UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Grošelj

Optimizacija sestave športnih obrokov glede na vsebovanost hranil

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Zoran Bosnić
Somentor: pred. dr. Boštjan Slivnik

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: 00321/2012
Datum: 03.09.2012

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

Kandidat: **ANDREJ GROŠELJ**

Naslov: **OPTIMIZACIJA SESTAVE ŠPORTNIH OBROKOV GLEDE NA VSEBOVANOST HRANIL**

OPTIMIZING THE NUTRIENT CONTENT OF SPORT MEALS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Zaradi športa in zdravja mora veliko ljudi skrbeti za pravilno sestavo obrokov, ki naj bi uravnoteženo vsebovali vse tri skupine makrohranil (beljakovine, ogljikovi hidrati in maščobe). Čeprav na spletu obstaja množica javnodostopnih baz podatkov o sestavi živil, je vsakdanjim uporabnikom na razpolago malo programskih rešitev, ki bi lahko samostojno predlagale izvirne kombinacije živil.

Kandidat naj v diplomski nalogi izdela aplikacijo, ki s preiskovanjem prostora kombinacij živil poišče primerne obroke za izbrani način prehranjevanja in jih v obliki predlogov ponudi uporabniku. Kakovost delovanja aplikacije in predlaganih obrokov naj kandidat oceni v sodelovanju s strokovnjakom področja športne prehrane.

Mentor:

doc. dr. Zoran Bosnić

Somentor:

pred. dr. Boštjan Slivnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Andrej Grošelj,

z vpisno številko 63050251,

sem avtor/-ica diplomskega dela z naslovom:

Optimizacija sestave športnih obrokov
glede na vsebovanost hranil

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. Zorana Bosnića

  in somentorstvom (naziv, ime in priimek)

  pred. dr. Boštjana Slivnika

- 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 04.12.2012 Podpis avtorja/-ice: 

Zahvala

Najprej bi se rad zahvalil mentorju doc. dr. Zoranu Bosniću in somentorju pred. dr. Boštjanu Slivniku za hitro odzivnost in izčrpno pomoč pri izdelavi diplomskega dela. Prav tako se zahvaljujem osebnemu trenerju Bojanu Justinu za ocenitev izbranih predlogov obrokov jedilnika v sklopu ovrednotenja rešitve diplomske naloge.

Zahvala gre tudi podjetju Halcom za potrpežljivost, sodelavcem, prijateljem in sošolcem, ki so mi stali ob strani tekom študija, mojemu dekletu Maji in njeni družini za vso podporo in motivacijo ter seveda moji družini, da je verjela vame in mi študij sploh omogočila.

Še enkrat iskrena hvala vsem.

Kazalo

Seznam uporabljenih kratic in simbolov

API – (angl.) Application Programming Interface

CSS – (angl.) Cascading Style Sheets

GNU GPL – (angl.) General Public License, splošno dovoljenje GNU

HTML – (angl.) HyperText Markup Language, jezik za označevanje nadbesedila

HTTP – (angl.) HyperText Transfer Protocol, glavna metoda za prenos informacij na spletu

JAR – (angl.) Java ARchive, javanska arhivska datoteka

Java EE – (angl) Java Enterprise Edition, Java, namenjena spletnemu razvoju

JSP – (angl.) JavaServer Pages, javanska tehnologija za izdelavo dinamičnih spletnih strani

JUnit – (angl.) Java Unit, javanska tehnologija za avtomatizirano testiranje enot

POM –(angl.) Project Object Model, konfiguracijska datoteka za Apache Maven

SERVLET – (angl.) Java Servlets, javanski razredi za delo z aplikacijskimi strežniki

SQL – (angl.) Structure Query Language, programski jezik za delo s podatkovno bazo

WAR – (angl.) Web Application Archive, spletno-aplikacijska arhivska datoteka

XML – (angl.) Extensible Markup Language

Povzetek

Športniki in ljudje, ki imajo predpisan poseben način prehranjevanja, morajo natančno skrbeti za sestavo svojih obrokov in s tem paziti na razmerje vnosa hranil posameznih živil: beljakovin, ogljikovih hidratov in maščob.

Cilj diplomske naloge je izdelava spletne aplikacije za optimizacijo sestave športnih obrokov glede na vsebovanost hranil. Rešitev s pomočjo zbirke živil na eni strani in uporabniških zahtev na drugi strani pripravi različne predloge obrokov, ki se uporabniku ponudijo. Pri tem smo analizirali priporočila ter principe zdrave in uravnotežene prehrane, na podlagi katerih smo svojo rešitev izdelali.

Osrednji del diplomske naloge predstavlja opis zasnove algoritma za izračun predlogov obrokov in njegova integracija v programsko rešitev. Preiskovanje prostora s pomočjo algoritma A* in njegove hevristične funkcije predstavlja osnovo pri implementaciji naše rešitve. Uspešnost delovanja smo ovrednotili v komunikaciji s strokovnjakom s področja športne prehrane, rezultati pa so pokazali uporabnost aplikacije.

Ključne besede: športna prehrana, hranila v živilih, preiskovanje prostora, algoritem A*, hevristična funkcija, spletna aplikacija

Abstract

Athletes and people who have special diets must take care of their meals' composition and pay attention to the intake ratio of basic food nutrients: proteins, carbohydrates and fats.

The aim of the thesis is to develop a Web application which is able to propose various sports meals according to the required composition of the food menu. The solution uses a database of food items, combinations of which it optimizes using a constrained search algorithm in the input space. In doing so, we analyze the recommendations and principles of a healthy and balanced diet on which basis we developed our solution.

The central part of the thesis presents a description of the design of an algorithm for proposing the meal templates and its integration in the software solution. Input space search with an adapted A* algorithm is the basis for the implementation of our solution. The resulting application was evaluated by an expert in the field of sport nutrition. The results have shown usefulness of the developed application.

Keywords: sport nutrition, nutrients in foods, input space search, A* algorithm, heuristic function, Web application

1 Uvod

Zdrava in uravnotežena prehrana je sestavni del zdravega življenja. Danes poznamo številna priporočila za zdravo prehranjevanje, katerih upoštevanje zagotavlja zadosten vnos energije in hranilnih snovi za normalno delovanje našega organizma ter ohranjanje zdravja. Seveda pa je uravnoteženje obroka z upoštevanjem priporočil za zdravo prehranjevanje zahteven večkriterijski problem s številnimi omejitvami, ki so si lahko tudi v nasprotju. Izbor kombinacije živil v obroku mora ustrezati kriterijem cene in kakovosti, ki je pogojena s sezono in z zaščitnimi snovmi (s funkcionalnostjo). Omejitve so določene na osnovi priporočil za zdravo prehranjevanje in vključujejo energijsko in hranilno vrednost obroka, razmerje med beljakovinami, ogljikovimi hidrati in maščobami ter velikost obroka. Še posebej pazljivi pri tovrstnih izborih morajo biti športniki in ljudje, ki imajo predpisan poseben način prehranjevanja. Na splošno pa je znano, da je pri zdravem in uravnoteženem prehranjevanju zelo pomembno kombiniranje živil, saj nobeno živilo ne vsebuje vseh hranilnih snovi, ki jih naše telo potrebuje.

