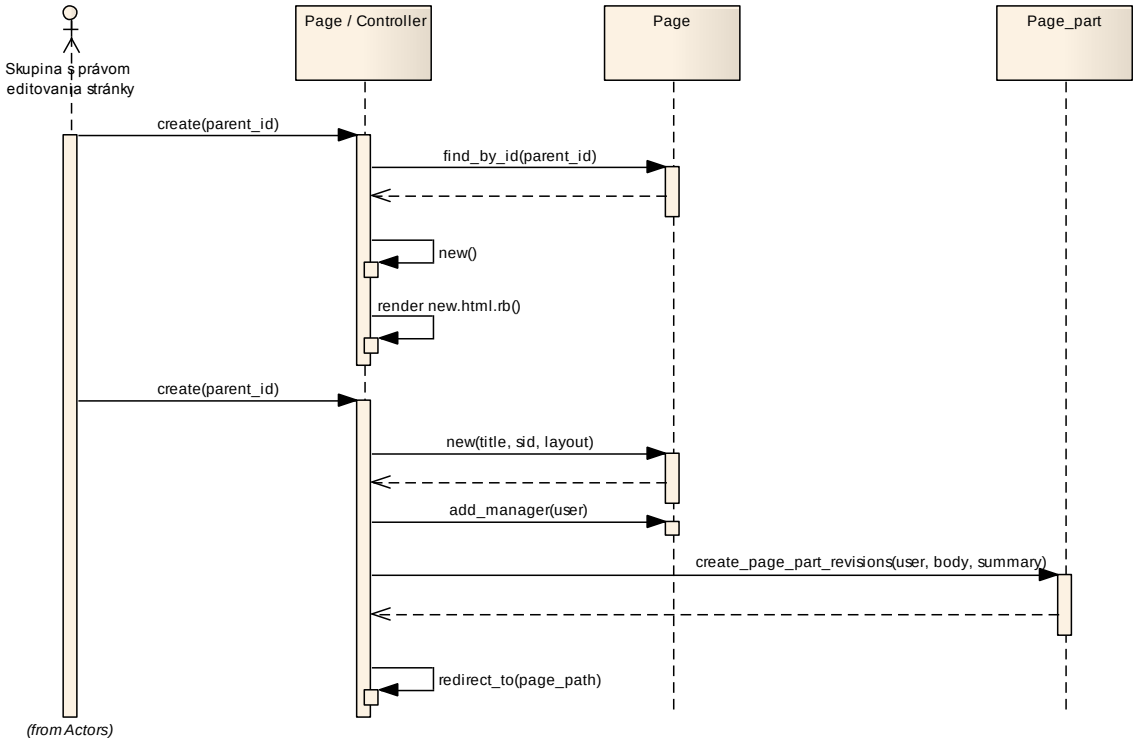
Obr. 13 Stavový diagram - Používateľské práva na stránku

V rámci aplikácie sú použité 3 typy požívateľských práv. Na čítanie, editovanie a manažment stránky. Všetky práva môžu byť používateľovi pridelené aj odobraté. Zároveň môže všetky práva zdediť, podľa toho, ktoré má pridelené pre rodičovskú stránku. Dedičné práva sa nedajú odobrať. Pre právo čítania platí výnimka, že pokiaľ nie je nikomu pridelené, stránka je verejná a pre právo editovania platí, že pokiaľ nie je niekomu toto právo pridelené, stránka je editovateľná všetkými prihlásenými používateľmi.

SPRÁVANIE SYSTÉMU

V nasledujúcich kapitolách je opísané správanie systému a interakcie medzi objektmi.

