Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Optimalizácia priradenia tímových projektov riešiteľským skupinám

Bakalárska práca

2016 **Martin Mečiar**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Optimalizácia priradenia tímových projektov riešiteľským skupinám

Bakalárska práca

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava, 2016 **Martin Mečiar**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Martin Mečiar
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Optimalizácia priradenia tímových projektov riešiteľským skupinám
Optimization of Team Projects Assignment to Working Groups

Cieľ: Výsledkom tejto implementačno - výskumnej práce bude webová aplikácia na výber projektov, napr. pre predmet Tvorba informačných systémov alebo podobný. Zadávatelia vyplnia názvy a popis projektov, svoje kontaktné informácie a potrebné znalosti a zručnosti. Študentské skupiny si projekty vyberajú, zadávajú svoje priority i zručnosti. Systém automaticky poprirađuje projekty tak, aby sa maximalizovala spokojnosť, pošle kontaktné informácie zadávateľom aj riešiteľom, vytvorí verejne dostupné zoznamy, na ktoré je možné sa odkazovať z externej stránky. Súčasťou práce je vygenerovanie modelových dát a porovnanie viacerých algoritmov priradenia úloh skupinám a vyhodnotenie tohto porovnania. Dôraz je na bezúdržbovosť systému, prívetivé rozhrania, možnosť k zadaniam pridať obrázky a prílohy.

Literatúra: Petrovič P. (2012) Prirodzený život sa skončil, Kognice a umělý život, Praha, Jún 2012.

Kľúčové slová: bipartitné párovanie v grafe, webová aplikácia, evolučný algoritmus

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 01.10.2015

Dátum schválenia: 28.10.2015

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím literatúry a zdrojov na internete, ktoré sú uvedené v zozname použitých zdrojov.

V Bratislave, 30.5.2016

Martin Mečiar

Poďakovanie

Chcem sa poďakovať svojmu školiteľovi Mgr. Pavlovi Petrovičovi, PhD. za trpezlivosť, cenné rady a vedenie počas vypracovávania tejto práce.

ABSTRAKT

Cieľom tejto bakalárskej práce je vytvoriť webovú aplikáciu na výber projektov, napríklad pre predmet Tvorba informačných systémov alebo podobný. Zadávatelia v tomto systéme vyplnia názvy a popis projektov, svoje kontaktné informácie a potrebné znalosti a zručnosti. Študentské skupiny si projekty vyberajú, zadávajú svoje priority a zručnosti. Systém automaticky popriraďuje projekty tak, aby sa maximalizovala spokojnosť, pošle kontaktné informácie zadávateľom aj riešiteľom, vytvorí verejne dostupné zoznamy, na ktoré je možné sa odkazovať z externej stránky. Dôraz je na bezúdržbovosť systému, prívetivé rozhrania, možnosť k zadaniam pridať obrázky a prílohy.

Kľúčové slová: bipartitné párovanie v grafe, webová aplikácia, evolučný algoritmus

ABSTRACT

The aim of this bachelor thesis is to create a web application for selecting projects for a school subject, for example Development of Information Systems or other. In this system, the submitters will fill out the names and descriptions of their projects, their contact information and skills needed for their projects. Student groups choose from these projects, enter their priorities and skills. The system will automatically assign the project while maximizing the satisfaction of groups. The system will also automatically send contact information to submitters as well as to the assignees and create publically accessible lists to which one can refer to from an external site. A maintenance-free system with friendly interfaces and the ability to upload attachments to the projects is emphasized.

Key words: bipartite matching in graph, web application, evolution algorithm

Obsah

Úvod

V súčasnej dobe je na program aplikovanej informatiky zapísaných takmer 90 študentov, pričom každý z nich musí, okrem iných, absolvovať aj predmet Tvorba informačných systémov. Súčasťou tohto predmetu je aj vypracovanie tímového projektu (obvykle je tím zložený zo štyroch študentov), ktorý im je momentálne prideľovaný vyučujúcim daného predmetu ručne. Keďže pri prideľovaní projektov chceme byť čo najviac féroví, treba pri procese prideľovania zvážiť schopnosti daného tímu a podľa neho projekty zadeliť. Tento úkon je ohromne časovo neefektívny a často nie triviálny. Preto je vhodné na takýto výber projektov realizovať aplikáciu, ktorá by pomohla čas strávený prideľovaním podstatne skrátiť a zefektívniť tak samotný výber projektov pre tímy ako aj chod predmetu.

Momentálne existuje webová aplikácia, ktorá umožňuje registráciu užívateľov, vytváranie tímov a vypĺňanie ich schopností. Zadania projektov, ktoré si môžu tímy vyberať musí do systému ručne pridávať vyučujúci po tom, ako mu ich samotní zadávatelia pošlú na email. Po tomto všetkom nasleduje už spomínaný ručný výber projektov pre tímy.

Zo strany vyučujúceho prichádza požiadavka na vytvorenie takého systému, ktorý by ho čo možno najviac odbremenil od tohto každoročne časovo náročného úkonu. Riešením takejto požiadavky je vznik webovej aplikácie, ktorá by bola schopná automatizovať takéto prideľovanie projektov. Takýto systém by nielen ušetril čas, ale aj zabezpečil férovosť pridelenia projektov a uľahčil a zefektívnil by tak chod samotného predmetu.

Do systému by mali mať prístup tri typy užívateľov – vedúci tímu (študenti), zadávatelia a administrátor. Po zaregistrovaní sa si bude môcť daný užívateľ spravovať svoje aktivity: vedúci tímu bude môcť vytvoriť tím, kde určí jeho meno, členov a schopnosti tímu; zadávateľ bude do systému môcť rovno zadávať svoje projekty; administrátor schvaľuje zadania projektov a v prípade potreby ich vie upravovať. Taktiež by mal systém vedieť automaticky navrhnúť priradenie projektov ku skupinám, ktoré môže administrátor v prípade potreby meniť.

Práca je rozdelená na niekoľko kapitol. V prvej kapitole sa venujeme východiskám, teda nájdeme tu prehľad použitých technológií, zadefinovanie nášho problému a taktiež sú v nej spomenuté už existujúce riešenia. V druhej kapitole budeme skúmať, ktorý zo širokej palety algoritmov založených na princípoch umelého života nakoniec použijeme vo našej práci. Ďalšia kapitola obsahuje analýzu požiadaviek na systém, entitno-relačný diagram

a use case diagram. Štvrtá kapitola obsahuje zhrnutie postupu pri implementácii a taktiež ukážky z aplikácie.

1. Východiská

V tejto kapitole opíšeme technológie, ktoré budú neskôr použité pri implementácii a už existujúce softvérové riešenia.

1.1. Výber programovacieho jazyka a technológií

Technológie, ktoré v prípade realizácie webovej aplikácie prichádzajú do úvahy sú PHP, Python alebo Java. Aj keď PHP nepatrí medzi najnovšie jazyky, plne postačuje na realizáciu tejto aplikácie. Na vývoj bude použitá softwarová architektúra Model-view-controller (MVC), databáza bude realizovaná pomocou MySQL.

1.1.1. JavaScript

JavaScript je vysoko-úrovňový, dynamický, beztypový a interpretovaný programovací jazyk. Spoločne s HTML a CSS tvoria tri základné technológie webu.

1.1.2. Model-view-controller (MVC)

Táto softwarová architektúra rozdeľuje dátový model aplikácie, užívateľské rozhranie a riadiacu logiku do troch nezávislých komponentov tak, že modifikácia niektorej z jeho častí má len minimálny vplyv na ostatné. MVC je niekedy chápaný ako návrhový vzor, avšak architektúry aplikácií sa týka viac ako klasický návrhový vzor.

1.1.2.1. Komponenty

Hlavným komponentom MVC je *model*, ktorý zachytáva chovanie sa aplikácie z hľadiska oblasti problému nezávisle od užívateľského rozhrania.

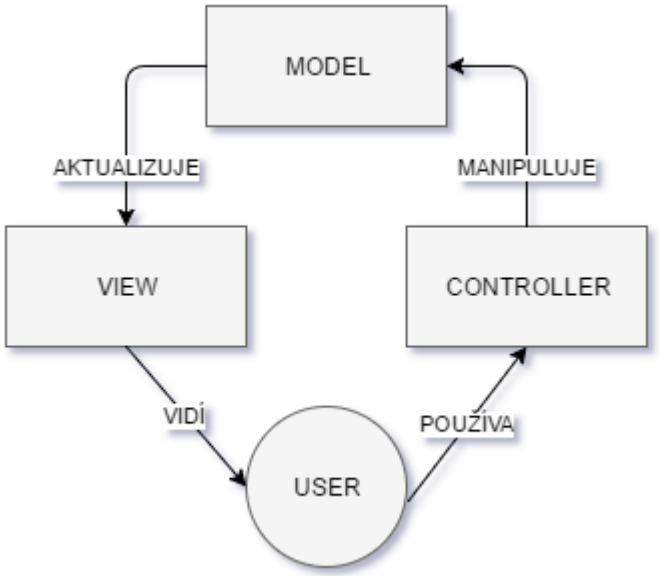
- *Model* priamo spravuje dáta, logiku a pravidlá aplikácie.
- *View* môže predstavovať akúkoľvek reprezentáciu informácií, napríklad tabuľka alebo diagram. Tá istá informácia môže mať rôzne náhľady, napr. stĺpcový diagram pre správu a tabuľkový náhľad pre účtovníctvo.
- Tretia časť, *controller,* preberá vstup a konvertuje ho do príkazov pre *model* alebo *view*.

1.1.2.2. Interakcie

Architektúra MVC okrem rozdelenia aplikácie do troch rôznych druhov komponentov definuje aj interakcie medzi nimi. [1]

- *Model* uchováva údaje, ktoré boli získané na základe príkazov z *controller*-a
- *View* generuje nový výstup pre používateľa na základe zmien v modeli

- *Controller* môže posielať príkazy do modelu na aktualizovanie stavu modelu (napríklad editácia dokumentu). Taktiež môže posielať príkazy príslušnému *view*-u aby zmenil s ním spojenú reprezentáciu modelu



Obrázok 1. Typická spolupráca MVC komponentov.

1.1.3. MySQL

MySQL je voľne dostupný systém na správu relačných databáz, ktorý je napísaný v jazykoch *C* a *C++*.

1.1.4. Bootstrap

Bootstrap je bezplatný, voľne dostupný súbor nástrojov na vytváranie webových stránok. Obsahuje dizajnové šablóny založené na HTML a CSS pre typografiu, formuláre, tlačidlá, navigáciu a iné prvky rozhrania. Taktiež obsahuje voliteľné JavaScriptové rozšírenia. Zameriava sa na uľahčenie vývoja dynamických webových stránok a webových aplikácií.

