Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Martin Jakomin

Sistem za avtomatsko ocenjevanje algoritmov

diplomsko delo
univerzitetni študijski program prve stopnje
računalništvo in matematika

Mentor: doc. dr. Tomaž Dobravec

Ljubljana 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00010/2012
Datum: 11.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **MARTIN JAKOMIN**

Naslov: **SISTEM ZA AVTOMATSKO OCENJEVANJE ALGORITMOV**
THE SYSTEM FOR AUTOMATIC EVALUATION OF ALGORITHMS

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V diplomskem delu izdelajte spletni sistem, ki bo omogočal avtomatsko ocenjevanje algoritmov. V sistemu opredelite pojem abstraktnega algoritma, ki opisuje čimbolj splošen algoritem s prilagodljivimi vhodnimi podatki. Izdelajte koncept abstraktnega preverjanja kakovosti posameznega algoritma. Sistem naj uporabniku omogoča implementacijo konkretnih algoritmov in konkretnih metod za preverjanje kakovosti. Izdelajte spletni vmesnik, ki bo omogočal dodajanje, brisanje in spreminjanje testnih metod in algoritmov. Vmesnik naj omogoča prikaz rezultatov testiranja na čimbolj nazoren način (tekstoven in grafičen prikaz). Implementirajte tudi katerega od znanih algoritmov in ga vključite v sistem kot testni primer.

Mentor:

doc. dr. Tomaž Dobravec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 1

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Martin Jakomin, z vpisno številko **63090019**, sem avtor diplomskega dela z naslovom:

Sistem za avtomatsko ocenjevanje algoritmov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Dobravca
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 21. septembra 2012 Podpis avtorja:

Kazalo

POVZETEK

Glavni cilj diplomske naloge je bil ustvariti učinkovit sistem za avtomatsko ocenjevanje algoritmov. Sistem, ki je dovolj robusten in prilagodljiv, hkrati pa učinkovit ne glede na izbrani algoritem. Pri izdelavi sistema je bila ključnega pomena enostavnost uporabe tako za uporabnike, kot tudi za administratorje sistema. Še prej pa je bilo potrebno definirati, kako je v sistemu dejansko predstavljen algoritem. Potrebna je bila poenostavitev in abstrakcija algoritma.

V uvodu so na kratko predstavljeni cilji in motivacija, naša predstavitev algoritma v aplikaciji, ter kratek opis same aplikacije. Sledijo navodila za namestitev sistema, ter navodila za uporabnike in administratorje. V četrtem poglavju so opisani postopki ocenjevanja kakovosti algoritmov ter zahteve, ki jih morajo izpolnjevati oddani algoritmi. Nato so predstavljene uporabljene tehnologije in orodja. Sledi sama implementacija sistema, ter sklepne ugotovitve in priloge.

Na priloženi zgoščenki se nahaja spletna aplikacija, program za testiranje algoritmov za brezizgubno stiskanje podatkov ZipTest in enostavni primer takšnega algoritma MyZip.

ABSTRACT

The main goal of this diploma thesis was to create efficient system for automatic evaluation of algorithms. The system, which is sufficiently robust and flexible and at same time efficient regardless of the chosen algorithm. When making this system, it was essential to create system, that is easy to use for both users as well as system administrators. But first it was necessary to define how is algorithm actually presented in the system. It took a simplification and abstraction of algorithm.

In the introduction, there is a brief presentation of the goals and motivation, representation of algorithm, and a short presentation of application. Following up, are the instructions for installation, and instructions for users and administrators. The fourth chapter gives a description of the procedures for quality evaluation of algorithms, and requirements that must be met by submited algorithms. Then there is a presentantion of the technology and tools used. Following up is the implementation of the system itself. Last but not least there is conclusion and annexes.

The included CD-ROM contains web application, testing program for lossless data compression algorithms ZipTest, and simple example of such an algorithm MyZip.

Poglavje 1

UVOD

V življenju se soočamo z vrsto problemov, ki jih lahko rešujemo na najrazličnejše načine. Izdelujemo tako enostavne, kot tudi zapletene algoritme. Vendar ali nam zapletenost algoritma zagotavlja tudi boljšo rešitev od drugih? Kako lahko sploh nepristransko primerjamo dva ali več različnih algoritmov za nek isti problem? Kako lahko na koncu ocenimo algoritme in izberemo »najboljšega«?

Izbrati najboljši algoritem še zdaleč ni tako trivialno kakor se zdi na prvi pogled, še posebej zato, ker obstaja veliko različnih problemov. Lahko pa za različne vrste (kasneje v delu omenjene kot »skupine«) algoritmov razvijemo različne ocenjevalne tehnike, ter ocenimo algoritme na podlagi le teh.

Prav to je bil tudi namen diplomske naloge, zato smo razvili sistem oziroma spletno aplikacijo, ki omogoča ocenjevanje najrazličnejših vrst algoritmov. Pri tem nam je bila dodatna motivacija tudi dejstvo, da podobni sistemi zaenkrat ne obstajajo.

1.1 Kaj je algoritem

Na začetku moramo definirati, kaj algoritem sploh je, oziroma kako ga predstavimo v računalniškem svetu. Pojem algoritem lahko formalno opišemo kot »končno, deterministično in učinkovito metodo za reševanje problemov,

ki se jo da implementirati kot računalniški program≪[2].

Drugi način opisa algoritma pa pravi, da je algoritem zaporedje računskih korakov, ki pretvorijo vhodne podatke v izhodne [1].

Algoritem je torej lahko karkoli, kar zadošča tem pogojem. Za lažjo predstavitev v našem sistemu bomo morali definirati pojem ≫abstraktnega≪ algoritma.

1.2 Kako predstaviti algoritem

Algoritem lahko definiramo s tem, da mu natančno opišemo postopke oziroma korake za rešitev nekega problema. Natančnost definicije algoritma lahko zagotovimo s tem, da algoritem predstavimo v obliki računalniškega programa, saj je tako lažje preveriti ali je algoritem končen, deterministične in učinkovit.

Kako torej predstaviti algoritem v našem sistemu? Kakšno definicijo je treba uporabiti, da bomo zajeli, kar največ različnih vrst algoritmov? V ta namen bomo najprej definirali vrsto oziroma skupino algoritmov. Vsaka skupina algoritmov je objekt s svojim unikatnim imenom, z do deset različnih merljivih kriterijev, kateri lahko ocenijo primere algoritmov teh skupin in s programom, ki avtomatsko oceni te kriterije. Za definicijo samega algoritma potrebuje vsaka skupina tudi svoj vmesnik, kateri natančno predpisuje metode, ki jih mora definirati vsak primer dane skupine. Vmesnik natančno določi tudi vhodne in izhodne podatke.

S tem lahko definiramo algoritem kot vsak program, ki pravilno implementira vse metode, ki jih predpisuje nek vmesnik, ki je del neke skupine algoritmov v našem sistemu. Definiramo tudi objekt algoritem z unikatnim imenom in z do deset različnih številskih polj za rezultate.

Z vpeljavo teh dveh objektov, bomo v nadaljevanju lahko med seboj primerjali in ocenjevali algoritme istih skupin.

1.3 Kako oceniti algoritem

Kateri algoritem je najboljši pri danem problemu je težko določiti. Na to vpliva zelo veliko dejavnikov. Osredotočili se bomo na učinkovitost algoritma pri porabljanju računalniških virov, kot sta na primer poraba pomnilnika ali čas izvajanja.

Teoretični pristop testiranja te učinkovitosti bi bil s pomočjo matematičnih modelov in napovedovanjem same učinkovitosti algoritmov. Na primer asimptotično ocenjevanje računske zahtevnosti.

Mi pa bomo uporabili bolj praktični pristop. S pomočjo našega sistema in posebnimi programi za testiranje in ocenjevanje bomo testirali vse algoritme. Za vsako skupino algoritmov bomo uporabili različne programe, ter s tem zajeli kar največ različnih načinov ocenjevanja različnih vrst algoritmov. Samo oceno kakovosti algoritma lahko potem ocenimo s pomočjo do deset rezultatov oziroma kriterijev, katere vrne program za testiranje. Rezultate lahko nato primerjamo tudi z rezultati drugih algoritmov iste skupine s pomočjo razpredelnic in grafov.

1.4 O aplikaciji

Razvita aplikacija omogoča uporabnikom, da se prijavijo v sistem in preko spleta pošiljajo svoje algoritme na strežnik, kjer se nepristransko ocenijo. Uporabnik lahko nato preveri svoje rezultate in jih s pomočjo preglednic in grafov tudi primerja s standardnimi algoritmi (algoritmi, ki so že shranjeni v aplikaciji in predstavljajo neko mejo za ostale algoritme) in z algoritmi, ki so jih oddali drugi uporabniki.

Uporabniki lahko oddajajo le algoritme za probleme, za katere tisti trenutek obstaja podpora v sistemu. Nove probleme oziroma nove skupine algoritmov lahko dodaja skrbnik sistema oziroma administrator. Njegova naloga je tudi, da pri vsaki skupini algoritmov natančno določi zahteve, katere morajo izpolnjevati oddani algoritmi.

Administrator lahko kadarkoli izvrši ponovno testiranje vseh algoritmov.

S tem poskrbi, da imajo vsi algoritmi ob morebitni posodobitvi strojne opreme ali menjavi strežnika enake in nepristranske pogoje testiranja.

Poglavje 2

NAVODILA ZA NAMESTITEV SISTEMA

Navodila in postopki opisani v tem predelu se nanašajo na delo z operacijskim sistemom Ubuntu, ter strežnikoma Apache in MySQL. Sistem deluje tudi na drugih operacijskih sistemih, kot na primer Windows 7.

