Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interaktívny portál na vyučovanie dynamického programovania

Bakalárska práca

2016
Michal Smolík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interaktívny portál na vyučovanie dynamického programovania

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Michal Forišek, PhD

Bratislava, 2016
Michal Smolík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov:

Cieľ:

Literatúra:

Kľúčové slová:

Vedúci:
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania:

Dátum schválenia: doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie:

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Úvod

Dynamické programovanie je pre veľa začínajúcich programátorov zastrašujúce, pretože vyžaduje zmenu spôsobu rozmýšľania o probléme.

Preto vytvárame miesto, kde sa žiaci môžu aj zo svojho domova učiť o základných princípoch a konkrétnych využitiach rekurzívnych výpočtov, memoizácie a dynamického programovania interaktívnym spôsobom. Úlohou portálu je poskytnúť prostredie, kde sa používatelia môžu od základov naučiť riešiť úlohy na dynamické programovanie bez účasti učiteľa.

Koncentrujeme sa hlavne na poskytovanie priestoru pre aktívnu výučbu, pretože programovanie sa najlepšie učí skúšaním. Preto sa výučba na našom portáli zameriava na zadávanie a automatické testovanie implementačných úloh. Ďalšia z hlavných funkcií portálu je nteraktívna vizualizácia rekurzívnych výpočtov a memoizácie, ktorej úlohou je prehľadným spôsobom ukázať túto niekedy náročnú, ale mimoriadne dôležitú časť programovania.

Aby sa zaistila vysoká kvalita vyučovania a používatelia ostali motivovaní, zbierame dáta o ich postupe výučbou a zároveň im dávame možnosť nechať spätnú väzbu na každú úlohu ktorú správne vyriešili. Tým zaistíme, že administrátor vie o tom, ktorá úloha je populárna, ktorá nie a ktoré časti výčbového procesu treba vylepšiť.

Je dôležité spomenúť, že cieľom tejto práce je iba implementácia funkcií portálu, nie výčbového obsahu. Preto budeme rozprávať iba o návrhu a konkrétnej implementácii portálu, nie o výučbe ako takej.

Táto práca má štyri hlavné časti, v ktorých si povieme o návrhoch prednej aj zadnej strany nášho portálu a potom aj o ich konkrétnych implementáciách v prvej pracovnej verzii portálu.

Kapitola 1

Výučba

V tejto kapitole si predstavíme používateľské prostredie a ako bude výučba vyzerať z pohľadu používateľa.

1.1 Proces výučby

1.1.1 Postup úlohami

Hlavný spôsob výučby poskytovaný naším portálom je zadávanie implementačných úloh, ktoré používateľ na svojom zariadení vyrieši a pošle nám riešenie ktorého správnosť overíme a ďalšie správanie portálu zácisí na správnosti riešenia (podľa implementácie testovača (2.2) sa pod riešením môže rozumieť samotný kód alebo archivované súbory s výstupmi).

Úlohy sú rozdelené na povinné a nepovinné a zoradené do rôznych úrovní. Na každej úrovni je jedna povinná a ľubovoľný počet nepovinných úloh. Ak používateľ správne vyrieši povinnú úlohu, jeho úroveň sa zvýši o 1 a odomknú sa mu nové úlohy. Nepovinné úlohy neodomykajú nové úrovne, ale rátajú sa používateľovi do jeho skóre (1.1.3).

1.1.2 Rady

Ak si používateľ nevie rady s úlohou, bude mať možnosť požiadať o radu. Každá úloha má ľubovoľne veľa rád - podľa uváženia administrátora. Používateľ pri začatí riešenia úlohy nemá k dispozícii žiadnu radu, ale môže o ňu kedykoľvek požiadať tlačidlom na stránke úlohy, ktoré mu sprístupní jednu ďalšiu radu. Potom sa mu vždy pri zobrazení úlohy ukážu aj všetky rady ku ktorým má prístup.

1.1.3 Bodovanie a rebríček

Každý používateľ má úroveň a skóre ktoré ukazujú jeho postup učebným procesom.

Úroveň používateľa je jednoducho najvyššia úroveň úloh ktoré sa podarilo používateľovi odomknúť

Skóre je o niečo zložitejšie: všetky správne vyriešené úlohy (povinné aj nepovinné) pridajú používateľovi do skóre svoju úroveň, teda prvá úloha sa ráta za 1 bod a úloha na vyššej, povedzme piatej úrovni pridá 5 bodov.

Používatelia majú aj možnosť pozrieť si svoje poradie v rebríčku a porovnať sa s ostatnými používateľmi. Sú dva rôzne rebríčky: rebríček úrovní a rebríček skóre.

Každý používateľ vidí v rebríčkoch seba a všetkých ostatných používateľov ktorí súhlasia so zverejnením svojho skóre (používateľ môže tento súhlas vo svojom profile kedykoľvek zmeniť).

K rebríčkom má prístup aj používateľ ktorý nesúhlasí so zverejnením svojej úrovne a skóre. Takýto používateľ stále vidí svoje poradie v porovnaní s ostatnými používateľmi tak isto ako ostatní používatelia. Jediná zmena je, že sa nezobrazuje v rebríčkoch ostatným používateľom.

1.1.4 Hodnotenie úloh

Používatelia mau možnosť po správnom vyriešení úlohu ohodnotiť.

Budú mať možnosť (ale nie povinnosť) ohodnotiť jej obtiažnosť a zaujímavosť na stupnici od 1 do 5. Tieto hodnotenia sa budú priemerovať a ich priemery budú prístupné iba administrátorovi. Každé hodnotenie bude prístupné iba používateľovi ktorý ho vytvoril a administrátorovi.