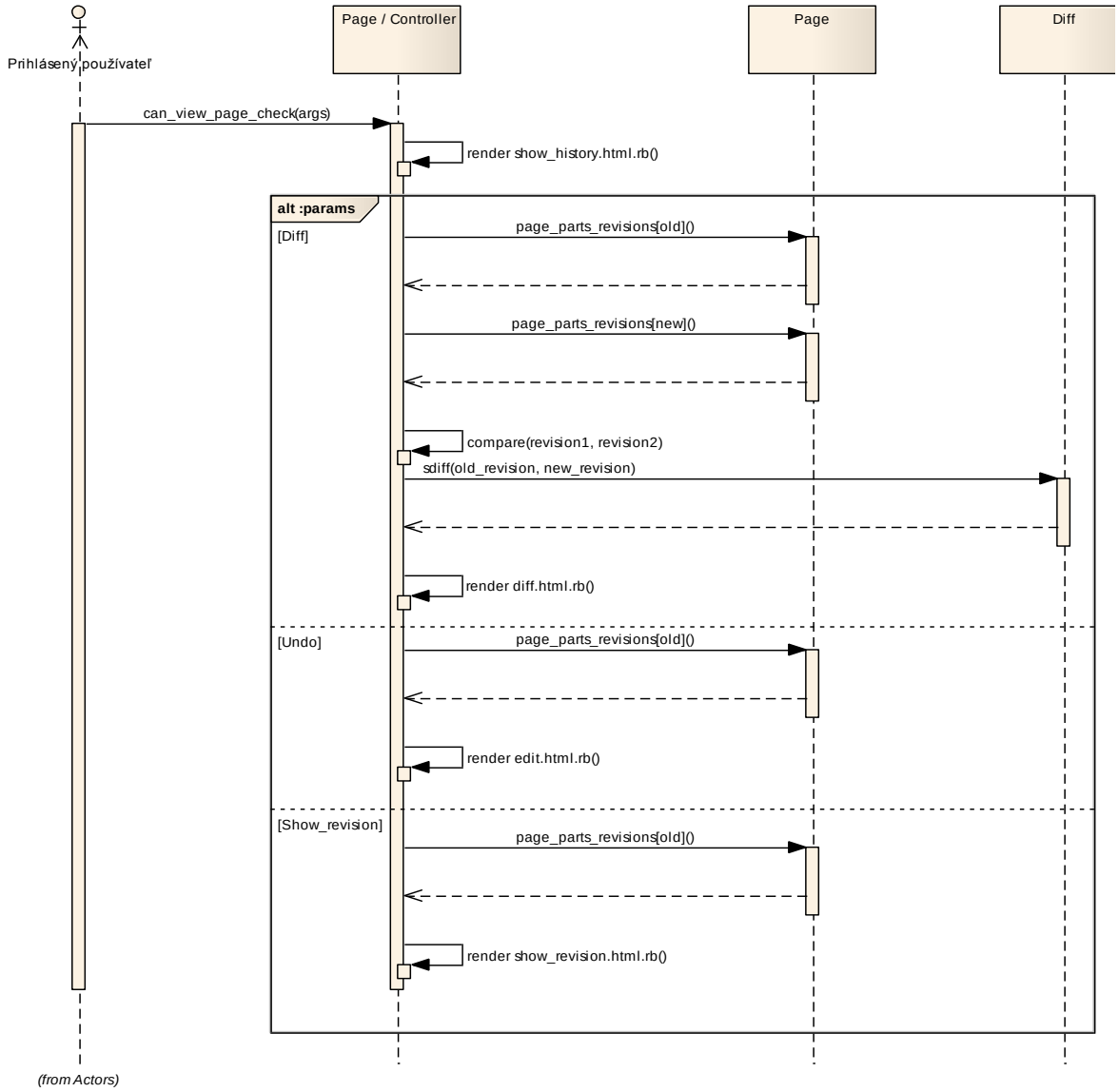
VYTVORENIE STRÁNKY



Obr. 14 Vytvorenie stránky

Proces najprv hľadá rodičov stránky, podľa cesty, na ktorej vytvárame stránku a poskytne používateľovi editovacie rozhranie (new.html.rb). Následne vytvorí stránku, priradí práva, vytvorí časť stránky body – tú každá stránka musí mať, a presmeruje používateľa na zobrazenie stránky.