2.1 Potrebna programska oprema za delovanje sistema

- Python 2.7
- Java Jdk
- MySQL
- Apache 2
- Apache WSGI
- Pip
- Django 1.4

- MySQL-python 1.2.3
- MySQL Connector J

2.2 Namestitev potrebne programske opreme

Python: Python verzije 2.7 je dostopen na spletnem naslovu: `http://www.python.org/download/`

Java: Potrebna orodja za delovanje z Javo (JDK - Java Development Kit) so dostopna na naslovu: `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

MySQL: Strežnik SQL je dostopen na spletnem naslovu `http://www.mysql.com/downloads/`

Apache: strežnik Apache je dostopen na spletnem naslovu: `http://httpd.apache.org/download.cgi`

Apache WSGI: Modul WSGI za Apache namestimo z naslednjim ukazom:

```
$ sudo apt-get install apache2 libapache2-mod-wsgi
```

Pip: Orodje Pip namestimo z naslednjimi ukazi:

```
$ sudo su
$ apt-get install curl
$ curl -O http://python-distribute.org/distribute_setup.py
$ python distribute_setup.py
$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-
   pip.py
$ python get-pip.py
```

MySQLPython: Django in MySQLPython namestimo tako, da se pomaknemo v mapo »Alg«. Nato je potrebno pognati naslednji ukaz:

```
$ pip install -r requirements.txt
```

MySQL Connector J: Knjižnica je dostopna na spletnem naslovu: `http://www.mysql.com/downloads/connector/j/`

Namestitev knjižnice »MySQL Connector J« ni potrebna, saj se ta že nahaja v mapi »Alg/additional_resources/lib«.

2.3 Nastavitev

2.3.1 Nastavitev podatkovne baze

Potrebno je zagnati strežnik SQL in ustvariti novo podatkovno bazo. To izvršimo z ukazom:

```
CREATE DATABASE <ime podatkovne baze>;
```

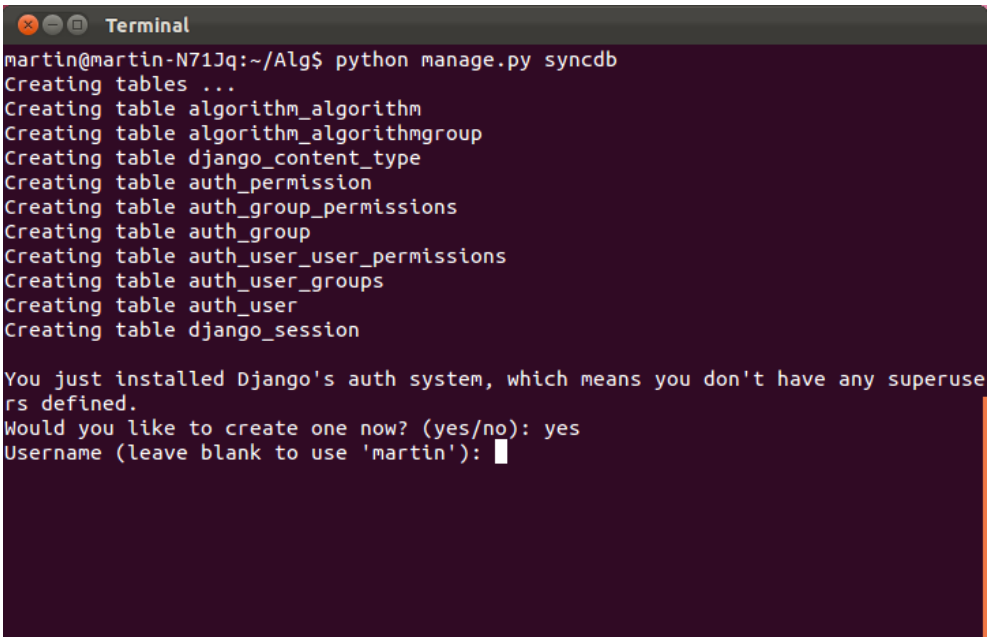
Za ustvaritev potrebnih tabel v podatkovni bazi in normalno delovanje sistema je potrebno pognati program »manage.py«. To izvedemo s premikom v mapo »Alg« in z zagonom ukaza:

```
$ python manage.py syncdb
```

Program nam pri tem ponudi možnost, da ustvarimo administratorski račun (slika 2.1). Potrebno je vnesti uporabniško ime, elektronski naslov in geslo. V primeru da se za ta korak ne odločimo bomo morali kasneje sami ustvariti administratorski račun.

2.3.2 Nastavitev datoteke settings.dat

Datoteka »settings.dat« se nahaja v mapi »Alg/additional_resources«, v katero je potrebno zapisati podatke o podatkovni bazi, ter nekatere druge podatke, ki omogočajo pravilno delovanje sistema. Vsi podatki morajo biti zapisani ločeno v novih vrsticah.



Slika 2.1: Ustvaritev tabel in administratorskega računa

V datoteko moramo vpisati ime podatkovne baze (katera je bila ustvarjena v prejšnjem koraku) skupaj s ključno besedo »name=«, uporabniško ime in geslo za dostop do baze (pred njima je potrebno zapisati »user=« in »password=«), ter ime gostitelja in vrata s ključnima besedama »host=« in »port=«.

Za pravilno delovanje prevajalnika je potrebno navesti tudi pot do mape, kjer se nahaja »Java JDK«. Pred potjo je treba navesti »path_to_jdk=«. Pot, kjer se med testiranjem ustvari mapa (in po končanju tudi izbriše), v kateri poteka samo testiranje algoritmov, je privzeto nastavljena na Alg/testing_env. Za spremembo poti je potrebno v datoteko zapisati »path_to_test_env=« in želeno pot.

Na koncu je potrebno zapisati še količino maksimalne dovoljene velikosti prenesenih datotek, izraženih v bajtih, s ključno besedo »max_size=«.

Primer datoteke settings.dat

```
//Database data: databaseName, username, password, host, port
name=alg
user=root
password=e2ff4wb
host=localhost
port=3306
//Path to java jdk
path_to_jdk=C:/Program Files/Java/jdk1.7.0_05
//Path to testing environment directory (directory must not yet
//exist, it will be created at start and delete itself at the
   end)
//If left blank, "ROOT_DIR"/testing_env/ will be used
path_to_test_env=
// maximum upload size (in bytes)
max_size=5242880
```

2.3.3 Testni zagon na lokalnem strežniku

Za testni zagon aplikacije na lokalnem strežniku je potrebno pognati ukaz:

```
$ python manage.py runserver [vrata ali naslov:vrata]
```

Če izpustimo naslov in vrata, se avtomatsko privzame naslov 127.0.0.1 in vrata 8000. Hkrati mora biti zagnan tudi strežnik SQL.

2.3.4 Zagon aplikacije s strežnikom Apache

Za zagon aplikacije na strežniku Apache je potrebno v mapi, kjer je nameščen Apache, odpreti datoteko »httpd.conf« in v njo dodati:

```
WSGIScriptAlias / <pot do mape Alg>/alg/wsgi.py
WSGIPythonPath <pot do mape Alg>
Alias /static <pot do mape Alg>/alg/frontend/static
```

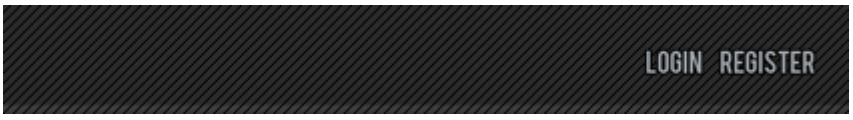
Poglavje 3

NAVODILA ZA UPORABO SISTEMA

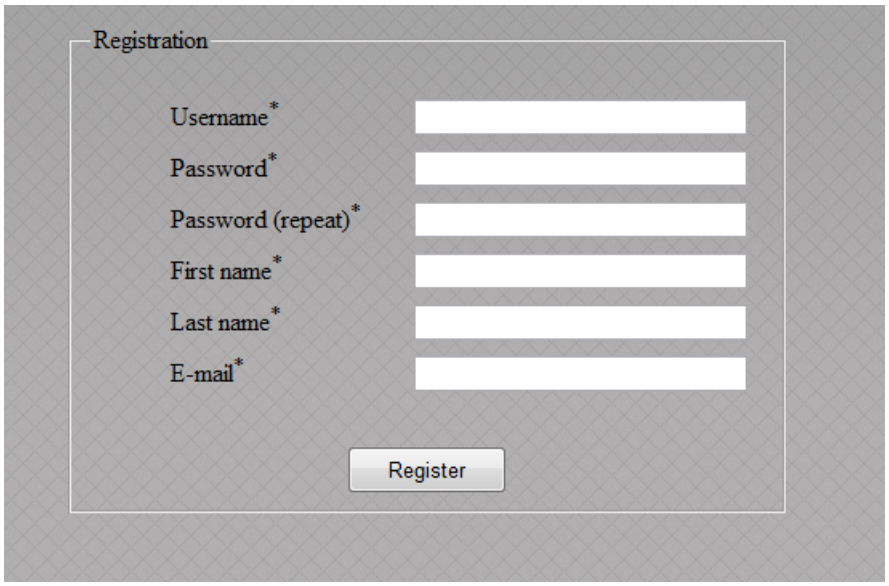
3.1 Uporabniška navodila

3.1.1 Registracija

Preden se lahko uporabnik prijavi v spletno aplikacijo, se mora najprej registrirati. To lahko naredi s klikom na gumb »Register« v desnem zgornjem kotu (slika 3.1). Prikaže se spletni obrazec (slika 3.2), v katerem je potrebno izpolniti vsa njegova polja: uporabniško ime, ki mora biti enolično in sestavljeno le iz črk, številk ali znakov »@,.,+,-,_«, geslo, geslo (ponovno), ime, priimek in elektronska pošta. Registracija se zaključi s pritiskom na gumb »Register« na obrazcu spodaj.



Slika 3.1: Prijava in registracija v zgornjem desnem kotu



Slika 3.2: Obrazec za registracijo

3.1.2 Prijava

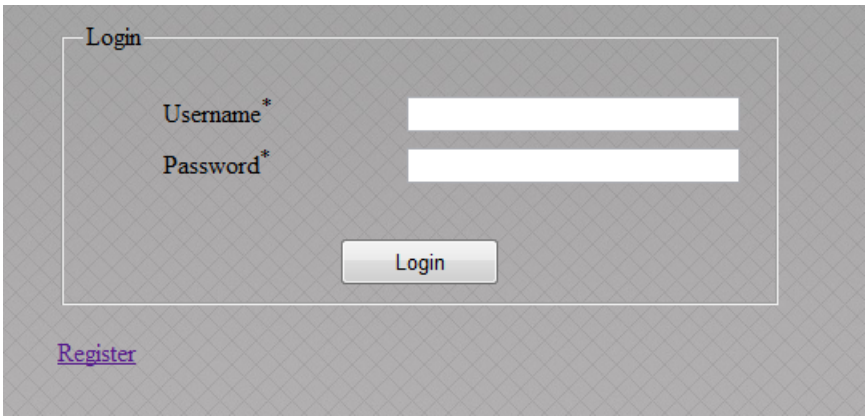
V aplikacijo se lahko uporabnik prijavi s klikom na gumb »Login« v zgornjem desnem kotu. Za prijavo sta potrebna uporabniško ime in geslo (slika 3.3). S prijavo imajo uporabniki (glede na dodeljene pravice) možnost dostopa do nekaterih delov aplikacij, ki so drugače nedostopni. Na primer možnost oddaje svojih algoritmov.