V diplomski nalogi bomo predstavili spletno aplikacijo za optimizacijo sestave športnih obrokov glede na vsebovanost hranil, s pomočjo katere si lahko preprosto sestavimo različne predloge obrokov. Aplikacija od uporabnika zahteva vnos nekaterih priporočil za zdravo prehranjevanje, in sicer skupno velikost oz. težo obroka, dovoljeno količino kalorij obroka, razmerje med beljakovinami, ogljikovimi hidrati in maščobami, število živil v obroku in število različnih predlogov obrokov. Na podlagi kasneje opisanega algoritma in izračuna med posameznimi živili se uporabniku ponudijo posamezni predlogi obrokov, ki vsebujejo vsebovana živila in podroben prikaz različnih hranilnih snovi in količin, tako za predlagani obrok, kot tudi za posamezno vsebovano živilo. Uporabnik lahko podrobno analizira že omenjene skupne in posamezne teže ter količine kalorij obroka in živil, energijsko vrednost obroka in živil, različne vitamine in minerale, količino vode v obroku in v vsebovanih živilih itd. Na ta način lahko izbere najbolj primerne predloge zase.

V drugem poglavju si bomo pobliže ogledali dejavnike zdrave in uravnotežene prehrane, na podlagi katerih je naša rešitev zasnovana. Pogledali si bomo piramido zdrave in uravnotežene prehrane ter izpostavili živila, ki so bodisi zdrava ali škodljiva za naš organizem.

Sledilo bo tretje poglavje z opisom zasnove algoritma. Zanimala nas bo dejanska formalizacija našega problema in kriteriji, ki smo jih določili pred izdelavo rešitve. Sledil bo podrobnejši opis algoritma, kjer bomo med drugim navedli tudi argumente, ki so botrovali izbiri algoritma A*, ki smo ga uporabili kot osnovo za našo rešitev.

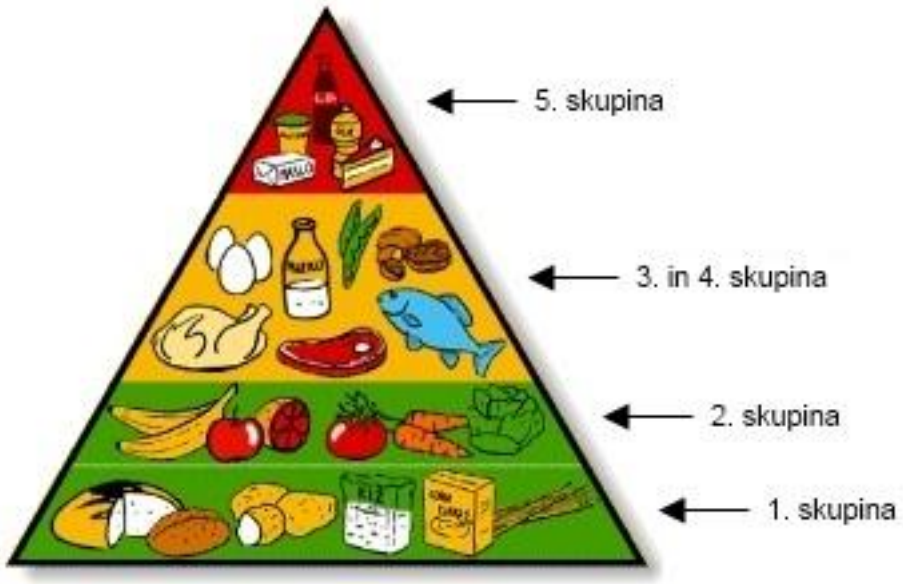
V četrtem poglavju se bomo posvetili izdelavi spletne aplikacije. Tu bomo opisali uporabljena programska orodja pri izdelavi aplikacije ter vzroke za njihovo izbiro. Natančneje si bomo ogledali arhitekturni načrt, razvoj aplikacije skupaj s primeri izvorne kode in primer testiranja določenega segmenta aplikacije. Nazadnje si bomo ogledali tudi ovrednotenje rešitve oz. primere ocenitve rezultatov naše aplikacije preko spletne ankete s strani strokovnjaka.

Za zaključek bomo v petem poglavju navedli kritike za našo rešitev, prav tako pa ne bomo pozabili na možnosti izboljšanja.

Za izdelavo tovrstne spletne aplikacije oz. samega algoritma za optimizacijo sestave športnih obrokov glede na vsebovanost hranil smo se odločili na podlagi dejstva, da je takih rešitev na trgu razmeroma malo. Že sama tematika diplomske naloge nas je zelo pritegnila, saj smo se v preteklosti aktivno, trenutno pa zgolj rekreativno, ukvarjali z vrhunskim športom in imamo glede zdravega in športnega prehranjevanja kar nekaj izkušenj.

2 Zdrava in uravnotežena prehrana

Zdrava in uravnotežena prehrana vsebuje vse hranljive snovi v količini, ki je ravno pravšnja, da naše telo maksimalno dobro deluje. To pomeni, da nobene sestavine ne sme biti preveč ali premalo. Uravnotežena prehrana še posebej ne sme vsebovati nobenih škodljivih snovi, kot so npr. pesticidi, gnojila, razne toksične snovi itd. Živila morajo biti higiensko neoporečna in smejo vsebovati le toliko dodatkov, kot jih dovoljuje zakon o zdravstveni ustreznosti živil in izdelkov ter snovi, ki prihajajo v stik z živili [1]. Poglejmo si vsem znano piramido zdrave prehrane, prikazano na sliki 1.



Slika 1: Piramida zdrave uravnotežene prehrane.

Kot lahko vidimo, je sestavljena iz petih skupin:

- **1. skupina:** živila z veliko ogljikovimi hidrati (kruh, žita, krompir, riž in testenine),
- **2. skupina:** sadje in zelenjava,
- **3. in 4. skupina:** beljakovinska živila (mlečni izdelki, meso, stročnice, jajca, oreščki),
- **5. skupina:** živila z veliko maščob in sladkorjev (živalske, rastlinske maščobe, slaščice).

Iz piramide je razvidno, da naj bi zdrava prehrana vsebovala največ energetskega vnosa od ogljikovih hidratov, sadja in zelenjave, nato beljakovin, najmanj omenjenega vnosa pa naj bi prispevale maščobe. Živila, ki smo jih navedli v 1. skupini, lahko jemo vsak dan. Tudi sadje in zelenjava prispevata k vnosu ustreznih ogljikovih hidratov, zato jih je prav tako priporočeno jesti vsak dan. Zdrave beljakovine naše telo dobi, če jemo živila, navedena v 3. in 4. skupini. Pri izbiri maščob pa se raje odločimo za rastlinske maščobe [1].

Zdrava prehrana naj bi človeku ohranjala in krepila zdravje, če pa smo bolni naj bi zdrava prehrana vsaj preprečevala, da se naše stanje ne bi še poslabšalo.

Uravnotežena prehrana je različna za različne starostne skupine. Kot primer lahko povemo, da so maščobe ključnega pomena v prehrani otrok, saj so raziskovalci ugotovili, da otroci porabijo več maščob kot odrasli glede na zaužite kalorije. Otroci potrebujejo maščobe za rast in razvoj. Pozabiti pa ne smemo, da je zelo važna kvaliteta maščob, ki jih otrok uživa. Zdrave so maščobe, ki jih najdemo v ribah in drugih naravnih virih, izogibati pa se je potrebno maščob v krompirčku in raznih prigrizkih [1].

Kakšna bo za nas uravnotežena in zdrava prehrana, je zelo odvisno tudi od naše športne aktivnosti. Poudarek v prehrani športnika je na ogljikovih hidratih, optimalni količini beljakovin in optimalnem odstotku določenih vrst maščob. Sadje in zelenjava ter napitki z vitamini so nujni. Tako je pravilna prehrana poleg kvalitetnega treninga oz. vadbe bistvenega pomena, da lahko dosegamo najboljše rezultate tako v profesionalnem športu kot na rekreativni ravni [1].