1.1.5 Diskusia

Každá úloha má diskusiu, v ktorej je možné rozoberať úlohu. V momentálnom návrhu je celá diskusia prístupná všetkým používateľom ktorí majú prístup k úlohe, z čoho ale môže vzniknúť problém prezrádzania správnych riešenie ostatným používateľom, ktorí ešte úlohu nevyriešili

Ak tento problém vznikne, budeme musieť rozdeliť diskusiu na dve: pre tých používateľov, čo úlohu vyriešili a pre tých čo ešte nie.

Ak budú stále pretrvávať problémy (napríklad používateľ prezradí riešenie skôr ako ho odovzdá na testovanie), môžme povoliť diskusiu iba pre úspešných riešiteľov.

Toto riešenie ale môže byť v rozpore s účelom diskusie, nakoľko sa môže stať, že používatelia sa po vyriešení úlohy o ňu prestanú zaujímať. Preto riešenie tohto problému

zatiaľ necháme na administrátorovi, aby zmazal všetky diskusné príspevky s radami, ktoré príliš zjednodušujú úlohu a iným nevhodným obsahom.

1.2 Prostredie

Predstavíme jednotlivé stránky ku ktorým má používateľ prístup a aké funkcie ktoré poskytujú

1.2.1 Základné stránky

Portál obsahuje stránky na registráciu nových používateľov, prihlásenie, zoznam úloh a používateľov profil ktoré sú jednoduché a ich funkcia evidentná, preto ich iba spomenieme bez podrobného popisu.

1.2.2 Detail úlohy

Stránka ktorá zobrazuje jednu úlohu: jej názov, zadanie, rady o ktoré používateľ požiadal, hodnotenie jeho riešení generované testovačom a nasledujúce možnosti:

- stiahnuť vstupné dáta alebo testovací skript - poskytnutie tejto možnosti závisí od implementácie testovača (2.2)
- odovzdať riešenie na otestovanie
- po správnom vyriešení úlohy ju ohodnotiť (1.1.4)
- zobraziť diskusiu a pridať komentár
- požiadať o ďalšiu radu

1.2.3 Vizualizácia rekurzívnych výpočtov

Jedna z najťažších častí učenia sa dynamického programovania je pochopenie rekurzívnych funkcií. Preto náš portál bude poskytovať možnosť ako si v prehľadnej forme zobraziť výpočet ľubovoľnej rekurzívnej funkcie.

Na stránke vizualizácie si používateľ bude môcť napísať svoju funkciu, ktorej stránka vytvorí a zobrazí strom výpočtu. Každý vrchol bude jedno zavolanie funkcie a bude ukazovať všetky argumenty.

Vrcholy budú vykresľované postupne, podľa poradia zavolania. Návratová hodnota zavolania sa zobrazí hneď, ak funkcia samú seba ďalej nevolá, alebo až keď všetci potomkovia jej vrchola v strome majú zobrazenú hodnotu (aby používateľ lepšie videl, ako postupoval výpočet a v ktorom momente boli spočítané ktoré návratové hodnoty).

Vizualizácia bude vedieť pracovať aj s memoizáciou, čiže pamätaním si návratových hodnôt pre argumenty a odpovedanie z pamäte pri každom ďalšom zavolaní funkcie s rovnakými argumentmi. Pri výpočte s memoizáciou budeme zobrazovať aj zoznam argumentov a ich hodnôt, ktoré už boli vypočítané V prípade vykreslenia vrcholu s argumentmi, pre ktoré hodnotu už poznáme, ukážeme aj na vrchol kde bola hodnota prvý krát vypočítaná.

Aj pre túto stránku si uvedieme možnosti, ktoré ponúka používateľovi:

- vybrať si, či výpočet bude používať memoizáciu alebo nie
- vyhodnotiť zadanú funkciu a spustiť vizualizáciu jej výpočtu
- zastaviť/znova spoustiť vizualizáciu
- krokovať vizualizáciu dopredu alebo dozadu (pri krokovaní vizualizáciu automaticky zastavíme)
- vykresliť celý strom výpočtu alebo začať vizualizáciu odznova

Uprednostníme počítanie a vytvorenie stromu pred vykreslením oproti počítaniu počas vykresľovania kvôli jednoduchšej implementácii počítania, ale aj vykresľovania samotného.

Je možné implementovať aj spoluprácu s úlohami, kde by sa úloha mohla odkazovať na konkrétny vstup vizualizácie, napríklad formou linku ktorý používateľa presmeruje na stránku vizualizácie kde bude zadaný potrebný kód.

V momentálnom návrhu ale pre túto funkciu nie je potreba, pretože rovnaký efekt docielime jednoducho napísaním potrebného kódu, ktorý používateľovi odporučíme skopírovať a vyhodnotiť vo vizualizácii manuálne. Oba spôsoby docielia rovnaký výsledok a automatické presmerovanie ušetrí iba málo používateľovho času takže ho považujeme za nepotrebné.

Ďalšia možnosť pre budúci vývoj je ukázať iba strom výpočtu bez kódu, čo sa dá využiť ako rada k úlohe alebo úplne nový typ úlohy, kde by používateľ musel nájsť funkciu, ktorá tento strom generuje.

Vizualizácia výpočtu nám zjavne ponúka veľké množstvo možností, ktoré určite v budúcnosti využijeme pre zlepšenie výučbového procesu.