3.1.3 Odjava

Po končanem delu v spletni aplikaciji je priporočeno, da se uporabnik odjavi iz aplikacije. To lahko naredi s klikom na gumb »Logout« v desnem zgornjem kotu.

3.1.4 Profil

Do lastne strani s profilom lahko uporabnik dostopa s klikom na svoje uporabniško ime v zgornjem desnem kotu. Odpre se stran s podatki o uporabniku



Slika 3.3: Obrazec za prijavo

in povezavo do njegovih oddanih algoritmov (slika 3.4).

Slika 3.4: Profil uporabnika

3.1.5 Osnovni meni

Osnovni meni (slika 3.5) aplikacije je sestavljen iz naslednjih komponent: domov, o sistemu, pregled skupin algoritmov in oddaja algoritmov. Privile-

girani uporabniki oziroma administratorji imajo v osnovnem meniju dostop tudi do administratorjeve plošče.

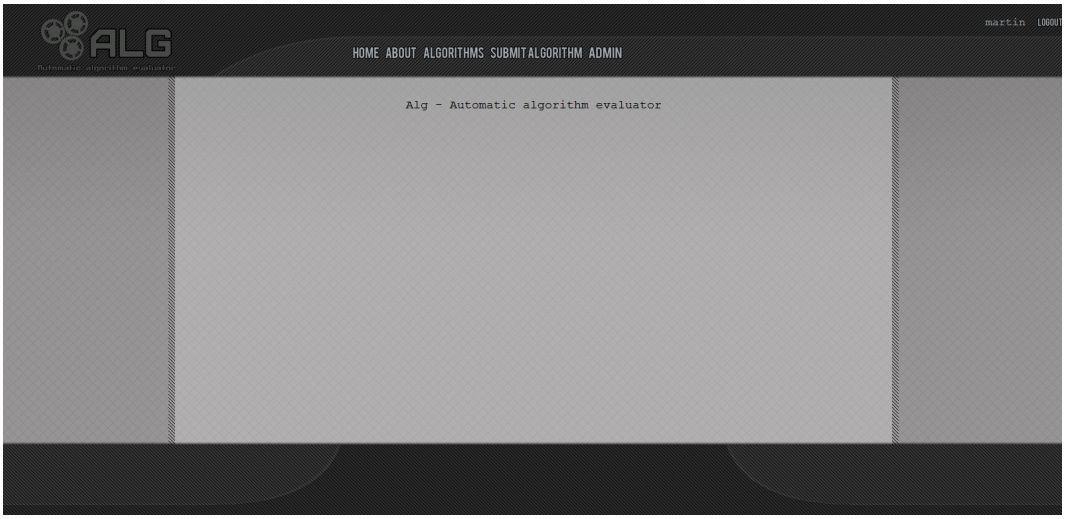
Slika 3.5: Osnovni meni

3.1.6 Domov

S klikom na gumb »Domov« se uporabniku odpre domača stran (slika 3.6).

3.1.7 O sistemu

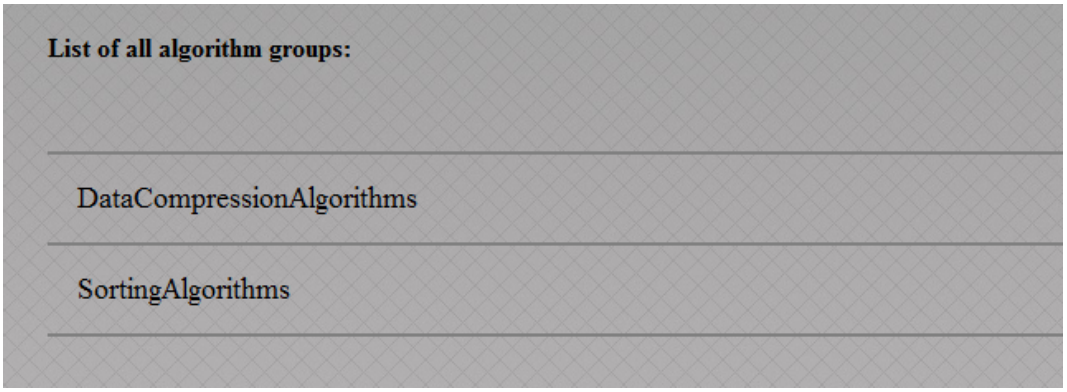
S klikom na gumb »About« se uporabniku odpre stran z informacijami o sistemu. Stran vsebuje kratek opis sistema z njegovimi osnovnimi podatki.



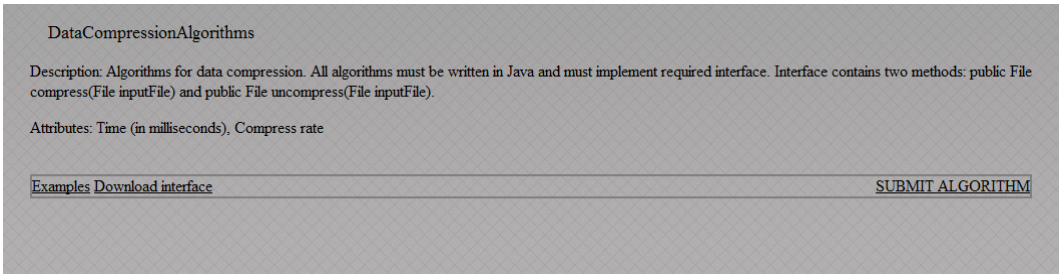
Slika 3.6: Domača stran

3.1.8 Pregled skupin algoritmov

Do seznama skupin algoritmov lahko pride uporabnik s klikom na gumb »Algorithms«. Odpre se stran s seznamom vseh skupin algoritmov urejenih v padajočem abecednem redu (slika 3.7). S klikom na ime skupine algoritmov se odpre nova stran (slika 3.8) z osnovnimi podatki in opisom, ter s tremi povezavami: primeri, vmesnik in oddaja algoritma. Povezava primeri (»Examples«) nudi dostop do vseh oddanih algoritmov, ki pripadajo tej skupini, povezava oddaja algoritma (»Submit algorithm«) pa dostop do spletnega obrazca za oddajo algoritma (obrazec ima polje skupina že nastavljeno). S klikom na povezavo »Interface« se prenese vmesnik, ki ga določa skupina algoritmov.



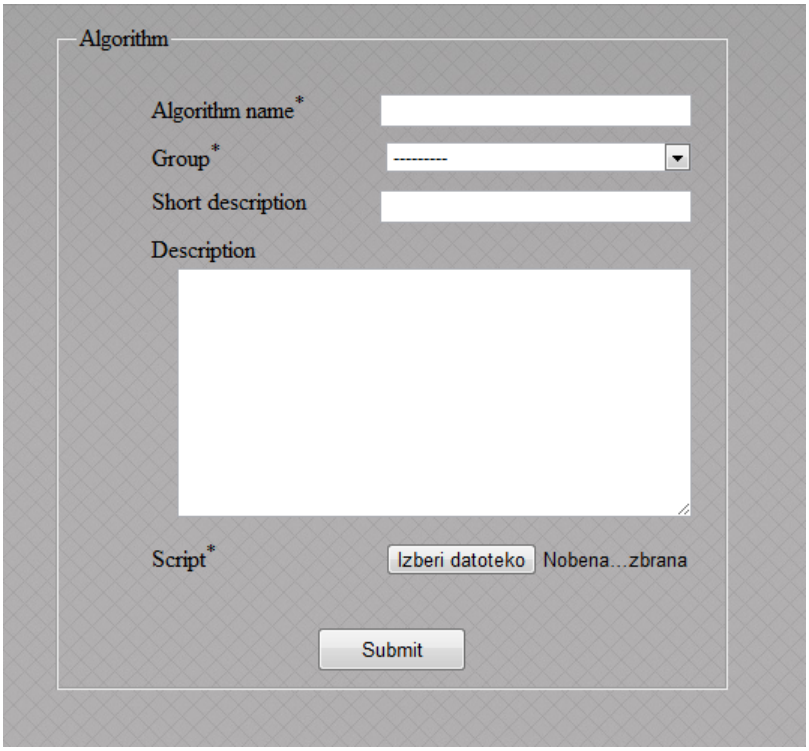
Slika 3.7: Pregled skupin algoritmov



Slika 3.8: Skupina algoritmov

3.1.9 Oddaja algoritmov

Oddaja algoritmov je možna na več načinov. S klikom na povezavo na strani izbrane skupine algoritmov (kot je opisano v prejšnjem odstavku) ali pa s klikom na gumb »Submit algorithm«. Oba načina sta dostopna le prijavljenim uporabnikom. Pri obeh načinih se odpre spletna stran z istim obrazcem (slika 3.9), le da je pri dostopu s povezavo s strani algoritma že izpolnjeno obvezno polje skupina z imenom izbrane skupine algoritmov. Poleg tega ima obrazec še dva obvezna polja: ime algoritma (ki mora biti enolično in sestavljeno le iz črk, številk ali podčrtajev ter polje za dodajanje datotek, kjer s klikom na gumb »Izberi datoteko« izberemo algoritem, ki ga želimo oddati. Obrazec ima nato še dve neobvezni polji, ki jih ni potrebno izpolniti za uspešno oddajo algoritma. To sta polji opis in kratek opis.