Danes so vse bolj pogoste tudi bolezni, ki so posledica neustrezne prehrane. Zbolimo lahko za sladkorno boleznijo, dobimo razne prebavne težave, čir na želodcu itd. Ljudje s sladkorno boleznijo imajo tako predpisan poseben način prehranjevanja. Paziti morajo, da je njihova prehrana bogata z ogljikovimi hidrati (približno 55 % dobljene energije), bogata z balastnimi snovmi (20 g na 1000 kalorij), revna z maščobami (približno 30 %) itd. [2].

Zdrava prehrana mora prav tako zagotoviti vse potrebne esencialne hranilne snovi, to so snovi, ki jih naše telo za delovanje nujno potrebuje (nekaterih od teh telo tudi ne zna samo proizvesti). Mednje spadajo vitamini, elementi, nekatere aminokisline (to so sestavine beljakovin) in nekatere maščobne kisline [1].

Nekatera živila imajo razen svoje hranilne vrednosti še poseben ugoden (varovalen) vpliv na zdravje in počutje človeka, zmanjšujejo pa tudi tveganje za nastanek bolezni. Probiotični jogurti, jajca, obogatena z omega-3 maščobnimi kislinami, ali pa različna živila z naravnimi ali dodanimi antioksidanti imajo celo zdravilni učinek, seveda če jih vključimo v svojo uravnoteženo prehrano [1].

Vsa omenjena priporočila ter dejavniki zdrave in uravnotežene prehrane predstavljajo temelje in so ključni pri zasnovi in izdelavi naše rešitve oz. aplikacije za optimizacijo sestave športnih obrokov glede na vsebovanost hranil.

3 Zasnova algoritma

Pri zasnovi algoritma bomo najprej formalizirali problem sestave jedilnika. Podrobneje si bomo ogledali izgradnjo dinamičnega drevesa in preiskovanje prostora. Zanimali nas bodo tudi kriteriji za izločanje neustreznih živil in obrokov ter sestava naše hevristične funkcije, ki na koncu vpliva na končne rezultate, se pravi predloge obrokov, ki se uporabniku ponudijo.

3.1 Formalizacija problema

Vsaki postavki oz. živilu smo najprej določili primarne in sekundarne kriterije (kot prikazuje tabela 1), s katerimi smo določili kakovost živila. Tu velja omeniti, da vsi kriteriji nekako predstavljajo pozitivne lastnosti živila, če so prisotni v svoji priporočeni količini.

Primarni kriteriji živil	Sekundarni kriteriji živil
Beljakovine	Vitamini in minerali
Ogljikovi hidrati	Energija
Maščobe	Holesterol

Tabela 1: Kriteriji za določitev kakovosti živila.

Sledijo kriteriji oz. zahteve uporabnika za sestavo predloga obroka, kot prikazuje tabela 2. Količina beljakovin, ogljikovih hidratov ali maščob ne sme presegati skupne velikosti oz. teže obroka.

Kriteriji uporabnika
Velikost oz. teža obroka
Količina kalorij obroka
Količina beljakovin, ogljikovih hidratov in maščob
Število živil v obroku
Število predlogov obrokov

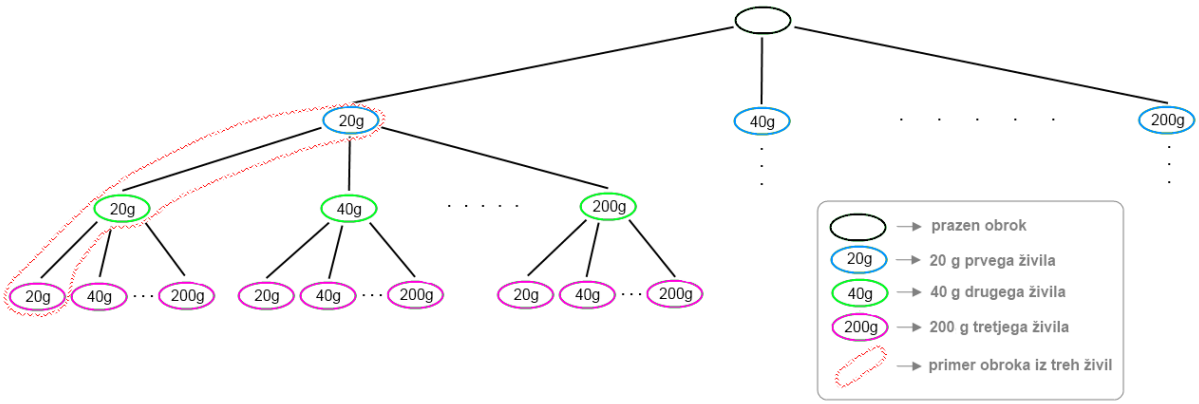
Tabela 2: Kriteriji uporabnika za sestavo obroka.

3.2 Opis algoritma

V sklopu opisa algoritma naše rešitve se bomo posvetili preiskovanju prostora hipotez. Pri tem zgradimo dinamično drevo preiskovalnega prostora in izvajamo usmerjeno iskanje z omejevanjem smiselnosti rešitev. Pri tem smo uporabili algoritem A* z ustrezno hevristično funkcijo.

3.2.1 Gradnja dinamičnega drevesa preiskovalnega prostora

Pri gradnji dinamičnega drevesa preiskovalnega prostora (slika 2) smo najprej določili začetno vozlišče oz. koren drevesa, ki predstavlja prazen obrok oz. živilo. Dodati mu želimo največ toliko živil, kolikor jih uporabnik določi (kriterij: število živil v obroku). Ker pa je dodajanje živil pogojeno s količino živila, ki ga želimo dodati, in ker med obroki, ki imajo npr. 20 g oz. 30 g kruha, nekako ni občutne razlike, smo vsako živilo predelali v enote po 20 g živila do maksimalne količine 200 g. Tako v prvem koraku dodamo prvo živilo v vseh njegovih možnih enotah oz. korenu drevesa dodamo 10 naslednikov od leve proti desni, naraščajoče. V naslednjem koraku razvijamo te naslednike tako, da vsakemu dodamo drugo živilo v vseh njegovih možnih enotah. Z razvijanjem oz. gradnjo drevesa zaključimo takrat, ko dosežemo kriterij - število živil v obroku. Za kombinacije novih živil in obrokov se zgradi novo drevo.



Slika 2: Primer dinamičnega drevesa preiskovalnega prostora.

