Slovenská technická univerzita Bratislava
Fakulta informatiky a informačných technológií
Študijný odbor: SOFTVÉROVÉ INŽINIERSTVO

Využitie sociálnych sietí pri vytváraní pracovných tímov

Tímový projekt

Dokumentácia k inžinierskemu dielu

Tím č. 6 – Awesome Legends
Bc. Stanislav Jurský
Bc. Martin Valašík
Bc. Lukáš Repka
Bc. Martin Sirota
Bc. František Chvostaľ
Bc. Dušan Zahoranský

Školský rok: 2008/2009
Vedúci tímového projektu : Ing. Michal Barla

Obsah

Slovník pojmov

LDAP	Lightweight Directory Access Protocol, je aplikačný protokol slúžiaci na dotazovanie a modifikovanie adresárových služieb. V tomto projekte slúži na účely overovania prihlasovacích údajov.
AJAX	Asynchronous JavaScript and XML. Ide o skupinu bežne používaných techník pri programovaní webových stránok na strane klienta. Cieľom je vytváranie stránok schopných dynamicky meniť svoj obsah bez nutnosti znovunačítania.

1. Zadanie

Výsledok snaženia študenta v predmete Tímový projekt je logicky veľmi závislý od toho, ako sa na začiatku povytvárajú jednotlivé tímy. Ak by bol tento proces úplne neriadený, mohlo by ľahko dôjsť k vzniku extrémnych situácií: vzniklo by niekoľko tímov zložených zo samých vynikajúcich študentov ale rovnako aj niekoľko tímov v ktorých by boli len študenti s podpriemernými výsledkami. Zatiaľ čo v prípade tímu vynikajúcich študentov by zrejme problém počas riešenia projektu nenastal (čo je z hľadiska cieľov predmetu Tímový projekt trošku problém :)), tím podpriemerných študentov by bez niekoho, kto tím v správnej chvíli potiahne, pravdepodobne neuspel. Ideálne je, keď sú tímy čo najviac vyvážené, aby si každý tím vyskúšal riešenie rôznych problémov, ktoré vyplývajú z rôznorodosti jednotlivých členov a kde sa slabší môžu niečo naučiť od tých lepších. Taktiež je potrebné myslieť na vyváženosť zručností jednotlivých členov tímu (tím zložený zo samých GUI dizajnérov bez databázistu sa bude trápiť).

Pri navrhovaní zloženia jednotlivých tímov treba zohľadniť:

- predchádzajúce skúsenosti, zručnosti budúcich členov tímu
- predchádzajúce spolupráce budúcich členov tímu
- preferencie, čo by kto chcel robiť v tíme
- preferencie, kto by s kým chcel/nechcel byť v tíme a z akých dôvodov
- povahové vlastnosti budúcich členov tímu
- a rôzne ďalšie atribúty...

Úlohou tímu je navrhnúť a vytvoriť systém, ktorý na základe priamych a nepriamych vstupov (formuláre, iné fakultné systémy a pod.) vytvorí profily jednotlivých študentov a prepojí ich do jednej sociálnej siete na základe rôznych zadaných aj odvodených vzťahov. Vhodnou vizuálizáciou takejto sociálnej siete a poskytnutím efektívnych nástrojov na jej ďalšiu analýzu (napr. známe algoritmy na sociálnu analýzu sietí, ktoré určia populárne a inak významné body siete) systém podporí proces vytvárania tímov študentov v predmete Tímový projekt na našej fakulte (čo však určite nie je jediná aplikácia, na ktorú sa takáto sociálna sieť dá efektívne využiť ;)).

2. Úvod

Tento dokument je výsledkom práce tímu č. 6 počas zimného semestra v predmete Tvorba softvérového systému v tíme. Cieľom bolo vytvoriť dynamickú sociálnu sieť tvorenú študentmi fakulty a slúžiacu pedagógom na uľahčenie práce spojenej s tvorbou tímov. Konkrétne išlo o zbieranie údajov pomocou formulárov a z rôznych informačných systémov školy, ich následné klasifikovanie, utriedenie, spracovanie a následné vyhodnocovanie.

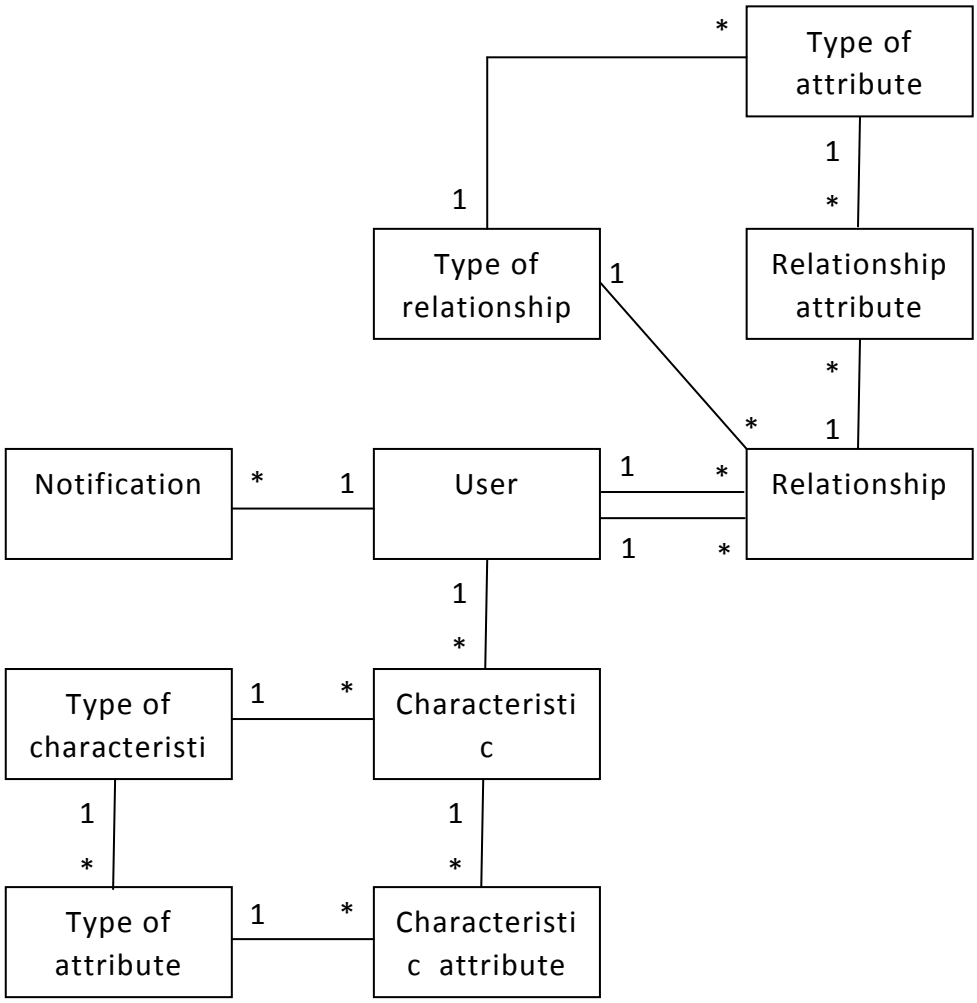
Na odporúčanie tímového vedúceho a aj na základe vlastných skúseností sme sa priklonili k agilnému spôsobu vývoja a to konkrétne k technike Scrum. Ide o iteratívny, inkrementálny proces, ktorý sa v súčasnej dobe hojne využíva pri tvorbe softvérových systémov, ale aj v iných oblastiach. Jeho hlavnou črtou sú tzv. šprinty, ktoré predstavujú jednu iteráciu procesu. Ďalším dôležitým rysom sú tímové stretnutia, z ktorých sa neskôr robia zápisnice. V tomto ohľade sa teda Scrum zhodoval s požiadavkami v tímovom projekte. Samotný proces sme si mierne upravili, šprinty sme skrátili z jedného mesiaca na 2 týždne a interval stretnutí naopak predĺžili, z jedného dňa na jeden týždeň.

Požiadavky zákazníka sú v technike Scrum riešené pomocou tzv. *user stories*, ktoré sa vždy viažu na šprint. Z nich sú následne vytvorené konkrétne úlohy pre členov tímu.

Nakoľko náš proces vývoja sa nezhodoval z bežným procesom vývoja počas tímového projektu, rozhodli sme sa modifikovať aj štruktúru dokumentácie. Namiesto obvyklých veľkých kapitol zodpovedajúcich vodopádovému vývoju budeme projekt opisovať podľa šprintov a samotných úloh v danom šprinte. Každá takáto úloha má svoju vlastnú malú analýzu, návrh a implementáciu a šprint má svoje ciele.

3. Šprinty

Ako už bolo spomenuté vyššie, každý šprint predstavuje jednu iteráciu v agilnom vývoji. Počas neho by mali v určitej miere prebehnúť všetky procesy bežné vo vodopádovom vývoji a výsledkom by mala byť do určitej miery funkčná aplikácia.
Pred začiatkom každého šprintu sme si definovali už vyššie spomenuté *user stories*, teda požiadavky na systém z hľadiska klienta. Z nich sa vytvorili úlohy, z ktorých každá spadá práve do jednej *user story*.
Ešte pred začatím prvého šprintu sme spoločne vytvorili logický dátový model aplikácie, ktorý je možno vidieť na obr. 1.



Obr. 1 – Logický dátový model aplikácie

3.1. Šprint č. 1

V prvom šprinte, ktorý trval od 15.10.2008 do 5.11.2008, sme sa sústredili najmä na dobré odrazenie sa a začatie projektu. Kvôli úvodnej chybe pri zadávaní šprintu trval 3 týždne namiesto dvoch. Išlo tu hlavne o vymyslenie a čiastočné implementovanie dátového modelu aplikácie a vytvorenie prvej, veľmi ranej verzie aplikácie. Počas tohto šprintu sme celkovo uzatvorili 3 user stories.

3.1.1. Prihlasovanie do systému cez LDAP

Používateľ sa prihlási s prihlasovacími údajmi z AIS aby sa dostal k funkcionalite systému. Na základe údajov LDAP servera a AIS databázy sa mu priradí rola v systéme.
Pre túto *user story* bola vytvorená len jedna úloha.

Nasadenie prihlasovania cez LDAP

Cieľom úlohy bola integrácia prihlasovania pomocou metódy LDAP. Študent, ktorý už má konto v školskom informačnom systéme si teda nemusí vytvárať nový účet pre našu aplikáciu, ale môže použiť prihlasovacie údaje z AIS.

Analýza problému

Pre správnu prácu väčšiny aplikácii je potrebné vedieť, kto danú aplikáciu používa. Ako riešenie sa bežne používa prihlasovanie, pričom aplikácie môžu používateľa overovať prostredníctvom svojej lokálnej databázy, alebo cez server tretej strany. V našom prípade použijeme kombinovaný spôsob. Používatelia, ktorí majú heslo uložené v lokálnej databáze sa prihlasujú cez ňu a v prípade, že používateľ nie je v lokálnej databáze, alebo tam nemá heslo, overujú sa údaje cez LDAP sever ldap.stuba.sk .

Návrh

Na prihlasovanie je potrebné v databáze uchovávať pre jednotlivých používateľov ich prihlasovanie meno a heslo. Heslo je uložené ako SH1 hash hesla kvôli bezpečnosti. V prípade, že používateľ nemá uložené heslo v lokálnej databáze, overia sa jeho prihlasovacie údaje cez ldap server ldap.stuba.sk . V prípade, keď sa používateľ, overený cez ldap server, nenachádza v lokálnej databáze vytvorí sa pre neho položka v databáze a z ldap serveru sa zistia informácie o používateľovi, konkrétne meno, priezvisko a rola v akej vystupuje.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Models: *user.rb*
- V zložke Controllers: login_*controller.rb*

Na prihlásenie používateľa slúži metóda login v *login_controller.rb* ktorá zavolá metódu autheticate z *user.rb*, ktorá vykoná logiku prihlasovania.

Testovanie

V zložke *Functional Tests* v súbore *login_controller_test.rb* je vytvorený testovací algoritmus pre overenie funkčnosti *login_controle.rb.* V zložke *Unit Tests* v súbore *user_test.rb* je vytvorený testovací algoritmus pre overenie funkčnosti *user.rb*

3.1.2. Vytváranie sociálnej siete študentom

Úloha požadovala vytvorenie formuláru, kde študent vidí ostatných študentov aktuálneho ročníka a môže vyjadrovať svoj vzťah k nim. Pri pridaní vzťahu sa podľa typu vzťahu môže, alebo nemusí čakať na potvrdenie vzťahu druhou stranou. Pri inverzných vzťahoch sa automaticky doplní aj opačný smer vzťahu.
Iba jedna úloha bola priradená k tejto *user story*.

Formulár a model na vytvorenie a upravovanie vzťahu zo zoznamu používateľov

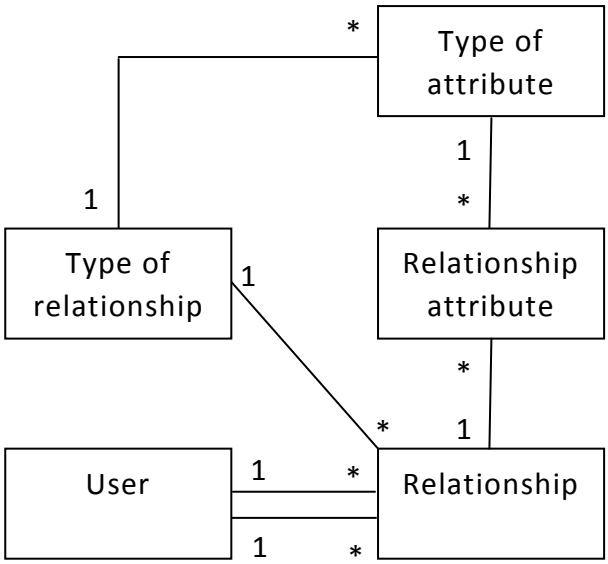
Táto úloha vyžadovala vytvorenie kompletnej aplikačnej logiky nad databázou z hľadiska vzťahov medzi používateľmi.

Analýza problému

Niekedy sa dá vytvoriť vzťah medzi používateľmi na základe ich charakteristík. Napríklad vzťah spolužiaci možno vytvoriť podľa navštevovaného ročníka. Niekedy je však potrebné, aby bol vzťah k inému používateľovi definovaný explicitne používateľom. Niektoré vzťahy je možné definovať ako inverzné (priatelia, kamaráti) či dokonca tranzitívne (spolužiaci).

Návrh

Na obr. 2 je vidieť časť databázového modelu použitú na reláciu vzťahu medzi používateľmi. Kvôli existencii inverzných resp. obojstranných vzťahov sa v modeli nachádza dvojitá hrana medzi používateľom a vzťahom. Relácia vzťahu je totiž navrhovaná ako orientovaná hrana od koho – ku komu.
Formulár vzťahov daného používateľa pribudne do formuláru pre profil jedného študenta. Bude obsahovať informáciu od koho a ku komu vzťah smeruje i typ vzťahu.



Obr. 2 – Model vzťahov

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Models: *relationship.rb, user.rb, type_of_relationship.rb*
- V zložke Controllers: *relationships_controller.rb, users_controller.rb, type_of_relationships_controller*
- V zložke Views: *layouts/users.html.erb, users/_show_relationship_detail.html.erb, users/_show_relationships.html.erb*

O zabezpečenie zobrazenia jedného vzťahu sa stará *_show_relationship_detail.html.erb*, ktorý je renderovaný z *_show_relationships.html.erb*. Ten je volaný zo zobrazenia profilu používateľa *users.html.erb*.

Testovanie

Testovanie prebieha nad databázou *2008team06isp2*, kde sa nachádzajú testovacie dáta pre vzťahy. V zložke *Functional Tests* v súbore *relationships_controller_test.rb* je vytvorený testovací algoritmus na overenie funkčnosti implementácie vzťahov nachádzajúcej sa v zložke Controllers.