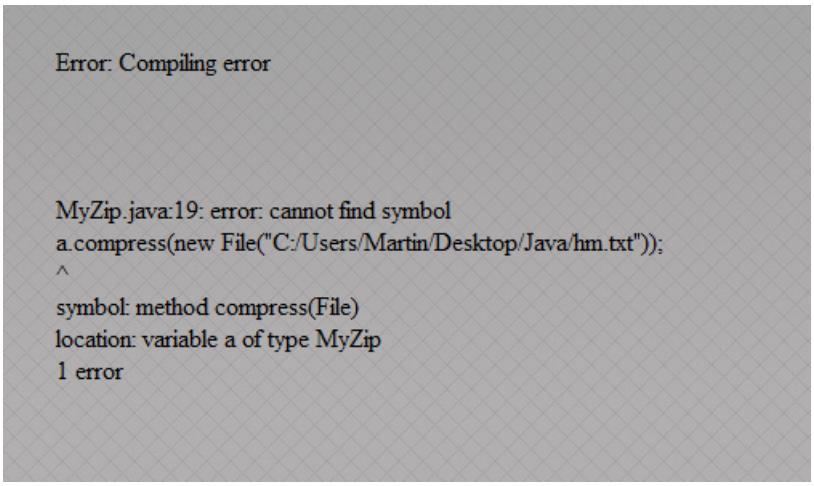
Slika 3.9: Obrazec za oddajo algoritmov

Zahteve za oddane algoritme

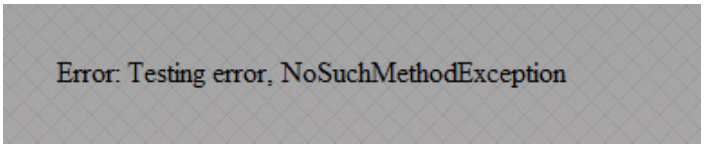
Oddani algoritmi morajo zadostiti vsem zahtevam opisanim v odstavku » 4.3 Zahteve za oddane algoritme«

Če oddani algoritem ne izpolnjuje danih zahtev (na primer ne implementira zahtevanih metod ali pa je sintaktično nepravilen), sistem avtomatsko javi napako. V tem primeru se oddani algoritem izbriše iz podatkovne baze, uporabniku pa se prikaže spletna stran z opisom napake (slika 3.10 in slika 3.11).

Ob uspešni oddaji in testiranju algoritma spletna stran avtomatsko preusmeri uporabnika na stran algoritma, katerega je oddal.



Slika 3.10: Napaka pri prevajanju programa



Slika 3.11: Napaka pri izvajanju algoritma

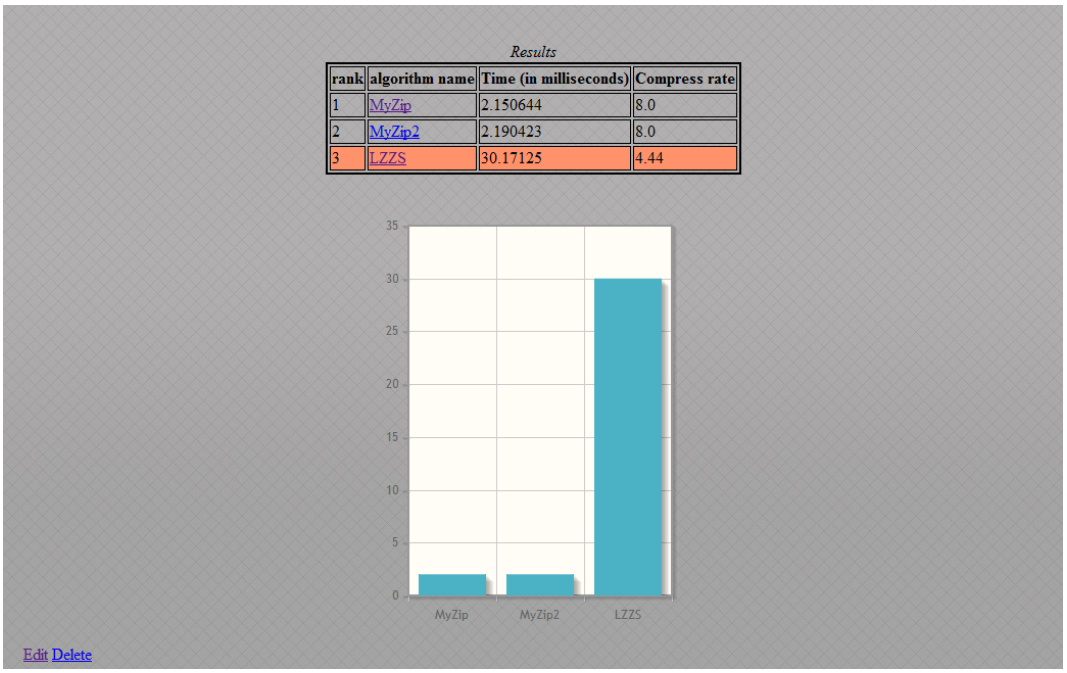
3.1.10 Stran algoritma

Do strani algoritma je možen dostop na več načinov. Prvi način je možen v pregledu skupin algoritmov, najprej s klikom na ime skupine, kateri želeni algoritem pripada, nato s klikom na povezavo »Examples« ter nazadnje s klikom na ime želenega algoritma. Drugi način zahteva klik na povezavo uporabnikovih oddanih algoritmov na strani s profilom uporabnika, ki je želeni algoritem oddal, ter klik na samo ime želenega algoritma. Tretji način predstavlja avtomatska preusmeritev ob uspešni oddaji algoritma, četrti pa kar direkten dostop preko naslova:

`/algorithms/<ime skupine algoritmov>/<id algoritma>/`

Stran vsebuje osnovne podatke o algoritmu (slika 3.12 in slika 3.13) ter njegove rezultate pri testiranju. Spodaj se prikažeta tudi razpredelnica z rezultati ostalih algoritmov iste skupine, ki je urejena glede na vrednost prvega kriterija v naraščajočem vrstnem redu in stolpičasti graf vseh algoritmov z vrednostmi prvega kriterija.

Slika 3.12: Stran algoritma



Slika 3.13: Rezultati

3.2 Navodila za administratorje

3.2.1 Dodajanje uporabniških in administratorskih računov

Uporabnike se dodaja tako, da se jih registrira (glej odstavek » 3.1.1 Registracija«). Za dodajanje novih administratorskih računov je potrebno obstoječim uporabniškim računom v podatkovni bazi spremeniti vrednost polja »is_superuser« na 1, kar se da storiti na več načinov. Najprej je potrebno zagnati strežnik SQL in nato izvršiti enega izmed naslednjih ukazov:

V primeru da poznamo uporabnikov id:

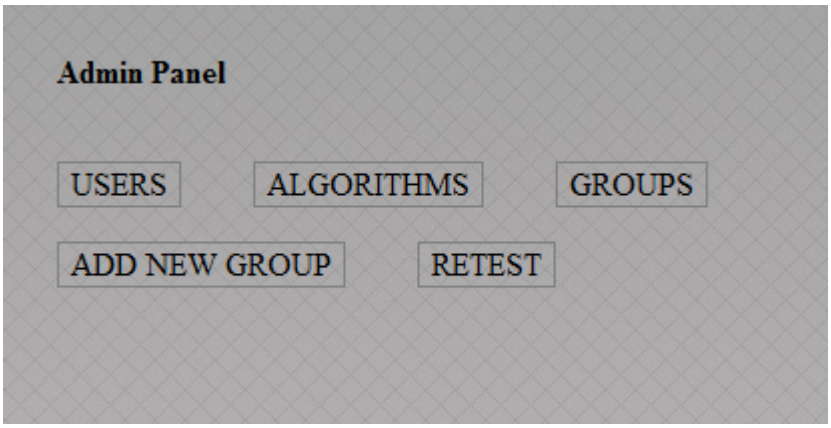
```
UPDATE <ime podatkovne baze>.auth_user SET is_superuser=1 WHERE
    id=<id uporabnika>;
```

V primeru da poznamo uporabnikovo uporabniško ime:

```
UPDATE <ime podatkovne baze>.auth_user SET is_superuser=1 WHERE
    username=<ime>;
```

3.2.2 Administratorjeva plošča

Do administratorjeve plošče lahko dostopajo le privilegirani uporabniki. S prijavo v spletno aplikacijo z administratorskim računom se administratorju v osnovnem meniju pojavi gumb »Admin«. S klikom nanj se odpre administratorjeva plošča (slika 3.14). Sestavljajo jo naslednje povezave: uporabniki, algoritmi, skupine algoritmov, dodaj novo skupino algoritmov in ponovno testiranje vseh algoritmov.



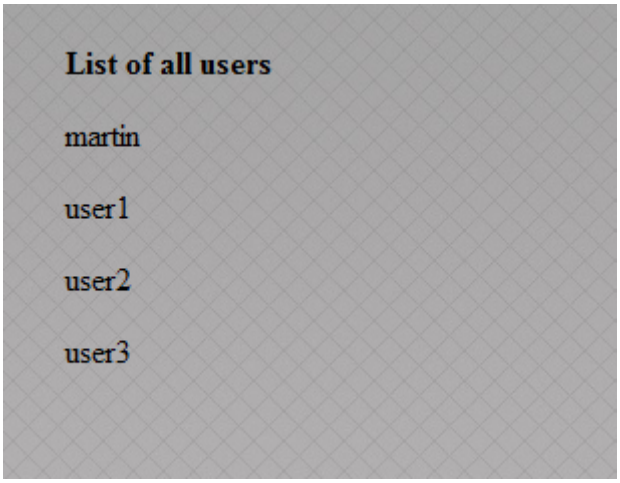
Slika 3.14: Administratorjeva plošča

3.2.3 Pregled uporabnikov

S klikom na povezavo »Users« se odpre spletna stran s seznamom vseh uporabnikov (slika 3.15). S klikom na uporabniško ime se odpre uporabnikova stran s profilom.