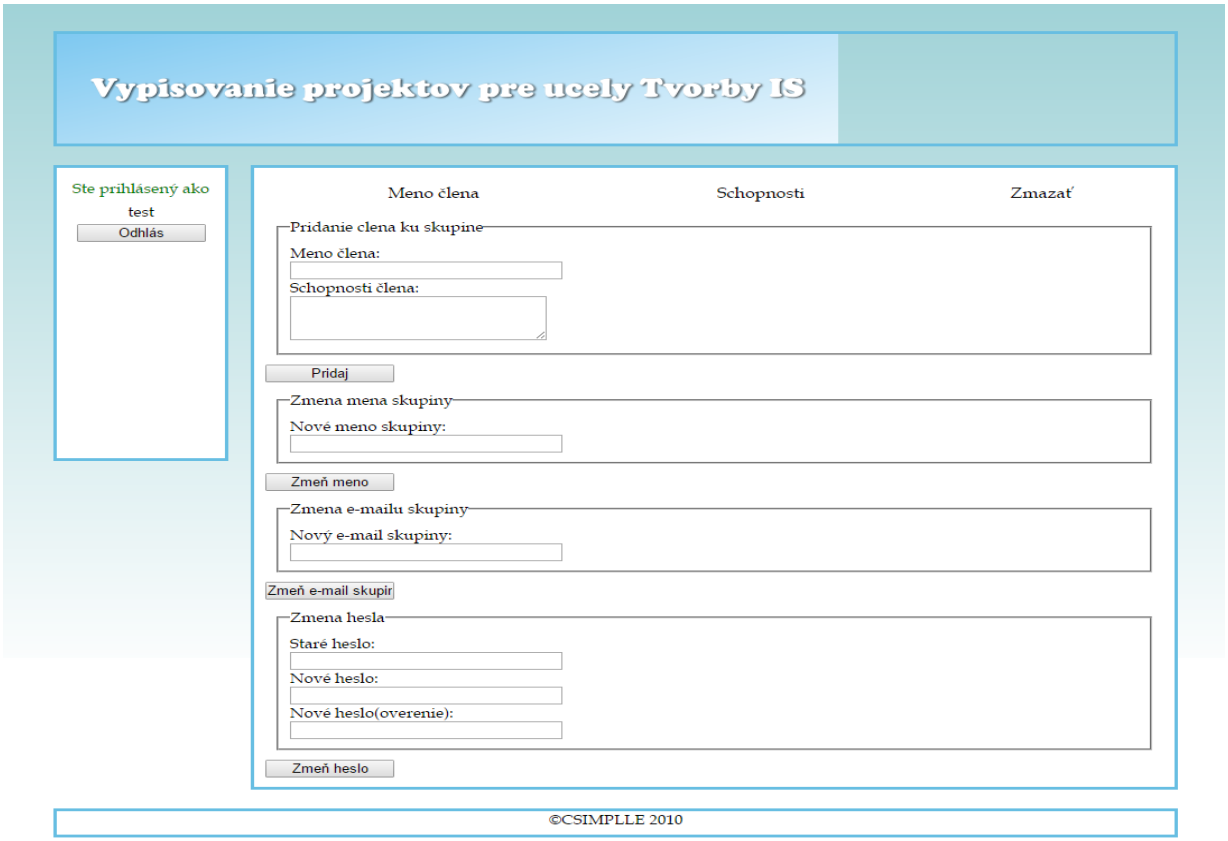
1.2. Existujúce riešenia

Táto časť bude venovaná už existujúcim a funkčným riešeniam.

1.2.1. TIS Select

Táto webová aplikácia je momentálne používaná vo vyučovacom procese na predmete Tvorba informačných systémov. Táto aplikácia slúži na zadávanie projektov, vytváranie a administráciu tímov a aj na priraďovanie projektov k týmto skupinám. Na obrázku nižšie je zobrazené rozhranie určené pre administráciu skupiny. Ako vidno, aplikácia je síce vizuálne jednoduchá, avšak je plne funkčná. Po tom, čo sa všetci študenti rozdelia do skupín a označia, ktoré projekty by potenciálne chceli vypracovať, samotný proces výberu zostáva

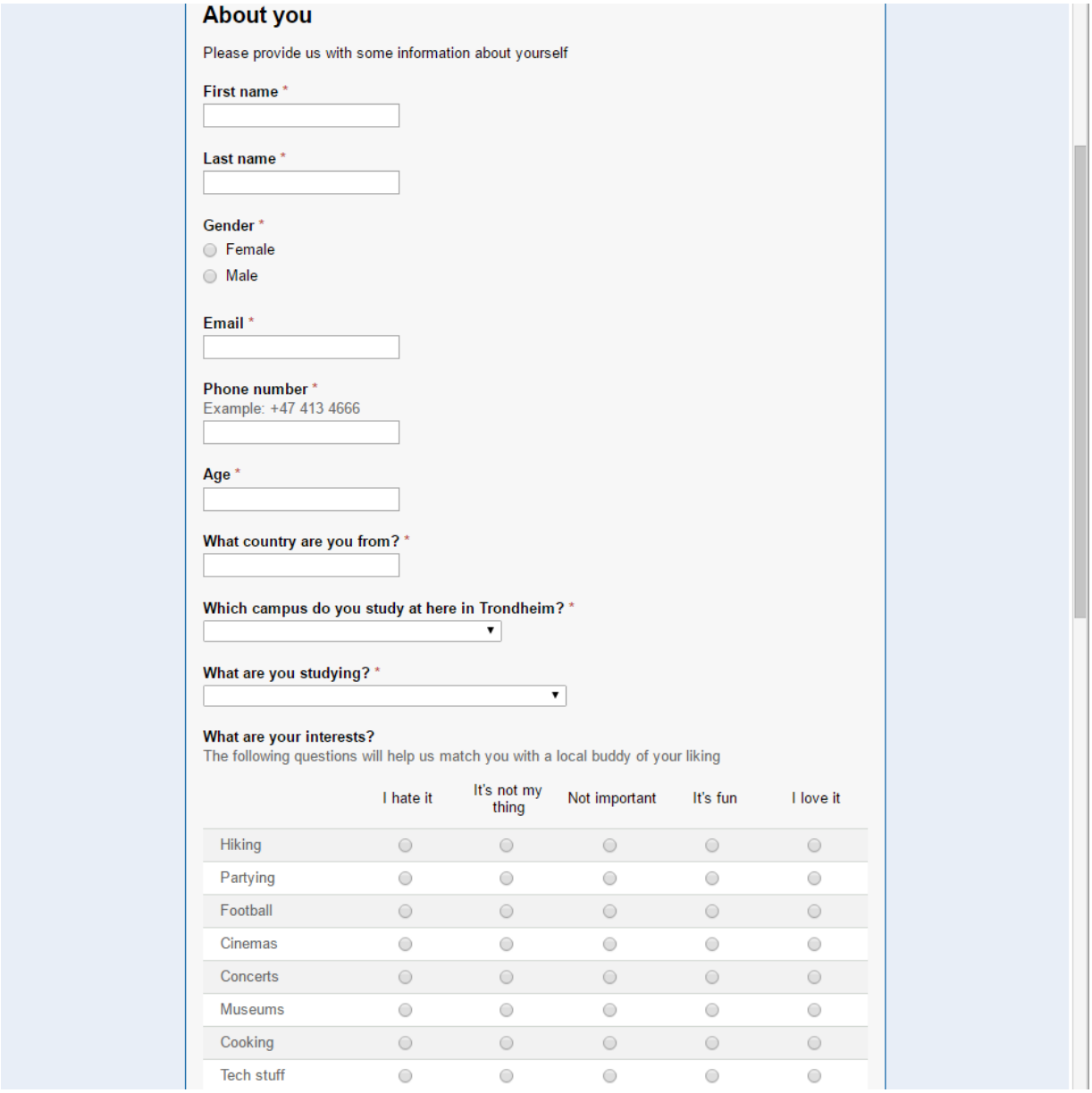
na vyučujúcom – ten musí zobrať do úvahy nielen to, aké predpoklady by mala spĺňať skupina pre daný projekt ale aj to, že sa snaží vyhovieť čo najviac skupinám podľa ich preferencií. Toto je nie vždy triviálna úloha, veď už len pri priemernom počte 20 skupín sa počet všetkých možností rovná $20! = 2.43 \times 10^{18}$ možností.



Obrázok 2 Select - Administrácia skupiny

1.2.2. Buddy program [2]

Cieľom tohto programu bolo zlepšenie integrácie zahraničných študentov do akademického a spoločenského života. Bol určený pre účastníkov študentského výmenného programu Erasmus a iných. Študenti domovských krajín a aj študenti prichádzajúci na výmenný program sa môžu do tohto systému zaregistrovať a po uzavretí registrácií systém vygeneruje najvhodnejšie dvojice domáci študent – hosť. Tieto dvojice nie sú vyberané náhodne – za ich výberom stojí riešenie založené na princípe umelého života, ktoré na základe preferencií a informácií vyplnených účastníkmi počas registrácie vyberá vhodné dvojice. Na obrázku nižšie možno vidieť časť registračného formuláru pre zahraničného študenta.



About you

Please provide us with some information about yourself

First name *

Last name *

Gender *

- Female
- Male

Email *

Phone number *
Example: +47 413 4666

Age *

What country are you from? *

Which campus do you study at here in Trondheim? *

What are you studying? *

What are your interests?
The following questions will help us match you with a local buddy of your liking

	I hate it	It's not my thing	Not important	It's fun	I love it
Hiking					
Partying					
Football					
Cinemas					
Concerts					
Museums					
Cooking					
Tech stuff					

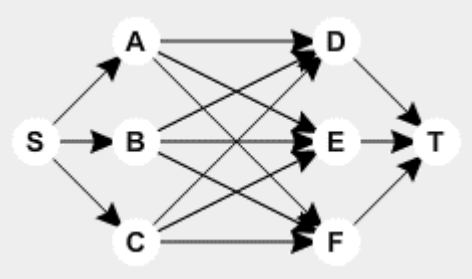
Obrázok 3 Buddy program – registrácia *[3]*

1.3. Formulácia problému

Každý projekt a tím vieme popísať n-ticou atribútov, ktoré sú uložené v tabuľke databázy. Chceme nájsť permutáciu tímov, podľa ktorej budú priradené k usporiadanej množine projektov. Každé potenciálne priradenie projektu k tímu vieme ohodnotiť pomocou funkcie, ktorá na základe požiadaviek projektu a schopností daného tímu určí vhodnosť takéhoto priradenia. Cieľom algoritmu je nájsť takú permutáciu tímov, pre ktorú bude platiť, že súčet vhodností všetkých vytvorených dvojíc tím—projekt bude maximálny.

1.3.1. Maximálne párovanie

Z matematického a algoritmického hľadiska ide o problém maximálneho párovania v bipartitnom grafe. Tento problém vieme vyriešiť tak, že tento graf prerobíme na sieť, v ktorej budeme hľadať maximálny tok. Pretože sa bavíme o sieti, tak do pôvodného grafu musíme pridať ešte dva vrcholy – vstup **s** a výstup **t**.



Obrázok 4 Sieť - ilustračný obrázok

Platí, že zo vstupu **s** vychádza hrana do každého vrcholu označujúceho projekt (vrcholy **A**, **B**, **C** na obrázku) a z každého vrcholu označujúceho tím (vrcholy **D**, **E**, **F**) vychádza hrana do výstupu **t**. Teraz nám na vyriešenie nášho problému stačí nájsť spomínaný maximálny tok. Na jeho nájdenie môžeme použiť napríklad Ford-Fulkersonovu metódu, respektíve Edmonds-Karpovu metódu [4]. Rozdiel medzi týmito dvoma metódami je len v tom, akým spôsobom sa v algoritme hľadá cesta zo vstupu do výstupu (kým vo Ford-Fulkersonovej metóde je použitý algoritmus „do hĺbky“, tak v Edmonds-Karpovej metóde je použitý algoritmus „do šírky“, ktorý je optimálnejší z hľadiska časovej zložitosti).

Ako už aj z obrázka vyplýva, tak ide o matematickú štruktúru s názvom *orientovaný graf*. V takomto grafe vieme potom každej hrane priradiť hodnotu, ktorá bude predstavovať maximálne množstvo, ktoré môže cez danú hranu pretekať a taktiež hodnotu, ktorá bude predstavovať množstvo, ktoré aktuálne hranou preteká (táto hodnota musí byť menšia alebo rovná maximálnej kapacite danej hrany). Taktiež v každom vrchole musí platiť, že súčet pritekajúceho a odtekajúceho množstva je rovnaký. Keď už máme takto vytvorený orientovaný graf, tak môžeme hľadať maximálny tok zo vstupného vrcholu **s** do výstupného vrcholu **t** s tým, že každej hrane nastavíme maximálnu kapacitu rovnú 1.