3.1.3. Prechádzanie profilov študentov

Pedagóg vidí zoznam študentov zvoleného ročníka spolu s informáciou o vyplnenom profile a možnosti prezerania daného profilu.
Pre túto *user story* boli vytvorené dve úlohy.

Vytvorenie databázového modelu pre profil

V tejto úlohe išlo o vytvorenie modelu profilu, ktorý sa neskôr použije pri práci s formulárom.

Analýza problému

Sociálna sieť bude uchovávať množstvo údajov o jednotlivých používateľoch. Či už sa jedná o dáta získané od samotného používateľa, alebo ide o dáta získané z externých informačných zdrojov, tieto dáta musia byť uložené spôsobom, ktorý bude umožňovať ich uchovávanie bez ohľadu na typ informácie a bude taktiež obsahovať základné metadáta a danej informácii.

Návrh

Model pre profil používateľa, ktorý je na obr. 3, obsahuje 5 entít:
Characteristics – Prepojovacia tabuľka. Bude prideľovať používateľovi atribúty jednej charakteristiky určitého typu.
Type of characteristic – Obsahuje základné údaje o type charakteristiky
Type of attribute – Obsahuje základné údaje o type atribútu. Každý typ charakteristiky môže mať jeden a viac typov atribútov.
Characteristic attribute – Atribút charakteristiky určitého typu.
User – samotný používateľ

Entitám modelu profilu používateľa prislúchajú nasledujúce Views:

1. Zobrazenie všetkých typov charakteristík
2. Vytvorenie nového typu charakteristiky
3. Zobrazenie jedného typu charakteristiky. Obsahuje taktiež formulár na vytvorenie typu atribútu a výpis typov atribútov prislúchajúcich zobrazenému typu charakteristiky.
4. Editovanie typu charakteristiky
5. Editovanie typu atribútu.
6. Vytvorenie charakteristiky určitého typu
7. Zobrazenie charakteristiky spolu s formulárom na vytvorenie atribútu a s výpisom všetkých atribútov danej charakteristiky.
8. Editovanie atribútu charakteristiky

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Models: *type_of_char.rb, type_of_attribute.rb, characteristic.rb, char_attribute.rb*
- V zložke Views/char_attributes: edit.*html.erb*
- V zložke Views/characteristics: new.*html.erb, show.html.erb*
- V zložke Views/type_of_attributes: edit.*html.erb*

- V zložke Views/chars: edit.*html.erb, index.html.erb, new.html.erb, show.html.erb*
- V zložke Controllers: char_attributes_controller.*rb, characteristics_controller.rb, type_of_attributes_controller.rb, type_of_chars_controller.rb*

Zobrazenie zoznamu študentov a ich profilov

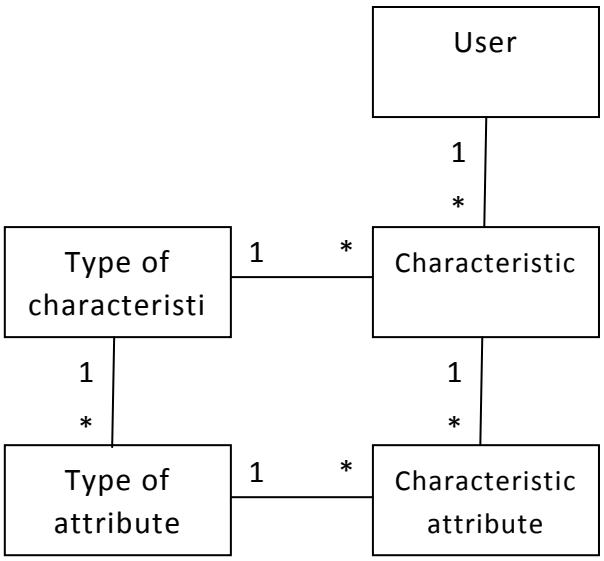
Táto úloha vyžadovala zobrazenie študentov pomocou modelu vytvoreného v predchádzajúcej úlohe. V zobrazenom zozname mala aplikácia automaticky identifikovať študentov, ktorí majú vytvorení profil a umožniť zobraziť tieto hodnoty.

Analýza problému

Pri práci z veľkým množstvom dát je pre používateľa prioritou ich prehľadné usporiadanie a zobrazenie kľúčových údajov. Sociálna sieť vo vyvíjanom systéme má potenciál uchovávania informácií o množstve študentov a podrobných charakteristík každého z nich. Z tohto dôvodu je potrebné používateľovi poskytnúť možnosti ako si tieto dáta prezerať, zobraziť podrobný profil každého z nich a umožniť mu tieto dáta filtrovať podľa potreby.

Návrh

Zobrazenie profilov využíva časť databázového modelu pre uchovávanie charakteristík jednotlivých študentov. Úlohou je zobrazenie dát, nie ich rozmiestnenie a formátovanie na formulári, tieto aspekty sú riešené v neskorších úlohách projektu. Obr. 2 zachytáva časť databázového modelu, ktorý je k tejto úlohe relevantný.



Obr. 3 – Model profilu používateľa

Zobrazenie profilov sa skladá z dvoch formulárov:

1. Formulár, ktorý zobrazí zoznam všetkých používateľov z tabuľky *User*. Tento formulár obsahuje aj komponenty na filtrovanie zobrazených používateľov, úloha však nepokrýva samotný algoritmus filtrovania, je to príprava formuláru pre použitie v neskorších fázach projektu. Časť formuláru pre filtráciu sa skladá z týchto komponentov:
   - Combobox na výber typu charakteristiky, dáta z tabuľky *Type of characteristic*
   - 3 krát combobox na výber typov atribútov, dáta z tabuľky *Type of attribute*
   - Tlačidlo pre spustenie filtrácie

2. Formulár profilu jedného študenta. Tento formulár zobrazuje jeden záznam z tabuľky *User* spolu so všetkými jeho charakteristikami, čo sú záznamy z tabuľky *Characteristic*, ktoré sa k nemu vzťahujú. Ku každej charakteristike sa zobrazia všetky vzťahujúce záznamy z *Characteristic attribute* spolu s ich typom, ktorý sa vyhľadá v tabuľke *Type of attribute*.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Models: *user.rb*
- V zložke Views/users: *index.html.erb, show.html.erb*
- V zložke Controllers: *user_controller.rb*

Zobrazenie zoznamu študentov zabezpečuje metóda index v *user_controller.rb* ktorá prečíta všetkých používateľov z databázovej tabuľky *User* a zobrazí ich na formulár. Zobrazenie detailného profilu je implementované v metóde show v *user_controller.rb*.

Testovanie

K testovaniu funkcionality je vytvorená štruktúra tabuliek z obr. 2 naplnená testovacími dátami v databáze *2008team06isp2*. V zložke *Unit Tests* je vytvorený testovací algoritmus pre zobrazenie študentov a ich profilov v súboroch *user_test* a *user_controller_test.*

3.2. Šprint č. 2

Druhý šprint sme začali 5.11.2008 a ukončili 12.11.2008. Kvôli kompenzácii prvého trojtýždňového šprintu sme tento skrátili na jeden týždeň. Cieľom bolo ďalšie vylepšovanie aplikácie a dopĺňanie funkčnosti. Celkovo sme definovali 6 rôznych *user stories*.

3.2.1. Získavanie informácií o študentovi od študenta

Vytvoriť formulár, kde študent môže zadať, skontrolovať, alebo zmeniť svoj profil a ďalší formulár, pomocou ktorého študent absolvuje psycho-test.
Táto *user story* zatiaľ nebola realizovaná.

Prepracovať formulár na zobrazenie charakteristiky

Úloha požadovala, aby pomocou modelu vytvoreného v úlohe „Vytvorenie databázového modelu pre profil“ bolo možné zobraziť zoznam atribútov a charakteristík priradených používateľovi. Ďalej zahrňovala vytvorenie rozhrania pre správu týchto informácií, ktoré bude umožňovať vytvorenie, zobrazenie, editáciu a vymazanie charakteristík a ich atribútov predstavujúcich profil študenta.

Analýza problému

Údaje o študentoch, ktoré budú tvoriť dôležitú súčasť sociálnej siete je potrebné do systému zadať. Niektoré údaje sa budú importovať automaticky z externých informačných systémov, iné je potrebné zadať priamo prostredníctvom formulára. Tieto údaje je potrebné zobraziť a v prípade potreby editovať alebo vymazať. Preto je nutné vytvoriť prehľadné rozhranie na správu týchto údajov.

Návrh

Vytvorenie novej charakteristiky:
Na vytvorenie novej charakteristiky je potrebné zadať iba typ charakteristiky. Po vytvorení, je možné začať s priraďovaním atribútov k tejto charakteristike, pričom je možné pridať len atribúty, ktorých typ je priradený danému typu charakteristiky.

Vytvorenie nového atribútu charakteristiky:
Pridá do zoznamu atribútov charakteristiky novovytvorený atribút. Na vytvorenie atribútu je potrebné zadať jeho názov a typ.

Zobrazenie atribútov zvolenej charakteristiky:
Zobrazí typ charakteristiky a používateľa, ktorému bola daná charakteristika priradená. Nasleduje výpis zoznamu atribútov zvolenej charakteristiky, ktorý obsahuje typ atribútu, názov, čas kedy bol vytvorený, a odkazy na jeho editáciu, resp. odstránenie.

Editácia atribútu:
Umožní používateľovi zmeniť názov atribútu.

Vymazanie charakteristiky:
Zmaže charakteristiku používateľa obsahujúcu atribúty.

Vymazanie atribútu:
Odstráni zvolený atribút zo zoznamu atribútov danej charakteristiky používateľa.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Views/characteristics: new.*html.erb, show.html.erb*
- V zložke Views/char_attributes: edit.*html.erb*
- V zložke Controllers: *characteristics_controller.rb, char_attributes_controller.rb*

3.2.2. Vizualizácia prepojení

Pedagóg má možnosť prezerať vizualizovanú sociálnu sieť študentov. Po kliknutí na uzol sa mu zobrazí profil študenta, pričom má možnosť aplikovať na danú sieť rôzne filtre, analyzátory, alebo zhlukovanie.
K tejto *user story* neboli pridelené úlohy, ale bola vypracovaná predbežná analýza. Jej implementácia sa presúva do ďalšieho šprintu.

Analýza nástrojov na vizualizáciu grafu

V analýze sú priblížené vybrané nástroje na vizualizáciu grafov. Všetky tieto nástroje sú voľne dostupné.

JUNG

Vykresľovanie grafu je možné manažovať a vykresľovať pomocou rôznych implementovaných rozložení. K dispozícii sú rozloženia pre hierarchické vizualizácie stromov a nehierarchické vizualizácie grafov so staticky vypočítanými pozíciami objektov a následným vykreslením alebo aj dynamicky počítanými pozíciami a priebežným vykresľovaním.
Pre hierarchické a nehierarchické vizualizácie je ale potrebné použiť inú reprezentáciu grafu. Tá sa vytvára pridávaním objektov (*Vertex*) a hrán (*Edge*). Vo verzii *JUNG 2.0* sa ako hrany a objekty pridávajú ľubovoľné objekty s tým, že sa každému objektu priradí aj jedinečný identifikátor. Ten môže byť opäť ľubovoľným objektom.
JUNG poskytuje možnosť vlastnej alebo úpravu už existujúcej vizualizácie objektov a hrán. Poskytuje aj dobrú podporu sledovania akcií na grafe a tak ponúka vytvorenie vlastnej interakcie s grafom. Podpora *GraphML* je žiaľ pre svoju zlú

implementáciu nepoužiteľná, ale k dispozícii je taktiež už použiteľná podpora Pajek formátu.
JUNG má API dokumentáciu k verzii 1.7.6 a aj pre verziu 2.0 . Manuál je žiaľ k dispozícii iba k verziám 1.0 a z externého zdroja k 2.0. Jednotlivé verzie však nie sú navzájom kompatibilné.

Prefuse

Prefuse využíva na spracovávanie a interakciu s grafom akcie. Pomocou tejto koncepcie ponúka mnohé vstavané akcie ako animácia vizualizácie, filter objektov a hrán grafu, atď.. Na vizualizáciu ponúka rozloženia ako so staticky vypočítanými pozíciami, tak aj s dynamicky prepočítavanými pozíciami počas vykresľovania. Umožňuje zobrazovať aj zoskupenia objektov grafu. Všetky vizualizácie pôsobia prívetivo.
Prefuse poskytuje nielen vizualizáciu grafov ale aj iné možnosti ako *FishEye, TreeMap* a iné.
Dokumentácia *Prefuse* je v rozpracovanom stave. K dispozícii je čiastočná prvá kapitola manuálu a API dokumentácia v JavaDoc.
Tento nástroj poskytuje plnú a kvalitnú podoru GraphML, čím dáva k dispozícii možnosť presunúť generovanie grafu inej aplikácii.

JavaScript Information Visualization Toolkit (JIT)

JIT je vizualizačný nástroj napísaný v jazyku JavaScript. To prináša výhody aj nevýhody. Poskytuje tak možnosť priameho prepojenia webovej stránky s vizualizáciou a tiež *AJAX* volania pre ďalšie potrebné súčasti. Definovanie grafu nad ktorým má pracovať je vo formáte *JSON*. *JIT* poskytuje aj možnosť upravenia a kontroly interakcie s grafom pomocou *ControllInterface*. K dispozícii je aj bočná lišta so záložkami, ktorej obsah je možné upraviť a prispôsobiť pre potreby vizualizácie.
Medzi hlavné nevýhody patrí fakt, že tento vizualizátor pracuje na úrovni webovej stránky, kde sa neočakáva potreba veľkého výkonu a teda zložitejšie grafy je nemožné vykresliť. V prípad, keď je požiadavka vykresliť grafy, ktoré je možné rozdeliť na jednoduché podgrafy s malým počtom hrán a objektov, je možné na takéto vykresľovanie použiť *AJAX* volania a vykresľovať tak len jeden podgraf.

Návrh na použitie v rámci projektu

Pre potreby vizualizácie grafov s počtom objektov rovnajúcemu sa počtu študentov, ktorí majú zapísaní predmet Tvorba softvérového systému v tíme (cca. 100) a počtom hrán rovnajúcemu sa počtu všetkých vzťahov študentov medzi sebou odporúčam použiť nástroj *Prefuse*. Implementovať daný nástroj ako *Java Applet* a využiť vstavanú podporu *GraphML* pre generovanie grafu.

3.2.3. Filtrovanie zoznamu študentov

Pedagóg má možnosť filtrovať zoznam študentov na základe ich charakteristík, pričom programovo nie je obmedzený ich počet a kombinácie (AND/OR).
Doposiaľ bola vytvorená jedna úloha.

Filtrovanie používateľov podľa charakteristiky

Úlohou je doplniť filtrovanie pomocou techniky AJAX pri zobrazení všetkých používateľov.

Analýza problému

Pri zobrazovaní zoznamu študentov musí byť poskytnutá možnosť nastaviť si zobrazenie určitého okruhu študentov na základe atribútov. Toto filtrovanie má umožniť tvorcovi sociálnej siete zobraziť študentov, ktorí sú si podobní na základe požadovaného atribútu. Aby zobrazovanie nového zoznamu študentov netrvalo dlho a nemusel byť zakaždým načítavaný celý obsah stránky, je potrebné pri filtrovaní použiť AJAX.