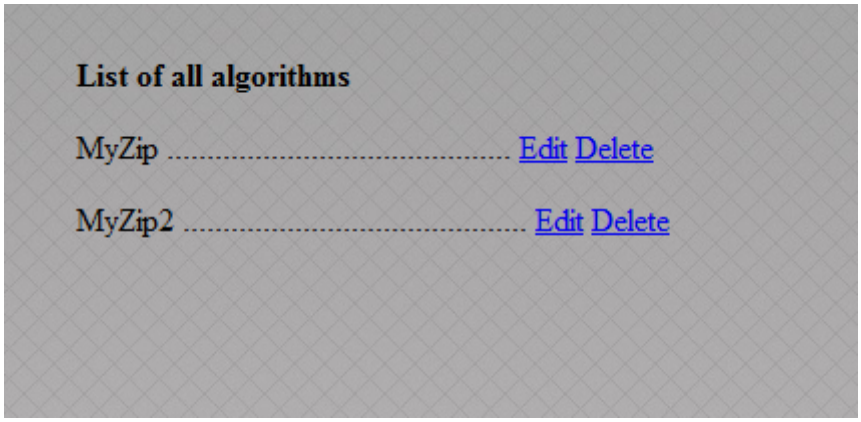
3.2.4 Pregled algoritmov

S klikom na povezavo »Algorithms« se odpre spletna stran s seznamom vseh algoritmov (slika 3.16). Pri vsakem algoritmu so na voljo tri povezave: povezava do algoritma ter povezavi uredi in izbriši. Za urejanje algoritmov služi povezava »Edit«. S klikom nanjo se odpre nova stran s spletnim obrazcem,



Slika 3.15: Pregled uporabnikov

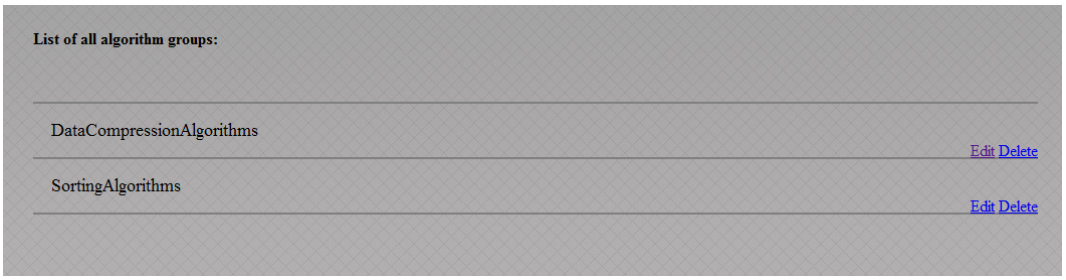
kjer je možno spremeniti polja kratek opis, opis in polje »Default algorithm«, ki označuje ali algoritem spada v skupino standardnih algoritmov. S klikom na povezavo »Delete« je možno izbrisati izbrani algoritem. Odpre se nova stran, kjer je potrebno potrditi izbris s klikom na gumb »Delete«.



Slika 3.16: Pregled algoritmov

3.2.5 Pregled skupin algoritmov

S klikom na povezavo »Groups« se odpre spletna stran »Pregled skupin algoritmov« (glej odstavek » 3.1.8 Pregled skupin algoritmov«). Administratorjem sta na voljo še dve dodatni povezavi: uredi in izbriši (slika 3.17). S klikom na povezavo »Edit« se odpre spletna stran z obrazcem, ki vsebuje enaka polja kot obrazec za dodajanje skupin algoritmov. Urejanje se zaključi s klikom na gumb »Edit«. Izbrano skupino algoritmov je možno izbrisati s klikom na povezavo »Delete«. Pred izbrisom je potrebno potrditi izbiro s klikom na gumb »Delete«. Pri izbrisu skupine algoritmov, se hkrati izbrišejo tudi vsi oddani algoritmi, ki pripadajo tej skupini algoritmov.



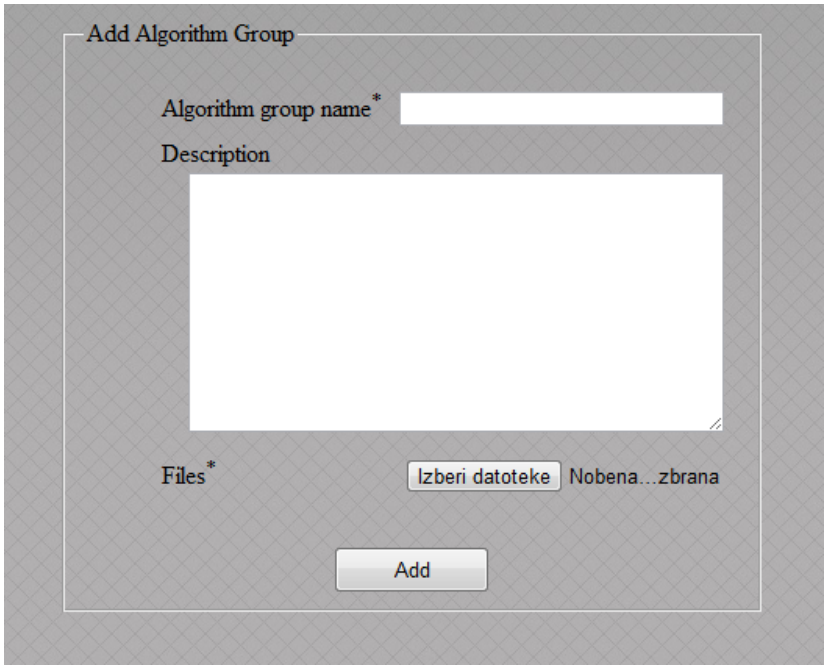
Slika 3.17: Administratorski pregled skupin

3.2.6 Dodajanje nove skupine algoritmov

S klikom na povezavo »Add new group« na administratorjevi plošči se odpre stran s spletnim obrazcem za dodajanje novih skupin algoritmov (slika 3.18). Potrebno je izpolniti dve obvezni polji: ime skupine algoritmov in polje za dodajanje datotek. Polje opis ni obvezno.

Pri vsakem dodajanju novih skupin algoritmov je potrebno oddati naslednje datoteke (primere vseh potrebnih datotek najdemo v mapi »ZipTest«):

- datoteka »meta.dat«;
- program za testiranje algoritmov;



Slika 3.18: Dodajanje nove skupine algoritmov

- morebitne testne datoteke, ki so potrebne za delovanje programa za testiranje algoritmov;
- vmesnik za algoritme.

Datoteka meta.dat

Datoteka »meta.dat« vsebuje osnovne podatke o skupini algoritmov. V prvih vrsticah mora vsebovati opise rezultatov ločene z novimi vrsticami. Nato mora slediti prazna vrstica in ime programa za testiranje algoritmov, ter nova prazna vrstica in ime vmesnika.

Primer datoteke meta.dat:

```
Time (in milliseconds)
Compress rate
```

`ZipTest.jar`

`Zip.java`

Program za testiranje algoritmov

Program, ki testira in ocenjuje oddane algoritme ter nato vpiše rezultate testiranja v podatkovno bazo. Zahteve katere mora izpolniti vsak program za testiranje algoritmov najdemo v odstavku » 4.2 Zahteve programa za testiranje algoritmov«.

Testne datoteke

Oddati je potrebno vse datoteke, ki jih program za testiranje algoritmov morebiti potrebuje.

Vmesnik

Vmesnik, ki definira vse potrebne metode, katere mora implementirati vsak oddani algoritem.

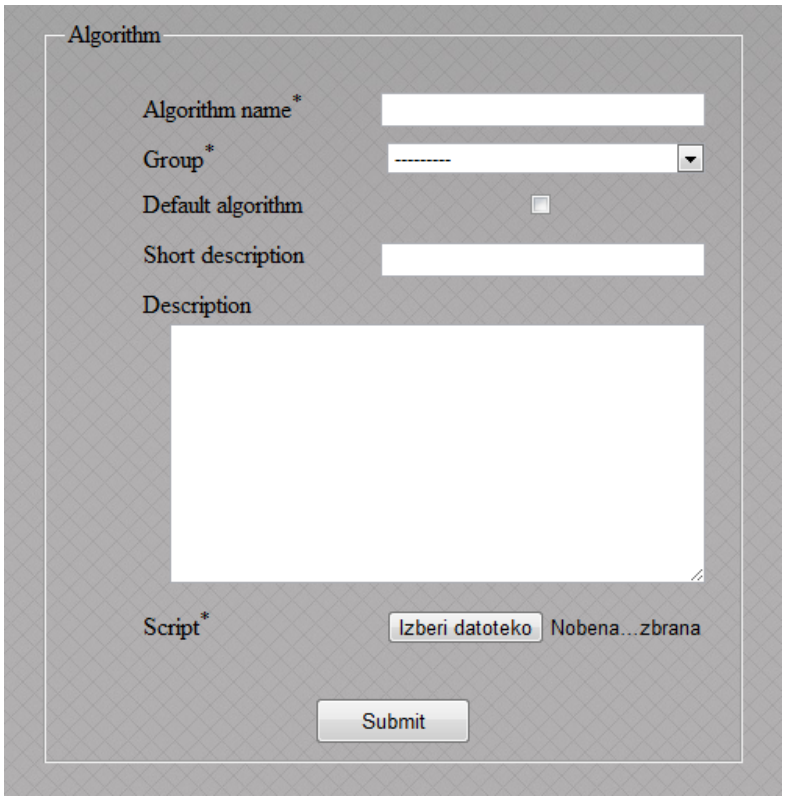
3.2.7 Ponovno testiranje vseh algoritmov

S klikom na povezavo »Re-Test« se odpre spletna stran za potrditev izbrane izbire. Administrator lahko potrdi izbiro s klikom na gumb »Re-Test«.

Ponovno testiranje algoritmov lahko traja dlje časa. Čas izvajanja je odvisen od števila in zahtevnosti vseh algoritmov.

3.2.8 Oddaja algoritmov z administratorskim računom

Uporabniki z administratorskimi pravicami oddajajo algoritme enako kot ostali uporabniki (glej odstavek » 3.1.9 Oddaja algoritmov«). Pri tem imajo še dodatno možnost, da oddani algoritem označijo za standardnega. To lahko storijo tako, da v spletnem obrazcu (slika 3.19) odkljukajo polje »Default algorithm«. Za administratorje prav tako veljajo enaka pravila glede priloženih datotek.



Slika 3.19: Administratorsko dodajanje algoritmov

Poglavje 4

OCENJEVANJE KAKOVOSTI ALGORITMOV

4.1 Postopek testiranja algoritmov

Za ocenjevanje kakovosti algoritmov v spletni aplikaciji skrbi poseben program imenovan »program za testiranje algoritmov«. Za vsako skupino algoritmov se ta program razlikuje od ostalih. Na ta način lahko ocenjujemo zelo različne vrste oziroma skupine algoritmov. Vse te programe pa mora sprogramirati in dodati v sistem administrator sam. Da to ne bi bila prezahtevna naloga mu je v pomoč abstraktni razred »Tester«, ki definira vse potrebne metode za pripravo algoritma na testiranje.

Samo testiranje oziroma ocenjevanje algoritmov poteka po vnaprej določenemu vrstnemu redu. Najprej uporabnik na spletni strani izpolni obrazec in doda svoj program oziroma algoritem ter klikne na gumb »Submit«. Spletna aplikacija nato preveri vsebino obrazca in če ne najde nobenih napak v podatkovno bazo shrani nov objekt tipa algoritem.

Sledi kopiranje vseh potrebnih datotek za delovanje programa za testiranje algoritmov v novo mapo, katere pot je določena v nastavitvah sistema. V to mapo se kopirajo vse datoteke, ki se nahajajo v mapi skupine algoritmov, kateri novi objekt pripada, uporabnikov program, ter datoteke iz mape

»additional_resources«.