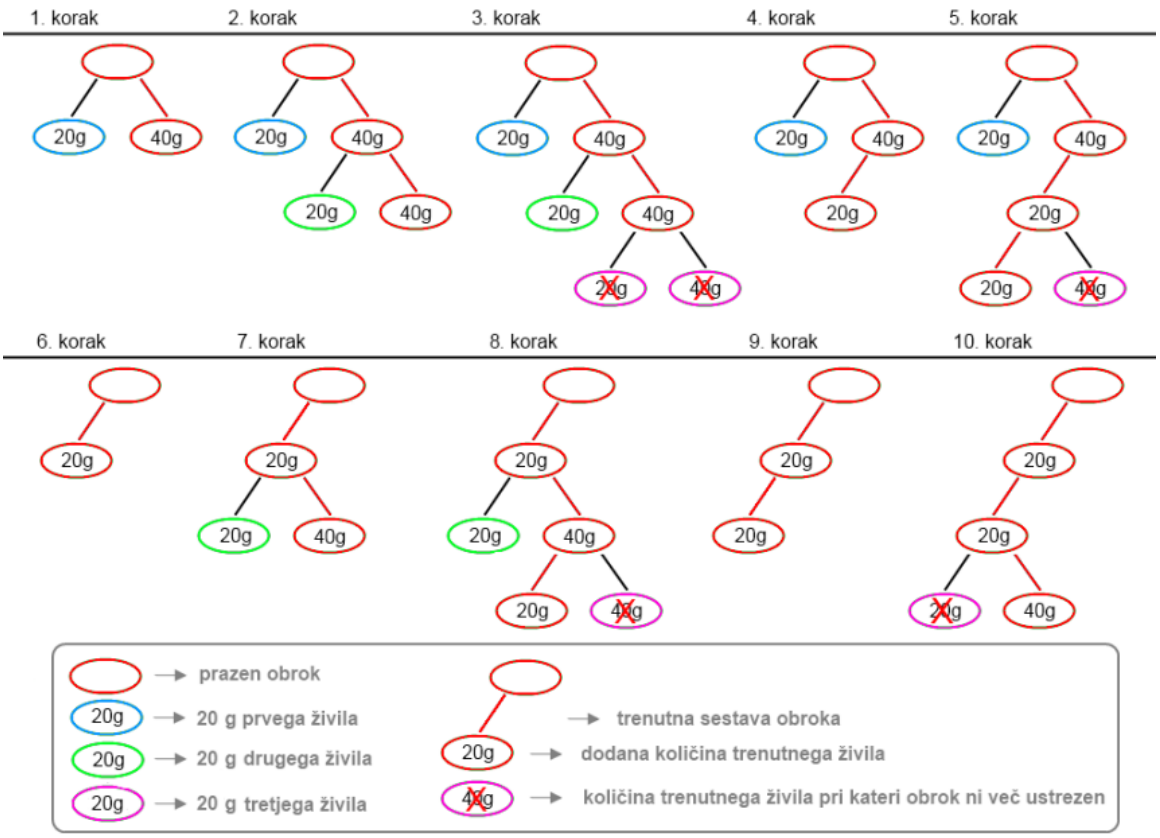
3.2.2 Preiskovanje prostora

Pri preiskovanju prostora izvajamo usmerjeno iskanje z omejevanjem smiselnosti rešitev. Kot osnovo za preiskovanje smo vzeli metodo algoritma A*, ki je eden od primerov preiskovalne

metode »najprej najboljši« (angl. best-first search), in ji pravimo tudi hevristično preiskovanje (angl. heuristic search). Njena hevristična funkcija *f(n)* (opisana v poglavju 3.2.3) je predstavljena kot seštevek dveh ocen, *g(n)* ter *h(n)*. Ocena *g(n)* predstavlja ceno oz. vrednost najboljše poti od začetnega do trenutnega vozlišča, *h(n)* pa vrednost optimalne poti od trenutnega do končnega vozlišča, kar omogoča usmerjanje iskanja v bolj obetavne dele preiskovalnega prostora [3]. V našem primeru to pomeni, da ocena *g(n)* predstavlja trenutno sestavo obroka, ocena *h(n)* pa glede na kriterije, ki so opisani v poglavju 3.2.3, izbira naslednjo enoto živila, ki jo dodamo k trenutnemu obroku. V tej točki smo osnovnemu algoritmu A* dodali še korak, kjer preiskovanje omejujemo in brišemo vozlišča oz. obroke, ki jih ni več smiselno razvijati naprej oz. jim dodajati živila.

Podrobno delovanje samega algoritma bomo predstavili na naslednjem primeru (slika 3). Tu je potrebno omeniti, da smo v primeru za lažjo predstavo predelali živila le v dve enoti, 20 g in 40 g. Prav tako bomo za upoštevanje kvalitete obroka oz. za hevristično funkcijo vzeli le trenutno težo obroka. Definirana kriterija obroka v naslednjem primeru sta:

- velikost oz. teža obroka: **90 g**
- število živil v obroku: **3**



Slika 3: Primer preiskovanja dinamičnega drevesa po korakih.

1. **korak:** Praznemu obroku dodamo prvo živilo v vseh enotah in izberemo enoto oz. vozlišče, ki ga je najbolje razvijati naprej (40 g živila je »bližje« želenim 90 g torej boljše od 20 g).
2. **korak:** Izbranemu živilu oz. obroku iz 1. koraka dodamo drugo živilo v vseh enotah in izberemo vozlišče, ki ga je najbolje razvijati dalje (40 g drugega živila, torej skupno 80 g obroka, kar je še dovoljena teža obroka, vendar še nismo dosegli števila živil v obroku).
3. **korak:** Izbranemu obroku iz 2. koraka želimo dodati zadnje, tretje živilo, vendar s tem prekoračimo dovoljeno skupno težo obroka (90 g) pri vseh dodanih enotah tretjega živila.
4. **korak:** Ker smo v 3. koraku prekoračili omenjeno skupno težo obroka, zavržemo izbrani obrok iz 2. koraka in izberemo drugi najboljši obrok iz 2. koraka.
5. **korak:** Trenutno najboljšemu obroku dodamo tretje živilo, pri čemer z enoto 40 g prekoračimo skupno težo obroka, z enoto 20 g pa dobimo prvi predlog obroka, sestavljenega iz 40 g prvega, 20 g drugega in 20 g tretjega živila, ki ga shranimo za končni izbor obrokov za trenutno preiskovano drevo. Dalje takšnega obroka seveda ne razvijamo zaradi prekoračitve števila živil v obroku.
6. **korak:** Za razvijanje naslednjega najboljšega obroka lahko izberemo samo živilo oz. obrok iz 1. koraka, in sicer 20 g prvega živila.
7. **korak:** Obroku iz 6. koraka dodamo drugo živilo v vseh enotah, vendar kot naslednje vozlišče za razvijanje na podlagi kvalitete izberemo obrok, ki vsebuje 40 g drugega živila.
8. **korak:** Izbranemu obroku iz 7. koraka dodamo tretje živilo, pri čemer z enoto 40 g prekoračimo skupno težo obroka, z enoto 20 g pa dobimo drugi predlog obroka, sestavljenega iz 20 g prvega, 40 g drugega in 20 g tretjega živila, ki ga zopet shranimo za končni izbor obrokov za trenutno preiskovano drevo. Nadaljnji razvoj omenjenega obroka prekinemo zaradi prekoračitve števila živil v obroku.
9. **korak:** Za razvijanje naslednjega vozlišča izberemo obrok iz 7. koraka, ki tokrat vsebuje 20 g drugega živila.
10. **korak:** Obroku iz 9. koraka dodamo tretje živilo, pri čemer dobimo z dodajanjem enote 40 g živila tretji predlog obroka, sestavljenega iz 20 g prvega, 20 g drugega in 40 g tretjega živila. Predlog shranimo za končni izbor obrokov za trenutno preiskovano dinamično drevo in obroka ne razvijamo dalje zaradi prekoračitve števila živil v obroku. Obrok z 20 g tretjega živila pa ni več smiseln oz. je slabši od omenjenega tretjega predloga obroka, zato ga zavržemo.

V opisanem primeru smo torej dobili tri shranjene končne predloge obrokov za trenutno preiskovano dinamično drevo. Te predloge na koncu preiskovanja s pomočjo hevristične

funkcije razvrstimo po kvaliteti in najboljšega shranimo v zbirko vseh obrokov vseh preiskanih dinamičnih dreves pod pogojem, da v tej zbirki še nismo dosegli uporabniškega kriterija, tj. števila predlogov obrokov, oz. da se najboljši predlog obroka za trenutno preiskovano drevo po kvaliteti uvršča v omenjeno končno zbirko obrokov. Uporabniku na ta način posredujemo predloge obrokov, sestavljene iz povsem različnih živil.