Potom na základe [2] hľadáme maximálny tok tak, že vytvoríme graf, kde ku každej dvojici vrcholov, ktoré boli spojené v pôvodnom grafe aspoň jednou hranou (v našom prípade, kedy ide o párovanie, je táto hrana najviac jedna), vytvoríme hrany oboma smermi. Tieto potom ohodnocujeme dvoma hodnotami – maximálnou kapacitou a aktuálnym tokom, ktorý cez hranu tečie (tento nikdy nie je väčší ako maximálna kapacita hrany a na začiatku je nulový).

Celkový tok na hrane medzi vrcholmi **(u, v)** potom môže nadobúdať buď kladnú hodnotu – vtedy tečie viac vody z vrcholu **u** do vrcholu **v**, alebo zápornú hodnotu – v tomto prípade tečie viac vody opačným smerom, čiže z vrcholu **v** do vrcholu **u**. Postupujeme podľa Edmonds-Karpovej metódy, kde v každej ďalšej iterácii cyklu najskôr skonštruujeme zvyškový graf, v ktorom potom pomocou prehľadávania do šírky nájdeme cestu zo zdrojového do výstupného vrcholu a na tejto ceste vyhľadáme hranu s najmenšou zvyškovou kapacitou (táto kapacita je vždy rovná 1), o túto kapacitu navýšime tok vo všetkých hranách na nájdenej ceste. V okamihu, kedy už nedokážeme nájsť ďalšiu cestu zo zdroja do cieľa, sme našli maximálny tok.

Zvyškový graf má takú istú štruktúru ako pôvodný graf, teda je to tiež orientovaný graf s hranami, ktoré vedú obojsmerne, a zvyškovú kapacitu hrany vypočítame ako rozdiel maximálnej kapacity v danom smere a celkového toku medzi dvoma vrcholmi berúc do úvahy smer. V prípade párovania v bipartitnom grafe sú všetky toky rovné len -1, 0 alebo 1 a zvyškové kapacity budú vždy len 0 alebo 1 (zvyškové kapacity sú vždy kladné). Zvyškový graf prehľadávame do šírky tak, že uvažujeme len hrany s nenulovou zvyškovou kapacitou. Začíname zo vstupného vrcholu a všetky vrcholy s ním susediace (to sú všetky vrcholy, do ktorých vedie hrana zo vstupného vrcholu) pridávame na FIFO (First-In-First-Out) front, z ktorého sa vždy vyberá ďalší vrchol na spracovanie, až kým sa nedosiahne výstupný vrchol. Ak sa front vyprázdni skôr ako dosiahneme výstupný vrchol, tak neexistuje cesta do výstupného vrcholu.

Náš problém párovania dvojíc projekt—tím je podobný, avšak váha každej priradenej dvojice nie je 1, ale záleží od výsledku funkcie, ktorá počíta vhodnosť projektu pre daný tím na základe schopností tímu a požiadaviek projektu. Mohlo by sa zdať, že stačí len namiesto jednotkovej kapacity medzi každým projektom a tímom nastaviť kapacitu na spomínanú hodnotu z funkcie. Takto by však došlo k znehodnoteniu vlastnosti algoritmu, ktorý počíta s tokom veľkosti jedna zo vstupného vrcholu do každého vrcholu na ľavej strane párovania. Taktiež by toto zväčšenie kapacity viedlo k možnému rozdeleniu toku medzi viaceré hrany a to by narušilo tú vlastnosť párovania, že každý tím má priradený práve jeden projekt. Preto musíme nájsť nejaké iné riešenia na náš problém a ako najlepší kandidát sa ponúkajú biologicky-inšpirované algoritmy umelého života.

2. Riešenie na princípe umelého života

V tejto časti si rozoberieme možné prístupy riešenia nášho problému, ktoré sú založené na princípe umelého života.

2.1. Hill-Climbing metóda

V tejto metóde sa iteratívne porovnávajú nové kandidátske riešenia v okolí aktuálneho kandidáta a ponechávajú sa tie riešenia, ktoré sú lepšie. Takto sa šplháme do kopca až kým nenarazíme na lokálne optimum. Táto metóda má aj agresívnejšie varianty, kedy na rozdiel od generovania len jedného kandidátskeho riešenia sa vygeneruje až *n* kandidátskych riešení naraz a ponecháva sa to, ktoré je najlepšie. Tento variant je nazývaný aj Steepest Ascent Hill-Climbing [5], pretože vďaka tomu, že generujeme viacej kandidátov v okolí nášho pôvodného riešenia a vyberáme toho najlepšieho, prakticky stúpame do kopca prudšie ako v pôvodnej metóde. Doteraz sme však ešte nespomenuli ako sa generujú kandidátske riešenia. Pre jednoduchosť predpokladajme, že naše riešenie je reprezentované ako vektor reálnych čísel nejakej fixnej dĺžky. V takomto prípade vieme vygenerovať nové kandidátske riešenia tak, že s nejakou pravdepodobnosťou p ku každému prvku v našom vektore pripočítame nejaké malé množstvo šumu (čiže nejaké malé číslo). Vo väčšine prípadov sa však $p = 1$. Ak si zvolíme príliš malú hodnotu šumu, tak sa síce dostaneme na vrchol kopca, avšak tento vrchol môže byť len lokálnym maximom a práve kvôli malej hodnote šumu ho nebudeme môcť preskočiť, pretože nové kandidátske riešenia budú vždy len horšie. Na druhej strane, ak bude šum príliš veľký, tak v prípade, že sme blízko nejakého vrcholu, máme problém s konvergenciou k maximu, keďže väčšinou sa vygenerujú riešenia, ktoré tento vrchol preskočia. Tieto doteraz spomenuté metódy predstavujú lokálne optimalizačné algoritmy. Teraz sa budeme venovať tým globálnym.

2.2. Populačné metódy [6]

Populačné metódy sa od tých predchádzajúcich líšia tým, že si v pamäti uchovávame vzorku kandidátskych riešení (na rozdiel len od jedného). Každé kandidátske riešenie podstupuje úpravy a hodnotenie kvality, ale čo naozaj robí tieto metódy niečím viac, ako len paralelným hill-climbom je, že kandidátske riešenia ovplyvňujú ako sa budú šplhať ostatní kandidáti pomocou funkcie, ktorá počíta kvalitu. Takto buď dobré riešenia spôsobia, že tie slabšie budú odmietnuté alebo tie dobré budú modifikované smerom k lepším. Vo všeobecnosti sú základom každého generačného evolučného výpočtového algoritmu, po skonštruovaní počiatočnej populácie, tri kroky, ktoré sa opakujú v cykloch:

- ohodnotenie vhodnosti všetkých jedincov v populácii,
- na základe informácií o vhodnosti sa vyprodukuje nová populácia potomkov,
- nejakým spôsobom sa spojí pôvodná populácia a populácia potomkov.

Evolučné algoritmy sa od seba líšia najmä spôsobom, akým produkujú potomkov a ako ich potom pridávajú do pôvodnej generácie. Vytvorenie potomka sa väčšinou skladá z dvoch častí: výber rodičov zo starej populácie a ich následného pozmenenia (či už mutáciou alebo nejakou ich rekombináciou), čím sa vyprodukujú potomkovia. Nová generácia sa vytvára buď úplným nahradením pôvodnej generácie potomkami alebo spojením rodičov, ktorí sú dostatočne vhodní spolu s potomkami.

2.2.1. (μ, λ) algoritmus

Tento algoritmus patrí medzi najjednoduchšie evolučné algoritmy. Začíname s populáciou, ktorá má zvyčajne λ jedincov, ktorí sú vygenerovaní náhodne. Potom iterujeme nasledovným spôsobom. Najskôr ohodnotíme vhodnosť všetkých jedincov v populácii, potom z nej vymažeme toľko jedincov, že nám zostane len μ najlepších. Každý z týchto μ jedincov potom vyprodukuje λ/μ potomkov pomocou obyčajnej mutácie, čím vznikne λ potomkov. Teraz už len vytvoríme novú populáciu z týchto potomkov (pôvodní sú zahodení) a celý cyklus opakujeme. Teda v skratke: μ predstavuje počet rodičov, ktorí prežijú a λ predstavuje počet potomkov, ktorých produkuje spomínaných λ rodičov (hodnota λ by mala byť násobkom μ) [7].

2.2.2. $(\mu + \lambda)$ algoritmus

Tento algoritmus sa od (μ, λ) algoritmu líši len v jednom aspekte – operáciou spájania starej a novej generácie. Kým v (μ, λ) sú všetci rodičia jednoducho nahradení potomkami, tak v $(\mu + \lambda)$ algoritme sa nová generácia skladá z μ rodičov plus λ potomkov. Takýmto spôsobom rodičia súperia s potomkami v ďalšej generácii a teda každá ďalšia generácia má veľkosť $\mu + \lambda$. Vo všeobecnosti je $(\mu + \lambda)$ viac „vykorisťovateľský“ ako (μ, λ), pretože rodičia, ktorí majú vysokú vhodnosť zostanú súťažiť s potomkami. Toto však má aj svoje riziko: dostatočne vhodný rodič môže poraziť členov populácie znova a znova, čím eventuálne spôsobí, že celá populácia bude predčasne konvergovať k priamym potomkom takéhoto rodiča [8], čo znamená, že celá populácia zostala zaseknutá na lokálnom optime v okolí tohto rodiča.

2.3. Genetický algoritmus

Je v mnohom podobný ako stratégia v (μ, λ): iteruje cez ohodnotenie vhodnosti, výber rodičov a vytváranie potomkov, znovuposkladanie populácie. Hlavným rozdielom je však spôsob, akým prebieha výber rodičov a generovanie potomkov – kým v evolučných stratégiách sa vyberajú všetci rodičia a potom sa generujú všetci potomkovia, tak v genetickom algoritme vyberá len niekoľko rodičov a generuje len niekoľko potomkov. Pri rozmnožovaní začíname s prázdnou populáciou potomkov. Potom sa vyberú dvaja rodičia z pôvodnej populácie, skopírujú sa, skrížia sa medzi sebou a tieto výsledky sa zmutujú. Takto dostaneme dvoch potomkov, ktorých následne pridáme do populácie potomkov. Tento proces opakujeme dovtedy, kým nie je populácia potomkov úplne zaplnená.

2.3.1. Elitism [9]

V tomto prístupe vylepšíme pôvodný genetický algoritmus tým spôsobom, že do ďalšej populácie priamo pridáme najvhodnejšieho jedinca alebo jedincov z predchádzajúcej populácie. Týchto jedincov budeme volať elity. Tento algoritmus pripomína $(\mu + \lambda)$ tým, že sa ponecháva najlepší jedinec (alebo jedinci) v budúcich populáciách. Treba mať na pamäti, že pokiaľ si chceme udržať populáciu nejakej veľkosti, tak aby bolo možné krížiť jedincov, musí byť táto veľkosť, zmenšená o počet elít, deliteľná dvomi.

2.3.2. Steady-State genetický algoritmus

Tento algoritmus predstavuje alternatívu k tradičnému generačnému prístupu, pretože namiesto obnovenia celej generácie naraz túto obnovuje len po častiach. Jeho podstatou je iteratívne splodiť jedného, či dvoch potomkov, určiť ich vhodnosť a potom ich dosadiť do samotnej populácie tak, že sa nahradia niektorí už existujúci jedinci. Nižšie je pseudokód (obrázok 7), podľa ktorého neskôr prebiehala implementácia. Používa kríženie, mutáciu a produkuje dvoch potomkov naraz.