Kapitola 2

Server

V tejto kapitole sa oboznámime s návrhom a špecifikáciami návrhu vnútorného fungovania servera, bezpečnostných protokoloch a možnosťami poskytovanými administrátorovi.

Je možné, že niektoré z týchto špecifikácií nebudú implementované v prvej pracovnej verzii, ale budú prítomné vo finálnom produkte.

2.1 Použité technológie

2.1.1 Django 1.9

Django je open source framework nad programovacím jazykom Python, ktorý uľahčuje vývoj a údržbu webových aplikácií. Poskytuje funkcionalitu servera na nízkych úrovniach (správa databázy, HTTP komunikácia a iné), čo umožňuje vývojárovi sa sústrediť na tvorbu samotného obsahu. Django takisto ponúka jednoducho implementovateľné možnosti ochrany pred niektorými najbežnejšími útokmi, ako napríklad XSS, SQL injection alebo CSRF.

Server používa framework Django 1.9 pre jednoduchosť vývoja spravovania databáz, autentifikácie popoužívateľov a prostredia pre administrátora.

2.1.2 REST framework

Django REST framework je v projekte využitý na jedoduché a prehľadné zobrazovanie obsahu databázy, odpovedí na HTTP požiadavky a na implementáciu token autorizácie. Umožňuje priamo v prehliadači posielať a zobraziť odpovede na požiadavky, čo zjednodušuje debugovací proces. Uľahčuje aj implementáciu externej OAuth autorizácie (2.3.2)

Takisto obsahuje implementáciu pre_save a post_save signálov ktoré používame pri modeloch ktorých pridanie alebo zmeny priamo menia obsah iných modelov (napríklad

pri automaticko zarovnávaní úloh (4.1.1)).

2.2 Testovač

Testovač je najdôležitejšia časť portálu, pretože dokáže zmerať schopnosti používateľa. Sú tri možné spôsoby testovania:

2.2.1 Stiahnutie vstupov a poslanie výstupov

Toto je najjednoduchší spôsob implementácie testovača. Používateľ si z nášho portálu si stiahne testovacie vstupy na ktorých spustí svoj program, výstup zapíše do súbora a súbor pošle na server, ktorý skontroluje správnosť odpovede.

Veľká výhoda tohto prístupu je jeho jednoduchosť, ale má značnú nevýhodu: nezaručuje že používateľ napísal optimálny program. Účelom portálu je učiť používateľov dynamické programovanie, ale tento testovač umožňuje správne vyriešiť úlohu aj hrubou silou.

Tento problém by sa dal obísť používaním úloh, ktoré majú mimoriadne časovo zložité riešenie hrubou silou (napríklad $O(2^n)$), ktoré by na väčších vstupoch vyžadovali niekoľko dní počítania. Toto riešenie je samozrejme nežiadúce keďže veľmi obmedzuje výber úloh. Preto bude používané iba počas procesu vývoja a nie vo finálnom produkte.

2.2.2 Testovanie skriptom u používateľa

Druhý možný spôsob testovania je rovnaký ako predošlý, až na jednu zmenu: používateľ si stiahne testovací skript, ktorý spustí používateľom zadaný program na vstupoch, ktoré stiahne z nášho servera. Potom odošle výstupy a tie sú porovnané so správnymi odpoveďami.

Toto riešenie nám umožňuje merať čas potrebný na beh programu, čo je značné vylepšenie oproti predošlému riešeniu, pretože ak používateľ nijak nezasiahne do skriptu, vieme zistiť či sa mu podarilo úlohu riešiť dynamickým algoritmom.

Používateľ ale môže skript viacerými spôsobmi napadnúť, napríklad manipulovať s časovým obmedzením alebo získať vstupy, vyriešiť ich iným, pomalším algoritmom a vytvoriť program ktorý pre každý vstup vypíše správny výstup zo súboru alebo priamo zapísaný v zdrojovom kóde.

Tieto chyby sa dajú vyriešiť zabezpečením skriptu proti útokom, ale žiadna ochrana nie je stopercentná. Mohli by sme teda napríklad neakceptovať výsledky, ktorých podozrivo vyzerajúce časy (t.j. časy, ktoré sa nesprávajú podľa krivky, časy dlhšie ako limit, ktoré ale skript sa nezastavil atď.). Toto riešenie ale nie je akceptovateľné, pretože by mohlo prepustiť falšované výsledky alebo odmietnuť správne riešenie.

Na ochranu vstupov sa dá použiť procedurálna generácia alebo náhodné vyberanie z väčšej množiny vstupov, ale tie pridávajú prácu administrátorovi a komplikujú pridávanie nových úloh.

2.2.3 Testovanie na servri

Tretí spôsob testovania prebieha na našom serveri. Používateľ pošle zdrojový kód svojho riešenia, ten na serveri skompilujeme a vyhodnotíme s časovým obmedzením behu.

Veľkou výhodou je, že používateľ nemá prístup k testovaným vstupom a správnym výstupom, takže úloha nebude vyhodnotená za správne vyriešenú pri neoptimálnom programe kvôli vyprchaniu časového limitu (závisí od dĺžky vstupu a časového limitu, ktoré zadáva administrátor). Rovnako je oveľa ťažšie napadnúť a upraviť testovač.

Nevýhodou testovača je napríklad zložitá implementácia, pretože musí vedieť kompilovať čo najviac programovacích jazykov a zároveň musíme dávať pozor aby poslaný program nijako nenarušoval bezpečnosť servera (nečítal z pamäťe ktorú nealokoval, nespúšťal žiadne iné programy, nepoužíval niektoré systémové volania a podobne). Ďalšou nevýhodou sú nároky na výpočtový čas servera: keďže proces testovania spúšťa program na serveri, môže server spomaliť.