3.2.3 Hevristična funkcija in izločanje neustreznih obrokov

Že v poglavju 3.2.2 smo na kratko opisali algoritem A* in njegovo hevristično funkcijo *f(n)*, ki se uporablja za usmerjanje algoritma A*. Njeno definiranje je za algoritem ključnega pomena, saj z njeno pomočjo lahko vplivamo na hitrost in pravilnost rezultata algoritma. V našem primeru hevristična funkcija predstavlja mero za to, kako dober je posamezen obrok glede na ideal, ki ga je nastavil uporabnik s svojimi zahtevami. Sestavili smo jo na podlagi dokaj preprostih kriterijev glede na naš problem, in sicer:

1. **kriterij:** vsota razlik med beljakovinami, ogljikovimi hidrati in maščobami, dejanskimi iz obroka in želenimi s strani uporabnika,
2. **kriterij:** količina vitaminov v obroku,
3. **kriterij:** količina mineralov v obroku,
4. **kriterij:** količina oz. nivo holesterola v obroku,
5. **kriterij:** količina energije obroka.

Naj spomnimo, da smo hevristično funkcijo uporabili pri preiskovanju prostora v segmentu, ko določamo, katero vozlišče oz. obrok je najbolje razvijati naprej in mu dodajati novo živilo. Prav tako se hevristična funkcija uporabi pri razvrščanju živil in pri vmesnem ter končnem razvrščanju predlogov obrokov, ki jih uporabniku ponudimo.

Kot smo že omenili v poglavju 3.2.2, smo metodi algoritma A* dodali tudi korak, kjer brišemo vozlišča oz. obroke, ki jih ni več smiselno razvijati naprej in jim dodajati živila. Tu smo definirali kriterije na podlagi uporabniških zahtev, opisanih v poglavju 3.1, in sicer:

1. **kriterij:** število živil v obroku še ni preseženo glede na želeno število živil uporabnika,
2. **kriterij:** skupna velikost oz. teža obroka ne presega želene teže s strani uporabnika,
3. **kriterij:** količina kalorij obroka ne presega želene količine s strani uporabnika,
4. **kriterij:** količina beljakovin, ogljikovih hidratov ali maščob v obroku še ni presežena glede na želeno količino s strani uporabnika.

Tu moramo dodati še poseben kriterij, ki ga med preiskovanjem uporabimo in ki nekako ne spada med zgoraj naštete. Gre za primer stanja med preiskovanjem s slike 3, in sicer 10. korak, ko smo trenutnemu obroku dodali še zadnje živilo v takšni enoti, da obrok še ustreza zgoraj naštetim kriterijem. V tem primeru vse obroke z manjšo enoto tega zadnjega živila zavržemo, saj so manj ustrezni od omenjenega obroka, ki še ustreza kriterijem.

4 Izdelava aplikacije

Pri izdelavi aplikacije si bomo najprej podrobneje ogledali uporabljena orodja in tehnologije pri razvoju aplikacije. Nato bosta sledili poglavji z opisom arhitekturnega načrta in z opisom razvoja aplikacije po segmentih. Na koncu pa bomo predstavili še primer testiranja sklopa aplikacije za optimizacijo sestave športnih obrokov glede na vsebovanost hranil.

4.1 Uporabljena programska orodja in tehnologije

Pri uporabi programskih orodij in tehnologij za razvoj aplikacije smo se predvsem osredotočili na izbor odprtokodne in prosto dostopne programske opreme. Današnji standardi za programsko opremo nekako narekujejo, da so aplikacije platformno neodvisne, tj. neodvisne od operacijskega sistema. To smo najlažje dosegli s tem, da je naša rešitev predstavljena v obliki spletne aplikacije. Temu primerno smo tudi izbirali programska orodja in tehnologije. Poleg opisov programskih orodij in tehnologij bomo v nadaljevanju podali tudi argumente za njihovo izbiro.

4.1.1 MySQL

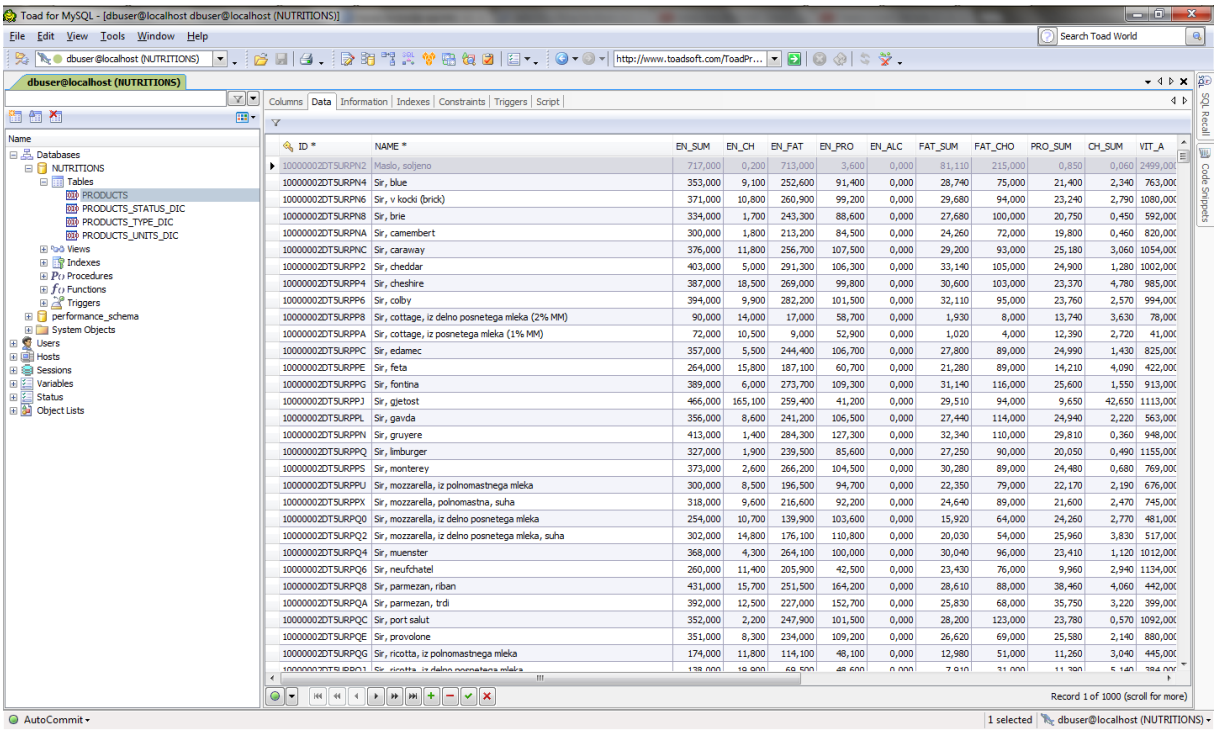
MySQL je podatkovna baza z večuporabniško in centralno nadzorovano zbirko podatkov. Predstavlja klasično (relacijsko) zbirko podatkov, ki lahko deluje kot samostojen podatkovni strežnik ali pa v kombinaciji z drugimi tehnologijami. Poizvedbe v podatkovni bazi MySQL se vršijo z jezikom SQL.
Pred približno desetimi leti je MySQL v jezikih C in C++ razvilo švedsko podjetje MySQL AB in ga ponudilo trgu pod pogoji uporabe javne licence GNU GPL (http://www.gnu.org). Postal je svetovno najbolj priljubljena podatkovna baza odprtega tipa, katere uporaba po zaslugi visoke zanesljivosti, hitrega delovanja, prilagojenosti in nenehnega razvoja še vedno strmo raste. Trenutno je nameščen na več kot 8 milijonov sistemov in deluje na več kot 20 operacijskih sistemih (Linux, Windows, OS/X, HP-UX, AIX, Netware, …) [4].