Návrh

Filtrovanie ma ponúknuť možnosť výpisu zoznamu študentov na základe konkrétneho atribútu. Časť s filtrovaním obsahuje jeden combobox na výber typu charakteristiky, jeden combobox na výber typu atribútu priradeného zvolenému typu charakteristiky, ktorý sa zobrazí po zvolení typu charakteristiky a textové políčko na zadanie názvu zvoleného atribútu. Po odoslaní formulára sa zobrazí výpis zoznamu študentov, ktorým je pridelená charakteristika zvoleného typu obsahujúca atribút zvoleného typu so zadaným obsahom.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Views/user: *index.html.erb, _user.html.erb*
- V zložke Controllers: *users_controller.rb*

View _user.html.erb obsahuje zdrojový kód na výpis jedného študenta, ktorý sa vloží do view *index.html.erb* po odoslaní požiadavky na výpis zoznamu filtrovaných používateľov. Filtrovanie a výpis zabezpečuje metóda *filter* v súbore *user_controller.rb*. V tom istom súbore sa nachádza aj metóda *show_attributes_combobox*, ktorá zobrazí, resp. aktualizuje combobox s typmi atribútov na základe zvoleného typu charakteristiky.

3.2.4. Vyžiadanie potvrdenia vzťahu druhou stranou

Typ vzťahu má pri sebe informáciu o tom, či vyžaduje potvrdenie druhou stranou. Vzťah má následne atribút, či je alebo nie je potvrdený. V prípade, že A zadefinuje vzťah V k používateľovi B a pritom typ V vyžaduje potvrdenie, tak sa pri najbližšom prihlásení používateľa B tomuto zobrazí výzva na potvrdenie vzťahu. Ak ju príjme, tak sa relácia označí ako potvrdená. V opačnom prípade sa vymaže.
Pre túto *user story* bola zatiaľ vytvorená jedna úloha.

Rozšíriť vzťahy o potvrdenie

Úloha pozostáva z nasledujúcich vecí:

- Doplnenie do typu vzťahu informáciu o tom, či vyžaduje potvrdenie
- Doplniť do vzťahu informáciu o tom či je potvrdený
- Zapracovať potvrdzovaciu logiku a zobrazenie pre vzťahy, ktoré treba potvrdiť.

Analýza problému

Pri niektorých typoch vzťahu môže nastať situácia, že sa ocitneme vo vzťahu k niekomu, s kým si neželáme byť. Takémuto vytvoreniu je možné zabrániť, ak má druhá strana možnosť tento vzťah potvrdiť alebo zamietnuť.

Návrh

Pri vytvorení vzťahu aplikácia overí, či sa jedná o vzťah, ktorý treba potvrdiť. Ak je tomu tak, potom sa druhej strane ponúkne možnosť tento vzťah potvrdiť alebo zamietnuť. V prípade, že bol vzťah potvrdený, tak sa jeho vytvorenie riadi rovnakými pravidlami ako doteraz (napr. pri inverznom vzťahu sa automaticky vytvorí obrátený vzťah). V prípade, že nebol vzťah prijatý druhou stranou, tak sa tento vymaže z databázy.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Controllers: *relationships_controller.rb, type_of_relationships_controller*
- V zložke Views: *users/_show_relationship_detail.html.erb*

Do tabuľky *type_of_relationships* je doplnený atribút *acknowledge,* ktorý určuje, či konkrétny typ vzťahu vyžaduje potvrdenie druhou stranou. Do tabuľky *relationships* je pridaný atribút *status* označujúci, či bol daný vzťah potvrdený druhou stranou, ak je to potrebné.

Testovanie

V zložke *Functional Tests* sa v súbore *relationships_controller_test.rb* nachádzajú testy pre overenie správania sa implementácie v zložke Controllers.

3.2.5. Rozdelenie aplikácie na zóny s rozdielnou úrovňou ochrany prístupu

Niektoré časti aplikácie sú prístupné bez prihlásenia, niektoré sú prístupné len prihláseným používateľom, niektoré iba používateľom v roli pedagóga a niektoré časti len pre administrátora.
Zatiaľ boli pre túto *user story* vytvorené dve úlohy.

Rozdelenie prístupu na metódy kontrolerov podľa role prihláseného používateľa

Úloha zahŕňa dve veci:

- Vytvorenie role administrátora
- Úprava možnosti volania metód podľa role prihláseného používateľa

Analýza problému

Po prihlásení používateľa do systému je potrebné rozlišovať medzi rolami používateľov a poskytnúť im podľa ich rolí aj rôzne možnosti, ktoré prostredie poskytuje. Používatelia sú rozdelení na 3 skupiny a to skupinu administrátorov, učiteľov a študentov. Najväčšie práva ma skupina administrátorov a najmenšie skupina študentov. Každá vyššia skupina zároveň obsahuje aj práva nižších skupín.

Návrh

Na obmedzenie prístupu používateľa na metódy kontroléru sa vytvorí before_filter metóda, ktorá funguje tak, že sa vykoná vždy pred spustením samotnej metódy, na ktorú sa viaže. V tejto before_filter metóde sa vyhodnotí akú rolu má prihlásený používateľ a keď má dostatočné práva, povolí mu vykonanie metódy. V opačnom prípade ho presmeruje naspäť.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Controllers: application.rb, char_attributes_controller.rb, characteristics_controller.rb, *login_controller.rb, relationships_controller.rb,*

type_of_attributes_controller.rb, type_of_chars_controller.rb, type_of_relationships_controller.rb, users_controller.rb

V *application.rb* sa nachádzajú *before_filter* metódy *staff_filter* a *admin_filter*, ktoré overujú rolu prihlasného používateľa a podľa nej povolia alebo zamietnu prístup k metóde. Vo všetkých controller triedach sa nachádza nastavenie, na ktoré metódy sa ma použiť *staff_filter* alebo *admin_filter*.

Testovanie

V zložke *Functional Tests* sa v súboroch nachádzajú Unit testy pre jednotlivé kontrolery.

Zahrnúť všetky časti aplikácie (okrem prihlasovania) do chránenej zóny

Úloha zatiaľ nebola realizovaná.

3.2.6. Definovanie navigácie v aplikácii a úvodná obrazovka

Prihlásený používateľ musí mať možnosť jednoduchej navigácie, bez potreby intenzívneho používania tlačidiel späť a vpred. Už tu musí byť oddelená navigácia pre študenta, pedagóga a administrátora. Po prihlásení používateľ vidí uvodnú obrazovku s informáciami a notifikáciami od posledného prihlásenia.
Pre túto *user story* boli zatiaľ vytvorené tri úlohy.

Kompletná navigačná logika

Úloha vyžadovala zapracovanie navigačnej logiky podľa role používateľa.

Analýza problému

Z každého formulára, ktorý má používateľ zobrazený musí mať možnosť navigácie do iných častí systému.

Návrh

Každý formulár musí obsahovať navigáciu na úvodnú obrazovku používateľa. Okrem toho formulár zobrazí dynamicky tie navigačné odkazy, na ktoré má používateľ právo v závislosti od role v akej je zaradený.

Implementácia

Funkcionalita úlohy implementačne zasahuje do všetkých views v systéme. Každý view obsahuje odkaz na úvodnú obrazovku a explicitne vytvorenú množinu odkazov na views so súvisiacim obsahom. Každý odkaz má priradené role, ktorým sa tento odkaz zobrazí a majú právo ho použiť.

Úvodná obrazovka

Úloha spočívala v navrhnutí a implementovaní úvodnej obrazovky podľa role používateľa a zahrnúť do nej vykreslenie informácii od posledného prihlásenia.

Analýza problému

Po prihlásení používateľa do systému je potrebné poskytnúť mu úvodnú obrazovku, kde bude mať zobrazené možnosti práce, ktoré mu systém ponúka. Používateľ musí mať na úvodnej obrazovke prehľad o zmenách vo vzťahoch v sociálnej sieti, ktorých je súčasťou.

Návrh

Štruktúra úvodnej obrazovky je závislá od role pod akou sa používateľ prihlási do systému. Používateľovi v roli študent sa zobrazia možnosti súvisiace s prezeraním a editovaním svojho profilu. Používateľ v roli pedagóg bude mať navyše možnosť vytvárať skupiny študentov na základe vzťahov v sociálnej sieti a filtrovať študentov. Administrátor bude mať okrem toho zobrazené možnosti pre editáciu sociálnej siete a jej dát v databáze.

Každému používateľovi nezávisle od role sa zobrazia na úvodnej obrazovke zmeny v sociálnej siete, ktoré sa k nemu vzťahujú. Tieto zmeny sa vyhľadajú v tabuľke *notification*.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Models: *login.rb*
- V zložke Views/login: *index.html.erb*
- V zložke Controllers: *login_controller.rb*

Vykreslenie úvodnej obrazovky zabezpečuje metóda index v *login_controller.rb*, táto metóda je implicitne zavolaná z metódy login po úspešnom prihlásení používateľa.

Testovanie

V zložke *Unit Tests* v súboroch *user_test* a *user_controller_test* sa nachádzajú metódy pre otestovanie zobrazenia úvodnej obrazovky pre jednotlivé typy rolí (študent, pedagóg a administrátor).

Vytvorenie notifikačnej tabuľky

Úloha spočíva vo vytvorení tabuľky, ktorá bude obsahovať notifikáciu pre používateľa. Notifikácie sa zobrazia na úvodnej obrazovke a budú obsahovať odkaz na vzťahy.

Analýza problému

Používateľ by mal po prihlásení byť oboznámený o nových udalostiach, ktoré sa ho týkajú. Konkrétne sa jedná o nové vzťahy.

Návrh

Používateľ po prihlásení uvidí úvodnú obrazovku, na ktorej budú vo forme viet s odkazmi zobrazené nové vzťahy. Tieto notifikácie zatiaľ nebudú obsahovať len informácie od posledného prihlásenia ale informácie o všetkých udianých udalostiach od registrácie používateľa. Funkcionalita schovávania a mazania notifikácií môže byť implementovaná v neskorších fázach projektu, rovnako ako aj definovanie notifikačnej správy žiadajúcou stranou.

Implementácia

Funkcionalita úlohy je implementovaná v týchto zdrojových súboroch:

- V zložke Models: *notification.rb*
- V zložke Controllers: *notifications_controller.rb*
- V zložke Views: *login/index.rhtml, users/_notification_detail.html.erb, users/_notifications.html.erb*

Zobrazenie *_notifications.html.erb* sa renderuje v *index.rhtml*, ktorý obsahuje úvodnú obrazovku po prihlásení.

Testovanie

V zložke *Functional Tests* sa v súboroch nachádzajú testy pre jednotlivé implementácie zo zložky Controllers.

3.3. Šprint č. 3

Tretí šprint sme mali v období 17.11.2008 až 1.12.2008. Od tohto šprintu sme sa rozhodli dokumentovať vždy user story namiesto úlohy. Riešilo sa hlavne dopĺňanie funkcionality a dolaďovanie už existujúcej.

3.3.1. Vytvorenie kritérií pre dobrý a zlý tím

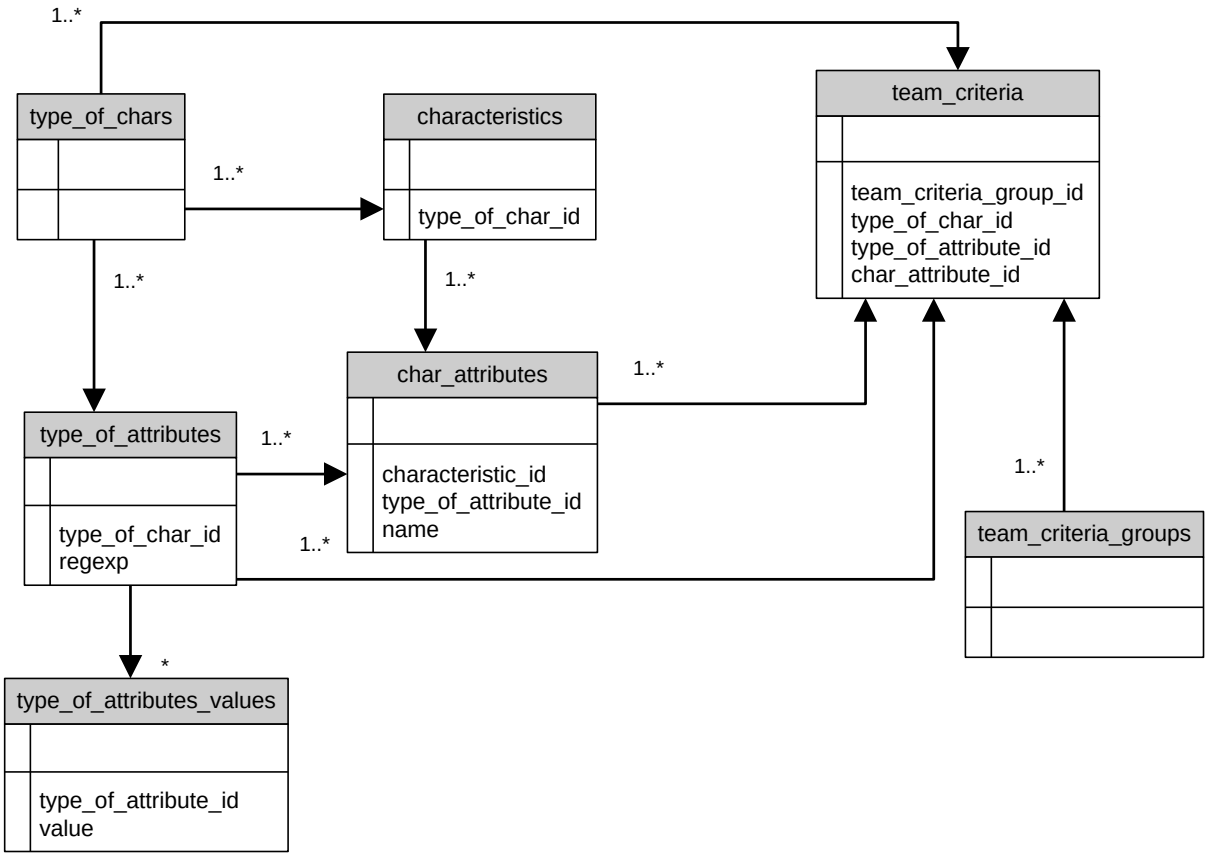
Cieľom tejto user story je vytvoriť v našej aplikácii prostriedok na vytvorenie kritérií, podľa ktorých sa bude posudzovať, či je tím dobrý alebo zlý. Na zápis týchto kritérií budeme používať normálnu konjunktívnu formu (NKF), teda má tvar konjunkcie konečného počtu disjunkcií, ktoré obsahujú konečný počet výrokových premenných, alebo ich negácií.

Analýza problému

Používateľ v roli učiteľa by si mal mať možnosť vybrať kritéria, podľa ktorých mu bude systém pomáhať pri zostavovaní tímu. Tieto kritéria môžu byt pozitívne (napr. aspoň jeden člen musí vedieť pracovať s databázami) alebo negatívne (napr. najviac 3 členovia tímu musia vedieť pracovať s databázami). Na základe týchto kritérií bude systém hodnotiť vytvorené tímy ako spĺňajú/nespĺňajú dané kritéria. Na základe tohto hodnotenia môže učiteľ posúdiť, či dobre rozdelil študentov do tímov, prípadne mu môže systém navrhnúť rozloženie študentov v tímoch.

Návrh

Na uloženie kritérií v databáze som navrhol vytvoriť tabuľku team_criterias, ktorá obsahuje cudzie kľúče na tabuľky type_of_chars , type_of_attributes, char_attributes a taktiež na tabuľku team_criteria_groups, ktorá slúži na vytvorenie skupín jednotlivých kritérií. Medzi pravidlami, patriacej do jednej skupiny sa aplikuje disjunkcia a medzi jednotlivými skupinami sa aplikuje konjunkcia. Časť dátového modelu je zobrazená na obr. 4.



Obr. 4 – Relevantná časť dátového modelu

Implementácia

Táto user story je implementovaný ako stránka v aplikácii, ktorá sa stará o vytváranie kritérií, ich zobrazovanie a editáciu a taktiež aj pripadné mazanie. Ďalej sa stará aj o zadeľovanie kritérií do skupín a editáciu a mazanie skupín. Pri vytváraní a editácii využívam komponentu selectbox, ktorá je naplnená vždy aktuálnymi údajmi z databázy pomocou AJAX technológie. Týmto je zabezpečené to, aby si používateľ mohol vybrať len kritéria, s ktorými má zmysel pracovať, teda sa nachádzajú v databáze.