Nakoľko pri tomto testovači je najťažšie mať správne výsledky v časovom limite a nemať optimálne riešenie, v portáli je implementovaný tento testovač. Ak ale sa zvýši popularita nášho portálu a vznikne problém s rýchlosťou testovania, bude možno treba zvážiť buď vylepšenie hardvéru servera alebo implementáciu predošlého testovača (2.2.2)

2.3 Bezpečnosť

2.3.1 Autorizácia

Portál používa token autentifikáciu ktorá z pohľadu používateľa vyzerá nasledovne:

Používateľ pri prihlasovaní pošle prihlasovacie meno a heslo (tento krok je iný pri prihlasovaní cez externú doménu) a ako odpoveď dostane náhodne vygenerovaný reťazec o dĺžke približne 40 znakov. Potom všetky požiadavky, pre ktoré server overuje totožnosť používateľa musia v hlavičke uviesť tento token, inak budú zamietnuté. Server si pamätá používateľov token až kým sa používateľ neodhlási alebo mu nevyprší platnosť po dlhšej neaktivite.

Všetky požiadavky musia obsahovať token (buď používateľov alebo CSRF token), čo pomáha používateľa chrániť pred CSRF útokom.

2.3.2 Prihlasovanie cez externé portály

Portál má viacero možností prihlasovania sa cez externé portály. Použijeme prihlasovaciu schému OAuth 2.0, aby komunikácia s externým portálom prebiehala iba pri prihlasovaní po ktorom používateľov prehliadačl komunikuje so serverom ako keby sa prihlásil priamo na náš portál bez využitia externej služby.

Prihlasovanie cez OAuth 2.0 prebieha nasledovne:

1. Používateľ sa prihlási na externý portál a potvrdí našej aplikácii vyžiadané oprávnenia
2. Od externého portálu získa prístupový token na ten externý portál
3. Tento token pošle nášmu serveru
4. Náš server pošle token externému portálu spolu s tajným ID našej aplikácie, aby externý portál vedel, že požiadavka prišla od našej aplikácie
5. Ak externý portál potvrdí požiadavku, náš server vráti používateľovi náš token a ďalej komunikujú iba s týmto novým tokenom.

Pomocou tejto schémy sa bude dať prihlásiť cez Facebook, Google+ a GitHub. (Tajné ID použité v kroku 3 je prístupné iba administrátorom)

2.3.3 Protokoly

Aby sme chránili používateľové dáta a token, všetky požiadavky obsahujúce osobné dáta používajú šifrovaný protokol HTTPS. Keďže HTTPS je pomalší na spracovanie, používame ho iba na potrebné požiadavky. Preto verejne známe informácie, ako napríklad zadania úloh alebo ich zoznam, nemusia byť chránené a používajú rýchlejší HTTP.

2.4 Administrácia a zbieranie dát

Administrátorovi poskytneme prostredie, v ktorom bude môcť prezerať, pridávať, upravovať ale aj mazať všetky modely ku ktorým má prístup. Oprávnenie používať toto prostredie bude mať samozrejme iba administrátor.

2.4.1 Pridávanie a hodnotenie úloh

Pridávanie a spravovanie úloh je hlavnou povinnosťou administrátora. Administrátor okrem zadávania úloh musí poskytnúť aj vzorové riešenie, prípadne nejaké rady k riešeniu úlohy a zaradiť ju podľa obtiažnosti.

Administrátor by sa mal snažiť, aby každá úloha mala ideálne používateľské hodnotenie (1.1.4) Je najlepšie, aby každá úloha mala čo najvyššie hodnotenie zaujímavosti, nakoľko zaujímavé úlohy lepšie motivujú používateľov.

S hodnotením zložitosti je to ale inak: žiadna úloha by nemala byť príliš ľahká (aby bola pre používateľa výzvou) ani príliš ťažká (aby ju používateľ zvládol), preto ideálny priemer hodnotenia zložitosti je 3.

Server bude poskytovať administrátorovi možnosť posúvať úlohy vyššie a nižšie v poradí, v ktorom ich používatelia riešia. Takisto bude označovať úlohy, ktoré majú príliš vysokú alebo príliš nízku obtiažnosť vzhľadom na úroveň riešiteľa a navrhovať nové miesto v poradí, kam ich zaradiť. Podobne bude označovať úlohy, ktoré sú považované za najmenej zaujímavé a bude na administrátorovi aby posúdil, či daná nezaujímavá úloha je dôležitá pre proces výučby, alebo či ju možno zmeniť alebo odstrániť.

2.4.2 Zbieranie dát o používateľoch

Náš portál bude okrem vyučovania dynamického programovania schopný aj zbierať údaje o schopnostiach používateľov a ich zlepšovaní sa postupom cez naše vyučovanie. Server bude zbierať viacero možných údajov o používateľskej aktivite:

- Počet navštívených návodových a teoretických stránok na našom portáli
- Frekvencia návštev počas riešenia úloh (ako často používateľ používa naše návody na pomoc s riešením)
- Priemernú frekvenciu a dĺžku návštev nášho portálu
- Čas od prečítania úlohy po jej správne vyriešenie (nie veľmi dôležitý, pretože neberie do úvahy prestávky pri riešení)
- Počet neúspešných pokusov o riešenie
- Rozdiel priemerného a používateľovho hodnotenia úlohy (až keď bude dostatočne veľa hodnotení aby tento údaj niečo znamenal)

Z týchto údajov budeme vedieť zistiť, ako efektívne je vyučovanie (napríklad, žiak sa pravdepodobne zlepšil ak prvé úlohy označoval ťažšie ako priemer a neskoršie ľahšie) a prípadne implementovať automatické zadávanie úloh podľa používateľovych schopností.