Za MySQL smo se odločili predvsem zaradi njegove popularnosti in enostavnega upravljanja. Upravljamo ga lahko z ukazno vrstico ali pa z grafičnim uporabniškim vmesnikom, kjer smo uporabili programsko orodje Toad for MySQL.

4.1.2 Toad for MySQL

Toad for MySQL (slika 4) je brezplačen program za upravljanje podatkovne baze MySQL. Razvilo ga je podjetje Quest Software. Omogoča administracijo podatkovne baze, pregledovanje vsebine in pisanje ter testiranje poizvedb SQL. Program poleg omenjenih funkcionalnosti omogoča tudi izdelavo diagrama za podatkovni model, določanja posameznih podatkovnih tipov ter konfiguracijo relacij med entitetami v podatkovni bazi [5].

Omenjeno orodje smo izbrali predvsem na podlagi delovnih izkušenj in njegove preprostosti.

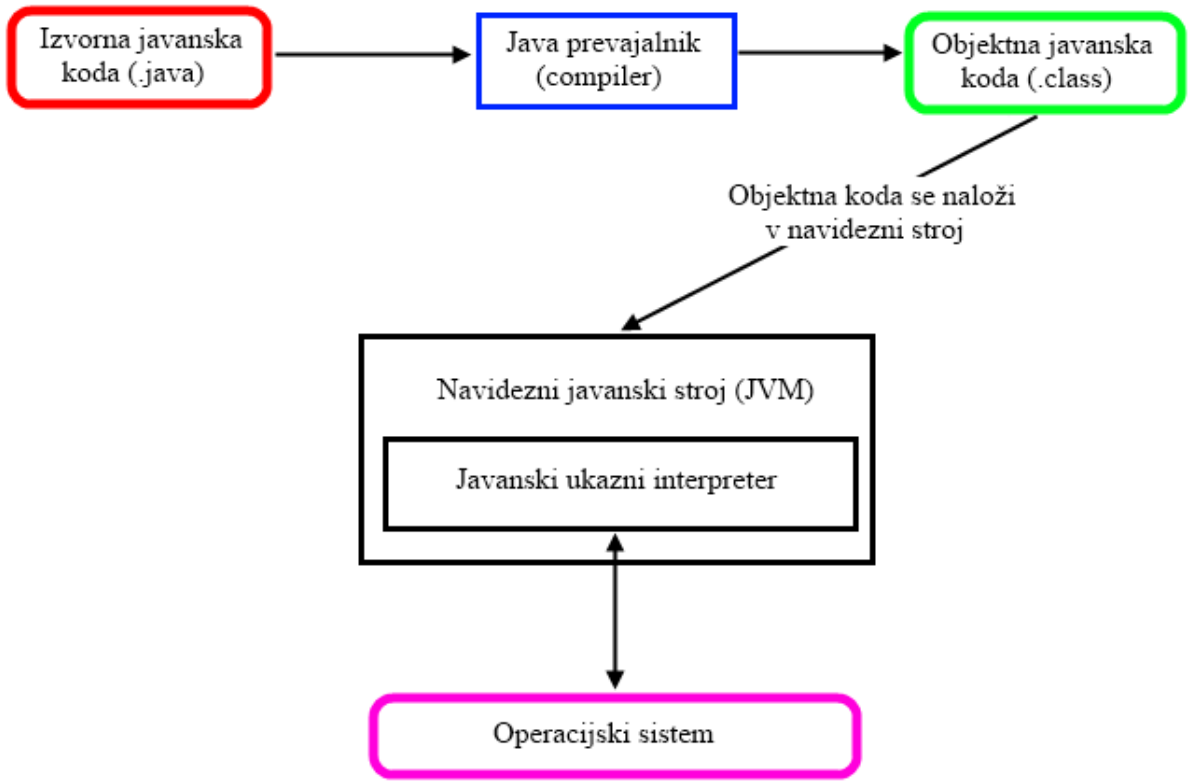


Slika 4: Podatkovna baza spletne aplikacije v Toad for MySQL.

4.1.3 Java EE

Java je trenutno eden izmed najbolj razširjenih in priljubljenih objektno usmerjenih prenosljivih programskih jezikov. Razvil ga je James Gosling s sodelavci v podjetju Sun Microsystems. Njena glavna prednost je neodvisnost od operacijskega sistema, kar pomeni, da javanska aplikacija, izdelana v okolju Windows, brez kakršnihkoli prilagajanj deluje tudi v okolju Unix. To omogoča JVM (angl. Java Virtual Machine) oziroma navidezni javanski stroj, katerega naloga je prevajanje vmesne kode v platformi prilagojene ukaze (diagram

delovanja na sliki 5) in upravljanje s spominom med izvajanjem, kar zmanjša obseg dela programerja.



Slika 5: Diagram delovanja JVM.

Java EE (angl. Java Enterprise Edition) [6] je javansko razvojno okolje, ki se od običajne oblike Jave (ang. Java Standard Edition) razlikuje v tem, da vsebuje knjižnice, ki omogočajo razvoj modularnih, porazdeljenih spletnih aplikacij, ki tečejo na aplikacijskem strežniku. Aplikacijski strežnik upravlja s transakcijami, skrbi za varnost, nadgradljivost in upravlja z vsemi komponentami, ki so na njem nameščene, in tako omogoča razvijalcem, da se osredotočijo na poslovno logiko sistema in ne na njegovo infrastrukturo.

Java je pri izdelavi naše spletne aplikacije predstavljala glavno izbiro glede programskega jezika. Izbrali smo jo predvsem zaradi naših programerskih izkušenj ter omenjene neodvisnosti od operacijskega sistema.

4.1.3.1 Servlet

Servlet se uporablja v programskem jeziku Java. Definiran je kot razred s funkcijami, ki razširijo zmožnosti strežnikov, ki gostujejo specifične aplikacije. Servlet lahko odgovori na kakršen koli tip zahteve, ampak v praksi se uporablja predvsem za nadgradnjo strežniških aplikacij. Te aplikacije so pogosto prisotne kot programi Java Applet, ki se namesto na brskalniku izvajajo na strežniku. Servlete lahko v Javi sami sprogramiramo. Za ta namen se uporablja paketa *javax.servlet* in *javax.servlet.http*, ki vsebujeta vmesnike in razrede za pisanje servletov. Servlet mora biti v skladu s protokolom Java servlet API, po katerem se lahko odziva na zahteve. Najpogosteje se servleti uporabljajo v kombinaciji s protokolom HTTP, zaradi česar se z besedo servlet označuje tudi HTTP servlet [7].

V naši aplikaciji smo uporabili HTTP servlet, in sicer za dodajanje dinamične vsebine na strežnik.

4.1.3.2 JSP

JSP je javanska tehnologija za izdelavo spletnih strani z dinamično vsebino. Stran JSP je v osnovi dokument HTML, ki vsebuje Java kodo, uvedeno s posebnimi oznakami. Ti odseki so elementi JSP, medtem ko je vse ostalo predloga, ki je neposredno posredovana do brskalnika. V nasprotju s statičnimi spletnimi stranmi omogoča JSP spremembo vsebine med izvajanjem z uporabo spremenljivk. Ko spletni strežnik prejme zahtevo za določeno stran JSP, se predloga in elementi JSP združijo. Rezultat je servlet, ki se prevede in nato požene [8].

V naši aplikaciji smo se za strani JSP odločili zato, ker jih je najbolje uporabljati v kombinaciji s servleti. S tem pristopom smo ločili obdelavo zahtev in poslovno logiko (servleti) od uporabniškega vmesnika (strani JSP), kar nam je bistveno olajšalo delo.