3.3.2. Filtrovanie používateľov podľa vzťahov

Mám možnosť si vybrať iba tých používateľov, ktorí vystupujú v určitej roli (from, to alebo ľubovoľná z nich) vo vzťahu k ľubovoľnému, resp. jednému konkrétnemu používateľovi.

Analýza problému

Filtrovanie používateľov podľa vzťahov, by bolo malou alternatívou k zobrazeniu vizualizovaných vzťahov. Pomocou vzťahu je tak možné bližšie špecifikovať používateľov a tak značne urýchliť vyhľadávanie používateľa s danými vlastnosťami.

Návrh

Doplniť pri zobrazení všetkých používateľov filter na vzťahy medzi používateľmi. Je možné vychádzať z už existujúcej funkcionality pre charakteristiku. Je však ale nutné zapracovať AND a OR logiku. V prípade, že v danom comboboxe nie je prázdna hodnota, tak sa tento parameter tvári ako AND. Z dôvodu integrácie do existujúceho filtra charakteristík, si táto implementácia vyžiada jeho revíziu implementácie.

Implementácia

Funkcionalita je implementovaná ako 2 comboboxy, z ktorých jeden zobrazuje hodnoty pre typ vzťahu a druhý jeho smer od (angl. from) alebo do (angl. to). Hodnoty v comboboxoch stráži AJAX komponenta. Hodnoty v comboboxoch sú teraz zoradené podľa abecedy (osobitne malé a osobitne veľké začiatočné písmená) pomocou zoradenia množiny a je zakázané ich opakovanie (zabezpečené pomocou objektu Set) v prípade, že ich hodnota je rovnaká pre viacero záznamov v tabuľke. Z toho dôvodu sa charakteristika vyhľadáva nad tabuľkou podľa názvu a nie podľa jej identifikátora.

3.3.3. Filtrovanie používateľov podľa atribútov je používateľsky prívetivé a pohodlné

Pri vytváraní filtra pre používateľov na základe atribútu systém po výbere typu atribútu zobrazí zoznam možných hodnôt tohto atribútu. Tým bude nemožné zadať pri filtrovaní nezmyselný parameter a zároveň to používateľovi pomôže pri zadávaní parametrov filtra. Cieľom tejto user story je doplniť existujúci filter o výber hodnoty atribútu zo zoznamu.

Analýza problému

Pri zobrazovaní zoznamu používateľov, bude filtrovanie často používanou funkciou vytváranej sociálnej siete. Pomocou filtra je jednoduché zobraziť študentov s rovnakou vlastnosťou a na to, aby bolo filtrovanie používateľsky čo najprívetivejšie, je dobré, aby si mohol používateľ vybrať zo zoznamu možných

hodnôt. Pri filtrovaní na základe konkrétnej hodnoty atribútu nejakej charakteristiky je potrebné vybrať si z troch comboboxov, pričom obsah druhého comboboxu závisí od prvého a obsah tretieho závisí od prvého. Na to, aby pri výbere týchto parametrov filtra nebolo nutné obnovovať stránku, bude potrebné využiť technológiu AJAX. Súčasný filter obsahuje textové pole na zadanie presnej hodnoty atribútu. Úpravou filtra zabránime zadávaniu nezmyselných hodnôt a uľahčíme používateľovi proces filtrovania výberom hodnoty atribútu.

Návrh a implementácia

Pri filtrovaní je potrebné zvoliť najprv typ charakteristiky. Následne je potrebné zvoliť typ atribútu a potom aj hodnotu tohto atribútu. Prvý combobox obsahuje všetky typy atribútov. Druhý sa objaví až po výbere prvého a obsahuje typy atribútov zvoleného typu charakteristiky. Tretí combobox sa objaví súčasne s druhým a obsahuje iba hodnoty atribútov zvoleného typu atribútu (v tomto prípade prvého typu atribútu v zozname). Následne je možné na základe týchto hodnôt zobraziť zoznam študentov, ktorí spĺňajú zvolené kritéria. Tento výpis prebieha taktiež bez obnovenia stránky, teda prostredníctvom AJAXu.

3.3.4. Layout a štýl stránok

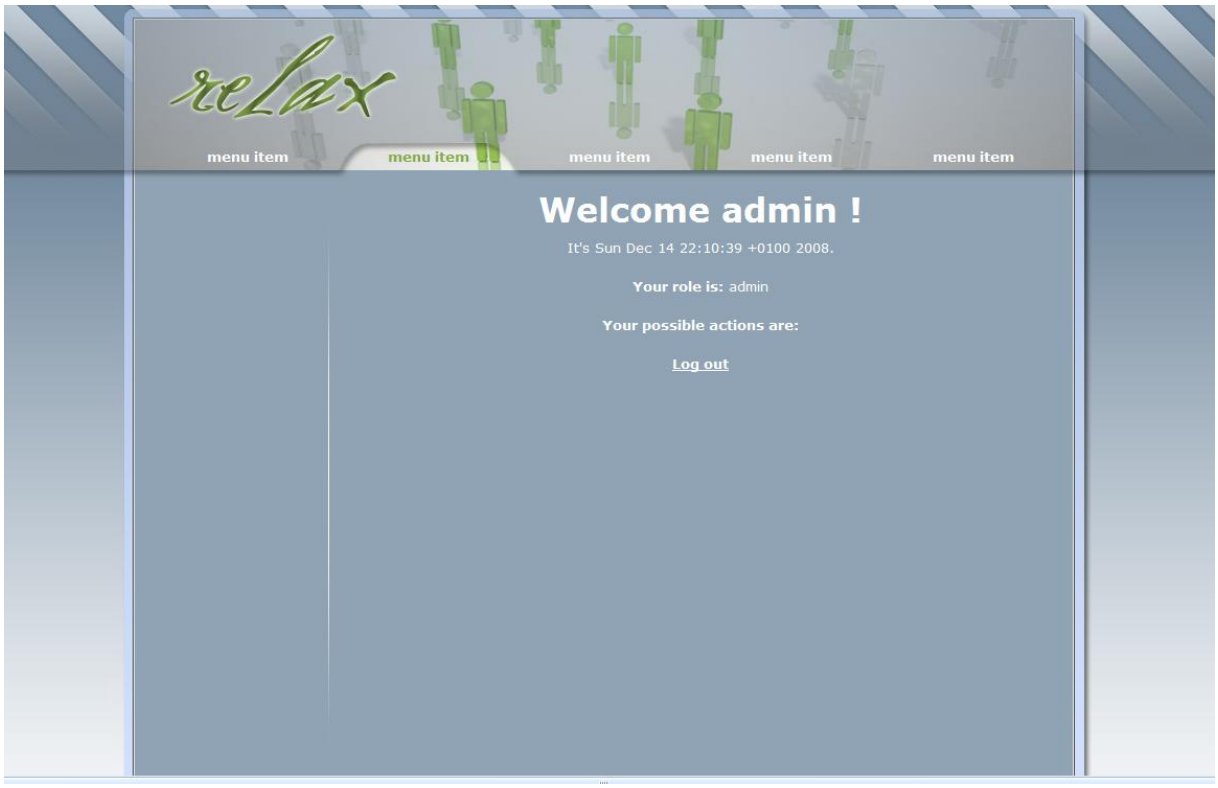
Cieľom tejto user story je vytvoriť rozloženie stránky v podobe, v akej bude dostupné používateľom systému, teda vytvoriť dizajn stránky a pripraviť priestor pre rozloženie jednotlivých prvkov.

Analýza problému

Každý používateľ uprednostňuje používateľsky prívetivé a prehľadné stránky, v ktorých sa dá rýchlo zorientovať. Do systému sa bude prihlasovať veľký počet študentov, a preto je dôležité aby bola stránka intuitívna a nebol problém nájsť to, čo používateľ práve potrebuje. Je potrebné vyčleniť priestor na hlavné menu, ktoré bude nemenné a vedľajšie menu, ktoré bude obsahovať kontextové odkazy na podstránky v závislosti na zobrazenej stránke.

Návrh a implementácia

Na obr. 5 je zobrazený implementovaný layout. V hornej časti sa nachádza logo a hlavné menu. Pod ním je naľavo vedľajšie menu, ktoré sa bude meniť v závislosti od obsahu stránky a aj samotný obsah stránky.



Obr. 5 – Layout aplikácie

3.3.5. Vizualizačný nástroj grafu vzťahov medzi používateľmi

Používateľ , ktorý má prístup do systému a môže vytvárať tímy, bude mať možnosť zobraziť vzťahy študentov medzi nimi. Dôvodom vizualizácie týchto vzťahov je lepšia predstaviteľnosť prepojenia jednotlivých študentov.

Analýza problému

Používateľ by mal mať možnosť vizualizovať graf, ktorý v sebe nesie informácie o používateľoch a ich vzťahoch. Vizualizácia bude prebiehať pomocou appletu, ktorý bude implementovaný pomocou vizualizačného nástroja *Prefuse*. Používateľ bude mať tiež možnosť zobraziť informácie o konkrétnom používateľovi. Taktiež bude umožňovať manipuláciu s daným grafom a rôzne filtrácie nad ním.

Návrh

Applet dostane pomocou vstupných parametrov. Tieto parametre budú vkladané cez HTML kód pri volaní appletu. V tejto fáze projektu boli navrhnuté dva povinné vstupné parametre:

1. GraphML

- URL adresa grafu ktorý má byť vizualizovaný

2. URL
    - prefix URL adresy, ktorá odkazuje na informácie o používateľovi. K tejto adrese bude pridané identifikačné číslo používateľa v rámci systému

V grafe budú samotní používatelia reprezentovaní ako vrcholy, a vzťahy medzi nimi ako hrany. GraphML formát bol vybraný pre svoju schopnosť niesť v sebe aj informácie o konkrétnych hranách a uzloch. Takto bude možné prenesenie informácií o používateľoch a ich vzťahoch do grafu. Tým získame informácie potrebné pre neskoršiu filtráciu v applete.

Návrh ovládania

Na ovládanie vizualizovaného grafu budú využité nasledovné možnosti :

1. presunutie vrcholu grafu
2. vycentrovanie vrcholu grafu
3. škálovanie grafu
4. pohyb po grafe
5. zobrazenie informácií o konkrétnom používateľovi

Návrh vstupného GraphML

Applet vyžaduje pre svoju funkčnosť, aby vstupný graf obsahoval informácie o používateľovi. Povinné sú dva údaje, ktoré sú definované v dátovej schéme GraphML, a to :

1. name
    - informácia o mene používateľa. Bude využitý pri vyhľadávaní používateľa na základe mena a taktiež na jeho identifikáciu v grafe.
2. Id
    - Identifikačné číslo používateľa v systéme. Pomocou tohto čísla sa bude odkazovať na informácie o používateľovi

Nasledujúce GraphML znázorňuje príklad a definíciu využívaného GraphML formátu.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
<graph edgedefault="directed">

<!-- data schema -->
     <key id="name"
          for="node"
          attr.name="name"
          attr.type="string"/>
     <key id="id"
          for="node"
          attr.name="id"
```

```
            attr.type="string"/>

<!-- nodes -->
<node id="1">
 <data key="name">Jeff Xantipa</data>
 <data key="id">25</data>
</node>
...
<!-- edges -->
<edge source="1" target="2"></edge>
...

</graph>
</graphml>
```

Implementácia

Vizualizátor je naimplementovaný ako applet, ktorý využíva na svoju funkcionalitu nástroj Prefuse. Tento applet získava pomocou HTML volania navrhnuté vstupné parametre. Hlavná trieda appletu je

SocialRelationshipVisualizator.class

a nachádza sa v balíku

sk.awesomelegends.valasik.socialrealitonshipvisualizator.

Celý applet sa volá pomocou HTML volania so vstupnými parametrami takto :