2.4.3 Zbieranie dát o úlohách

Pre kvalitnejší výučbový proces je nutné vedieť, ktoré úlohy sú populárne a koľko sa na nich žiaci naučia. Preto budeme zapisovať dáta z používateľskej aktivity aj pre úlohy. Medzi tieto dáta bude patriť:

- Priemerné hodnotenie zložitosti a zaujímavosti úlohy
- Priemerný čas od zadania do vyriešenia úlohy
- Priemerný počet nesprávnych riešení pred správnym
- Priemerný počet použitých rád

Tieto dáta bude môcť vidieť iba administrátor a na ich základe by mal meniť úlohy tak, aby používatelia boli s nimi čo najspokojnejší.

2.4.4 Oprávnenia používateľa a administrátora

Bežní používatelia majú prístup iba k učebným materiálom, odomknutým úlohám (vyriešeným aj nevyriešeným) a svojim riešeniam. Ďalšie úlohy sa používateľovi odomknú, iba ak správne vyrieši všetky prerekvizitové úlohy. Všetky dáta zozbierané servrom (2.4.2) budú od používateľa skryté, pretože by mohli spôsobiť frustráciu u pomalšie sa učiacich používateľov.

Administrátor bude mať prístup ku všetkým úlohám, riešeniam, zozbieraným dátam všetkých používateľov a ich priemerom. Bude mať aj možnosť manuálne meniť niektoré hodnotenia a odomykať alebo zamykať používateľom úlohy.

Túto možnosť obchádzať pravidlá administrátorovi dávame pre prípad, že vznikne chyba a používateľovi sa napríklad odomkne úloha ku ktorej nemal mať prístup alebo neodomkne taká, ktorej všetky prerekvizity splnil. Používateľ v prípade takejto chyby bude môcť kontaktovať administrátora, ktorý preverí či naozaj nastala chyba a bude ju môcť napraviť bez debugovania servera, čo môže trvať príliš dlho.

Kapitola 3

Dokumentácia stránok

V tejto kapitole si priblížime implementáciu prednej časti portálu, s ktorou budú používatelia interagovať. Povieme si komunikácii so servrom ale aj o implementácii vizualizácie rekurzívnych výpočtov.

3.1 Stránky a ich adresy

V prvej pracovnej verzii sú implementované tieto stránky (uvedené s URL adresami)

- registrácia - */user/register/*
- prihlasovanie - */user/login/*
- zoznam úloh - */lessons/*
- detail úlohy - */lesson/id/* (id je unikátne identifikačné číslo úlohy)
- vizualizácia - */visualisation/*
- administrátorské prostredie - */admin/*

Administrátorské prostredie je predvolené prostredie Django frameworku bez modifikácií. Toto prostredie poskytuje všetky požadované možnosti (prezeranie, pridávanie, úprava a mazanie objektov) vo veľmi prehľadnej a jednoduchej forme.

3.1.1 Komunikácia so servrom

Na komunikáciu so servrom používame hlavne knižnicu jQuery, konkrétne asynchronickú metódu AJAX, kvôli jej jednoduchému používaniu a podpore pridania autorizačnej hlavičky. Autorizačná hlavička je údaj v hlavičke HTTP požiadavky ktorý určuje totožnosť používateľa, v našom prípade napríklad `{'Authorization': 'Token a5b498587e33d1f44a97c2328618dd15373b0705'}`.

Tento token dostaneme ako odpoveď na prihlasovaciu požiadavku a uložíme si ho v lokálnej pamäti prehliadača pomocou zavolania `localStorage`, konkrétne takto: `localStorage.setItem('token', response['token'])`. Potom token môžme kedykoľvek získať pomocou zavolania metódy `getItem()`. Väčšina požiadaviek ktoré posielame na server sú si veľmi podobné, preto uvedieme iba jednu typickú požiadavku:

```
1  $.ajax({
2     url: "../../api/lesson/"+id+"/",
3     type: "GET",
4     headers: {'Authorization':'Token '+localStorage.getItem('
          ↪ token')},
5     success: function(data) {
6       title = document.getElementById("title")
7       title.innerHTML = "Lesson "+data['name']
8       problem = document.getElementById("problem")
9       problem.innerHTML = "Lesson "+data['problem']
10    },
11    error: function(data) {
12      if (data.status==401) {
13        window.location.replace("../../user/login/")
14      }
15    }
16 });
```

Požiadavka na získanie zadania jednej úlohy

(v tomto príklade je požiadavka trochu zmenená oproti implementácii, aby bola v texte prehľadnejšia)

Tento a viacero iných podobných požiadavok posielame vždy pri načítaní stránky prehliadačom, aby sme získali dáta ktoré chceme zobraziť.

Teraz si vysvetlíme niektoré časti tohto kódu:
V druhom riadku používame premennú id, ktorá označuje unikátne identifikačné číslo úlohy. Toto číslo získame z URL zobrazenej stránky ako prvú vec pri načítaní.

Vo štvrtom riadku pridávame autorizačnú hlavičku, podľa spôsobu, aký sme popísali vyššie.

V piatom až desiatom riadku deklarujeme správanie pri úspešnom získaní odpovede od servra, v tomto prípade zobrazujeme názov a zadanie úlohy v na to určených HTML elementoch stránky.