Nato strežnik zažene nov sistemski proces v katerem se izvede tisti program za testiranje algoritmov, ki ga določa skupina algoritmov. Programu doda še prvi argument, ki je enak »id«-ju algoritma. Program za ocenjevanje algoritmov iz podatkovne baze prebere ime uporabnikovega programa, katerega tudi prevede. Če med prevajanjem ne pride do nobenih napak program naloži vse razrede, ki jih uporabnikov algoritem potrebuje. Ti razredi oziroma njihovi objekti se nato uporabijo v samem ocenjevanju kakovosti algoritma.

Do te stopnje, so bili vsi koraki enaki za vse programe različnih skupin algoritmov. Za vse te korake poskrbi že abstraktni razred »Tester« sam. Sledijo koraki, ki so lahko pri vsaki skupini algoritmov različni. Ti koraki predstavljajo samo testiranje algoritma, ki se odvija v posebni metodi test(), katero mora definirati vsak program za testiranje algoritmov. V tej metodi lahko skrbnik sistema implementira želeno testiranje. Testiranje je lahko kakršnokoli, edina omejitev je število kriterijev oziroma rezultatov, ki ocenjujejo nek algoritem. Na voljo ima največ deset številskih mest za rezultate. Vse načine testiranja pa mora opisati oziroma sprogramirati sam. Na koncu mora zapisati rezultate v pravilnem vrstnem redu, da ti sovpadajo z opisi rezultatov, ki so nastavljeni ob ustvaritvi skupine algoritmov.

Primer takšne implementacije testiranja je program »ZipTest« (priloga 7.1). Program je namenjen testiranju algoritmov za brezizgubno stiskanje podatkov. Program najprej preusmeri vse podatkovne tokove v začasno datoteko. To stori zato, da na strežnikov standardni izhod ne prihajajo stvari iz uporabnikovega programa.

Nato s pomočjo naloženega objekta algoritma katerega testira, pokliče njegovo metodo »compress«, katera predstavlja samo stiskanje podatkov. Kot argument doda vnaprej določeno testno datoteko, pred tem pa si še zabeleži čas. Prav tako si zabeleži čas ob koncu izvajanja metode in tako dobi čas izvajanja algoritmovega stiskanja podatkov. Nato preveri velikost izhodne datoteke uporabnikove metode »compress« in s tem izračuna stopnjo

stiskanja. Potem pokliče še algoritmovo metodo »uncompress«, ter preveri če se izhodna datoteka ujema z originalno datoteko.

Na koncu zapiše stanje in rezultate v podatkovno bazo. V polje status zapiše »Done« če se je izvajanje uspešno izvedlo in če algoritem ustreza vsem pogojem. V nasprotnem primeru v polje status zapiše napako do katere je prišlo med izvajanjem. Nato v polje result1 vpiše čas izvajanja stiskanja podatkov, v polje result2 pa stopnjo stiskanja.

S tem se izvajanje programa za testiranje algoritmov zaključi. Zatem spletna aplikacija najprej preveri izhodno stanje, ki ga vrne podproces. Če je prišlo do notranje napake med izvajanjem samega programa, aplikacija izbriše objekt algoritem in vse njegove datoteke na strežniku, uporabniku pa prikaže spletno stran z obvestilom o napaki. Če pa je prišlo do napake med prevajanjem ali pa algoritem ni izpolnjeval vseh zahtev, aplikacija prav tako izbriše objekt in datoteke, ter prikaže spletno stran z opisom napake. Morebitno napako aplikacija odkrije z vpogledom v polje status v podatkovni bazi.

V primeru, da se je izvajanje programa uspešno zaključilo in da je testirani algoritem izpolnil vse zahteve, se uporabniku prikaže stran z rezultati (odstavek » 3.1.10 Stran algoritma«). Aplikacija iz podatkovne baze prebere vsa polja z rezultati algoritmov, ki pripadajo isti skupini algoritmov kot oddani algoritem ter jih prikaže v razpredelnici. Rezultati so urejeni glede na vrednost prvega kriterija v naraščajočem vrstnem redu. Algoritmi, ki pripadajo zbirki standardnih algoritmov so obarvani rdeče. Spodaj pod razpredelnico se nahaja tudi stolpičasti graf, ki prikazuje vrednosti prvega kriterija vseh algoritmov, ki pripadajo isti skupini algoritmov.

4.2 Zahteve programa za testiranje algoritmov

Program mora biti v napisan v programskem jeziku Java in oddan v formatu ».jar«. Program mora razširiti abstraktni razred Tester (datoteka »Te-

ster.java≪ je dostopna v mapi »Alg/additional_resources≪) in mora definirati abstraktno metodo test().

Metoda test() je glavna metoda sistema saj v njej poteka testiranje in ocenjevanje drugih algoritmov. V tej metodi je potrebno klicati metode algoritmov, ki se testirajo in jih kasneje tudi natančno opisati v vmesniku, katerega je potrebno priložiti programu. Poskrbeti je potrebno za pravilno delovanje teh metod, ter ocenit samo kakovost izvajanja (na primer: čas izvajanja ali poraba pomnilnika). Na koncu je potrebno nastaviti atributa »results≪ in »status≪.

V pomoč metodi test() so na voljo naslednje metode abstraktnega razreda Tester: statusError(), getId(), getCompileResult(), getStatus(), getAlgClass(), getAlgObject(), getRootDir(), setResults(String[] results) in setStatus(String[] status). Vse naštete metode so opisane v odstavku » 5.5.1 Abstraktni razred Tester≪.

V programu je potrebno definirati tudi konstruktor, ki sprejme niz »id≪, ter v njem kliče metodo super(id). Prav tako je potrebno definirati metodo main() v kateri mora program ustvariti objekt svojega tipa z atributom »id≪ katerega program dobi kot prvi argument metode main(). Preveriti je potrebno, če so vsi atributi pravilno nastavljeni. To se da preveriti s klicem metode t.statusError(), ki vrne »False≪, če ni prišlo do nobenih napak. Nato je potrebno poklicati metodi test() in writeResults().

Prav tako je v programu potrebno poskrbeti za vse možne izjeme in napake, ki bi lahko nastale med testiranjem algoritmov. Priporočljivo je, da v metodi test() nastavimo maksimalen čas izvajanja algoritma, katerega lahko potem tudi prekinemo če je to potrebno. Prav tako je tudi priporočljivo, da pred samim začetkom izvajanja programa preusmerimo podatkovne tokove, saj v nasprotnem primeru, če karkoli zaide na standardni izhod, sistem javi napako: »Internal error≪.

Primer programa za testiranje algoritmov, ki izpolnjuje vse dane zahteve najdemo v prilogi (priloga 7.1).

Za pravilno delovanje razreda »Tester≪ je potrebno programu dodati

knjižnico »MYSQL Connector J«, ki je dostopna v mapi »Alg/additional_resources/lib«.

4.3 Zahteve za oddane algoritme

Oddani program, ki definira uporabnikov algoritem mora biti napisan v programskem jeziku Java, ter mora biti oddan v eni sami datoteki tipa ».java« , ki ne sme biti večja od maksimalne predpisane veličine.

Program mora implementirati vmesnik, ki je natančno določen pri vsaki skupini algoritmov in je dostopen ter natančno opisan na strani vsake skupine algoritmov. To pomeni, da mora uporabnik v svojem programu natančno definirati vse metode, katere predpisuje dani vmesnik. Te metode natančno opisujejo tudi vhodne in izhodne podatke algoritma. V teh metodah mora uporabnik implementirati vse korake svojega algoritma.

Primer zelo preprostega algoritma za brezizgubno stiskanje podatkov, ki zadostuje vsem zahtevam, ki jih določa vmesnik te skupine algoritmov, najdemo v mapi »MyZip« (datoteka »MyZip.java«).

Poglavje 5

IMPLEMENTACIJA

5.1 Splošno o aplikaciji

Spletna aplikacija je večinoma napisana v programskem jeziku Python in deloma v programskem jeziku Java. Strežniški del, ki skrbi za izmenjavo spletnih zahtev sloni na ogrodju Django in je v celoti napisan v programskem jeziku Python. Del, ki testira same algoritme pa je napisan v programskem jeziku Java. Oba dela sistema uporabljata relacijsko podatkovno bazo MySQL.

5.2 O Djangu

Django je odprtokodno ogrodje za izdelovanje spletnih aplikacij napisano v programskem jeziku Python. Namenjen je poenostavitvi izdelovanja zapletenih spletnih aplikacij.

Django zagovarja hiter razvoj, programiranje s čim manj programske kode in s čim manj ponavljanja, čim večjo abstrakcijo, ter popolno neodvisnost med posameznimi plastmi ogrodja.

Ogrodje bazira na arhitekturi MVC (model-view-controller) oziroma model-pogled-krmilnik, vendar pa Django uporablja malce drugačno poimenovanje: MTV (model-template-view) oziroma model-predloga-pogled.

Pogled predstavlja povratno funkcijo, ki točno določenim naslovom URL preslika podatke. Ti podatki se potem prikažejo v obliki katero določi predloga. Model pa opisuje, kako so podatki shranjeni. Vse podatke lahko predstavimo kot objekte v programskem jeziku Python, saj Django avtomatsko nudi preslikavo med Pythonovimi objekti in relacijskimi bazami.

5.3 Podatkovna baza

Program uporablja Djangove tabele za avtentikacijo in sejo. Glavni tabeli v sistemu pa predstavljata tabeli algoritem in skupina algoritmov. Shema baze je predstavljena v sliki 5.1.

5.3.1 Algoritem

Tabela algoritem vsebuje polja:

- primarni ključ »id«;
- časovno polje datum oddaje algoritma;
- ime algoritma;
- tuji ključ avtor algoritma (kaže na tabelo uporabnikov);
- tuji ključ skupina algoritma (kaže na tabelo skupine algoritmov);
- polje ki označuje standardne algoritme;
- kratek opis;
- opis;
- status;
- pot do naložene datoteke;
- deset številskih polj za rezultate.

5.3.2 Skupina algoritmov

Tabela skupina algoritmov vsebuje polja:

- primarni ključ »id«;
- ime skupine;
- opis;
- polje za naložene datoteke;
- ime testnega programa;
- ime vmesnika;
- deset črkovnih polj za opise rezultatov.