```
<APPLET codebase="."
      code="sk/awesomelegends/valasik/socialrealitonshipvisualizator/
SocialRelationshipVisualizator.class"
      archive=http://relax.fiit.stuba.sk/relax/visualizator.jar
      width="600"
      height="600" >
      <PARAM NAME="graphml"
            VALUE=" http://relax.fiit.stuba.sk/relax/socialnet.xml">
      <PARAM NAME="url"
            VALUE="http://relax.fiit.stuba.sk/relax/users/">
</APPLET>
```

Implementácia ovládania

Jednotlivé ovládacie prvky boli implementované a do systému pridávané pomocou metódy `addControlListener` triedy `Display`. Triedy použité vo volaní tejto metódy rozširujú triedu `ControlAdapter`. Tieto triedy implementujú v sebe už samotné konkrétne ovládanie grafu.

Ovládanie prebieha pomocou myši a je nastavené nasledovným spôsobom :

1. Ľavé tlačidlo
    - Kliknutie na vrchol grafu – vycentrovanie konkrétneho vrcholu na stred grafu
    - Držanie a pohyb – posúvanie grafu a premiestňovanie sa v ňom
2. Pravé tlačidlo

- Kliknutie na vrchol grafu – otvorenie *Pop-up* okna s informáciami o používateľovi
- Držanie a pohyb – škálovanie grafu – zmena veľkosti

Adresa ktorá je vložená do *pop-up* okna je zložená zo vstupného parametra *url* a identifikačného čísla *id* vrchola grafu.

Implementácia vyhľadávania

Vyhľadávanie je implementované pomocou tried `SearchQueryBinding` a `PrefixSearchTupleSet` implementovanej v *Prefuse*. V triede `SearchQueryBinding` sa nastaví vlastnosť objektov, v ktorej sa bude vyhľadávať, v našom prípade meno používateľa. V triede `PrefixSearchTupleSet` sa nastaví charakter metódy, ktorá sleduje zmenu v nájdených výsledkoch.
Zmena farby a podfarbenia vyplýva z nastavení, ktoré sú definované v triedach rozširujúcich triedu `ColorAction`. Tieto triedy sú následne pridané do zoznamu akcií nástroja *Prefuse* pomocou metódy `putAction` v triede `Visualization`. Jednotlivé akcie sú volané a na základe týchto volaní nástroj vykresľuje výsledný graf.

Umiestnenie v systéme

Applet je v systéme umiestnený v priečinku *Public* aby bol prístupný z ľubovoľného miesta. Samotný nástroj nepotrebuje kontrolu prístupu, nakoľko samotný nezobrazuje a neumožňuje prístup k informáciam, ktoré by boli nejakým spôsobom kritické. Informácie ktoré zobrazuje sú dodávané pomocou GraphML súboru, ktorý dostáva ako vstupný parameter.

3.3.6. Výber zo zoznamu hodnôt pri vytváraní atribútu charakteristiky

Pri pridávaní atribútov charakteristiky zobraziť používateľovi zoznam hodnôt, z ktorého môže vyberať.

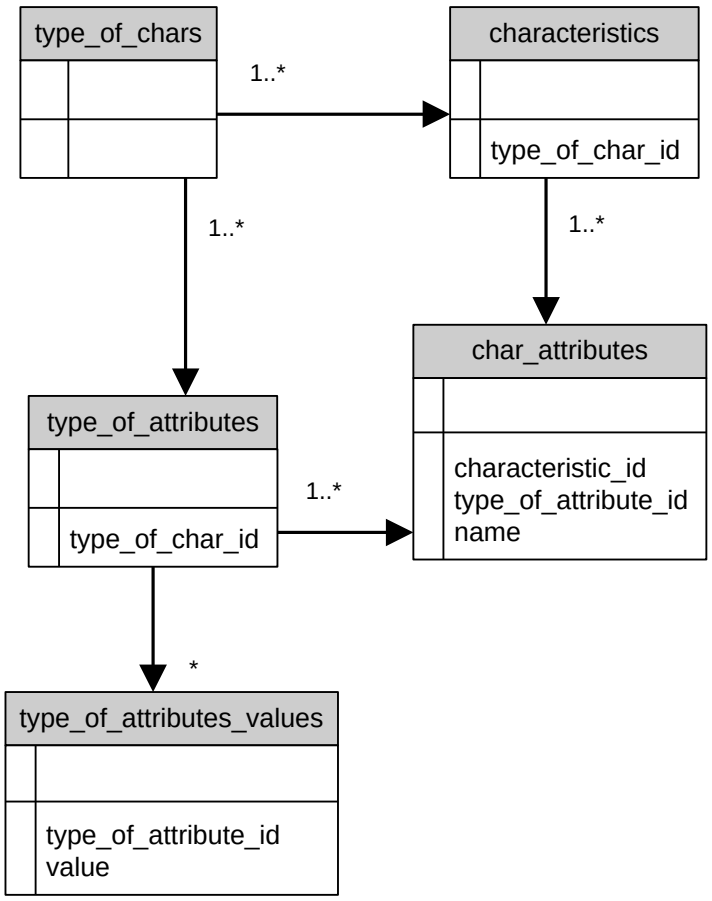
Analýza problému

Atribúty charakteristiky ktoré možno pridať na charakteristiku závisia od viacerých faktov:

- Akého typu charakteristika je, podľa jej typu závisí množina typov atribútov ktoré možno pridať.
- Či tabuľka *type_of_attributes_values* obsahuje predurčené hodnoty pre typ atribútu.

S týmito fakami súvisí aj databázový model, ktorý je zobrazený na obr. 6 (model zobrazuje iba fyzickú úroveň abstrakcie ktorá je potrebná pre túto úlohu).

Používateľovi sa zobrazí možnosť výberu hodnôt v prípade, ak typ atribútu ktorý chce pridať obsahuje predurčené hodnoty, ktoré môže nadobúdať.



Obr. 6 - Časť modelu sociálnej siete pre chrakteristiku používateľa

Návrh a implementácia

Zobrazenie množiny hodnôt s ktorých používateľ môže vyberať zabezpečí komponent *selectbox*, pričom používateľ môže vybrať práve jednu z hodnôt. Pri potvrdení formulára systém overí, či bola vybraná jedna z hodnôt, t.j. hodnota je rôzna od null.

Naplnenie *selectbox* komponenty sa získa nasledovným spôsobom:

1. Systém ponúkne používateľovi *selectbox* s typmi atribútov v závislosti od typu charakteristiky na ktorý chce nový atribút pridať.
2. Po výbere typu charakteristiky systém zistí, či pre tento typ atribútu tabuľka *type_of_attributes_values* obsahuje aspoň jednu hodnotu (znamená to, že atribút má predurčenú množinu hodnôt).

3. Pokiaľ áno, tak sa s tejto tabuľky načíta množina hodnôt pre tento typ atribútu a naplní sa nimi *selectbox* a ten sa následne vykreslí na GUI.
4. V opačnom prípade systém zobrazí *editbox* pre vpísanie hodnoty (tento prípad pridávania atribútu rieši user story *Kontrola vstupu pri vytváraní atribútov charakteristiky*).

3.3.7. Kontrola vstupu pri vytváraní atribútov charakteristiky

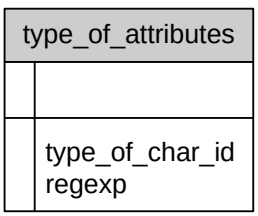
Pri pridávaní atribútov charakteristike je potrebné zabezpečiť platnosť vstupných dát zadaných používateľom.

Analýza problému

Pri zadávaní hodnôt atribútov je potrené aby systém vykonal striktnú kontrolu vstupných dát v závislosti od typu atribútu. Tým má znemožniť používateľovi zadať nezmyselné dáta ako napríklad pri charakteristike typu *programuje v*, ktorá obsahuje atribút typu *počet rokov*, aby v tomto prípade používateľ zadal reťazec „od strednej školy“. Z kontextu je pritom nutné, aby to bolo celé číslo väčšie ako 0. Tento problém sa netýka prípadu, kedy používateľ vyberá hodnoty z množiny predurčených hodnôt (validáciu vstupu v tomto prípade rieši user story *Výber zo zoznamu hodnôt pri vytváraní atribútu charakteristiky*)

Návrh a implementácia

V prípade že typ atribútu neobsahuje množinu predurčených hodnôt, systém zobrazí na GUI *editbox*, kde používateľ zadá hodnotu atribútu ktorý pridáva. Po potvrdení formulára systém overí, že hodnota, ktorú vložil je rôzna od *null* a súčasne reťazec nie je prázdny. Na to aby sa kontroloval typ a rozsah hodnoty ktorá sa uloží do databázy, je vhodné využiť techniku kontroly pomocou regulárnych výrazov, ktorá umožňuje zadať presný formát reťazca. Regulárny výraz sa viaže na jednotlivý typ atribútu, preto vznikol nový stĺpec v databáze v tabuľke type_of_attributes s názvom *reg_exp* (obr. 7). Pred uložením hodnoty atribútu do databázy systém skontroluje či hodnota spĺňa vzor predurčený regulárnym výraz. Pokiaľ nie tak systém hodnotu do databázy nevloží a zobrazí upozornenie pre používateľa na GUI.



Obr. 7 - Tabuľka type_of_attributes s pridaným stĺpcom regexp pre kontrolu hodnôt

3.4. Šprint č. 4

Posledný šprint v zimnom semestri trval od 1.12.2008 do 15.12.2008. Cieľom šprintu bolo testovanie a dolaďovanie už implementovanej funkčnosti. Preto obsahoval len jednu user story, ktorá sa preniesla zo šprintu č. 2.

3.4.1. Integrácia na YonBan

Napojiť sa na školský informačný systém a načítať dáta, ktoré sú relevantné pre tento projekt.

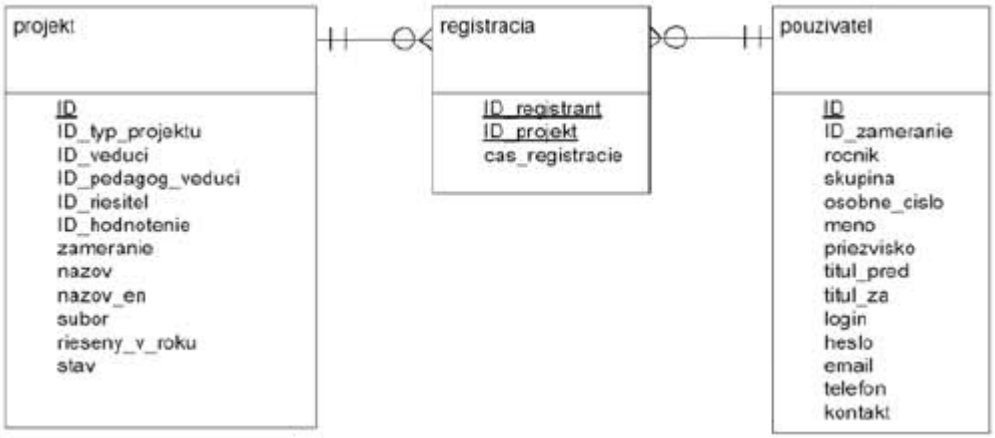
Analýza problému

Informačný systém YonBan nie je vo svojej podstate stavaný na udržiavanie informácií ohľadom vzťahov medzi používateľmi. Preto obsahuje pomerne málo informácií, ktoré by boli zaujímavé pre náš projekt. Jednou z týchto informácií je, akí študenti mali v rovnaký rok rovnakého projektového vedúceho ako daný študent. Existuje totiž predpoklad, že v jednom roku má jeden vedúci podobné, alebo rovnaké témy a preto aj jeho zverenci musia vykazovať isté spoločné známky.

Návrh a implementácia

Systém by sa mal napojiť na databázu YonBan a zistiť, akí študenti mali v rovnaký rok rovnakého vedúceho ako študent, ktorý sa práve prihlasuje. Danému študentovi sa následne pridajú vzťahy s týmito študentmi, samozrejme za predpokladu, že už neexistujú. Typ vzťahu vyjadruje možnosť spoločných záujmov, čo môže byť využité pri vytváraní tímu.

Systém YonBan používa databázu PostgreSQL. Napájanie na ňu a dotazovanie je implementované hneď po prihlásení používateľa v *login_controller.rb*. Toto riešenie sa ukázalo ako nešťastné a bude ho potrebné v budúcnosti zmeniť, nakoľko pri nedostupnosti databázy sa na odpoveď o nedostupnosti čaká veľmi dlho. Samotný výber informácií z databázy je riešený pomocou jedného SQL SELECT príkazu. Na obr. 8 je zobrazená časť dátového modelu systému YonBan. Podľa osobného čísla študenta získame tabuľku používateľ. Podľa *ID* v tabuľke používateľ a *ID_riesitel* sa ďalej dostaneme ku všetkým projektom, ktoré používateľ riešil. Z nich vieme zistiť, aké všetky projekty mal v daný rok daný pedagogický vedúci (podľa *ID_veduci* a *rieseny_v_roku*). Z týchto projektov sa následne cez *ID_riesitel* vieme dostať ku študentom. Podľa osobného čísla zistíme, či už sa študent nachádza v našej databáze a pokiaľ áno, pridá sa vzťah s aktuálnym používateľom.



Obr. 8 – Časť dátového modelu systému YonBan

4. Prototyp

Nakoľko sme si zvolili techniku agilného vývoja, mali sme prvý prototyp pomerne skoro, už mesiac po začatí semestra. S každou ďalšou iteráciou sme ho potom vylepšovali a dopĺňali. Na konci zimného semestra máme implementované väčšinu z toho, čo sme chceli.

Prototyp dokáže prihlásiť používateľa pomocou mena a hesla zo školského informačného systému, umožňuje prezeranie a editovanie profilu, pridávanie vzťahov (kamarátov) a pridávanie charakteristík. Niektoré vzťahy sa načítavajú zo systému YonBan. Pre pedagóga je k dispozícií možnosť filtrovania používateľov podľa zadaných kritérií, alebo vzťahov. Samotná vizualizácia je zimplementovaná, jej integráciu sme už ale nestihli.

5. Šprinty (letný semester)

V letnom semestri sme pokračovali v agilnom vývoji projektu. Spolu sme mali len 3 šprinty, nakoľko sme posledný natiahli až do konca semestra. Zmena oproti letnému semestru spočívala v dohode tímu o dokumentovaní *user stories* a nie pridelených úlohách (*tasks*).

5.1. Šprint č. 1

Prvý šprint v letnom semestri trval od 6.3.2009 do 19.3.2009. Sústredili sme sa v ňom ďalšie rozvíjanie funkcionality systému.

5.1.1. Vyhľadávanie zoznamu používateľov

Popis

Pri zadávaní filtra mi systém ponúka zoznam používateľov a umožňuje automatické doplňovanie postupne ako zadávam jednotlivé písmená mena používateľa.

Analýza problému

Používateľ nemusí poznať presné znenie mena vyhľadávanej osoby, čo v prípade zadeľovania veľkého počtu ľudí to tímov ani nie je možné. Pre efektívnu prácu v systéme je vhodné, aby používateľ nemusel tráviť zbytočne veľa času vyhľadávaním informácií o používateľoch. Preto je v našom systéme implementovaná podpora automatického doplňovania mena a priezviska používateľa okamžite počas písania mena. Systém ponúkne používateľovi viacero možných existujúcich mien, z ktorých sa následne vyberie hľadaná položka. Vyhľadávanie pre automatické doplňovanie pracuje s krstným menom a s priezviskom.

Návrh a implementácia

Keďže automatické doplňovanie si vyžaduje priamu podporu klientského jazyka, ktorý by ale vedel odosielať požiadavky na backend bol použitý jazyk JavaScript formou technológie Ajax. JavaScript sa uplatnil pri získavaní kódu stlačenej klávesy a pri vykresľovaní políčok so získanými menami hľadaných osôb. Prostredníctvom Ajaxu je posielaná požiadavka, na metódu v *user* kontroleri, ktorá následne vracia zoznam osôb, ktorých krstné meno, resp. priezvisko sa sčasti zhoduje so zadávaným menom v textovom poli umožňujúcom automatické doplňovanie. Metóda *searchUsersByName* vracajúca zoznam osôb pracuje nad filtrom, a teda zoznam vrátených osôb je prienikom osôb vrátených filtrov. Týmto spôsobom je docielené,

že osoby, ktoré automatické doplňovanie prehľadáva sú iba vyfiltrované osoby.
Aby bol zoznam týchto osôb pre používateľa ľahko spracovateľný, teda aby si mohol intuitívne zvoliť a zobraziť detaily o osobe, ktorú dal vyhľadávať, bolo potrebné implementovať metódy na prehľadné vykresľovanie zoznamu osôb priamo pod textovým políčkom. Na tento účel bol vytvorený JavaScript súbor *autocomplete.js*. Po vykreslení zoznamu je používateľovi umožnené rýchlo sa pohybovať po položkách zoznamu použitím kláves "hore", "dole", zvoliť požadovanú osobu klávesou "enter", zrušiť automatické doplnenie klávesou "ESC". Rovnako je možné zvoliť si vyhľadanú osobu kliknutím myši na odkaz v zozname.
Na obr. 9 je zobrazený doplnený zoznam používateľov. Zadaným vstupom bolo písmeno "m". Vyhľadané osoby obsahujú tento vstup buď v krstnom mene, alebo v priezvisku. Po stlačení klávesy "dole", bola zvýraznená druha osoba v zozname a jej meno bolo doplnené do textového políčka. Po stlačení klávesy "enter" je osoba vypísaná na stránke a je možné nahliadnuť do daného profilu, prípadne ho editovať.

Obr. 9 – Ukážka automatického dopĺňania

5.1.2. Vytvorenie kritérií pre dobrý a zlý tím

Popis

Cieľom tejto *user story* je vytvoriť používateľské rozhranie pre vytváranie kritérií, podľa ktorých sa zadeľujú študenti do tímov.

Analýza problému

Kritéria môžu byt dvojakého typu, teda kritéria nad charakteristikami a kritéria nad vzťahmi. Používateľ musí definovať, či kritéria majú byt chápané ako pozitívne alebo ako negatívne a tiež rozhodne, koľko študentov má minimálne, alebo maximálne spĺňať kritérium. Pri charakteristikách používateľ vyberá typ charakteristiky a typ atribútu a jeho hodnotu. Táto hodnota nemusí byt presná, používateľ môže vybrať hodnotu väčšiu, alebo menšiu než zadaná hodnota. Pri kritériách vzťahov používateľ vyberá len typ vzťahu. Ďalej používateľ môže vytvoriť rôzne skupiny kritérií pričom platí, že medzi dvoma skupinami sa použije logická spojka alebo a medzi kritériami v rámci skupiny sa použije logická spojka a. Týmto spôsobom dokáže používateľ vytvoriť disjunkciu konjunkcii kritérií.

Implementácia

Implementácia sa nachádza v *TeamCriteria* kontroleri. Hlavnú časť implementácie tvoria Ajaxové volania, ktoré vytvárajú vyberateľné ponuky s predvyplnenými dátami. Kritéria sú uložené v databáze v tabuľke *team_criterias*, ktorá je naviazaná na tabuľky *type_of_chars, type_of_attributes, char_attributes* a *type_or_relationship*. Ďalej táto tabuľka obsahuje stĺpce pre počet ľudí, ktorý ma spĺňať toto kritérium, či je kritérium pozitívne alebo negatívne, ktorý používateľ ho vytvoril a do ktorej skupiny kritérií patrí. V prípade kritérií pre charakteristiky sa používajú previazania opäť na tabuľky *type_of_chars, type_of_attributes* a *char_attributes*. Naopak v prípade kritérií pre vzťahy sa používa iba naviazanie na tabuľku *type_of_relationship*. Ostatné stĺpce tabuľky sa používajú pre obidva typy kritérií. Na zobrazenie kritérií slúži stránka na kontroleri TeamCriteria. Táto stránka je dostupná cez url so sufixom „/team_criterias“ a okrem zobrazovania existujúcich kritérií umožňuje aj ich mazanie a pridávanie nových kritérií. Toto pridávanie je možné v dvoch podobách, buď pridaním nového kritéria do existujúcej skupiny alebo vytvorením nového kritéria v novej skupine. Pri druhom spôsobe musí používateľ zadať aj meno skupiny.

5.1.3. Editovanie typov charakteristík a typov atribútov

Popis

Administrátor systému má mať možnosť dynamicky meniť číselníkové dáta systému. Medzi tieto dáta patria aj typy charakteristík a typy atribútov.

Analýza problému

Existuje veľké množstvo potenciálnych oblastí využitia systému. Požiadavky na charakteristiky v profile používateľa môžu byť rôzne podľa potreby. Napríklad charakteristika *absolvoval predmet,* ktorá je dôležitá pri tvorbe tímov v predmete Tímový projekt, by zrejme nemala žiadny význam pri tvorbe pracovných tímov v súkromnej firme. Podobne je to aj s typmi atribútov pri jednotlivých charakteristikách, kde je tiež potrebné umožniť zadať rôzne položky, ktoré sa k charakteristike budú v systéme evidovať. Možnosť dynamickej konfigurácie číselníkových hodnôt v tabuľke typov charakteristík a typov atribútov je preto dôležitou funkcionalitou systému. Umožní sa tým tvorba flexibilnej bázy znalostí používateľov systému.

Návrh a implementácia

Možnosť editácie je dostupná z hlavného formulára systému, pokiaľ je používateľ v roli administrátora. Tomuto používateľovi je umožnené typy charakteristík pridávať, meniť a mazať. Tieto možnosti má aj v prípade editovania typov atribútov.

5.1.4. Ručné zadeľovanie študentov do tímov v grafickom móde

Popis

Výberom jednotlivých uzlov vo vizualizátore môžem študentov zadeľovať do tímov. Systém mi pritom farebne notifikuje splnenie kritérií pre dobrý/zlý tím, ak sú nejaké definované.

Analýza problému

Používanie vizualizačného nástroja uľahčuje prácu so študentmi, nakoľko umožňuje zobrazenie ich vzťahov medzi sebou. Pri takomto nástroji je vhodné, aby práca so študentmi zahŕňala aj funkcionality pridávania používateľov do tímov.
Pre takúto požiadavku je ale potrebné špecifikovať postup, ako je možné študenta alebo študentov zaradiť do tímu. Ten bol špecifikovaný nasledovne:

1. Výber študentov vo vizualizačnom applete
2. Výber tímu pre študentov
3. Pridanie študentov do tímu

Vyber tímu pre zadeľovanie ľudí do tímu

Z databázy sa pomocou Ruby on Rails získa zoznam aktuálnych tímov. Tie sa nakoniec zobrazia do select boxu, pričom select box ma nasledovnú špecifikáciu:

```
<select name="team[select]" id="team_select">
     <option value="#id_timu">#nazov_timu</option>
</select>
```

Tento select box sa nachádza presne nad appletom a bude následné použitý JavaScript-om, ktorý je volaný prostredníctvom appletu. K tomu aby bolo možné volať bezproblémovo JavaScript z appletu, je potrebné upraviť HTML inicializáciu tohto appletu takto :

```
<APPLET MAYSCRIPT codebase="."
code="sk/awesomelegends/valasik/socialrealitonshipvisualizator/Social
RelationshipVisualizator.class"
archive="http://localhost:3000/applet/visualizator.jar" width="600"
```

```
height="600" >
    <PARAM NAME="graphml"
VALUE="http://localhost:3000/services/gen_graph.xml">
    <PARAM NAME="url" VALUE="http://localhost:3000/relax/users/">
</APPLET>
```

Podstatným rozdielom je pridanie atribútu `MAYSCRIPT`. Ten zabezpečuje, že spôsob, ktorým je JavaScript volaný z appletu je funkčný vo väčšine internetových prehliadačov.

Vytvorenie Ruby funkcionality na tvorbu tímov

Implementácia funkcionality na pridávanie používateľov do tímu pomocou Ajax volania. Vytvorila sa špecifická URL adresa, ktorá túto funkcionalitu obsahuje a zabezpečuje. Pomocou nastavení modelu v Ruby on Rails vlastnosťou:

```
validates_uniqueness_of  :scope => [:team_id, :user_id]
```

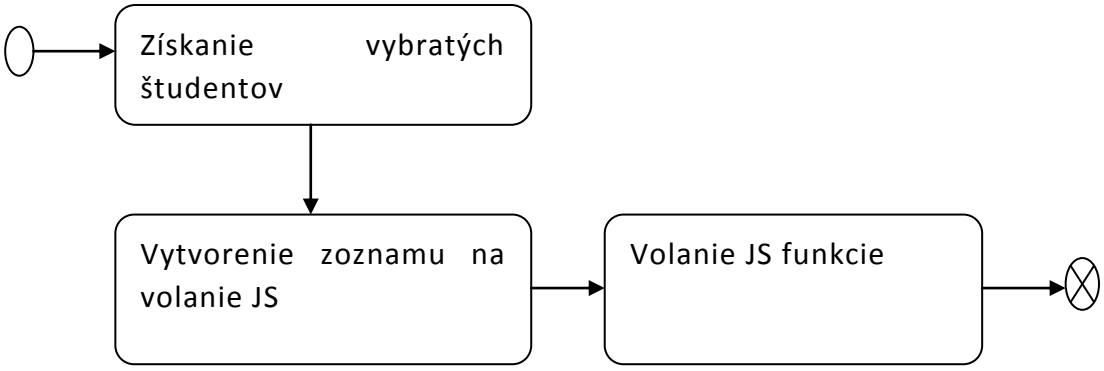
, sa zabezpečí, že kombinácia tím-polužívateľ bude iba jedna, a teda používateľ bude v tíme zaregistrovaný iba raz.
Návratová hodnota pre toto volanie je zoznam používateľov, oddelených čiarkou.

Vytvorenie JavaScript funkcionality

Táto úloha spočíva v integrovaní predchádzajúcich dvoch úloh do JavaScript funkcie. Tá zabezpečí, aby vybraní používatelia, boli zadelení do vybraného tímu. Vybraný tím sa identifikuje na základe výberu zo select boxu a používatelia sú vstupom do funkcie.
Funkcia pre komunikáciu so serverom využíva Ajax volanie za pomoci frameworku Prototype. Tento je využívaný v rámci celého systému. Využíva sa funkcia Ajax.Request, kde je ale potrebné k vstupným parametrom pridať aj *authenticity_token*, kvôli kontrole unikátnosti volania Ruby on Rails.
Vďaka návratovej hodnote, ktorú volaná Ruby funkcia vracia, je možné zobraziť zostavu tímu do ktorého sa aktuálne pridávali používatelia. To sa zobrazuje do DIV elementu nad appletom.

Volanie JavaScript funkcionality z appletu

Pre potreby tohto *user story* bolo potrebné pridať do appletu funkcionalitu na pridávanie používateľov do tímu. Pridať je možné viacero používateľov naraz. Výber tímu prebieha pomocou už opísaných zložiek, select boxu a JS funkcie. JS funkcia je volaná prostredníctvom appletu.
Túto akciu treba ale taktiež aj vyvolať. Na to slúži pridané tlačidlo „Pridať do tímu“, ktoré takýmto spôsobom zavolá JavaScript funkciu. Celý proces po kliknutí na tlačidlo znázorňuje obr. 10.



Obr. 10 – Proces vyvolaný stlačením tlačidla „Pridať do tímu“

5.1.5. Pri vytváraní vzťahu mi systém ponúka zoznam používateľov a dáva automaticky dopĺňa vhodných kandidátov

Popis

Pri vytváraní vzťahov doplniť vyhľadanie používateľov (cez Ajax) počas písania mena a tento zoznam aktualizovať po každej zmene zadávaného vstupu.

Analýza problému

V prípade, že systém obsahuje veľké množstvo používateľov je veľmi nepraktické umiestniť ich výber do comboboxu. Hoci combobox poskytuje hľadanie pomocou zadávania začiatočných písmen, aj tak je takéto vyhľadávanie nepraktické. Preto treba navrhnúť iný spôsob zadávania resp. vyhľadávania konkrétneho používateľa do vzťahu.

Návrh a implementácia

Jeden zo spôsobov ako vyriešiť automatické dopĺňanie je použiť už existujúci JavaScript kód, ktorý sme použili na filtrovanie používateľov pri ich zozname na *show* view. Keďže úlohou tímového projektu je osvojiť si viacero rôznych prístupov, tak sme použili existujúci plugin do Ruby on Rails prostredia s názvom *autocomplete*. Tento vyhľadáva nad konkrétnym stĺpcom v databáze podľa zadávaných písmen a možné výsledky zobrazuje do rolovacieho boxu. Celá implementácia tak nezaberá viac ako 5 riadkov s tým, že podporuje aj definíciu vlastných CSS štýlov, ktorú sme samozrejme využili.

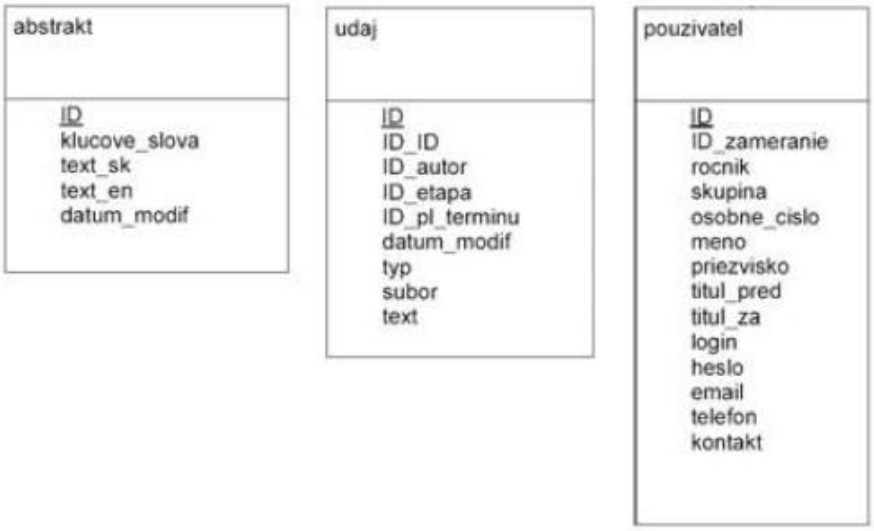
5.1.6. Integrácia na YonBan #2

Popis

V zimnom semestri bol údaje zo systému YonBan ťahané prostredníctvom rovnakých projektových vedúcich v danom roku. Druhú metódy sme zvolili filtráciu podľa kľúčových slov.

Analýza

Na obr. 11 je možné vidieť 3 entity zo systému YonBan, ktorých sa týka náš problém kľúčových slov. V originálnej schéme medzi sebou nemali žiadne vzťahy, takže nie sú zobrazené ani tu. V skutočnosti spolu ale vzťahy majú. Podstatná je tabuľka *udaj*, ktorá pokiaľ má *typ* nastavený na hodnotu 4, *ID_ID* určuje ID v tabuľke *abstrakt*. Tabuľke *abstrakt* už vieme z zistiť kľúčové slová pomocou *klucove_slova* a podľa *ID_autor* v tabuľke údaj sa vieme dopracovať k *pouzivatel*.



Obr. 11 – Entity relevantné pre import podľa kľúčových slov

Implementácia

Mierny problém bol v nekonzistentnosti oddeľovania jednotlivých kľúčových slov, niekedy pomocou medzery, niekedy cez čiarku. Systém teda hľadá čiarku a oddeľuje podľa nej, pokiaľ čiarku nenájde, oddeľuje podľa medzery. Funkcionalitu zabezpečujú tri metódy v *Users* kontroleri. Metóda *yb_add_method2* prechádza všetkých používateľov v databáze systému a podľa osobného čísla hľadá kľúčové slova cez YonBan cez metódu *yb_get_keywords*. Pokiaľ nájde, vnorený cyklus prejde opäť všetkých používateľov (s výnimkou práve spracúvaného) a opäť k nim hľadá kľúčové slová, ktoré porovnáva cez metódu *yb_keywords_match*. Pri aspoň jednej zhode pridá do databázy vzťah *klucove_slova* pre týchto dvoch používateľov.

5.2. Šprint č. 2

5.2.1. Zložené filtrovanie

Popis

Ako používateľ pracujúci so zoznamom študentov chcem mať možnosť spájať filtre na atribúty a charakteristiky a tak si postupne ohraničovať množinu študentov. Samozrejme mám možnosť jednotlivé filtre aj odoberať.

Analýza

Pri práci s veľkým počtom ľudí, ktorých je potrebné zaradiť do tímov je potrebné vedieť týchto ľudí rýchlo a prehľadne odlíšiť na základe nejakých zvolených kritérií. Aby sa zamedzilo hromadnému prezeraniu všetkých profilov študentov kvôli potrebe nájsť študentov s určitou charakteristikou, ktorí by sa hodili do vytváraného tímu, môžem si aplikovať ľubovoľný filter a vypísať tak iba študentov, ktorí ma zaujímajú. Keďže používateľ systému zaraďujúci študentov do tímu môže potrebovať filter na študentov na základe viacerých kritérií, je potrebné mu umožniť vytváranie zložených filtrov. Jednou z požiadaviek je aj prehľadné ponúkanie atribútov, ktoré budú použité pri filtrovaní, na základe zvolených typov atribútov. Z používateľského hľadiska je vhodné, aby nebolo potrebné pri vytváraní zložených filtrov čakať po každom spresnení výberu na načítanie stránky. Ďalšou požiadavkou je, aby bolo možné vytvárané filtre individuálne odoberať a meniť podľa potrieb používateľa.

Popis procesu zloženého filtrovania

Pre aplikovanie zloženého filtra je potrebné najprv navoliť požadované parametre filtra. Ak chce používateľ vytvoriť filter na základe charakteristiky, je potrebné zvoliť typ charakteristiky. Následne sa dynamicky zobrazí zoznam typov atribútov, na ktoré sú namapované základné atribúty. Po zmene typu atribútu je prispôsobovaný aj zoznam atribútov.
Ak chce používateľ aplikovať filter na vzťah medzi študentmi, je potrebné zvoliť si typ vzťahu a jeho orientáciu. Kliknutím na tlačidlo „Add filter" je filter pridaný do zoznamu filtrov a zoznam vyfiltrovaných študentov je prispôsobený novému zloženému filtru. Súčasne je popis nového filtra spolu s odkazom na jeho odobratie zobrazený vo výpise aplikovaných filtrov. Používateľovi je umožnené odoberanie individuálnych filtrov ale aj zmazanie celého zoznamu filtrov po kliknutí na odkaz „Clear filters". Po zmazaní filtra je možné aktualizovať zoznam používateľov na základe aktuálneho zloženého filtra.

Návrh a implementácia

Filtrovanie je z časti implementované v *users* kontrolleri a sčasti v *users* modeli. Skladá sa z dvoch častí:

- Metódy práce s filtrom
- Metódy umožňujúce filtrovanie na základe zvolených filtrov

Metódy práce s filtrom sú implementované v *users* kontrolleri. Jedná sa konkrétne o metódu *filter*, ktorej úlohou je vytvorenie zložených filtrov a ich uloženie do session. Ďalej je to metóda *delete_filter*, ktorá zmaže vytvorený filter po kliknutí na zmazanie. Pre vytvorenie filtra je potrebné dynamicky meniť *select* boxy s výberom atribútov na základe zvolených typov atribútov. Na dynamické zobrazovanie týchto vstupov pre vytváranie filtra boli vytvorené metódy *show_attribute_types_combobox* a *show_attributes_combobox*.

Základnou metódou umožňujúcou filtrovanie na základe zvolených filtrov je metóda *getFilteredUsersSet*, ktorej vstupom je zoznam filtrov a výstupom zoznam vyfiltrovaných osôb.

Pre výpis zoznamu filtrov bol vytvorený nový view *_filter.html.erb*, ktorý sa vkladá nad formulár s filtrom a aktualizuje sa po každej zmene zloženého filtra.

Pre prácu s jednotlivými filtrami a ich uchovávanie bola vytvorená trieda *MyFilter*.

5.2.2. Spustenie pageranku

Popis

Spustenie Pagerank algoritmu nad sociálnou sieťou študentov, aby používateľ v role pedagóg (staff) a vyššie mohol z charakteristík odčítať túto hodnotu pre každého študenta.

Analýza problému

Pre riešenie sme sa rozhodli využiť knižnicu JUNG napísanú pre programovací jazyk Java a preto sme museli zistiť ako spustiť javovské triedy v jazyku Ruby. Tento problém sme vyriešili použitím interpretera jazyka Ruby v jazyku Java, ktorého názov je JRuby. Tento interpreter umožňuje spúšťať okrem štandardného Ruby kódu aj Java príkazy a triedy.

Implementácia

Implementácia spočíva vo vytvorení reprezentácie grafu študentov za pomoci tried JUNG knižnice. Pri vytváraní tohto grafu je nevyhnutné, aby bol do úvahy vzatí len študenti, ktorí boli vybraní na vytvorenie grafu a neboli tam zbytočné hrany so študentmi, ktorí sa nenachádzajú v grafe. To by malo za následok nepresný výsledok. Následne je tento graf použitý na výpočet hodnoty Pagerank. Výstup

algoritmu je potrebné spracovať a následne vytvoriť nové charakteristiky alebo prepísať existujúce charakteristiky pre Pagerank študentov.

5.2.3. Vytvorenie „teams views“

Popis

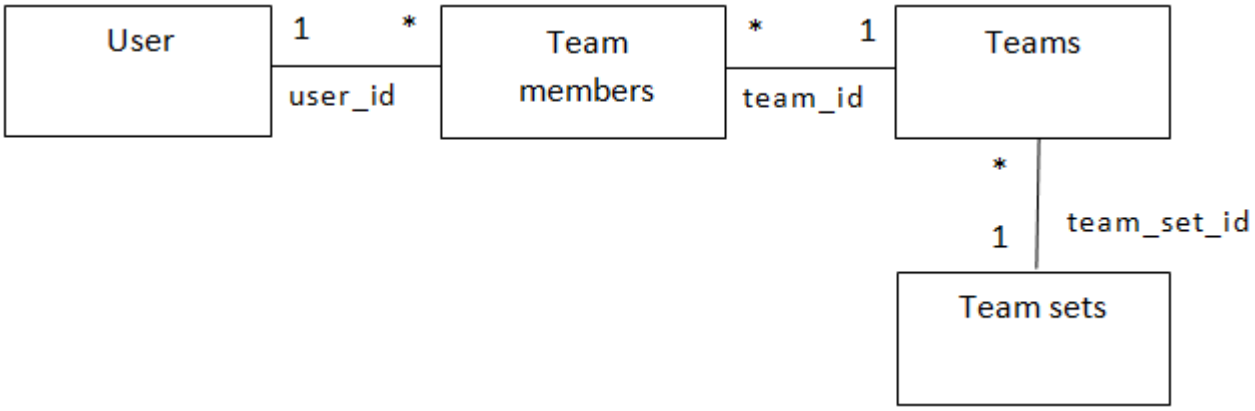
Hlavnou úlohou systému je tvorba pracovných tímov. Používateľ, preto má mať možnosť tímy prezerať, upravovať ich, alebo mazať priamo s grafického rozhrania systému.

Analýza problému

Tvorba tímov si vyžaduje vytvorenie dátových štruktúr v modeli, kde budú tímy uložené. Dáta vytvorených tímov bude možné editovať z grafického rozhrania. Dátové štruktúry v modeli je potrebné navrhnúť tak, aby ich vedeli využiť aj algoritmy pre automatické vytváranie tímov.

Návrh a implementácia

Pri riešenie tejto úlohy bolo potrebné začať od vytvorenia *modelu*, cez vytvorenie *kontrolerov* až po samotné *views.* V databázovom modeli pribudla tabuľka *teams* a *team_members* . *Tabuľka teams* obsahuje vytvorené tímy, presnejšie, statické údaje o nich. Tabuľka *team_members* ukladá relácie medzi tímami a používateľmi, teda určuje, kto sa nachádza v akom tíme.
Aby bolo možné vytvárať tímy viacerými používateľmi súčasne, alebo vytvorenie viac skupín tímov pre jedného používateľa, bola navrhnutá ďalšia tabuľka *team_sets*, ktorá slúži pre spájanie tímov do skupín (skupina tímov má rovnakú hodnotu atribútu *team_set_id*). Časť modelu, ktorá zachytáva všetky vyššie spomenuté tabuľky je zobrazená na obr. 12.



Obr. 12 – Časť DB modelu pre reprezentáciu tímov

Možnosť ručnej editácie je dostupná cez GUI systému. Funkcionalita je dostupná pre používateľov v roli administrátor, alebo pedagóg (staff).

5.2.4. Vytvorenie Ruby funkcionality na tvorbu tímov a URL, ktorá vracia zoznam tímov

Popis

Cieľom tejto úlohy je zabezpečiť API, ktoré by bolo možné volať z *appletu* pre vizualizáciu siete. Takto by malo byť umožnené z *appletu* pridávať členov tímu, odoberať členov z tímu a zobraziť tímy a ich členov.

Analýza problému

Metódy pre úpravu a zobrazenie členov tímu je potrebné poskytnúť *appletu*. To znamená, že musia byť pridané medzi *routes* systému a vystavené v kontroleroch *teams*, alebo *team_members*, aby ich bolo možné prostredníctvom HTTP požiadavky zavolať.

Návrh a implementácia

Metódy umožňujúce pridanie a odobratie člena tímu sú vytvorené v kontroleri *team_members.* Umožňujú jedným volaním pridať/odobrať *n* používateľov v rámci jedného tímu. Metódy vracajú vo výsledku, mená členov tímu po vykonaní operácie pridania/odobratia.

Metóda, ktorá zobrazuje aktuálny prehľad tímov, čiže tímy a členov, ktorí sú v tíme zaradení je pridaná v *teams* kontroleri. Výsledkom volania metódy je zoznam tímov v XML formáte v tomto tvare:

```
<teams>
	<team>
		<team-name>team[0].name</team-name>
		<team-id>team[0].id</team-id>
	</team>
	<team>
		<team-name>team[1].name</team-name>
		<team-id>team[1].id</team-id>
	</team>
```

...
</teams>

Príklady volaní metód:

- pridanie člena do tímu
  - [RELAX_ROOT]/teams/1/add/

  kde parametrom HTTP požiadavky je list obsahujúci identifikátory používateľov (user.id)
- odobratie člena z tímu
  - [RELAX_ROOT]/teams/1/remove/

  kde parametrom HTTP požiadavky je list obsahujúci identifikátory používateľov (user.id)
- výpis tímov
  - [RELAX_ROOT]/teams/xml_report/

5.2.5. Práca s hranami v grafe

Popis

Pri práci s grafom si môžem vyberať ktoré typy hrán (vzťahy) sa majú zobrazovať. A tie typy hrán by mali byt farebne odlíšené.

Analýza problému

Aby si používateľ mohol vo nástroji na vizualizáciu vzťahov vyberať, ktoré hrany (vzťahy) sa zobrazia, je potrebné aby dostal možnosť výberu týchto hrán. Tieto je možné dostať z GraphML vstupu *appletu*. Ten totiž obsahuje informácie o celom grafe, a teda aj o hranách (vzťahoch), ktoré sú v grafe obsiahnuté. Tým sa zabezpečí aj skutočnosť, že vyberať si je možné len z hrán, ktoré obsahuje vstupný graf.

Návrh a implementácia

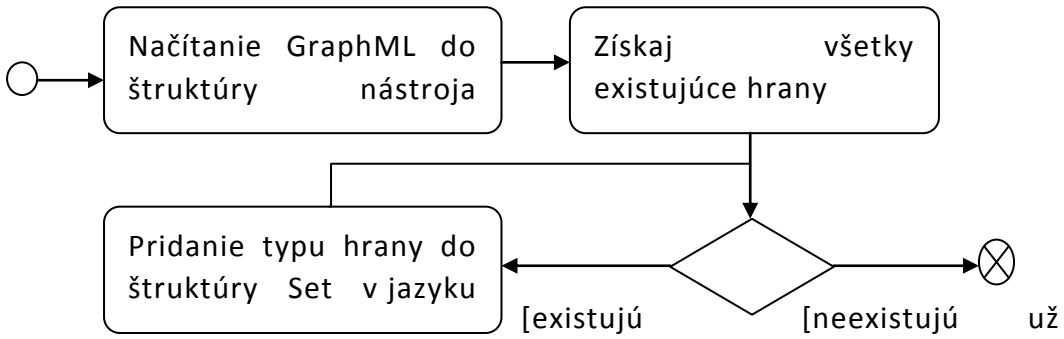
Pre tieto potreby je ale nutné, aby vstupné GraphML obsahovalo informácie aj o hranách. To sa zabezpečí nasledovným rozšírením:

- Pridaním do dátovej schémy GrpahML

  ```
  <key id="type" for="edge" attr.name="type" attr.type="string"/>
  ```
- Pridaním informácie o type do hrán