Steady-State genetický algoritmus má dve dôležité vlastnosti. Prvou je, že používa len polovicu pamäte oproti tradičným genetickým algoritmom, pretože si stačí pamätať súčasne len jednu populáciu. Druhou je, že je celkom „vykorisťovateľský“ oproti tradičnému prístupu: rodičia zostávajú v populácii, potenciálne po dlhý čas, a preto v prístupoch ako napríklad elitism hrozí nebezpečenstvo, že celý systém začne predčasne konvergovať k viac-menej len kópiám malej skupiny vysoko vhodných jedincov [10]. Tento stav sa môže ešte zhoršiť podľa toho, ako sa rozhodneme vyberať jedincov na nahradenie. Ak budeme mať tendenciu vyberať na nahradenie jedincov s najmenšou vhodnosťou (napríklad pomocou

metódy turnajov, v ktorých je „víťaz“ ten s najmenšou vhodnosťou v turnaji), tak môžeme z populácie vytlačiť rozmanitosť ešte rýchlejšie. Častejšie sa však deje to, že jedinci určení na nahradenie sa vyberajú náhodným spôsobom, čo pomáha, aby sa generácia nezahltila len priveľmi vhodnými jedincami (keďže pri náhodnom výbere je väčšia šanca, že bude na nahradenie vybraný aj priveľmi vhodný jedinec ako pri výbere turnajom).

3. Analýza požiadaviek

Táto kapitola bude venovaná stručnému opisu požiadaviek na našu webovú aplikáciu a očakávanej funkcionalite.

3.1. Cieľ aplikácie

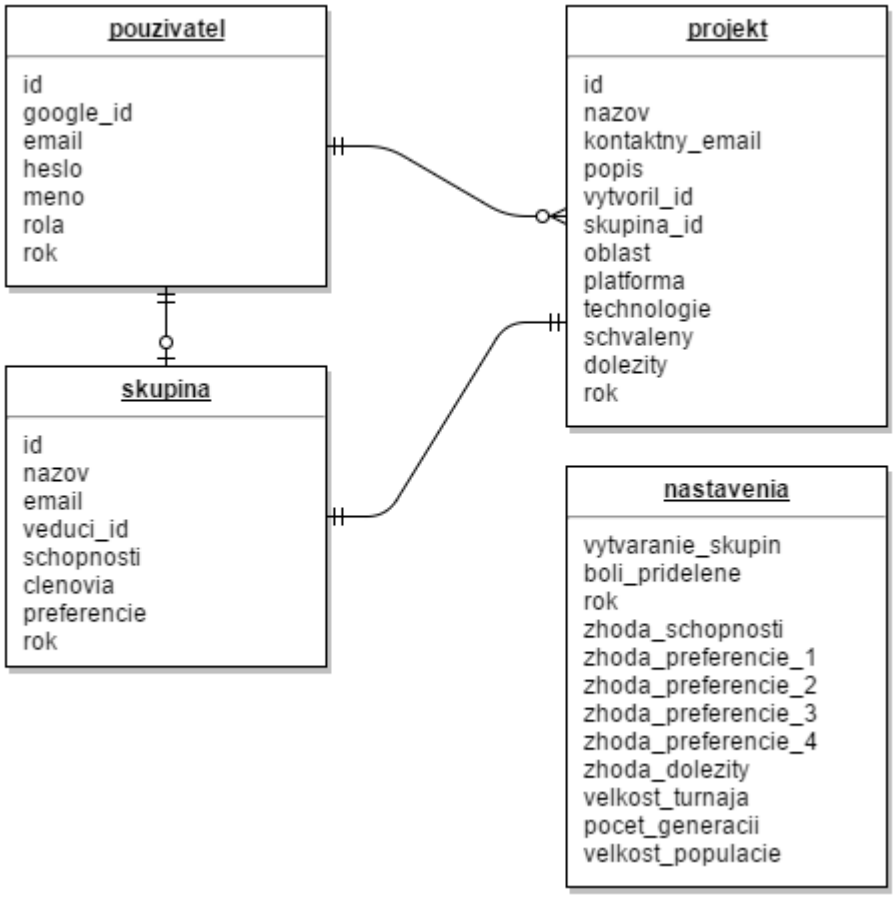
Cieľom aplikácie je uľahčiť a optimalizovať výber projektov riešiteľským skupinám. Táto aplikácia taktiež uľahčí zadávanie projektov a celkovú administráciu predmetu vo vyučovacom procese.

3.2. Entitno-relačný diagram

Entitno-relačný diagram je zobrazený na obrázku 5.

Keďže so systémom budú pracovať len autorizované osoby, je potrebné vytvoriť tabuľku *pouzivatel*, ktorá bude obsahovať záznamy o používateľoch. V systéme bude možno vytvárať skupiny (tímy) a projekty, teda je nutné vytvoriť aj tabuľky *skupina* a *projekt*. Medzi používateľom a projektom je vzťah *one-to-many(optional)*, pretože používateľ typu Zadavatel môže vytvárať viaceré projekty, no existujú aj používatelia, ktorí nevytvárajú žiadne projekty (títo sú typu Uzivatel a označujú obyčajných študentov). Vzťah medzi používateľom a skupinou je *one-to-one(optional)*, keďže každý používateľ má buď len jednu skupinu (študent) alebo nemá žiadnu skupinu (zadávateľ). Systém sa teda skladá z troch entitných množín:

1. pouzivatel: zoznam registrovaných používateľov
2. skupina: zoznam tímov
3. projekt: zoznam projektov
4. nastavenia: globálne nastavenia systému



Obrázok 5 Entitno-relačný diagram

Entitná množina *pouzivatel* má primárny kľúč id, ďalej nasledujú atribúty google_id, email, heslo, meno, rola a rok. Atribút google_id označuje identifikátor účtu Google, pokiaľ sa daný používateľ prihlásil pomocou Google účtu. Na základe atribútu rola budú mať rôzne typy používateľov rôzne možnosti v systéme – študenti si môžu označovať projekty na základe ich preferencií a vytvárať tímy, zadávatelia môžu do systému pridávať projekty. Každý projekt musí byť najskôr schválený používateľom typu administrátor, čo je vlastne vyučujúci predmetu.

Študentami vytvorené tímy zastrešuje entitná množina *skupina*, ktorá okrem iných obsahuje aj atribúty schopnosti (schopnosti tímu, na základe ktorých sa dá neskôr rozhodnúť, či je nejaký tím vhodnejší pre nejaký projekt ako druhý), clenovia (obsahuje zoznam členov daného tímu) a preferencie (kde sú uložené preferencie daného tímu pre rôzne projekty, čo neskôr taktiež pomáha systému pri prideľovaní projektov).

Entitná množina projekt, ktorá má aj tieto atribúty: oblast, platforma a technologie sú tie, ktoré taktiež pomáhajú pri prideľovaní projektov, atribút dolezity označuje, že projekt v danom roku – bez ohľadu na to, že systém môže na základe tímových preferencií,

schopností a vlastností projektu nájsť lepšie pridelenie projektov – musí byť pridelený nejakému tímu. Nakoniec atribút rok označuje, v ktorom roku bol projekt pridaný do systému.

Nakoniec entitná množina nastavenia, ktorá obsahuje nastavenia, ktoré môže admin meniť vo svojom rozhraní. Atribút vytvaranie_skupin nastavuje, do kedy je možné vytvárať a editovať študentské skupiny, boli_pridelene označuje, či už prebehlo automatické priradenie projektov, rok nastavuje aktuálny rok výučby, zhoda_schoposti, zhoda_preferencie_1, zhoda_preferencie_2, zhoda_preferencie_3, zhoda_preferencie_4 a zhoda_dolezity nám hovoria, koľko systém pripočíta v ohodnocovacej funkcii, v prípade, že nastane daná zhoda, velkost_turnaja označuje veľkosť použitého turnaja pri výbere v algoritme, pocet_generacii a velkost_populacie nastavujú požadovaný počet generácií a veľkosť populácie v priraďovacom algoritme.

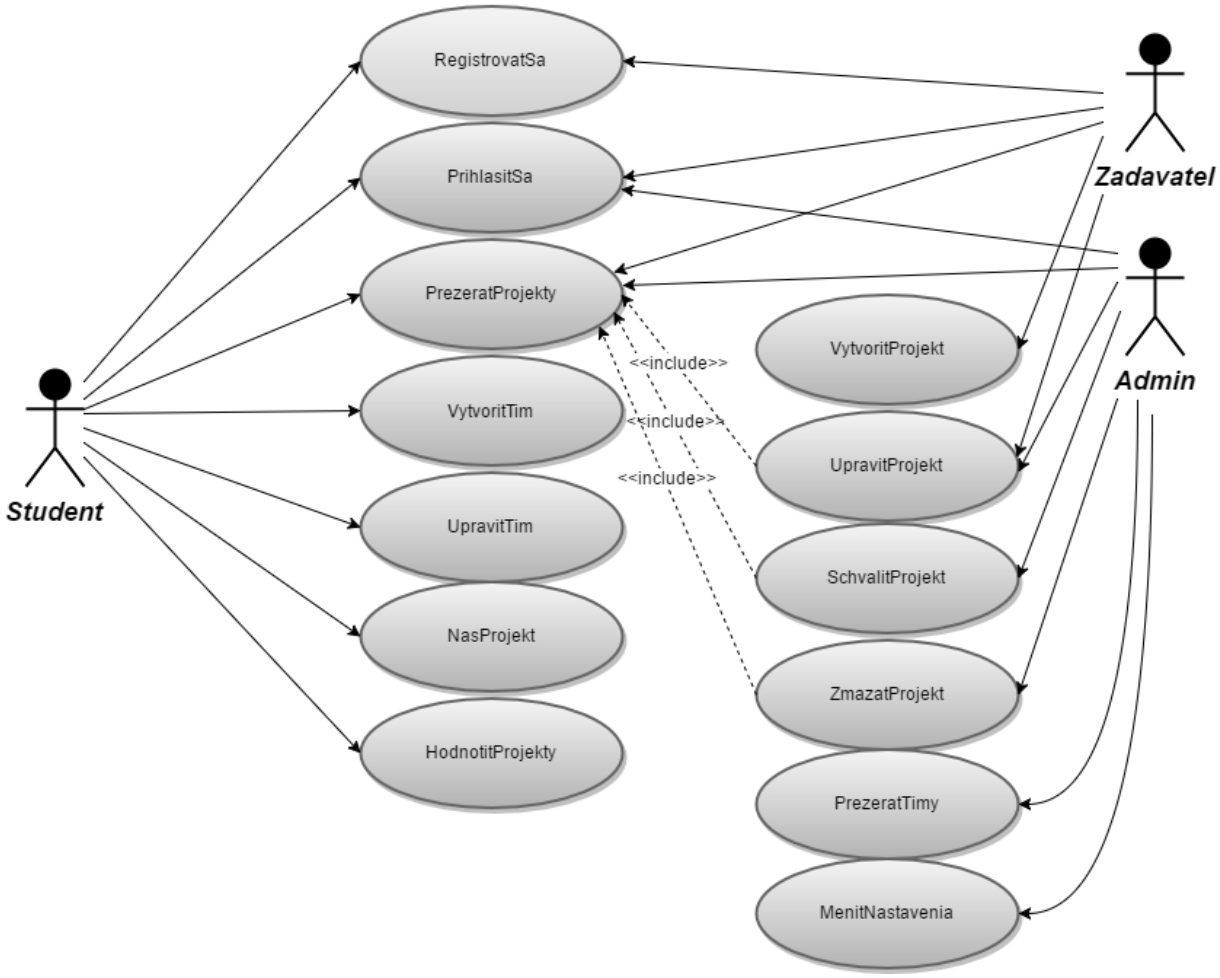
3.3. Use Case diagram

Jednotlivým rolám prislúchajú nasledovné Use Case (prípady použitia):

- Študent
    1. RegistrovaťSa
    2. PrihlásiťSa
    3. VytvoriťTím
    4. UpraviťTím
    5. PrezeraťProjekty
    6. HodnotiťProjekty
- Zadávateľ
    1. RegistrovaťSa
    2. PrihlásiťSa
    3. VytvoriťProjekt
    4. UpraviťProjekt
    5. PrezeraťProjekty
- Admin (vyučujúci)
    1. PrihlásiťSa
    2. VytvoriťProjekt
    3. UpraviťProjekt
    4. SchváliťProjekt

5. ZmazaťProjekt
6. PrezeraťTímy
7. MeniťNastavenia

Nasleduje podrobnejší popis jednotlivých Use Case-ov:



Obrázok 6 Use Case diagram

- RegistrovaťSa

  Po tom, čo si používateľ otvorí v prehliadači webovú stránku, bude mať na výber prihlásenie. Po kliknutí na odkaz *Vytvoriť užívateľa* sa mu zobrazí formulár, v ktorom si vyberie, či je zadávateľom a vyplní e-mailovú adresu a heslo. Potom tieto údaje potvrdí. Podľa údajov sa vykoná nasledovné:

  1. ak sa zadaná e-mailová adresa už používa, systém vypíše chybovú hlášku
  2. ak je niektorý z údajov v nesprávnom formáte, systém vypíše chybovú hlášku
  3. systém zaregistruje nového užívateľa.