4.1.3.3 JUnit

JUnit je odprtokodno ogrodje za avtomatizirano testiranje enot v programskem jeziku Java. Avtorja ogrodja sta Erich Gamma in Kent Beck. Nastalo je leta 1997 na podlagi ogrodja SUnit, s čimer je postalo prvo splošno znano ogrodje za testiranje enot [9].

JUnit smo uporabili za testiranje naše aplikacije oz. določenih enot izvorne kode. Uporabili smo ga predvsem zaradi dejstva, da programsko orodje Apache Maven (opisano v poglavju 4.1.5) avtomatsko izvaja testiranja, izdelana v ogrodju JUnit.

4.1.4 JavaScript

JavaScript je objektni skriptni programski jezik, razvit s strani podjetja Netscape z namenom, da bi spletnim programerjem pomagal pri ustvarjanju interaktivnih spletnih strani. Jezik je bil razvit neodvisno od Jave, vendar si z njo deli številne lastnosti in strukture. JavaScript lahko sodeluje s kodo HTML in s tem poživi stran z dinamičnim izvajanjem, kot so npr. strani JSP. JavaScript podpirajo velika programska podjetja in kot odprt jezik ga lahko uporablja vsakdo, ne da bi pri tem potreboval licenco. Podpirajo ga vsi novejši spletni brskalniki [10].

JavaScript nam je zelo olajšal delo s stranmi JSP, tako da se njegovi uporabi praktično ne bi mogli izogniti. Uporabili smo ga predvsem za preprosta preverjanja operacij, kot je npr. preverjanje, ali je uporabnik vnesel število, in ne katerega izmed drugih znakov v vnosno polje.

4.1.5 Apache Maven

Apache Maven je programsko orodje za gradnjo in upravljanje s programerskimi projekti oz. posameznimi moduli projekta. Zasnovan je v programskem jeziku Java, podpira pa tudi nekatere druge programske jezike, kot na primer C#, Scala in Ruby. Gradnja projekta z omenjenim orodjem temelji na projektno objektnem modelu oziroma datoteki POM (slika 6) ter na naboru vtičev (angl. plugin) in artefaktov (angl. artifact). Vtičniki so že vnaprej vgrajeni v orodje in jih lahko uporabljamo po potrebi, medtem ko same artefakte, ki jih potrebujemo pri svoji izdelavi, lahko avtomatsko prenesemo preko interneta iz t. i. oddaljenega skladišča (angl. repository), s čimer jih orodje shrani v naše lokalno skladišče na disk. Datoteka POM, ki je po obliki datoteka XML, vsebuje vse nastavitve projekta, od uporabljenih vtičev, artefaktov, njihovih medsebojnih odvisnosti in vrstnim redom prevajanja, do samih verzij, nazivov avtorjev, podjetja, itd. [12].
Njegova največja prednost je med drugim tudi avtomatsko prevajanje izvorne kode in določitev, v kakšni obliki želimo imeti zapakirano našo aplikacijo. Tako zapakirano aplikacijo lahko potem distribuiramo po potrebi.

V naši aplikaciji smo uporabili vtičnik *maven-war-plugin,* s čimer smo določili, da se naša aplikacija zapakira v t. i. datoteko WAR, pri čemer gre za javansko datoteko JAR s točno določeno strukturo, prilagojeno spletni aplikaciji.