  ```
  <edge source="12" target="7">
        <data key="type">Priatelia</data>
  </edge>
  ```

Následne je potrebné získať zoznam existujúcich hrán. Tento proces popisuje obr. 13.



Obr. 13 – Proces získania existujúcich hrán

Aby bolo možné skryť vybrané hrany, je potrebné vytvoriť filter *EdgeVisibilityFilter*, ktorý hrany skryje.
Pre získaný zoznam typov hrán je následne pomocou *Swing* komponenty vytvorený *select* box pre zobrazenie vybranej hrany. Tomu sa priradí *ActionListener* na odchytávanie zmeny výberu. Pri zmene je potrebné, aby sa vytvoril *Tuple* zoznam hrán, ktorý bude obsahovať hrany, ktoré budú zobrazené. Tým sa vo vypĺňaní hrán farbou, priradí požadovaná farba. Pre ostatné je farba nastavená na bielu.
Aby filter fungoval, je potrebné ho pridať do zoznamu filtrov v akciách.

5.2.6. Ovládanie appletu cez menu

Popis

Ako admin alebo pedagóg (staff) chcem mať na pravý klik dostupné menu, v ktorom si viem vyberať, ktoré typy hrán majú byť zobrazené.

Analýza problému

Presunutie výberu zobrazených hrán do menu bolo zapríčinené predovšetkým tým, aby bolo možné zobraziť alebo nezobraziť viacero typov hrán. Výber bude prebiehať pomocou zaškrtávacích boxov v menu. Toto menu sa zobrazí pri stlačení pravého tlačidla na plochu vizualizačného nástroja, avšak nie na uzol grafu. To je rezervované pre inú funkcionalitu.

Implementácia a návrh

Pre potrebu menu v nástroji je potrebné vytvoriť vlastné menu, ktoré bude asociované na pravé tlačidlo, v prípade, že nie je použité na uzol grafu. Menu bude pozostávať z triedy *MyPopupMenuForEdges*, ktorá obsahuje inicializáciu ako aj spracovávanie informácií pre celé menu, a komponenty *JPopupMenu*. Toto menu bude pozostávať zo submenu, do ktorého bude načítaný zoznam hrán ako položky

menu *JChoiceMenuItem*. Všetky hrany sú na začiatku zobrazené, teda aj všetky položky menu sú zaškrtnuté.
Jednotlivé typy hrán sú získane rovnakým spôsobom ako pri predchádzajúcom *User Story* „Práca s hranami v grafe“. Pri zmene stavu niektorej z položiek v menu, sa zavolá filter a spustí sa prekreslenie grafu.
Trieda *MyPopupMenuForEdges* je navrhnutá pomocou vzoru singleton, kvôli potrebe prístupu k nej z viacerých miest aplikácie.
Taktiež bolo potrebné upraviť samotný filter. Ten totiž bral do úvahy len jeden vybraný typ hrán. Bolo potrebné ho upraviť tak, aby vyberal hrany pre všetky vybrané typy hrán.

5.3. Šprint č. 3

5.3.1. Ohodnotenie tímu na základe kritérií

Popis

Pre proces automatického vytvárania tímov a zadeľovanie študentov do týchto tímov je potrebné vytvoriť metódy na spoľahlivé ohodnocovanie vytváraných tímov na základe zadefinovaných kritérií pre dobrý tím.

Analýza problému

Vytváranie tímu používa evolučné algoritmy, ktoré sa na základe fitnes musia vedieť rozhodnúť o kvalite vytvoreného tímu a týmto spôsobom môžu vytvárať tímy študentov s približne rovnakou úrovňou kvalifikácie študentov. Tímy študentov by mali byť rovnomerne vytvorené, tak aby nebol jeden tím výrazne silnejší ako iný.

Návrh a implementácia

Vytvorený databázový model poskytuje dostatočné množstvo informácií na výpočet ohodnotenia tímu na základe vytvorených kritérií pre dobrý tím. Zoznam kritérií s ohraničeniami sa nachádza v entite *team_criterias*, ktorá obsahuje typy atribútov, hodnoty atribútov, maximálne, alebo minimálne počty študentov, ktorí musia v rámci tímu spĺňať kritérium, skupinu kritérií, ktorá sa uplatňuje na tím a teda všetky kritéria v danej skupine sú aplikované na vytváraný tím. Okrem iného sa tam nachádzajú aj informácie potrebné pre definovanie kritéria pre vzťahy medzi študentmi a minimálnu, alebo maximálnu hodnotu v prípade špeciálnych atribútov obsahujúcich číselnú hodnotu. Pre výpočet ohodnotenia a uľahčenie práce algoritmom na vytváranie tímov boli vytvorené nasledujúce metódy v modeli *TeamCriteria*:

- **get_team_mark** – Vracia ohodnotenie tímu, ktoré je sumou dôležitostí kritérií zadefinovaných pre každé kritérium u členov tímu spĺňajúceho danú skupinu kritérií. Vstupom je zoznam členov tímu, skupina kritérií a používateľ, ktorý danú skupinu kritérií vytvoril. Pri výpočte ohodnotenia sa aplikuje iba skupina kritérií, ktorá bola vytvorená daným používateľom.
- **get_team_criteria_count** – Vracia počet členov tímu spĺňajúcich kritérium. Vstupom je zoznam členov tímu a jedno konkrétne kritérium.
- **get_team_members_based_on_criteria** – Vracia zoznam členov tímu spĺňajúcich kritérium. Vstupom je kritérium a zoznam členov tímu.
- **get_team_pagerank** – Vracia súčet Pagerankov členov tímu. Vstupom je zoznam členov tímu.

5.3.2. Spúšťanie analýzy grafov

Popis

Cieľom tejto *user story* bolo rozšíriť implementáciu počítania Pageranku o iné algoritmy nad grafom z JUNG knižnice.

Analýza problému

Najprv bolo nevyhnutne nájsť časti kódu z Pagerank algoritmu, ktoré sú rovnaké aj pre iné algoritmy nad grafmi z JUNG knižnice. Preto sa tento algoritmus rozdelil na tri časti, kde v prvej časti sa vytvorí graf študentov, v druhej časti sa vykoná samotný algoritmus a v tretej časti sa spracuje výsledok a uloží výstup do charakteristík študentov. Vďaka tomuto rozdeleniu stačí na pridanie ďalšieho algoritmu z JUNG knižnice len vymeniť strednú časť za iný algoritmus a nastaviť do akej charakteristiky sa má ukladať.

Implementácia

Implementácia spočíva vo vytvorení podporných metód *create_graph*, ktorá sa nachádza v triede *ServicesHelper* a *add_characteristic*, ktorá sa nachádza v modeli *User*. Metóda *create_graph* slúži na vytvorenie grafu študentov na základe listu študentov, ktorý dostáva ako parameter. Metóda *add_characteristic* slúži na vytvorenie alebo zmenenie, ak je už taká charakteristika vytvorená pre používateľa, ktorého dostala v argumente. Spúšťanie JUNG algoritmov je riešené cez odkaz na hlavnej stránke, ktorý po stlačení zavolá konkrétne metódy cez Ajax a pridá dané charakteristiky študentom. Táto funkcionalita je dostupná pre používateľov s rolou pedagóg (staff) a administrátor.

5.3.3. Umožniť pridávanie atribútov používateľom

Popis

Cieľom tejto úlohy je zabezpečiť API, ktoré by umožnilo volaním jednej metódy pridať atribút charakteristiky používateľovi.

Analýza problému

Pridanie atribútu používateľovi v sebe zahŕňa niekoľko operácií s údajmi charakteristiky, typom charakteristiky, atribútom a jeho typom. Metóda overí, či typ charakteristiky už existuje, ak nie, tak vytvorí nový. Podobne overí aj existenciu typu atribútu. Podľa toho buď vytvorí novú charakteristiku, a priradí jej nový atribút, alebo zmení existujúcu hodnotu atribútu.

Návrh a implementácia

Metóda pre pridanie atribútu používateľovi má štyri vstupné parametre:

- *userId*
- *typeOfCharacteristicName*
- *typeOfAttributeName*
- *atributeValue*

Postup pridania atribútu je nasledovný:

1. ak existuje typ charakteristiky s názvom typeOfCharacteristicName, tak ho načítaj z databázy, inak vytvor nový s názvom typeOfCharacteristicName
2. ak existuje typ atribútu s názvom typeOfAttributeName a k typu charakteristiky s bodu 1, tak ho načítaj z databázy, inak vytvor nový s názvom typeOfAttributeName
3. ak existuje charakteristika pre používateľa userId, s typom z bodu 1, tak ju načítaj z databázy, inak vytvor novú
4. ak existuje atribút charakteristiky z bodu 3 a typom z bodu 2, tak ho načítaj z databázy, inak vytvor nový
5. atribútu nastav hodnotu atributeValue

5.3.4. Algoritmus na vytvorenie tímov

Algoritmus je navrhnutý ako automatizovaná alternatíva k tvorbe tímov. Pri návrhu algoritmu sa vychádzalo z nasledovných predpokladov definície tímových kritérií:

- Tímové kritériá sú zoskupené do disjunkcií konjunkcií
- Jeden tím zodpovedá jednému zoskupeniu konjunkcií

Algoritmus sa snaží o priradenie ľudí do tímov tak, aby čo najviac zodpovedali kritériám. Na získanie potenciálnych členov tímu sa využíva funkcia *get_team_members_based_on_criteria*, ktorá vracia pre dané kritérium vhodných kandidátov. Celý algoritmus pracuje v nasledovných krokoch (pseudoalgoritmus) :

1. Načíta zoznam tímových kritérií zoradených podľa *confidence*
2. **Pre** každé kritérium :
    a. Zaraď kritérium do disjunktívnej normálnej formy na najbližšiu voľnú pozíciu v konjunkcii
3. **Pre** každú položku disjunktívnej normálnej formy :
    a. **Ak** pre danú položku neexistuje tím
        i. Vytvor ho
    b. **Ak** existuje
        i. Vyprázdni ho
4. **rob**{
    a. **Pre** každý tím :
        i. Zober prvé kritérium v zozname konjunkcií
        ii. Nájdi človeka, ktorý najviac zodpovedá danému kritériu
        iii. **Ak** človek existuje
            1. Priraď ho do tímu
            2. **Ak** je kritérium úplne splnené
                a. Odstráň ho
        iv. **Ak** človek neexistuje
            1. Presuň kritérium na koniec zoznamu v konjunkcii
    }**pokiaľ** bol pridaný aspoň jeden človek do niektorého tímu

5. **pokiaľ** existujú nezaradený ľudia

Takýmto spôsobom sa algoritmus snaží dosiahnuť nasledovné ciele :

- zaradiť čo najviac používateľov tak, aby zodpovedali kritériám
- zaradiť používateľov tak, aby nenastal prípad, kedy jeden používateľ zodpovedá všetkým kritériám a ostatní sú doplnení automaticky ako zvyšok.

5.3.5. Vyvolanie pop-up cez dropdown menu na pravý klik

Popis

Ako admin alebo pedagóg (staff) si na pravý klik v applete viem zobraziť menu, v ktorom si viem vybrať zobrazenie pop-up s informáciami o používateľovi.

Analýza problému

Podobne ako pri *user story* „Ovládanie appletu cez menu“ aj tu je potrebné vytvoriť menu, avšak v tomto prípade ide o menu, ktoré sa otvorí, po kliknutí pravým tlačidlom myši na uzol grafu. Týmto bude ale potrebné funkcionalitu, ktorá na tomto tlačidle už je na mapovaná presunúť do tohto menu.

Implementácia a návrh

Opäť, ako pri už spomenutom *user story*, je potrebné vytvoriť menu. Toto menu bude pozostávať taktiež z triedy, ktorá toto menu bude ovládať, a z komponenty *JPopupMenu*. V tomto prípade sa však trieda volá *MyPopupMenu*. Samotná funkcionalita pre zobrazenie detailu konkrétnej osoby z grafu je presunutá do tohto menu na prvú úroveň. Vyvolanie zobrazenia tohto menu je zas naopak presunuté do *UserInformationControl* triedy, ktorá odchytáva zmeny vyvolané na uzloch grafu, a teda ľuďoch.

5.3.6. Editovanie tímov cez dropdown menu v applete

Popis

Ako admin alebo pedagóg (staff) si na pravý klik v applete viem zobrazím menu v ktorom viem ručne editovať tímy (vybrať používateľa, odstrániť používateľa).

Analýza problému

Pridávanie používateľov do tímu už bolo rozoberané v kapitole „Ručné zadeľovanie študentov do tímov v grafickom móde“. V tomto prípade sa presunie časť funkcionality do menu, ktoré sa zobrazí nad uzlom grafu, teda používateľom. Tu bude možné pomocou výberu tímu z menu priradiť konkrétneho používateľa do konkrétneho tímu. Pridávanie viac používateľov bude prebiehať rovnakým spôsobom ako doteraz.
Jednotlivé tímy budú zaradené do menu pod položku „Pridanie do tímu“. Druhou alternatívou je „Odobranie z tímu“, kde ale nie je potrebné riešiť konkrétny tím.

Implementácia a návrh

Pridávanie funkcionality prebieha iba doplnením menu nad uzlom grafu o ďalšie položky. Pridávanie používateľov do tímu už taktiež implementované je. Problémom v tomto prípade ostáva iba získanie zoznamu tímov, ktoré budú ponúknuté.
Tie sa získajú prostredníctvom vstupného parametra URL adresy a jej zavolania. Návratové XML je vo formáte :

```
<teams>
    <team>
        <team-name>#nazov</team-name>
        <team-id>#id</team-id>
    </team>
</teams>
```

Pomocou tohto XML vizualizačný nástroj získa zoznam tímov, ktoré ponúkne na výber na pridanie používateľa do tímu.

5.3.7. Podpora exportu tímov

Popis

Je potrebné zapracovať export zoznamu tímov, ktorý vznikol buď ručným vytvorením alebo automatickou generáciou, do PDF formátu.

Analýza problému

Keď si používateľ, ktorý ma na to oprávnenie vytvorí sadu tímov (team set), tak je vhodné ak má možnosť túto sadu aj vyexportovať zo systému pre ďalšie využitie. Najvhodnejší formát predstavuje PDF, keďže je najrozšírenejší a platformovo nezávislý.

Návrh a implementácia

Na tvorbu PDF formátov je možné využiť rôzne pluginy do Ruby on Rails, ktoré sa líšia najmä svojím API, možnosťami, ktoré poskytujú a zložitosťou na používanie. Ako najlepšia voľba sa ukázal plugin do Ruby s názvom prawn a jeho nadstavba do Rails prawnto, ktorá poskytuje možnosť previazať view a kontroler. Všetko čo treba spraviť, je vytvoriť súbor s príponou .prawn, umiestniť ho do views a naplniť ho podľa zverejneného API k prawn, pridať malú úpravu do kontrolera a zavolať príslušné URL zo zvoleného view.

5.3.8. Integrácia na AIS

Popis

Pripojenie na databázu školského informačného systému by teoreticky systému mohlo poskytnúť veľa užitočných informácií o charakteristikách jednotlivých študentov aj o vzájomných vzťahoch medzi nimi.

Analýza

V praxi sme žiaľ dostali veľmi obmedzený prístup do systému, navyše bez akéhokoľvek dátového modelu. Ako užitočné dáta sa javia vzťahy na báze spolužiakov, keď vieme zistiť kto s kým chodí, prípadne chodil do ročníka. Ďalej sa dajú hromadne importovať používatelia, prípadne predmety ako atribút charakteristiky *absolvoval predmet*.

Návrh a implementácia

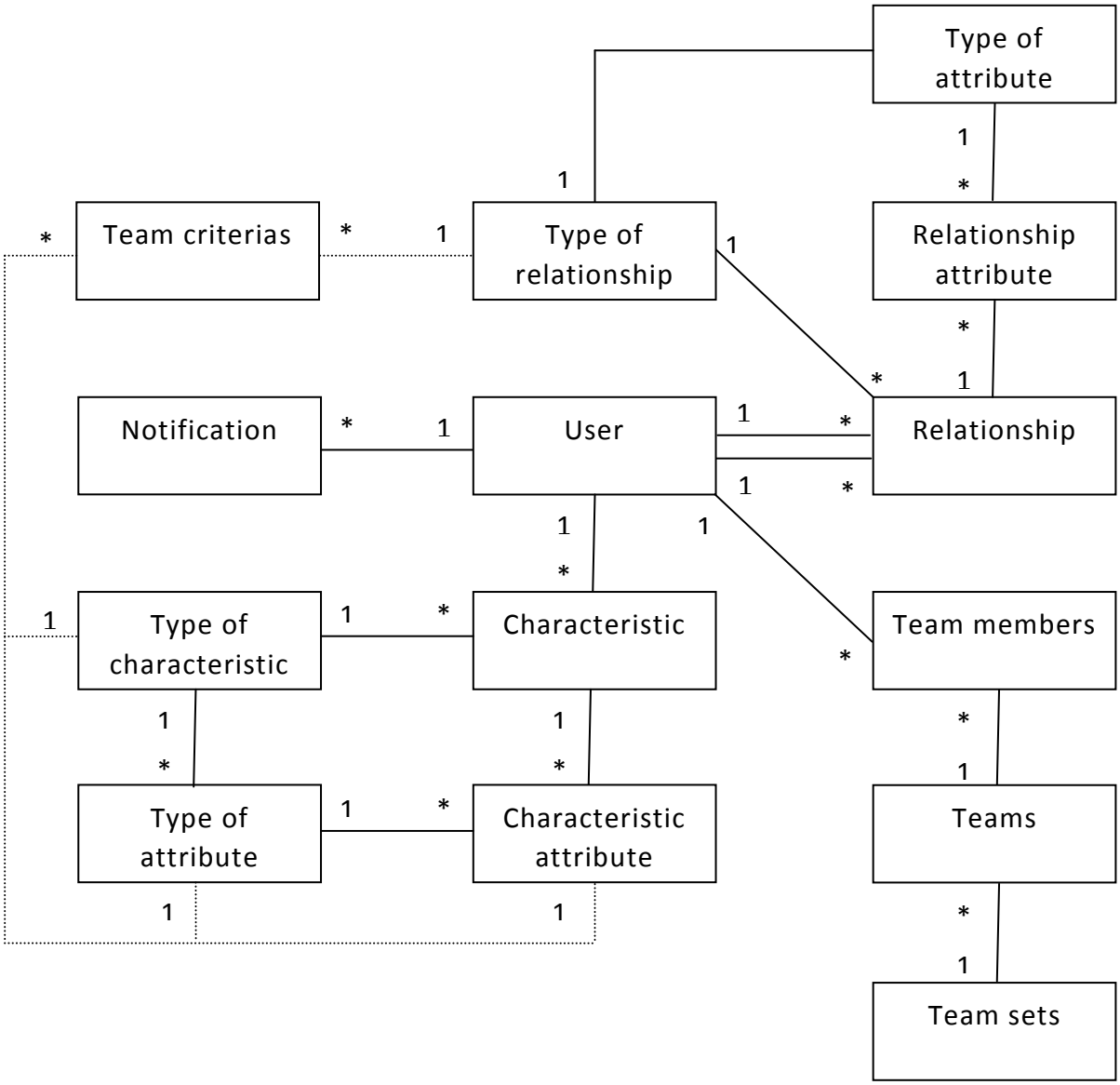
Kvôli používaniu JRuby ako interpretera bolo spojazdnenie pripojenia na AIS databázu, ktorou je Oracle, pomerne zložité. Nakoniec sa to podarilo pomocou adaptéru JDBC a jeho inicializovaní priamo Javou. Táto implementácia sa nachádza v *ais_connection.rb* v metóde *establishConnection*. Na hlavnej stránke sa nachádzajú tri odkazy, ktoré využívajú konektivitu na AIS. Update by classmates vytvorí vzťahy *spoluziaci* medzi používateľmi podľa toho, či spolu chodia do toho istého ročníka. *Import AID users* naimportuje študentov prvého ročníka inžinierskeho štúdia (tí majú predmet Tímový projekt). *Import AIS predmety* pridá hodnoty atribútu *nazov* pre charakteristiku *absolvoval predmet*. Posledné dve bežia po spustení v samostatných vláknach a o svojom skončení nechávajú notifikáciu.

6. Produkt

Produkt sme vyvíjali počas dvoch semestrov, dalo by sa povedať pol roka. To je približne 24 týždňov, takže podľa tejto logiky by sme mali mať za sebou 12 šprintov. V skutočnosti ich ale bolo 7, jednak kvôli chybám v prvom semestri a jednak kvôli veľkému natiahnutiu posledného šprintu.

Napriek odporúčaniam sme si nezvolili vodopádový vývoj projektu, ale moderný, agilný vývoj pomocou metódy Scrum. Tomuto rozhodnutiu sme prispôsobili aj dokumentáciu, pri ktorej sme upustili od klasického rozloženia na analýzu, návrh, implementáciu a testovania celého produktu, ale každú jednu úlohu (v prípade prvého semestra), alebo *user story* (v prípade druhého semestra) sme dokumentovali podľa tejto šablóny (podľa možností). Ťažko súdiť, ktorý prístup je lepší, tento dáva pekný prehľad o tom, ako sa postupne produkt iteratívne vyvíjal.

Na obr. 14 je možné vidieť výsledný dátový model aplikácie.



Obr. 14 – Výsledný dátový model systému

Prerušované prepojenia z tabuľky *Team criterias* značia dvojaké, resp. nejednoznačné naviazanie ako vysvetľuje kapitola 5.1.2.

Ciele, ktoré sme si dali na začiatku projektu môžeme považovať za splnené. Systém dokáže pracovať s tímami či už manuálne, alebo automaticky. Na tento účel využíva údaje z vytvorenej sociálnej siete. Mnoho vecí sme nestihli, alebo stihli len na poslednú chvíľu, ale zadanie sme splnili. Množstvo vecí ostáva nedotiahnutých, nedokončených, prípadne by sa dalo spraviť lepšie. To by sa ale pravdepodobne dalo povedať o tomto projekte aj o rok, stále by totiž bolo čo vylepšovať.