- PrihlásiťSa

  Hneď ako používateľ navštívi webovú stránku bude mať pred sebou prihlasovací

formulár, v ktorom vyplní e-mailovú adresu, ktorú zadal pri registrácii a svoje heslo. Systém následne skontroluje, či sa v databáze daný používateľ nachádza a skontroluje heslo:

1. ak boli zadané nesprávne prihlasovacie údaje, systém vypíše chybovú hlášku
2. ak používateľ zadal správne prihlasovacie údaje, je pustený do systému s príslušnými právami, ktoré závisia od jeho roly.

- VytvoriťTím

  Používateľ v systéme klikne na *Administráciu skupiny*, kde následne klikne na *Vytvoriť*.

  1. ak už používateľ vytvoril tím, môže ho už len meniť a systém ho na to upozorní
  2. používateľ vo formulári vyplní príslušné údaje popisujúce nový tím a potvrdením takýto nový tím vytvorí.

- UpraviťTím

  Používateľ klikne na *Administráciu skupiny* a klikne na *Upraviť*.

  1. ak používateľ ešte nevytvoril tím, systém ho upozorní, že musí najskôr vytvoriť tím
  2. používateľ zmení požadované údaje o tíme a následným potvrdením ich uloží do systému.

- NášProjekt

  Ak používateľ klikne na *Administráciu skupiny* a potom na *Náš projekt*, tak v prípade, že už prebehlo automatické priradenie projektov, zobrazí projekt, ktorý bol danej skupine priradený.

- PrezeraťProjekty

  Po kliknutí na *Zoznam projektov* je požívateľovi prezentovaný zoznam projektov.

  1. ak používateľ nie je prihlásený, tak sa zobrazí len zoznam schválených projektov v danom roku
  2. ak je používateľ študentom, tak sa zobrazí len zoznam už schválených projektov, po kliknutí na projekt sa mu zobrazí stránka s detailnejším popisom projektu.
  3. ak je používateľ zadávateľom, môže si po kliknutí na *Zoznam projektov* vybrať, či chce zobraziť len jeho projekty (*Moje*) alebo chce vidieť zoznam všetkých projektov (*Všetky*), po kliknutí na jeho projekt ho môže editovať

4. ak je používateľ adminom, pribudne mu na výber k *Moje* a *Všetky* ešte položka *Na schválenie*, kde sa mu zobrazí zoznam ešte neschválených projektov a po kliknutí na projekt môže tento schváliť, prípadne upraviť ho.

- HodnotiťProjekty

  Po kliknutí na *Výber projektu* sa používateľovi zobrazí zoznam už schválených projektov, ktoré môže ohodnotiť podľa preferencie.

- VytvoriťProjekt

  Po kliknutí na *Vytvoriť projekt* sa zobrazí formulár, po ktorého vyplnení a potvrdení bude vytvorený nový projekt, ktorý čaká na schválenie adminom.

- UpraviťProjekt

  Po kliknutí na *Zoznam projektov* → *Všetky* a následnom kliknutí na projekt sa zobrazí stránka, na ktorej je možné daný projekt upraviť a po potvrdení sa zmeny uložia.

- SchváliťProjekt

  Ak admin klikne na *Zoznam projektov* → *Na schválenie*, zobrazí sa mu zoznam projektov, ktoré čakajú na schválenie. Po kliknutí na vybraný projekt ho možno schváliť.

- ZmazaťProjekt

  Ak admin klikne na ktorýkoľvek projekt v zozname projektov, má možnosť tento projekt zmazať.

- PrezeraťTímy

  Po kliknutí na *Zoznam tímov* si admin môže prezerať zoznam doteraz vytvorených tímov a ich členov.

- MeniťNastavenia

  Ak admin klikne na *Nastavenia*, tak môže meniť rôzne parametre algoritmu automatického priraďovania a systémové parametre.

4. Implementácia

V tejto kapitole sa budeme venovať implementácii algoritmu spomenutého v kapitole 2.3.2 (na obrázku 7 je vidno jeho pseudokód) a ukážeme, ako sme vytvárali systém s použitím technológií, ktoré sme opísali v kapitole 1.1.

4.1. Reprezentácia permutácie

Naše permutácie, ktoré predstavujú priradenia projektov ku skupinám budeme reprezentovať nasledovne: majme nejakú abecedu Σ, potom reťazec $(a_1, a_2, \dots, a_n)$ dĺžky $n = počet\_projektov$ predstavuje priradenia k projektom nasledovne. Poradie písmena v reťazci označuje poradie projektu, o ktorý sa jedná a znak na tomto mieste prestavuje skupinu. Poradie tejto skupiny získame ako poradie tohto znaku v abecede. Čiže napríklad reťazec "$dceab$" nám hovorí, že prvý projekt dostala štvrtá skupina, druhý projekt tretia, atď.

```
popsize = desired population size
P = {}
for popsize times do
      P = P ∪ {new random individual}
Best = null
for each individual Pi ∈ P do
      AssessFitness(Pi)
      if Best == null or Fitness(Pi) > Fitness(Best) then
            Best = Pi
repeat
      Parent Pa = SelectWithReplacement(P)
      Parent Pb = SelectWithReplacement(P)
      Children Ca, Cb = Crossover(Copy(Pa), Copy(Pb))
      Ca = Mutate(Ca)
      Cb = Mutate(Cb)
      AssessFitness(Ca)
      if Fitness(Ca) > Fitness(Best) then
            Best = Ca
      AssessFitness(Cb)
      if Fitness(Cb) > Fitness(Best) then
            Best = Cb
      Individual Pd = SelectForDeath(P)
      Individual Pe = SelectForDeath(P)      // Pd ≠ Pe
      P = P - {Pd, Pe}
      P = P ∪ {Ca, Cb}
until Best is the ideal solution or we have run out of time
return Best
```

Obrázok 7 Pseudokód Steady-State algoritmu

4.2. Inicializácia populácie

Populácia je na začiatku zaplnená náhodne vygenerovanými jedincami.

4.3. Operátor mutácie

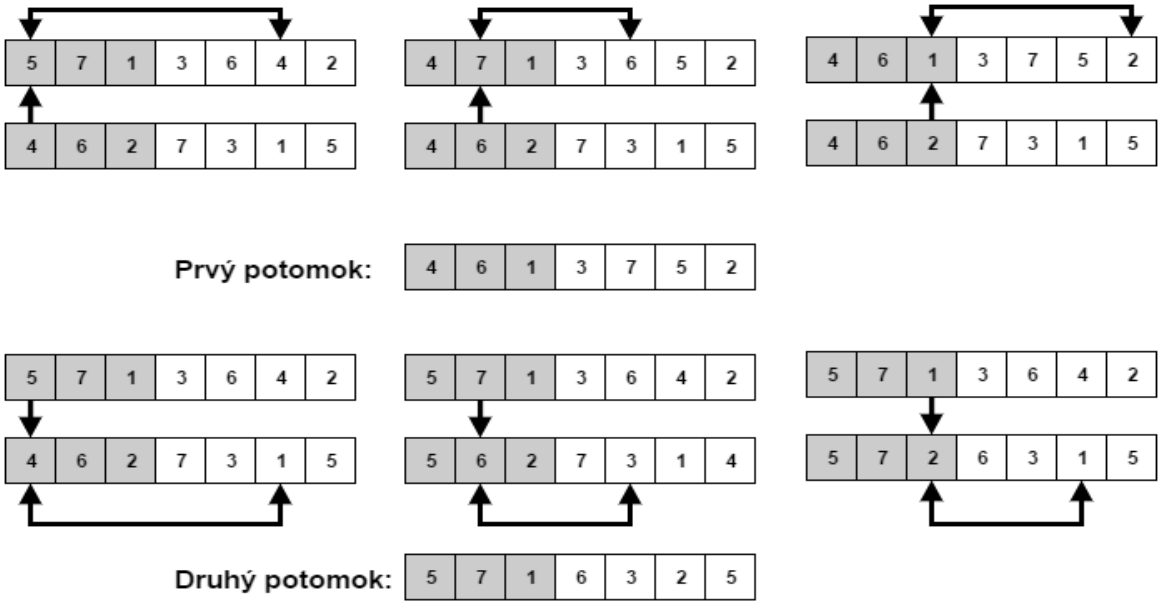
Na realizáciu mutácie jedinca nie je možné použiť jednoduchý bitový flip, preto je mutácia vyriešená výmenou náhodných dvoch prvkov vrámci jednej permutácie.

4.4. Výber rodičov pre kríženie

Rodičia sú pre potreby kríženia vyberaní spôsobom turnaja, pričom veľkosť turnaja je voliteľná. Jedinci sú do turnaja vyberaní náhodne a nesmú sa opakovať (pri náhodnom výbere môže nastať situácia, kedy by sa vybrali opakujúci sa jedinci – toto treba ošetriť).

4.5. Operátor kríženia

Keďže v našom prípade sa jedná o permutácie, nie je možné použiť štandardný operátor pre kríženie (či už one-point alebo two-point), keďže po krížení by mohla nastať situácia, kedy by sa v potomkovi nachádzali opakujúce sa gény – čo je neprípustné (pre nás by to znamenalo, že jednej skupine by boli pridelené dva projekty). Ako riešenie pripadá do úvahy napríklad Partially-Matched-Crossover (PMX). Postup je naznačený na obrázku 8 a jeho



Obrázok 8 Partially-Matched-Crossover

fungovanie si vysvetlíme teraz. Majme dvoch rodičov *s* a *t*, PMX náhodne vyberie miesto kríženia – podobne ako one-point kríženie. Potomka potom skonštruujeme nasledovne. Začneme s kópiou *s*, pozície medzi bodmi kríženia sú jeden po druhom nastavené na hodnoty *t* na týchto pozíciách. Keďže potrebujeme, aby daný reťazec zostal permutáciou, tak nemôžeme hodnoty na týchto pozíciách len prepisovať – pred tým, ako nastavíme hodnotu

v c (čo označuje potomka) si hodnota na pozícii p a c vymenia miesto. Na už spomenutom obrázku je znázornený tento postup pre permutácie 5, 7, 1, 3, 6, 4, 2 a 4, 6, 2, 7, 3, 1, 5. V našej implementácii však použijeme metódu popísanú v [11]. Táto používa na popis permutácie jej inverzie. Majme permutáciu $i_1, i_2, \dots, i_n$ množiny $\{1,2, \dots N\}$, nech a_j označuje počet celých čísel v permutácii, ktoré sú pred j, no sú väčšie ako j. Potom a_j vyjadruje ako veľmi je j mimo poradia. Postupnosť čísiel $a_1, a_2, \dots, a_n$ nazývame *inverznou postupnosťou* permutácie $i_1, i_2, \dots, i_n$. Napríklad inverznou postupnosťou permutácie 4, 6, 2, 7, 3, 1, 5, je 5, 2, 3, 0, 2, 0, 0. Tu napríklad 2, ktorá je piatym prvkom v inverznej postupnosti vyjadruje, že existujú presne dva prvky v permutácii, ktoré sú vľavo od 5 a sú väčšie ako 5 (v tomto prípade ide o čísla 6 a 7). Inverzná postupnosť $a_1, a_2, \dots, a_N$ spĺňa podmienku $0 \leq a_i \leq N - i$ pre $i = 1,2, \dots, N$. Ako vidíme, tak nie je žiadne obmedzenie pre prvky, ktoré by hovorilo, že $a_i = a_j$ nie je povolené pre $i \neq j$. Nasledujú pseudokódy, kde prvý popisuje ako vygenerovať inverznú postupnosť z danej permutácie a druhý popisuje opačný postup, teda ako z inverznej postupnosti získame naspäť permutáciu.

```
for i = 1…N do
        inv_i = 0
        m = 0
        while perm_m ≠ i do
                if  perm_m > i then inv_i = inv_i + 1
                m = m + 1
```