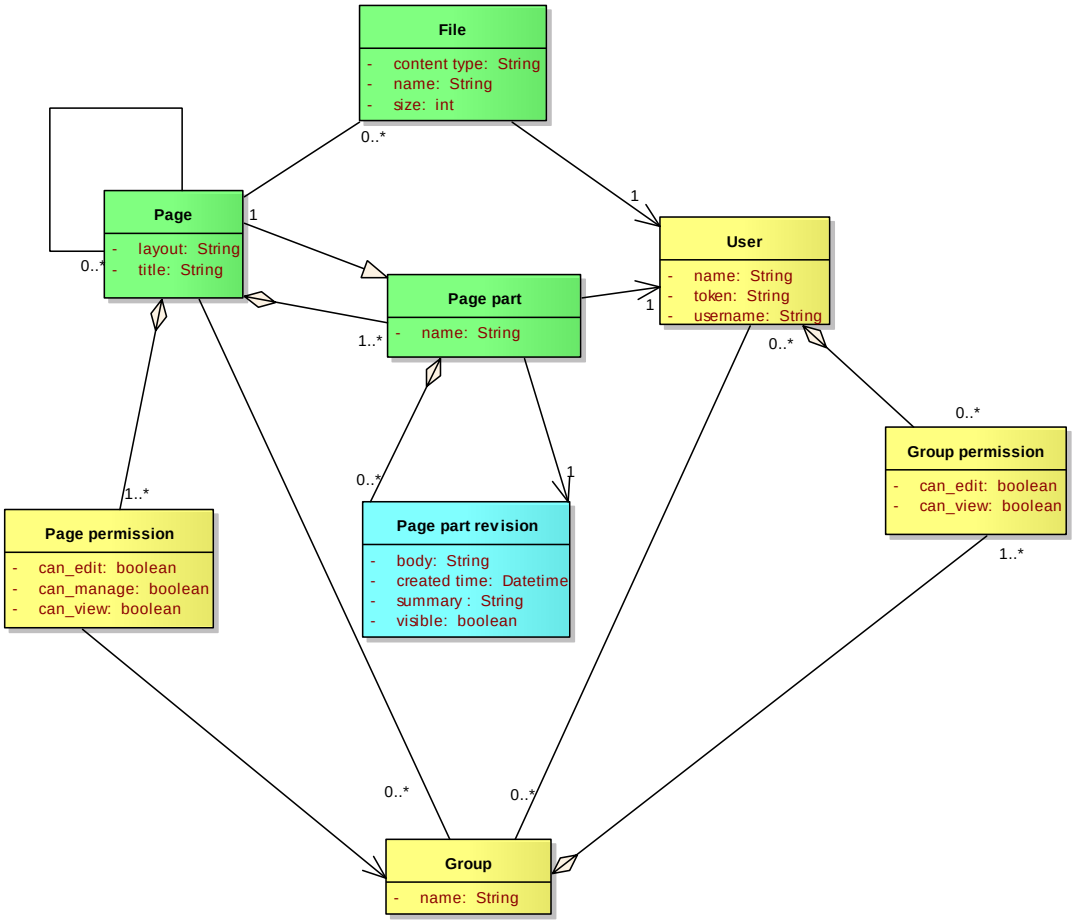
MANAŽOVANIE HISTÓRIE STRÁNOK



Obr. 15 Manažovanie histórie stránok

Diagram popisuje nastavenie a zmenu prístupových, editovacích a manažovacích práv na stránku, vymazanie správ a nastavenie stránky na public – viditeľnú všetkými a editable – editovateľnú všetkými používateľmi.

DÁTOVÝ MODEL



Obr. 16 Dátový objektový model

Ako v klasickej wikipédii, základný a najdôležitejší objekt je page (stránka). Objekty stánky sú v stromovej štruktúre, ktorej hlavný vrchol predstavuje hlavná stránka wiki a jej deti sú jednotlivé podstránky.

Stránka sa skladá s viacerých page partov (častí stránky). Každá stránka má minimálne jednu časť – body (telo), pričom je možné jej dodefinovať iné časti s ľubovoľným menom. Každý page part je samostatne editovateľný, t.j. pokiaľ je treba zmeniť len časť textu stránky, nie je treba updatovať celú.

Každý page part má navyše vlastnú množinu revízií, obsahujúcu históriu zmien v danom page parte. Každá stránka má navyše zoznam príloh, súborov, ktoré reprezentuje objekt Files(súbor). Používateľa predstavuje objekt user, ktorý môže vkladať súbory a vytvárať page party.

Všetko však závisí od jeho prístupových práv. Každý používateľ je vždy v aspoň jednej skupine, ktorá sa volá privátna a nesie jeho meno. Samozrejme okrem iného, môže patriť skupinám, ktoré sú vytvorené nejaký používateľom. Práva na stránku sa dedia.

Pokiaľ má používateľ práva na rodiča stránky, nikto mu tie práva neodníme. Skupina je v dátovom modely reprezentovaná objektom group. Každá grupa môže mať práva na úpravu, čítanie, či manažovanie stránky. Tie reprezentuje objekt page permission. Objekt group permission reprezentuje práva členov skupiny na to či vidia ostatných členov, prípadne či ich môžu upravovať a meniť obsadenie skupiny.

ZOZNAM PRÍLOH

1. Zápisnica z 1. stretnutia
2. Zápisnica z 2. stretnutia
3. Zápisnica z 3. stretnutia
4. Zápisnica z 4. Stretnutia
5. Zápisnica z 5. stretnutia
6. Zápisnica z 6. stretnutia
7. Zápisnica z 7. stretnutia
8. Zápisnica z 8. Stretnutia
9. Zápisnica z 9. Stretnutia
10. Zápisnica z 10. Stretnutia
11. Zápisnica z 11. Stretnutia
12. Zápisnica z 12. Stretnutia
13. Zápisnica z 13. Stretnutia
14. Zápisnica z 14. Stretnutia
15. Zápisnica z 15. Stretnutia
16. Zápisnica z 16. Stretnutia
17. Zápisnica z 17. Stretnutia
18. Zápisnica z 18. Stretnutia
19. Zápisnica z 19. Stretnutia
20. Zápisnica z 20. Stretnutia
21. Vývoj produktového backlogu počas riešenia tímového projektu