5.4 Implementacija strežnika

Kot že omenjeno je strežniški del implementiran s pomočjo ogrodja Django. Programiranje sistema smo razdelili na štiri dele (»apps«): »frontend«, »account«, »admin« in »algorithm«.

Front end mapa »Alg/alg/frontend«

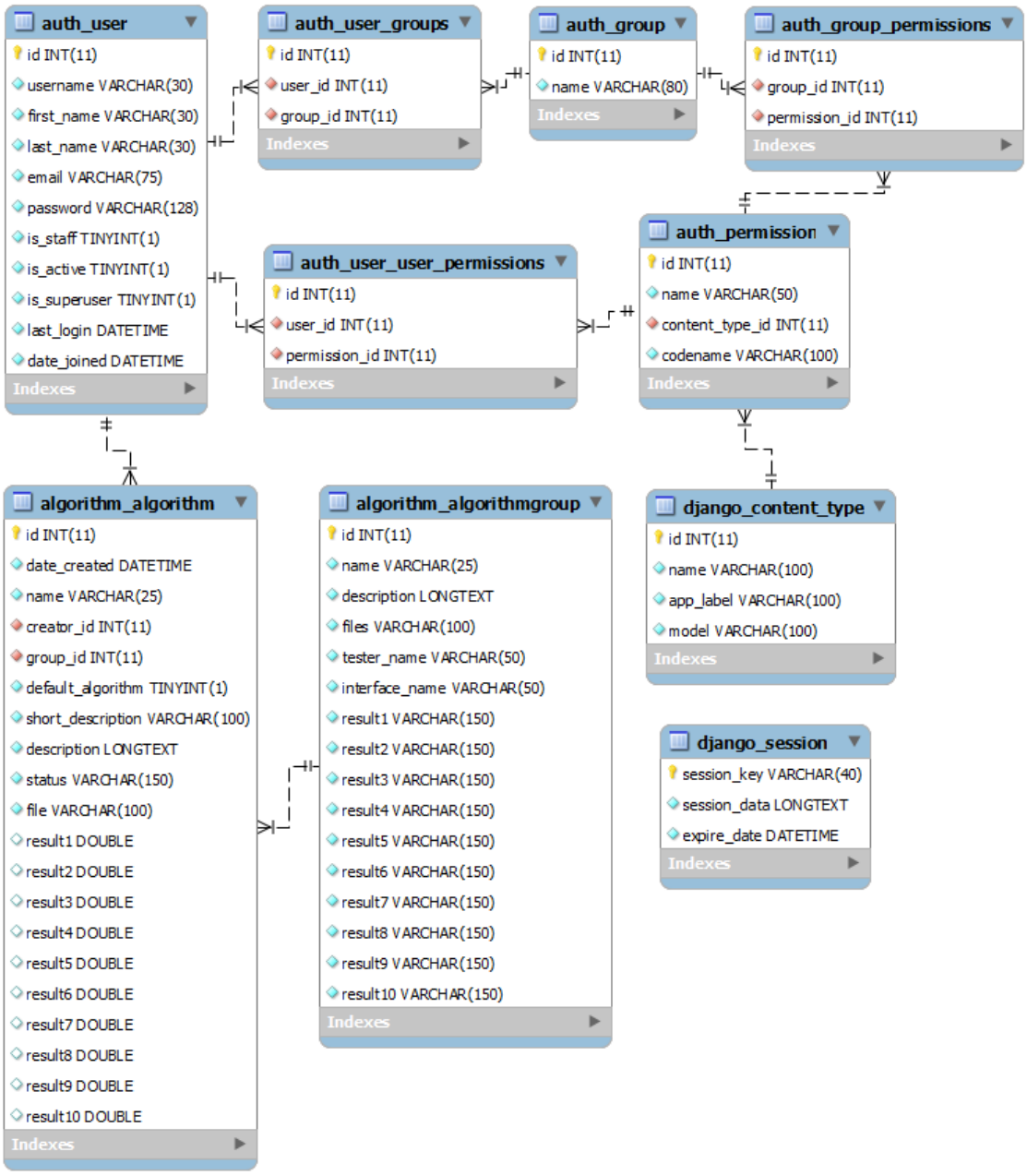
Skrbi za zunanjo podobo spletne aplikacije. Definira osnovni izgled strani ter vsebuje pogleda za domačo stran in stran o sistemu. Hrani tudi vse statične datoteke.

Pogledi (datoteka »views.py«):

- class HomeView - pogled za domačo stran;
- class AboutView - pogled za stran o sistemu.

Predloge (mapa »templates«):

- plain - osnovni izgled strani. Vse strani razširjajo to datoteko;



Slika 5.1: Shema baze

- forms - izgled spletnih obrazcev;
- home in about - izgled domače strani in strani o sistemu.

Vsebuje tudi mapo »static« s statičnimi datotekami kjer se nahajajo mape: Slike, datoteke CSS, datoteke Javascript in mapa z vmesniki.

Account mapa »Alg/alg/account«

Skrbi za registracijo in uporabnikovo stran s profilom. Za prijavo in odjavo skrbi Django sam. Definira izglede uporabnikove strani, registracijo, prijavo in odjavo.

Pogledi (datoteka »views.py«):

- class RegistrationView - pogled, ki preverja spletne obrazce za registracijo in ustvarja nove uporabnike v podatkovni bazi;
- class ProfileView - pogled za uporabnikovo stran s profilom.

Predloge (mapa »templates«):

- user - izgled uporabnikove strani;
- registration, login - izgled strani registracija in prijava (razširja form).

Datoteka »forms.py«:

- class RegistrationForm - razred, ki ustvari spletni obrazec za registracijo.

Admin mapa »Alg/alg/admin«

Skrbi za administracijo sistema. Vsebuje poglede administratorjeve plošče in skrbi za dodajanje novih skupin algoritmov.

Pogledi (datoteka »views.py«):

- class MainView - pogled za stran z administratorjevo ploščo;
- class UserListView - pogled, ki v podatkovni bazi poišče vse uporabnike;

- class AlgorithmListView - pogled, ki v podatkovni bazi poišče vse algoritme;
- class AddGroupView - pogled, ki preverja spletne obrazce za dodajanje novih skupin algoritmov in ustvarja nove skupine;
- ReTestView - pogled za ponovno testiranje algoritmov. Kliče metodo reTest();
- setInterface(group,name) - metoda, ki ustvari novo mapo z vmesnikom skupine v mapi »static/Interfaces«;
- setAdditionalInfo(id) - metoda, ki v podatkovni bazi nastavi ime programa za testiranje algoritmov in ime vmesnika;
- reTest() - metoda, ki na novo testira vse algoritme.

Predloge (mapa »templates«):

- admin_home - izgled strani z administratorjevo ploščo;
- add_new_group - izgled strani za dodajanje novih algoritmov (razširja file_form).

Datoteka »forms.py«:

- class AddGroupForm - razred, ki ustvari spletni obrazec za dodajanje novih skupin algoritmov.

Algorithm mapa »Alg/alg/algorithm«

Skrbi za delovanje aplikacije z algoritmi. Definira objekta algoritem in skupina algoritmov. Prav tako skrbi za klicanje programov za testiranje algoritmov. Definira izglede strani algoritmov in predstavo rezultatov v razpredelnicah in grafih.

Modeli (datoteka »models.py«):

- class Algorithm - model, ki predstavlja algoritem (vsebovana polja so opisana v pododstavku » 5.3.1 Algoritem« odstavka » 5.3 Podatkovna baza«);
- class AlgorithmGroup - model, ki predstavlja skupino algoritmov (vsebovana polja so opisana v pododstavku » 5.3.2 Skupina algoritmov« odstavka » 5.3 Podatkovna baza«);
- class RestrictedFileField - Polje za dodajanje datotek z omejitvami. Razred tudi definira vse omejitve;
- setData() - metoda, ki pokliče program za testiranje algoritmov;
- get_upload_folder(instance, filename) - metoda, ki vrne pot do mape, kamor se shranjujejo naložene datoteke;
- copyData(path1, path2, path3, destination) - metoda, ki skopira vse datoteke v mapah path1, path2 in path 3 v mapo destination.

Pogledi (datoteka »views.py«):

- class MainView - pogled, ki v podatkovni bazi poišče vse skupine algoritmov;
- class groupView - pogled za skupino algoritmov;
- class groupListView - pogled, ki v podatkovni bazi poišče vse algoritme, ki pripadajo določeni skupini algoritmov;
- class SubmitAlgorithmView - pogled, ki preverja spletne obrazce za dodajanje novih algoritmov in ustvarja nove algoritme;
- class UserAlgorithmsView - pogled, ki v podatkovni bazi poišče vse algoritme, ki pripadajo določenemu uporabniku;
- class AlgorithmView - pogled, ki v podatkovni bazi poišče vse algoritme, ki pripadajo isti skupini kot določeni algoritem;

- class EditGroupView - pogled za urejanje skupin algoritmov (preverja spletne obrazce in ureja skupine);
- class EditAlgorithmView - pogled za urejanje algoritmov (preverja spletne obrazce in ureja algoritme);
- class DeleteAlgorithmView - pogled za izbris algoritmov;
- class DeleteGroupView - pogled za izbris skupin algoritmov;
- metoda get_content() - metoda, ki iz datoteke »error.log« prebere napake in jih vrne;
- deleteDirectory(name) - metoda, ki izbriše mapo algoritma z imenom podanim kot argument;
- rename(oldname, newname) - metoda, ki preimenuje mapo skupine algoritmov iz starega imena v novega, podanega kot drugi argument.

Predloge (mapa »templates«):

- algorithm - izgled strani algoritma, vključno s tabelami in grafi;
- group - izgled strani skupina algoritmov;
- error - izgled strani, ki prikazuje napake.

Datoteka »forms.py«:

- class SubmitForm - razred, ki ustvari spletni obrazec za dodajanje novih algoritmov.

Datoteka »results.py« (nahaja se v mapi »templatetags«):

- getAtributes(context) - metoda, ki vrne vse opise rezultatov določene skupine;
- get_results(context) - metoda, ki vrne vse rezultate določenega algoritma;

- get_table(context) - metoda, ki vrne tabelo z rezultati vseh algoritmov ;
- get_algorithm_url(name) - metoda, ki vrne povezavo do določenega algoritma.

Poleg tega aplikacija vsebuje še datoteke:

- settings.py - datoteka z nastavitvami sistema;
- urls.py - datoteka, ki vsebuje spremenljivko »urlpatterns«, ki definira vse možne naslove URL;
- wsgi.py - konfiguracijska datoteka za delovanje s strežnikom Apache.