V jedenástom až päťnástom riadku deklarujeme správanie sa pri chybe. Zatiaľ jediná chyba, na ktorú reagujeme je 401-UNAUTHORIZED ktorú dostaneme ak posielame správu s nesprávnym alebo úplne chýbajúcim tokenom. V tom prípade používateľa

presmerujeme na prihlasovaciu stránku. Toto správanie je použíté v každej požiadavke, ktorá vyžaduje autentifikáciu.

Odosielanie na server je o trochu iné, pretože používa metódu POST a musíme pridať aj posielané dáta. To najčastejšie robíme pomocou pridania
`data : $("#form").serialize()`, čo nám dáta získa priamo z hodnôt vo formulári.

jQuery má ale obmedzenie, že kvôli bezpečnosti nedokáže zapisovať do súborov na disku. To znamená, že sťahovanie súborov je náročné na implementáciu. Preto na sťahovanie vstupných súborov (pre implementáciu jednoduchého testovača 2.2.1) používame formulár, ktorý pošle GET požiadavku na požadovanú adresu. Tento formulár ale nemôže mať hlavičku s tokenom, preto sťahovanie vstupov nebude autentifikovať používateľa.

To síce znamená, že používateľ môže mať prístup aj k vstupom úloh, ktoré si ešte neodomkol. Tento problém nepovažujeme za závažný, pretože bez zadania úlohy je malá šanca že používateľ túto úlohu vyrieši. Ďalej, vo finálnom produkte testovač nebude vyžadovať sťahovanie dát, čím bude tento problém úplne odstránený.

Rovnako, jQuery má ťažkosti s posielaním súborov (skoršie verzie dokonca neodkážu posielať súbory). Preto na posielanie riešení používame formát XMLHttpRequest, ktorý funguje podobne ako AJAX, preto si ho ďalej približovať nebudeme.

Kapitola 4

Dokumentácia servra

V tejto kapitole sa pozrieme na prvú pracovnú implementáciu návrhu servra. Oboznámime sa s modelmi, pohľadmi a niektorými funkciami, ich možnosťami a použitím.

4.1 Modely

Django servery sledujú návrhový vzor Model-View-Controller (MVC), kde model je tabuľka databázy a view je spôsob zobrazenia (napríklad vo forme HTML stránky). Pre pochopenie funkcionality je dôležité vedieť aké modely a pohľady používame a ako medzi sebou interagujú, preto sa s nimi oboznámime.

Tu sa oboznámime s dôležitými modelmi, teda tabuľkami databázy. Pri každom modeli spomenieme jeho názov v implementácii a slovenský ekvivalent tohto názvu.

(Každý model obsahuje aj pole id, čo je jedinečné identifikačné číslo automaticky generované Django frameworkom)

4.1.1 Lesson - úloha

Úlohy majú tieto polia:

- **name**: meno úlohy ktoré sa zobrazuje používateľovi.
- **problem**: znenie úlohy.
- **pub_date**: dátum publikácie úlohy.
- **number**: úroveň úlohy, používateľ musí mať vyriešené všetky povinné úlohy s na nižších úrovniach aby mal k tejto úlohe prístup.
- **optional**: voliteľnosť úlohy. Povinné aj voliteľné úlohy sa odomykajú rovnako, voliteľné ale neodomykajú neskoršie úlohy.
- **inputs**: vstupy na ktorých prebieha testovanie úlohy.

- **correct_solution**: správne riešenie, oproti ktorému sa budú testovať používateľské riešenia.

Pridávanie úloh je jednoduché - stačí vyplniť polia v administrátorskom prostredí a uložiť. Na každej úrovni ale musí byť práve jedna povinná úloha, preto po upravení alebo pridaní novej povinnej úlohy sa poradie upraví tak, aby pridaná alebo zmenená úloha mala po uložení číslo, aké jej administrátor zadal. Viac si o spôsobe zarovnania úloh povieme v časti 4.2.2

Na úlohy máme tri rôzne pohľady (views): Jeden ktorý vráti všetky úrovne, voliteľnosti a názvy úloh ku ktorým má používateľ prístup, druhý meno, znenie a úroveň s voliteľnosťou jednej úlohy a tretí iba vstupy alebo skript na stiahnutie (ak používame testovač 2.2.1 alebo 2.2.2)

4.1.2 User - používateľ

Model s osobnými údajmi používateľa

Polia:

- **username**: prezývka používateľa, nutná na registráciu a prihlásenie, jedinečná
- **email**: emailová adresa používateľa, tiež nutná a jedinečná
- **password**: heslo používateľa, ukladané pomocou hash funkcie poskytovanej REST frameworkom
- **first_name**: prvé meno používateľa, nepovinné
- **last_name**: priezvisko používateľa, taktiež nepovinné

Pri vytvorení nového používateľa sa automaticky vytvorí jeho **UserStat** (4.1.8)
Tu takisto môžem spomenúť model **Token**, ktorý má polia **user** a **key**. **Token** je používaný pri autorizácii (2.3.1) a je vygenerovaný pri každom prihlásení a zmazaný pri odhlásení.

4.1.3 Submit - riešenie

Riešenie ktoré používateľ pošle na server na otestovanie.