Obrázok 9 Pseudokód pre získanie inverzie

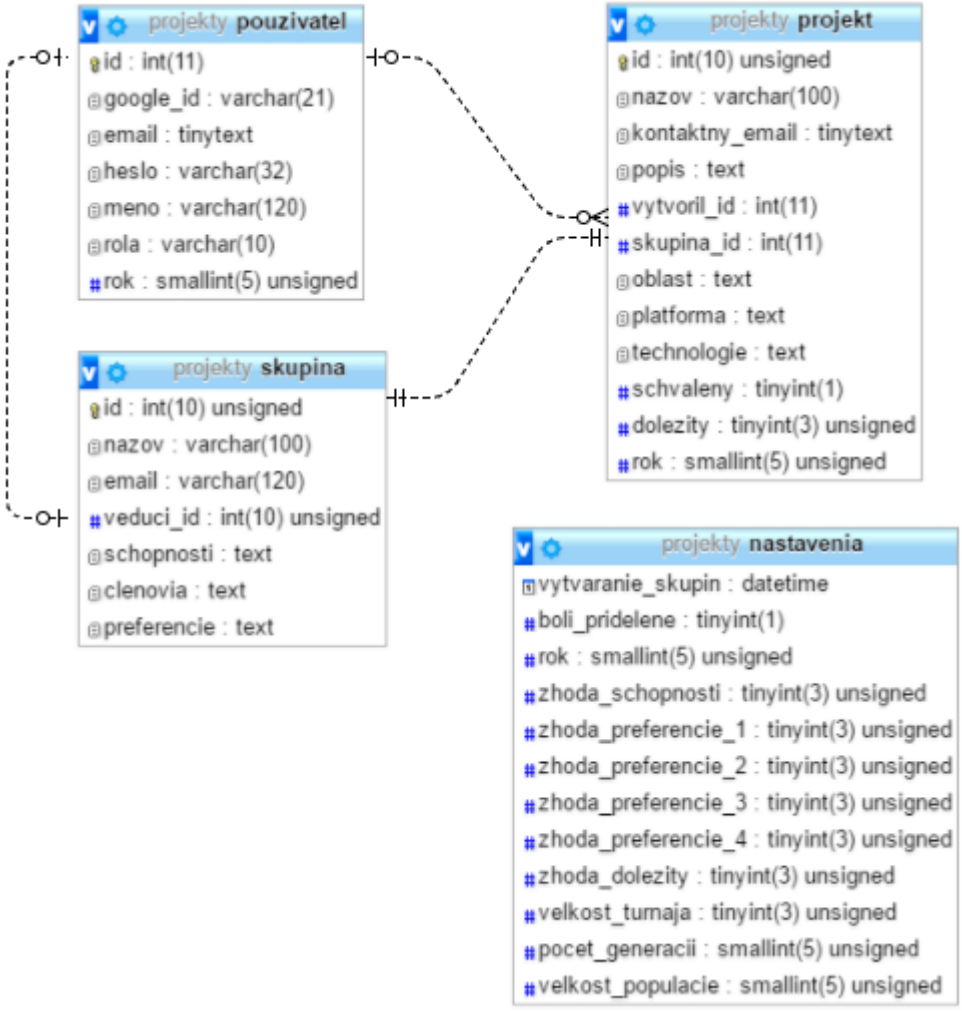
V kóde na obrázku 9 je vstupom pole $perm$, ktoré obsahuje samotnú permutáciu a výstupom je pole inv, obsahujúce inverznú postupnosť.

```
for i = N..1 do
        for m = i + 1..N do
                if pos_m ≥ inv_i + 1 then pos_m = pos_m + 1
                pos_i = inv_i + 1
for i = 1..N do perm_{pos_i} = i
```

Obrázok 10 Pseudokód pre získanie permutácie z inverzie

V kóde na obrázku 10 je vstupom pole inv obsahujúce inverznú postupnosť a výstupom pole $perm$, ktoré obsahuje permutáciu. Taktiež je použité pomocné pole pos, ktoré obsahuje medzivýsledok (je možné napísať kód, ktorý by takéto pomocné pole nepoužíval, avšak bol by zložitejší a časovú zložitosť by aj tak neznížil).

4.6. Databázový dátový model



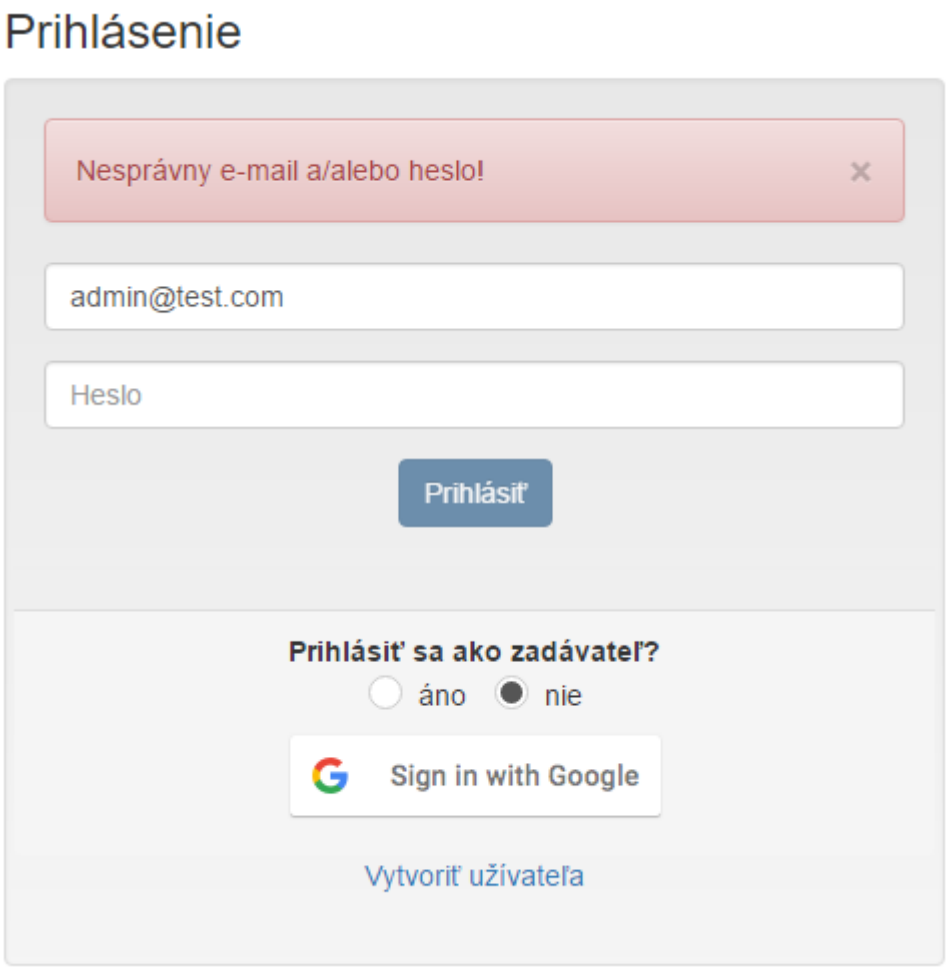
Obrázok 11 Databázový dátový model

4.7. Všeobecná funkcionalita

Tu spomenieme funkcionality, ktoré sú k dispozícii všetkým používateľom – teda ide najmä o prihlasovanie/odhlasovanie sa do systému a registrácia nového používateľa.

4.7.1. Prihlasovanie

O prihlasovanie do systému sa stará controller *Login*, ktorý používa metódy controllera *API* a modelu *User*. Používateľ si môže vybrať, či sa chce prihlásiť klasickým spôsobom (teda si najskôr vytvorí účet pomocou registrácie) alebo použije prihlásenie pomocou účtu Google. Na obrázku 12 môžeme vidieť ako vyzerá formulár v prípade, že sa užívateľ pokúsil prihlásiť so zlým e-mailom a/alebo heslom.

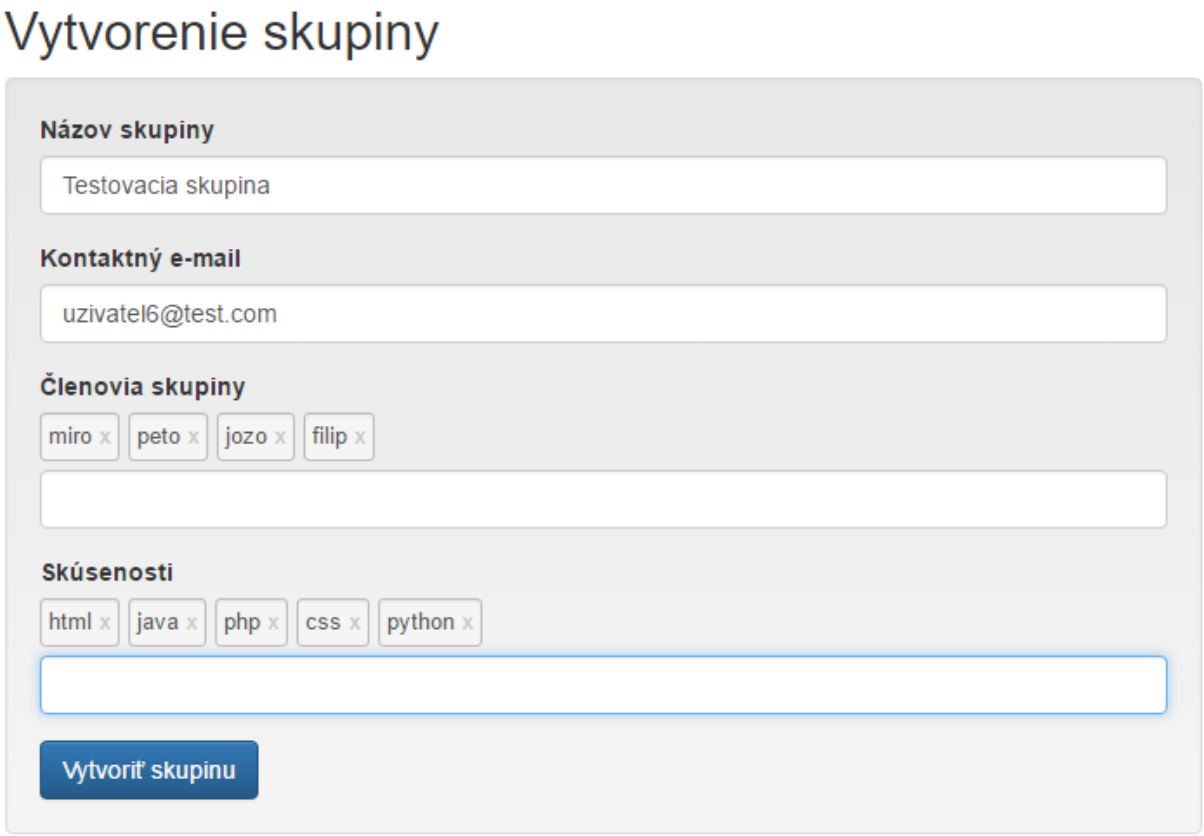


Obrázok 12 Prihlasovací formulár

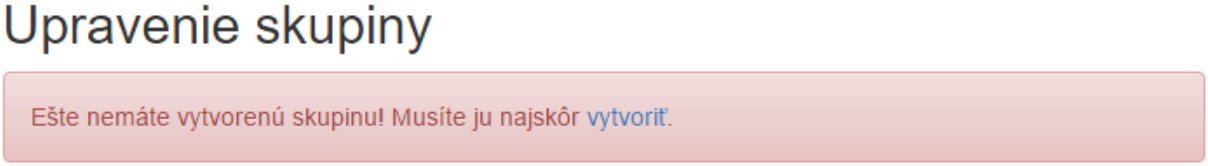
4.7.2. Vytváranie a editácia skupín

Vytváranie a editáciu skupín popisuje controller *Groups*. V prípade, že sa ešte dajú vytvárať a editovať skupiny (dátum, do kedy sa tak dá urobiť nastavuje admin) si môže používateľ vytvoriť skupinu (ak ešte nemá), prípadne ak už má vytvorenú skupinu, tak ju môže editovať.