5.5 Implementacija testiranja algoritmov

Za samo testiranje algoritmov skrbi poseben program napisan v programskem jeziku Java. Za vsako skupino algoritmov se ta program razlikuje od ostalih, vsi pa so podrazredi abstraktnega razreda »Tester«. Programi morajo implementirati abstraktno metodo test(), konstruktor in metodo main(), kot je zapisano v odstavku » 4.2 Zahteve programa za testiranje algoritmov«. Za vse ostalo poskrbi »Tester« sam.

5.5.1 Abstraktni razred Tester

datoteka »Tester.java« (ki se nahaja v mapi »Alg/additional_resources«).

Razred tester vsebuje atribute: id, scriptName, compileResult, status, algClass, algObject, results, rootDir, jdkPath, dbName, username, password in url.

Razred Tester vsebuje javne metode:

- public Tester(String id)- konstruktor, ki iz datoteke »setting.dat« nastavi atribute potrebne za povezavo s podatkovno bazo. Nato glede na

dani »id« v podatkovni bazi poišče ime programa ter ga na koncu tudi prevede ter nastavi atribute algClass in algObject. Če med izvajanjem metode ni prišlo do napake na koncu zapiše v atribut status vrednost »OK«;

- public boolean statusError() - metoda, ki vrne »True« če je vrednost atributa status »OK«, če ni, vrne »False«;
- public void writeResults() - metoda, ki zapiše vrednosti polja results in status v podatkovno bazo;
- public String getId() - metoda, ki vrne vrednost atributa id;
- public int getCompileResult() - metoda, ki vrne vrednost atributa compileResult, če je vrednost atributa 0 pomeni, da se je prevajanje zaključilo brez napak;
- public String getStatus() - metoda, ki vrne vrednost atributa status;
- public Class getAlgClass() - metoda, ki vrne razred testirajočega algoritma;
- public Object getAlgObject() - metoda, ki vrne objekt testirajočega algoritma;
- public String getRootDir() - metoda, ki vrne vrednost atributa rootDir;
- public void setStatus(String status) - metoda, ki nastavi vrednost atributa status glede na podani argument;
- public void setResults(String[] results)- metoda, ki nastavi vrednost atributa results glede na podani argument.

privatne metode:

- private String getScriptName() - metoda, ki v bazi preveri ime algoritma, ki se trenutno testira in ga na koncu tudi vrne;

- private Connection connectToDb() - metoda, ki se poveže s podatkovno bazo in vrne povezavo;
- private int compileScript() - metoda, ki prevede program algoritma, ki se trenutno testira. Če se prevajanje zaključi brez napak vrne 0, sicer pa neničelno število, ter hkrati zapiše napako v datoteko »error.log«;
- private Class getAlgoClass() - metoda, ki naloži in vrne vse razrede testirajočega programa. Razrede najde preko metode findClasses() in jih naloži z uporaba razreda MyClassLoader;
- private List¡String¿ findClasses() - metoda, ki poišče vse datoteke tipa ».class«.

in abstraktno metodo:

- abstract void test() - abstraktna metoda, ki dejansko poskrbi za ocenjevanje algoritmov. Definirati jo morajo vsi podrazredi razreda Tester.

Datoteka »Tester.java« prav tako vsebuje tudi razred »MyClassLoader«, ki je podrazred razreda java.lang.ClassLoader.

Vsebuje metodi:

- public MyClassLoader(String loadFolder, List¡String¿ restrict) - konstruktor, ki nastavi atributa loadFolder in restrict;
- public Class loadClass(String name) - metoda, ki naloži razred z imenom podanim kot argument te metode.

5.5.2 program ZipTest

datoteka »ZipTest.java« (ki se nahaja v mapi »ZipTest«)

ZipTest je podrazred razreda Tester, ki testira in ocenjuje algoritme za brezizgubno stiskanje podatkov. Med testiranjem preveri, če je izhodna datoteka razširjanja enaka originalni datoteki, ter nato poda hitrost (v mili sekundah) in stopnjo stiskanja. Program je priložen v prilogi 7.1.

ZipTest vsebuje metode:

- public ZipTest(String id) - konstruktor;
- public static void main(String[] args) - metoda main, ki ustvari objekt tipa ZipTest z atributom id podanim kot prvim argumentom. Nato spremeni podatkovne tokove za standardni izhod in standardni izhod za napake. Zatem pokliče metodo test(), ponastavi podatkovne tokove, ter na koncu pokliče metodo writeResults(), ki zapiše rezultate v podatkovno bazo;
- void test() - metoda, ki oceni dani algoritem za stiskanje (katerega program testira). Najprej pokliče algoritmovo metodo »compress()« in meri čas njenega izvajanja zatem pa še izmeri dolžino dobljene datoteke. Nato pokliče algoritmovo metodo »uncompress()«, ter preveri, če sta dobljena datoteka in originalna datoteka enaki. Če se postopek zaključi brez napak metoda nastavi atribut results z rezultati testiranja (čas izvajanja in stopnja stiskanja), ter atribut status na »Done«. Ob napaki (na primer če metoda »uncompress« ne obstaja, ali pa vrne napačno datoteko) pa metoda v atribut status zapiše opis napake;
- private static boolean sameFile(File testFile, File outputFile) - metoda, ki vrne »True« če sta podani datoteki v argumentu enaki. Če nista metoda vrne »False«.

Poglavje 6

SKLEPNE UGOTOVITVE

Definirali smo pojem algoritem, ter predstavili, kako smo ga poenostavili in implementirali v našem sistemu, ter ga nato uporabili pri samem testiranju in ocenjevanju.

Ustvarili smo sistem, ki lahko testira najrazličnejše vrste algoritmov. Uspešno smo realizirali vse zastavljene cilje. Sistem je dovolj robusten in prilagodljiv, kar pa ne vpliva na zapletenost uporabe. Spletna aplikacija je tako enostavna in prijazna do uporabnikov. Sistem je zaradi zmožnosti ponovnega testiranja vseh algoritmov hkrati tudi prenosljiv.

Implementirali smo program za testiranje in ocenjevanje kakovosti algoritmov za brezizgubno stiskanje podatkov in ga tudi uspešno umestili v sam sistem.

Poglavje 7

PRILOGE

7.1 Program ZipTest

Program ZipTest za testiranje brezizgubnih algoritmov za stiskanje.

Priloga 7.1: ZipTest

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.NoSuchElementException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Martin
 */

public class ZipTest extends Tester {
```

```
// Every program for algortihm testing must define
   constructor containing "super(id)"
public ZipTest(String id) {
    super(id);
}

public static void main(String[] args) {
    // Every  program for algortihm testing must create its
       own object and then check for status errors. If there
        are no errors, program must then call method test()
       and at the end method writeResults()
    // It is also advisable to redirect output streams (
       Internal error will raise, if anything comes to
       output stream during the program execution).
    PrintStream origOut = System.out;
    PrintStream origErr = System.err;
    ZipTest t = new ZipTest(args[0]);
    if (!t.statusError()) {
        try {
            PrintStream stdout = new PrintStream(new
               FileOutputStream(t.getRootDir() + "output.log
               "));
            PrintStream stderr = new PrintStream(new
               FileOutputStream(t.getRootDir() + "error.log
               "));
            System.setOut(stdout);
            System.setErr(stderr);
        } catch (FileNotFoundException ex) {
            Logger.getLogger(ZipTest.class.getName()).log(
               Level.SEVERE, null, ex);
        }
        t.test();
        System.setOut(origOut);
        System.setErr(origErr);
    }
    t.writeResults();
}
```

```
@Override
void test() {
    String[] res = new String[11];
    String stat = "Error: Testing error";
    double end = -1;
    long rate2 = 8;
    try {
        File testFile = new File(getRootDir() + "test.txt");
        Method m = getAlgClass().getDeclaredMethod("compress
            ", File.class);
        long start = System.nanoTime();
        Object methodOutput = m.invoke(getAlgObject(),
            testFile);
        end = ((double)(System.nanoTime() - start))/1000000;
        long length1 = testFile.length();
        long length2 = length1;
        File outputFile = (File) methodOutput;
        length2 = outputFile.length();
        Method m2 = getAlgClass().getDeclaredMethod("
            uncompress", File.class);
        Object methodOutput2 = m2.invoke(getAlgObject(),
            outputFile);
        File outputFile2 = (File) methodOutput2;
        stat = "Done";
        if (!sameFile(testFile, outputFile2)) {
            stat = "Error: Testing error, outputfile is not
                the same";
        }
        outputFile.delete();
        outputFile2.delete();
        long rate = (length1 - length2) * 100 / length1;
        rate2 = (length2 * 8) / length1;
    } catch (IllegalAccessException ex) {
        stat = "Error: Testing error, IllegalAccessException
            ";
    } catch (IllegalArgumentException ex) {
        stat = "Error: Testing error, IllegalArguments";
    } catch (InvocationTargetException ex) {
```

```
            stat = "Error: Testing error,
                InvocationTargetException";
        } catch (NoSuchMethodException ex) {
            stat = "Error: Testing error, NoSuchMethodException
                ";
        } catch (SecurityException ex) {
            stat = "Error: Testing error, SecurityException";
        } catch (NullPointerException ex) {
            stat = "Error: Testing error, no output file";
        }
        res[1] = Double.toString(end);
        res[2] = Long.toString(rate2);
        setResults(res);
        setStatus(stat);
    }

    private static boolean sameFile(File testFile, File
        outputFile) {
        try {
            Scanner sc = new Scanner(testFile);
            Scanner sc2 = new Scanner(outputFile);
            while (sc.hasNextLine()) {
                try {
                    if (!sc.nextLine().equals(sc2.nextLine())) {
                        sc.close();
                        sc2.close();
                        return false;
                    }
                } catch (NoSuchElementException ex) {
                    return false;
                }
            }
            sc.close();
            sc2.close();
        } catch (FileNotFoundException ex) {
            Logger.getLogger(ZipTest.class.getName()).log(Level.
                SEVERE, null, ex);
        }
```

```
        return true;
    }
}
```

7.2 Kazalo slik

LITERATURA

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to algorithms, third edition, Massachusetts: The MIT Press, 2009

[2] R. Sedgewick, K. Wayne. Algorithms, fourth edition, Massachusetts: Pearson Education, 2011

[3] The Django Book. Dostopno na: `http://www.djangobook.com/`