Polia:

- **user**: používateľ ktorý riešenie poslal
- **lesson**: úloha ktorú používateľ rieši týmto riešením
- **submittedFile**: súbor s riešením, ktoré bude odovzdané testovaču
- **result**: výsledok - reťazec znakov obsahujúci výpis z testovača

Hneď ako server dostane nové riešenie, spustí testovač a jeho odpoveď uloží v poli **result**. Ak nie je správne, riešenie uložíme a ďalej s ním nič nerobíme. Ak testovač ale vyhodnotí riešenie ako správne (výpis `"OK"`), pred jeho uložením zvýši používateľovu úroveň o 1 (ak je úloha povinná) a označí úlohu ako vyriešenú v modeli **UserLessonWrapper** (4.1.9).

4.1.4 Comment - komentár

Komentár od používateľa na úlohu. Polia:

- **user**: používateľ ktorý vytvoril komentár
- **lesson**: úloha ku ktorej bol komentár pridaný
- **text**: text komentára
- **date**: čas vytvorenia komentára, potrebný na zoraďovanie pri zobrazení

Pri komentároch sa bude zobrazovať aj úroveň autora a to, či úlohu vyriešil. Zatiaľ sú všetky komentáre prístupné všetkým používateľom (ak majú prístup k ich úlohe)

4.1.5 Hint - rada

Rada k úlohe, pre prípad že ju používateľ nevie riešiť.

Polia:

- **lesson**: úloha ku ktorej je táto rada
- **number**: poradové číslo rady
- **text**: text rady

Používateľovi sa pri zobrazení úlohy ukazujú iba toľko rád koľko si požiadal. Táto informácia sa ukladá v modeli **UserLessonWrapper** (4.1.9). Používateľ môže požiadať o ďalšiu radu a kým existuje nepoužitá rada, zväčšíme číslo použitých rád v **UserLessonWrapper**-i a odpovieme novým, o jednu randu dlhším zoznamom rád.

4.1.6 Rating - hodnotenie

Hodnotenie úlohy v kategóriach zložitosti a zábavnosti.

Polia:

- **user**: používateľ ktorý zaslal toto hodnotenie
- **lesson**: úloha ktorú hodnotil
- **fun**: hodnotenie zaujímavosti
- **difficulty**: hodnotenie zložitosti

Pre každú dvojicu **úloha-používateľ** môže existovať iba jedno hodnotenie, čo znamená že každý používateľ môže ohodnotiť každú úlohu iba raz.

Kým používateľ nevyriešil úlohu, prehliadač mu nezobrazí formulár na hodnotenie, preto by nemal mať spôsob ako poslať hodnotenie úlohy ktorú nevyriešil. Napriek tomu, zavedieme opatrenie proti možnosti manuálne poslať požiadavku a tým obísť podmienku správneho vyriešenia úlohy: pred uložením hodnotenia overíme aj na servri či používateľ úlohu správne vyriešil. Ak a mu to ešte nepodarilo, ako odpoveď na poslanie hodnotenia dostane chybu UNAUTHORIZED.

Ak sa mu ale podarilo úlohu predtým vyriešiť, vytvoríme nové alebo upravíme existujúce hodnotenie a obnovíme model **LessonStat** (4.1.7) aby mal aktuálne dáta.

4.1.7 LessonStat - štatistika úlohy

Model ktorý zbiera štatistické dáta o úlohe, ktoré budú neskôr spracované a ukázané administrátorovi. K tomuto modelu má prístup iba administrátor.

Polia:

- **lesson**: úloha ku ktorej štatistika patrí
- **avg_fun**: priemerné hodnotenie zaujímavosti
- **avg_diff**: priemerné hodnotenie zložitosti
- **good_solutions**: počet používateľov ktorí úlohu správne vyriešili (nie počet správnych riešení, pretože účelom tohto poľa je pomôcť zistiť koľko riešení používatelia skúšajú pred správnym vyriešením)
- **bad_solutions**: počet zlých riešení úlohy pred správnym vyriešením úlohy

4.1.8 UserStat - štatistika používateľa

V momentálnej implementácii zbierame iba jeden údaj pre používateľov, ktorý ale používame na zisťovanie, ktoré úlohy používateľovi sprístupniť. Narozdiel od modelu **LessonStat** (4.1.7), táto štatistika je prístupná používateľovi, pretože ide o jeho výsledky.

Tento model bude vo finálnej obsahovať napríklad aj skóre používateľa, ktoré bude podľa priania používateľa zverejnené v rebríčku (1.1.3)

Polia:

- **user**: používateľ
- **progress**: úroveň používateľa - počet správne vyriešených povinných úloh

4.1.9 UserLessonWrapper - obal používateľ-úloha

Model na určovanie postupu používateľa jednou úlohou. Používa sa iba na vnútorné overovanie prístupu používateľa k niektorým modelom. Tento model je unikátny pre každú dvojicu používateľ-úloha

Polia:

- **user**: používateľ
- **lesson**: úroveň používateľa - počet správne vyriešených povinných úloh
- **hints_used**: počet použitých rád k úlohe
- **completed**: či používateľ správne vyriešil úlohu

Tento model sprístupníme administrátorovi na upravovanie, pretože chceme mať možnosť opravovať nepredvídané chyby (napríklad ak používateľ pošle správne riešenie ale úloha sa mu neoznačí za vyriešenú).

4.2 Pomocné funkcie

Teraz si popíšeme implementáciu automatického testovača a zarovnávania pridaných úloh, pretože obe sú dôležité funkcie, ktoré by sme mali ovládať.

4.2.1 Testovač

Testovač je v prvej pracovnej verzii implementovaný podľa najjednoduchšieho prístupu 2.2.1

Uvedieme si popis kódu testovača a potom si ho aj uvedieme.

Tento testovač otvorí používateľove a správne riešenie úlohy ako .zip súbor. Najprv overí, či počet súborov je rovnaký. Ak nie, vypíše chybovú hlášku.