Na obrázku 13 vidíme rozhranie, ktoré umožňuje vytvoriť/editovať skupinu. V prípade, že by chcel užívateľ spraviť nejakú nepovolenú akciu, čo sa týka skupiny (vytváranie/editácia mimo povoleného dátumu, pokus o editáciu skupiny, aj keď ešte žiadnu skupinu nemá), tak je na toto upozornený (viď. obrázok 14).



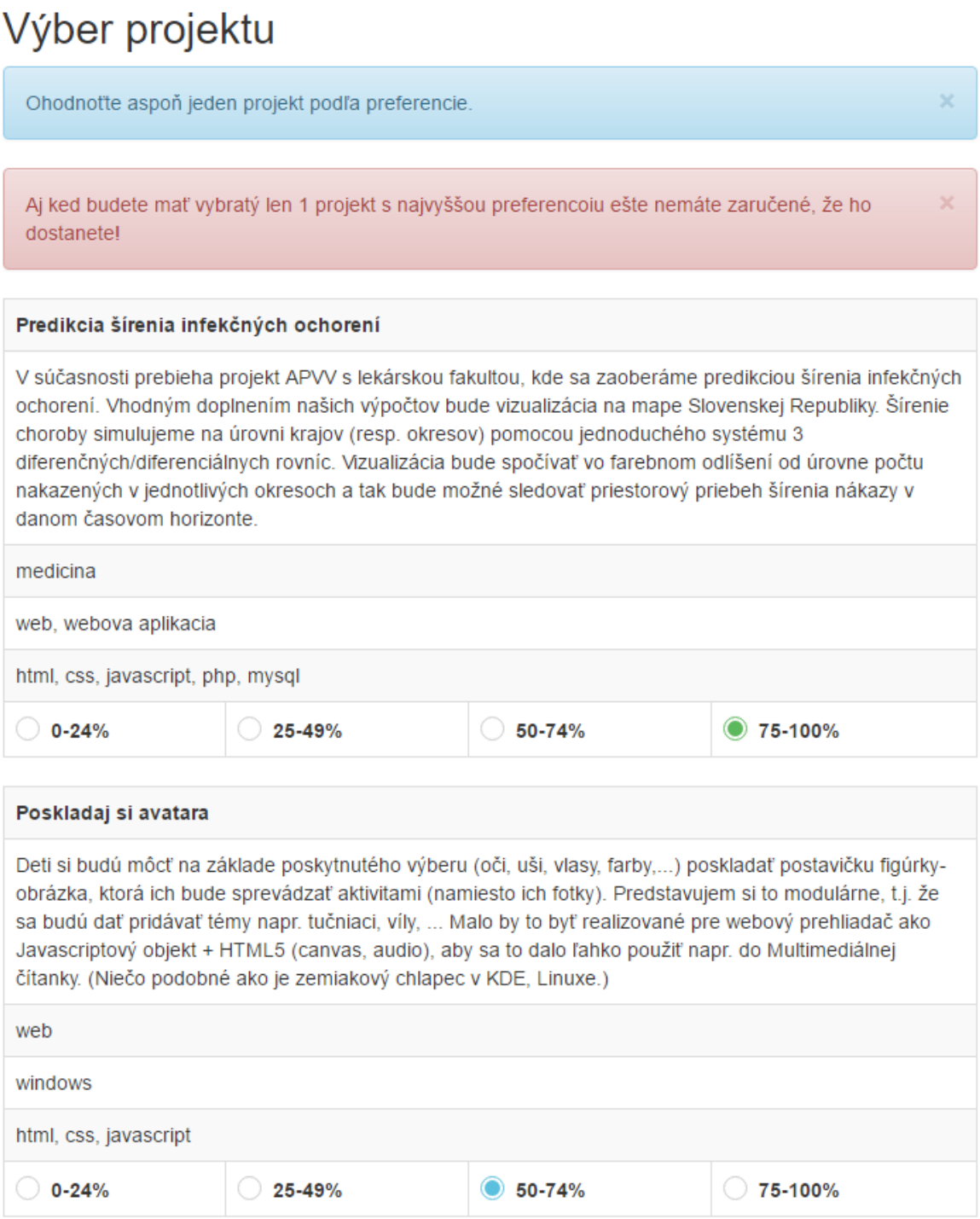
Obrázok 13 Formulár na vytvorenie skupiny



Obrázok 14 Chybová hláška - editácia skupiny

4.7.3. Výber projektu

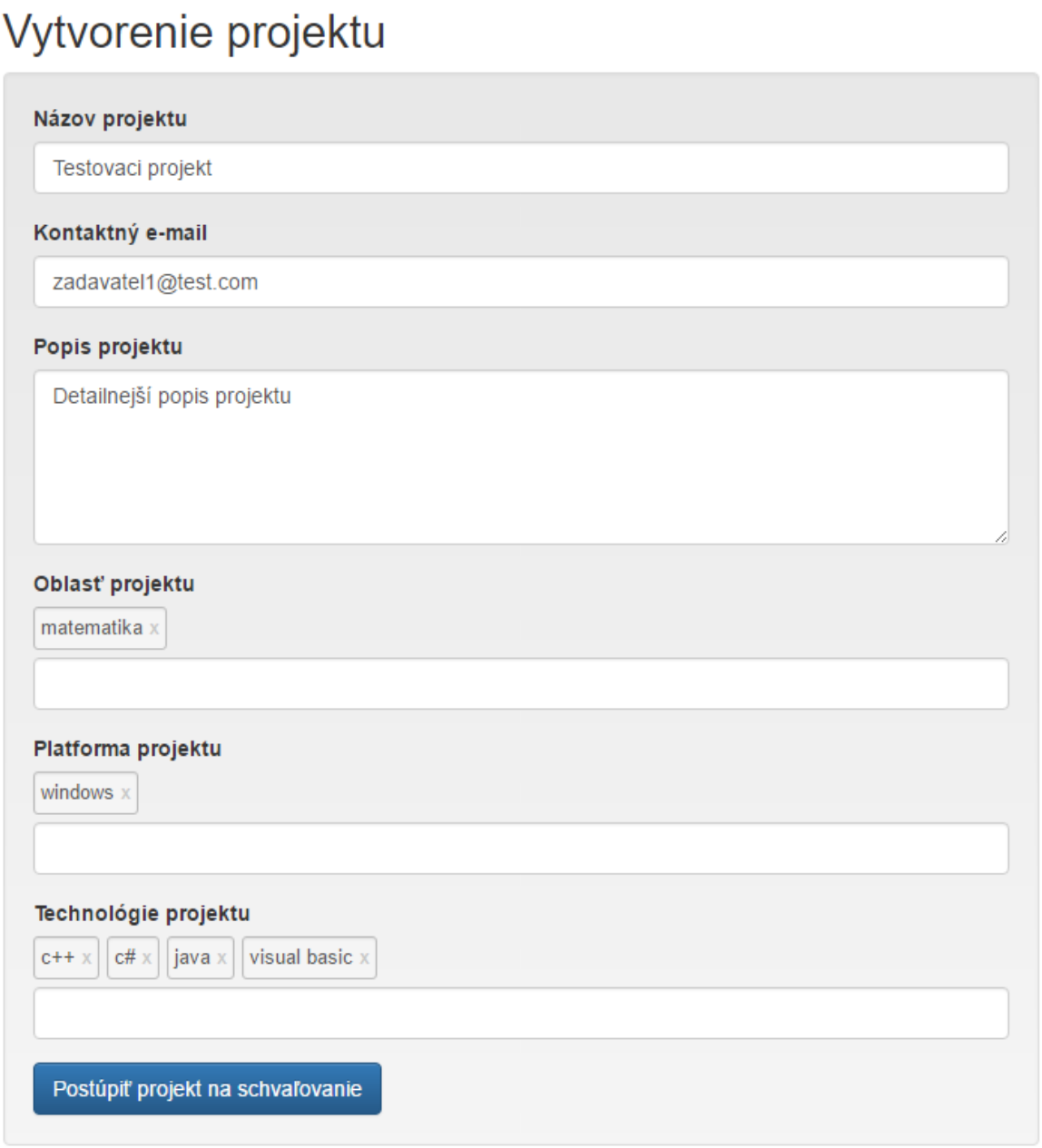
O označovanie preferencií pre jednotlivé projekty pre skupinu sa stará controller *Select*, ktorý používa metódy modelu *User*, *Group* a *Project*. V prípade, že by si chcel používateľ vyberať projekty bez vytvorenej skupiny, je na to upozornený a systém mu nedovolí pokračovať. Na obrázku 15 je vidno, ako si skupina môže jednoducho podľa preferencie ohodnocovať projekty.



Obrázok 15 Výber projektu - zadávanie preferencií

4.7.4. Vytváranie a editácia projektu.

Vytváranie a editáciu projektu má na starosti controller *Create_project*, ktorý používa model *User* a používa metódy controllera *API*. Po vyplnení formulára sa projekt uloží do databázy a čaká na schválenie adminom. Príklad formulára je vidno na obrázku 16.