Potom pre všetky súbory zo správneho riešenia skúsi nájsť súbor s rovnakým menom v používateľovom riešení. Ak ho nenájde, vypíše chybu. Ak ho nájde, ale obsah súboru sa líši v používateľovom a správnom riešení, vyhodnotí riešenie za nesprávne.

Ak všetky súbory z používateľovho riešenia prejdú týmito testami, vieme povedať, že všetky súbory sa zhodovali s tými v správnom riešení a teda riešenie je správne.

```
1  def compare_files(correctFile, submitFile):
2      correct = zipfile.ZipFile(correctFile)
3      submit = zipfile.ZipFile(submitFile)
4      if len(correct.namelist()) != len(submit.namelist()):
5          return "Wrong file structure"
6      for name in correct.namelist():
7          if name not in submit.namelist():
8              return "Wrong file structure"
9          if submit.read(name) != correct.read(name):
10             return "Wrong answer"
11     return "OK"
```

testovač

Nevýhodou je, že používateľove súbory musia mať rovnaké meno ako tie v správnom riešení, preto je na autorovi úlohy aby v zadaní popísal správny formát riešenia.

My odporúčame ako konvenciu používanie vstupných súborov s príponou .in a výstupov s rovnakým menom, ale príponou .out (napríklad vstupný súbor 01.a.in a výstupný 01.a.out)

4.2.2 Posúvanie úloh

Každá úroveň úloh musí mať presne jednu povinnú úlohu. Keďže chceme, aby administrátor nemusel úlohy posúvať sám pred pridaním novej alebo pred presunutím existujúcej na inú úroveň, tento proces zarovnávania si automatizujeme.

Prejdeme tieto kroky:

- Všetky povinné úlohy posunieme tak, aby neboli žiadne medzery v poradí.
- všetky povinné úlohy čo majú úroveň väčšiu alebo rovnú úrovni upravovanej/pridanej úlohy posununieme o jednu úroveň vyššie (upravovaná/pridaná úloha nie je posunutá).
- všetky úlohy znovu posununieme tak, aby sme vyplnili medzery (pretože predošlý krok mohol vytvoriť nové medzery ak presúvame existujúcu úlohu na vyššiu úroveň)

- nakoniec pridáme ukladanú úlohu a zarovnáme ju tiež, nakoľko administrátor mohol urobiť chybu a pridať úlohu s číslom väčším ako počet povinných úloh a teda by bola nedosiahnuteľná.

```
def PushOtherLessons (sender, instance, *args, **kwargs):
  if not(instance.optional):
    rank = instance.number
    non_optionals = [x for x in Lesson.objects.all().order_by('number')
      if not(x.optional) and x.id!=instance.id]
    #fill the gaps
    for l in non_optionals:
      while len(Lesson.objects.filter(number = (l.number-1)))==0 and l.number>1:
        Lesson.objects.filter(id=l.id).update(number = l.number -1)
        l.number -= 1
        print(l)
    #push later lessons one level up
    if len(Lesson.objects.filter(number=rank))>0:
      Lesson.objects.filter(number__gte = rank, optional=False).exclude(id=instance.id).update(number = F('number') +1)
    non_optionals = [x for x in Lesson.objects.all().order_by('number') if not(x.optional) and x.id!=instance.id]
    #fill the gaps again
    for l in non_optionals:
      while len(Lesson.objects.filter(number = (l.number-1)))==0 and l.number>1:
        Lesson.objects.filter(id=l.id).update(number = l.number -1)
        l.number -= 1
    #adjust saved lesson
    while len(Lesson.objects.filter(number = (instance.number-1) ))==0 and instance.number>1:
      Lesson.objects.filter(id=instance.id).update(number = instance.number-1)
      instance.number -= 1
post_save.connect(PushOtherLessons, sender=Lesson)
```

Posúvanie úloh

Vypĺňanie medzier sa uskutočňuje pomocou `while` cyklu, v ktorom dekrementujeme úroveň zarovnávanej úlohy, až kým nenarazíme na prvú povinnú úlohu ktorá má úroveň o 1 menšiu ako práve zarovnávaná úloha. Vtedy cyklus skončíme a môžme si byť istí, že po túto úlohu žiadne medzery neexistujú.

Na ukladanie zmien používame metódu `Lesson.update()` miesto `Lesson.save()` pretože `Lesson.save()` zavolá `post_save` signál, ktorý by spustil proces zarovnávania, ktorý by tiež spustil ďalší proces zarovnávania a funkcia by nikdy neskončila.

Ako nové číslo úlohy v metóde `Lesson.update()` používame pri posúvaní úloh hore výraz F, konkrétne `F('number')`. Tento výraz F nám dovoľuje získať hodnotu poľa práve obnovovaného objektu bez toho, aby sme mu priradili premennú čím skracujeme a sprehľadňujeme kód. V ostatných prípadoch výraz F nie je potrebný, pretože už máme obnovovanú úlohu označenú premennou l alebo instance.

Záver

Literatúra

[1] Django documentation. https://docs.djangoproject.com/en/1.9/. [2015-12-07].

[2] Facebook login for the web with the javascript sdk. https://developers.facebook.com/docs/facebook-login/web. [2015-12-07].

[3] Javascript promise. https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise. [2015-12-07].

[4] React documentation. https://facebook.github.io/react/index.html. [2015-12-07].

[5] Daniel Roy Greenfield and Auderey Roy Greenfield. *Two Scoops of Django: Best Practices for Django 1.8*. Two Scoops Press, 2015.