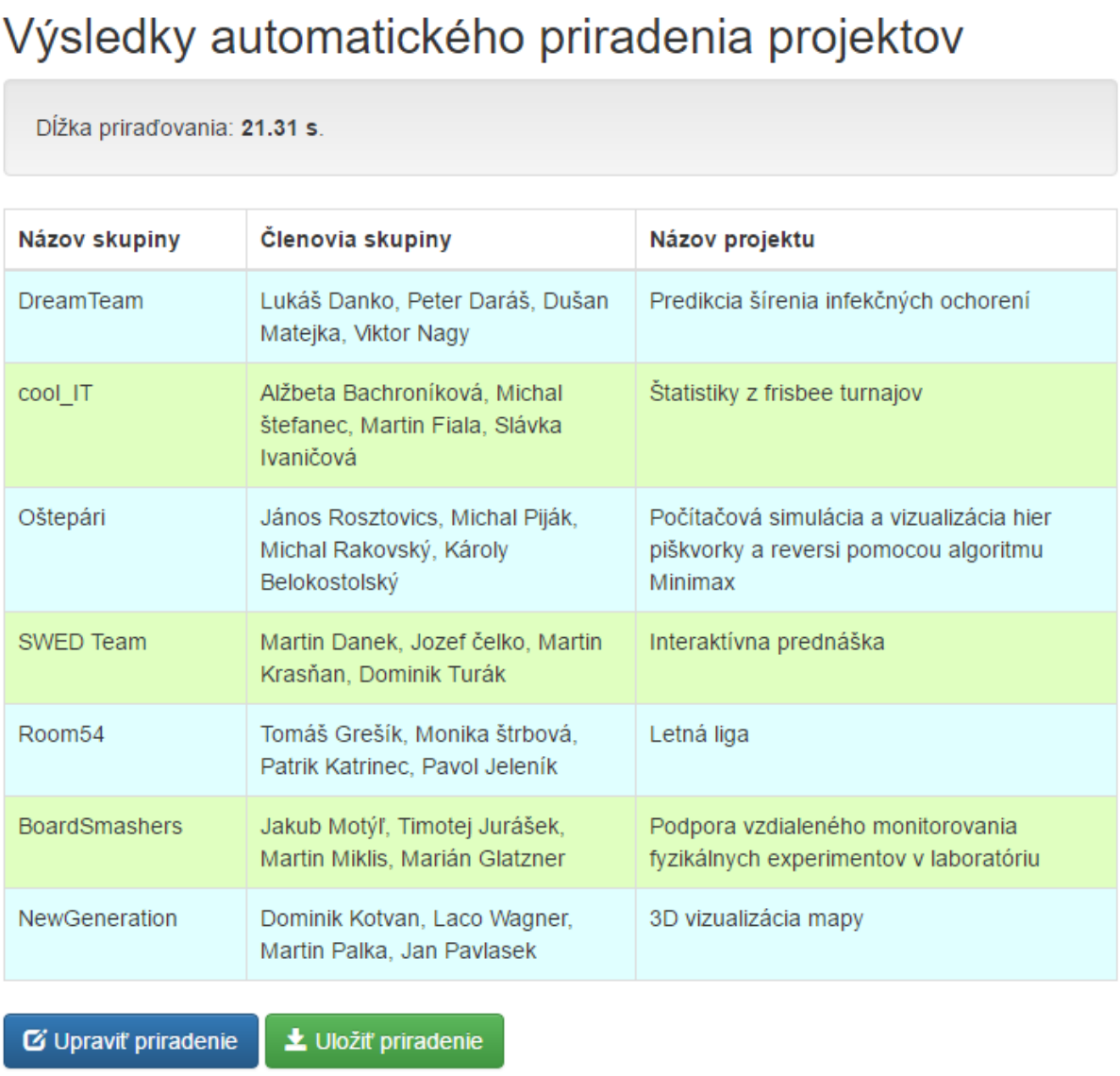


Obrázok 16 Formulár na vytvorenie projektu

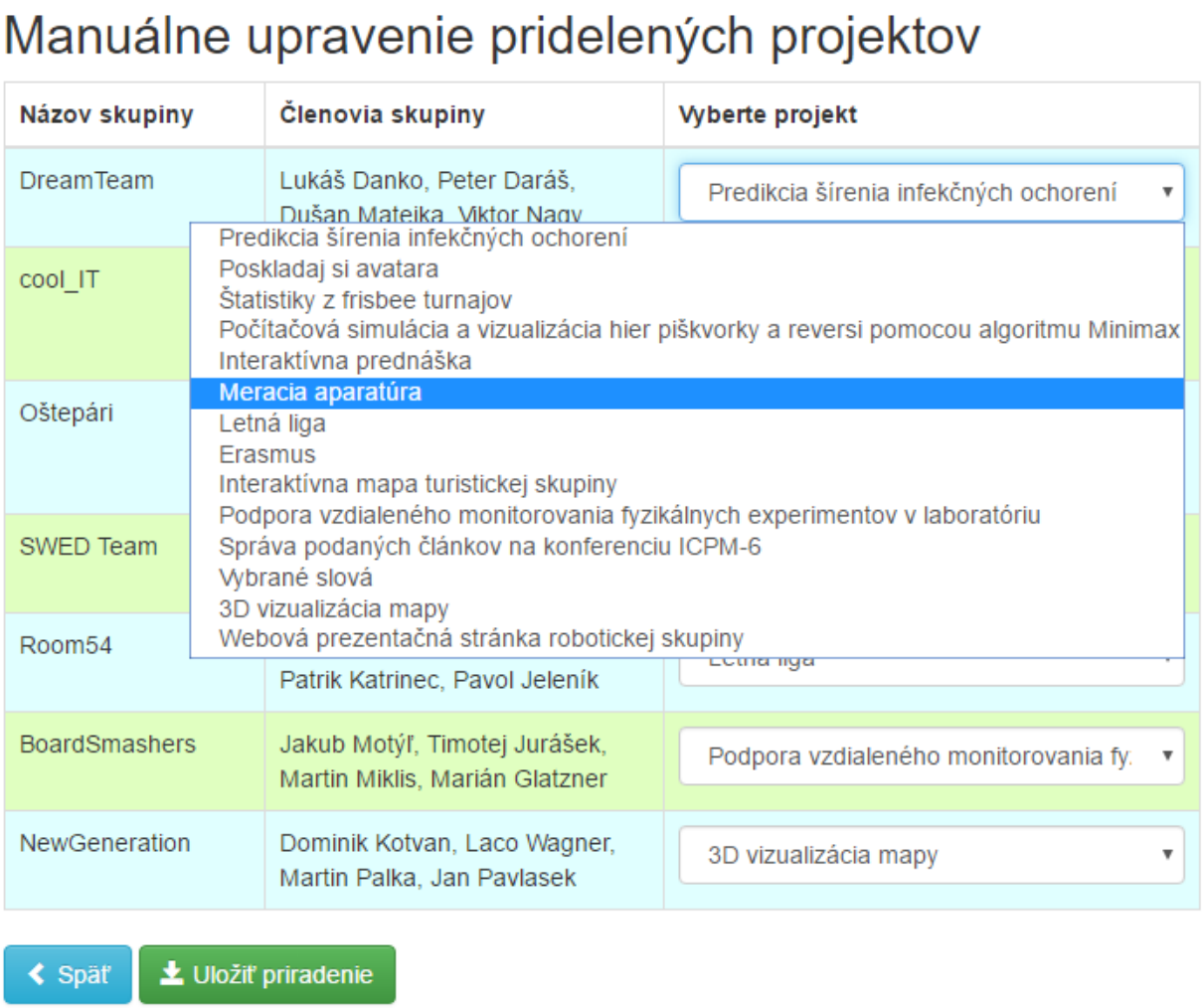
4.7.5. Automatické priradenie projektov

Admin, po tom, čo si študenti vytvoria skupiny a pridajú sa do nich, môže spustiť automatické priradenie projektov. Toto priradenie používa algoritmus spomenutý v časti 2.3.2, ktorý je neskôr detailnejšie popísaný v časti 4. Tento na základe požiadaviek projektu, schopností skupín a ich preferencií navrhne priradenie projektov ku skupinám. Admin si môže tieto výsledky pozrieť a v prípade potreby toto priradenie ešte meniť. O celý tento proces sa stará controller *Selection*, ktorý používa modely *User*, *Project* a *Group* a taktiež používa metódy zo súboru *GA.php*, ktorý obsahuje implementáciu Steady-State genetického algoritmu. Nižšie sú obrázky, ktoré ukazujú, ako takéto priradenie vyzerá – obrázok 17

zobrazuje navrhované priradenie projektov a obrázok 18 zasa editáciu navrhovaného priradenia. V prípade, že by sa admin snažil priradiť projekt viacerým skupinám, je na túto skutočnosť upozornený a systém mu nedovolí takéto priradenie uložiť. Po uložení systém rozpošle automaticky e-maily skupinám s ich priradenými projektami a zadávateľom e-maily o tom, ktorý ich projekt bol priradený akej skupine. Príklad takéhoto e-mailu je na obrázku 19.



Obrázok 17 Navrhované priradenie projektov



Obrázok 18 Úprava priradenia

```
Tato sprava bola vygenerovana automaticky.

Vas projekt "Podpora vzdialeneho monitorovania fyzikalnych
experimentov v laboratoriu" bol uspesne priradeny.
Projekt dostala skupina "BoardSmashers", ktoru je mozne kontaktovat na
emailovej adrese: mato.meciar@gmail.com

Dakujem.
```

Obrázok 19 Text e-mailu pre zadávateľa

5. Záver

Cieľom tejto bakalárskej práce bolo vytvoriť webovú aplikáciu na výber projektov. Po preskúmaní už existujúcich riešení a skúmaní dostupných algoritmov, ktoré by bolo možné použiť na riešenia nášho problému sme vybrané riešenie aplikovali na náš problém. Implementovali sme všetku požadovanú funkcionalitu, takže sa nám cieľ podarilo splniť.

Okrem toho, že sme vytvorili samotný systém bolo aj výzvou oboznamovanie sa evolučnými algoritmami a ich fungovaním. Na rozdiel od pôvodného rozhodnutia nepoužiť žiaden framework na implementáciu systému by sme v prípade vytvárania podobného systému siahli po niektorom z dostupných frameworkov (je vysoko pravdepodobné, že čas strávený učením sa práce s takýmto frameworkom by bol menší ako nutnosť všetko implementovať od základov).

Ako ďalšie možné rozšírenia systému by som považoval implementáciu viacerých algoritmov na výber priradení a možnosť administrátora sa rozhodnúť, ktoré použije. Taktiež by bolo dobré doimplementovať detailné logy z priebehu samotného priraďovacieho algoritmu.

6. Zdroje

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad a M. Stal, Pattern-Oriented Software Architecture, Volume 1, A System of Patterns, Chichester: John Wiley & Sons, Ltd., 1996.

[2] P. Petrovič, „Prirodzený život sa skončil," rev. *Kognice a umělý život*, Praha, 2012.

[3] „Buddy Programme," 20 Máj 2016. [Online]. Available: http://buddy.trondheim.esn.no/#section-signup. [Cit. 27 Máj 2106].

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest a C. Stein, Introduction to Algorithms, Third Edition ed., MIT Press, 2009.

[5] A. D. Marshall, „Steepest Ascent Hill Climbing," [Online]. Available: http://www.cs.cf.ac.uk/Dave/AI1/steep.html. [Cit. Apríl 2016].

[6] A. E. Eiben a J. E. Smith, Introduction to Evolutionary Computing, Springer, 2003.

[7] G. Jones, „Genetic and Evolutionary Algorithms," [Online]. Available: http://www.wiley.com/legacy/wileychi/ecc/samples/sample10.pdf. [Cit. Apríl 2016].

[8] A. Ter-Sarkisov a S. Marsland, „Convergence Properties of (μ + λ) Evolutionary Algorithms," rev. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*.

[9] R. C. Purshouse a P. J. Fleming, „Why use Elitism and Sharing in a Multi-Objective Genetic Algorithm?," [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.9699&rep=rep1&type=pdf. [Cit. 12 Marec 2016].

[10] G. J. E. Rawlins, Foundations of Genetic Algorithms, San Mateo: Morgan Kaufmann Publishers, 1991.

[11] G. Üçoluk, „Genetic Algorithm Solution of the TSP," [Online]. Available: http://www.ceng.metu.edu.tr/~ucoluk/research/publications/tspnew.pdf. [Cit. 11 Máj 2016].

[12] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Boston: Addison-Wesley Longman Publishing Co., Inc., 1989.

[13] M. Mitchell, J. H. Holland a S. Forrest, „When Will a Genetic Algorithm Outperform," [Online]. Available: http://web.cecs.pdx.edu/~mm/nips93.pdf. [Cit. 22 Marec 2016].

[14] A. W. Moore, „Iterative Improvement, Hill Climbing, Simulated Annealing, WALKSAT, and Genetic Algorithms," Pittsburgh.

[15] A. Marczyk, „Genetic Algorithms and Evolutionary Computation," 23 Apríl 2004. [Online]. Available: http://www.talkorigins.org/faqs/genalg/genalg.html. [Cit. 16 Apríl 2016].

[16] Mausam, „Local Search and Optimization, Chapter 4," [Online]. Available: https://courses.cs.washington.edu/courses/csep573/11wi/lectures/04-lsearch.pdf. [Cit. 7 Marec 2016].

[17] D. Simon, Evolutionary Optimization Algorithms: Biologically-Inspired and Population-Based Approaches to Computer Intelligence, Wiley, 2013.