Na splošno smo se za orodje Apache Maven odločili na podlagi izkušenj in njegove razširljivosti, pa čeprav je mogoče naša aplikacija preveč enostavna za takšno orodje.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>MealNutritionOptimizer</groupId>
  <artifactId>MealNutritionOptimizer</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.2</version>
        </plugin>
    </plugins>
  </build>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.16</version>
    </dependency>
  <developers>
    <developer>
      <id>andrejg</id>
      <name>Andrej Groselj</name>
      <email>mordex007@gmail.com</email>
    </developer>
  </developers>
</project>
```

Slika 6: Primer Apache Maven datoteke POM.

4.1.6 Apache Tomcat

Apache Tomcat je odprtokodna strežniška rešitev, razvita s strani razvijalcev Apache Software Foundation. Predstavlja aplikacijski strežnik, ki podpira računalniško okolje Java. Omogoča izvajanje spletnih programov Java Servlet in strani JSP ter zagotavlja »čisto« Java strežniško okolje HTTP, v katerem se lahko izvaja koda, napisana v Javi. Uporabniki imajo prost dostop do izvorne kode strežnika Tomcat pod licenco Apache [13].

Apache Tomcat je bila nekako primarna izbira za naš aplikacijski strežnik predvsem zaradi popularnosti in preprostosti ter izkušenj z uporabo.

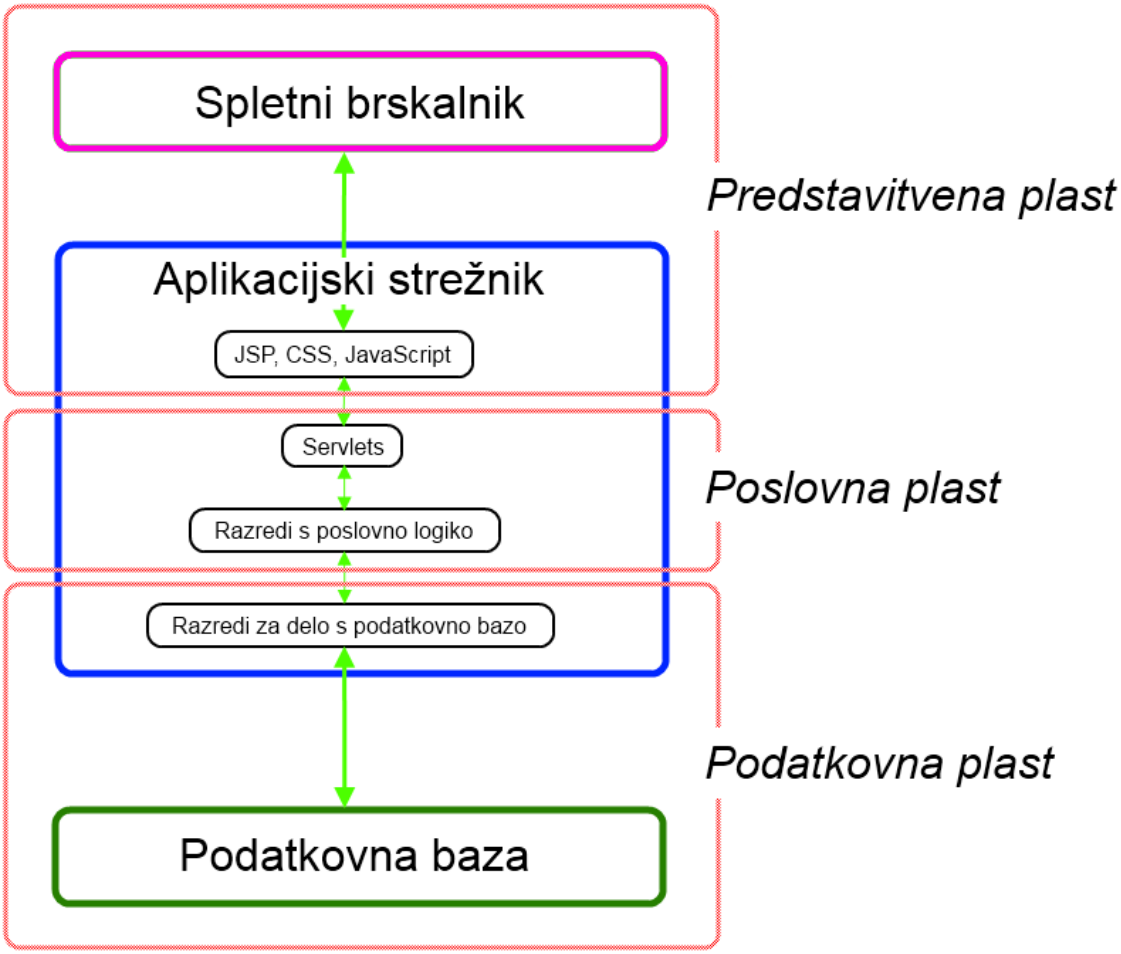
4.1.7 Eclipse

Eclipse je platformno neodvisno in odprtokodno razvojno okolje, primarno namenjeno razvijalcem v okolju Java, podpira pa tudi razvijanje v drugih programskih jezikih. Podpira razne vtičnike, ki jih razvijajo posamezniki in podjetja. Namestitev omenjenega orodja je preprosta, saj le razpakiramo namestitveni paket in poženemo zagonsko datoteko [14].

Za Eclipse smo se odločili zaradi izkušenj in dejstva, da podpira vsa orodja in tehnologije, ki smo jih uporabili pri izdelavi naše aplikacije. Odlično deluje skupaj z orodjema Apache Maven in Apache Tomcat s tem, da je potrebno seveda namestiti temu primerne vtičnike za Eclipse.

4.2 Arhitekturni načrt

V tem poglavju si bomo podrobneje ogledali arhitekturni načrt naše rešitve, kot ga prikazuje slika 7.



Slika 7: Arhitekturni načrt aplikacije.

Pri naši aplikaciji uporabljamo tristopenjsko arhitekturno zasnovo, in sicer:

- **podatkovna plast**: predstavljajo jo podatkovna baza in razredi v aplikacijskem strežniku za delo s podatkovno bazo,
- **poslovna plast**: predstavlja logiko aplikacije in vsebuje razrede s poslovno logiko (ki pridobijo podatke iz razredov za delo s podatkovno bazo) ter servlete (ki izvajajo obdelavo in rezultat posredujejo stranem JSP),
- **predstavitvena plast**: predstavljajo jo strani JSP, ki skrbijo za prikaz rezultatov, pridobljenih s strani servletov.

Dejansko delovanje arhitekture aplikacije si bomo ogledali na glavni funkcionalnosti naše aplikacije, na primeru izračuna predlogov obrokov:

1. uporabnik s pomočjo spletnega brskalnika izpolni vnosna polja s svojimi parametri za izračun predlogov obrokov (Spletni brskalnik, JSP, CSS, JavaScript → Servlets),

2. servlet preveri ustreznost uporabnikovih vhodnih podatkov in jih ob ustreznosti posreduje razredom, ki vsebujejo poslovno logiko,

3. razredi s poslovno logiko pridobijo z druge strani podatke o živilih s pomočjo razredov za delo s podatkovno bazo,

4. razredi s poslovno logiko napravijo izračun in servletu posredujejo predloge obrokov,

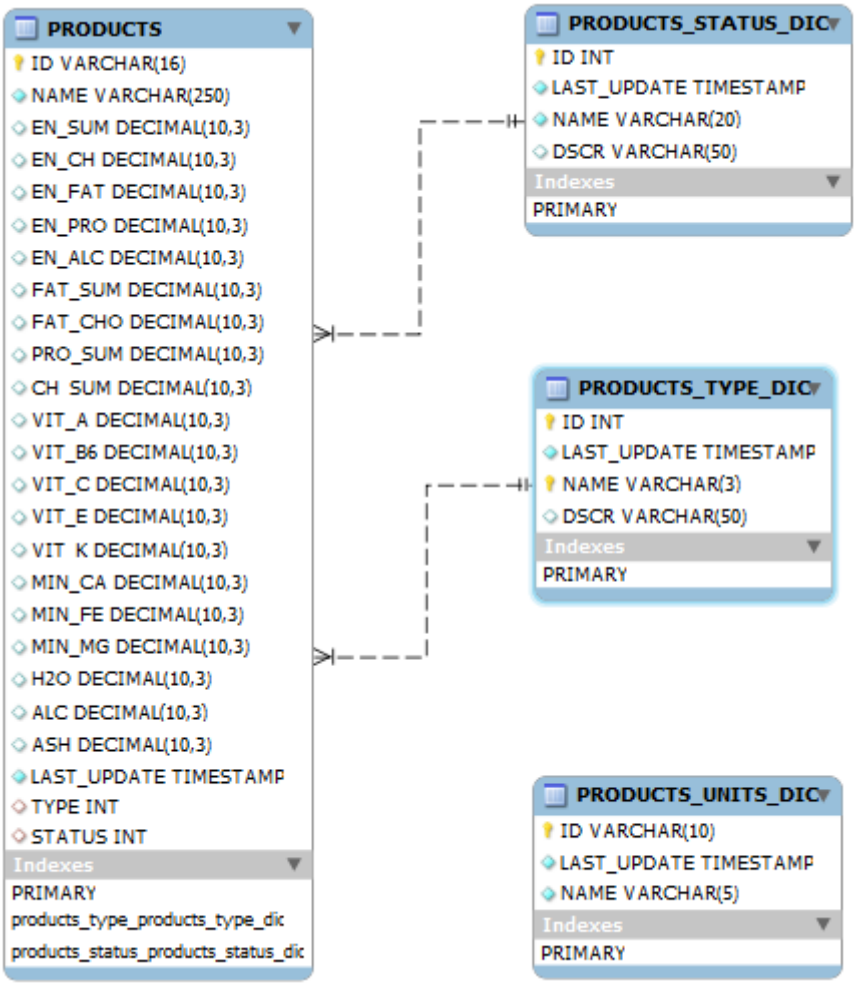
5. servlet pošlje predloge obrokov in uporabnikove podatke stranem JSP, ki predloge obrokov prikažejo spletnemu klientu.

4.3 Razvoj aplikacije

Razvoj aplikacije bomo razdelili na posamezne segmente. Ti nekako predstavljajo zaporedje razvoja aplikacije oz. prikazujejo razvijanje aplikacije glede na njen čas izdelave. Tu si za lažjo predstavo lahko pomagamo z arhitekturnim načrtom, ki smo ga podrobneje predstavili v poglavju 4.2.

4.3.1 Izdelava podatkovne baze

V poglavju 4.1 smo omenili, da smo za postavitev podatkovne baze uporabili MySQL. Slika 8 nam prikazuje fizični model naše podatkovne baze *NUTRITIONS*.



Slika 8: Fizični model podatkovne baze *NUTRITIONS*.

Kot vidimo, smo izdelali eno podatkovno tabelo z imenom *PRODUCTS* (tabela 3) za shrambo živil in tri podatkovne tabele z imeni *PRODUCTS_STATUS_DIC* (tabela 4), *PRODUCTS_TYPE_DIC* (tabela 5) in *PRODUCTS_UNITS_DIC* (tabela 6), ki nam služijo kot imeniki za lažjo interpretacijo točno določenih podatkov v tabeli za shrambo živil.

1. **Tabela *PRODUCTS***
   - namenjena shrambi živil in je glavni in edini vir za črpanje podatkov o živilih,
   - vsebuje tuji ključ, imenovan *products_type_products_type_dic*, ki povezuje numerični podatek *TYPE* s primarnim ključem iz tabele, imenovane *PRODUCTS_TYPE_DIC*,
   - vsebuje tuji ključ, imenovan *products_status_products_status_dic*, ki povezuje numerični podatek *STATUS* s primarnim ključem iz tabele, imenovane *PRODUCTS_STATUS_DIC*.

Ime stolpca	Tip podatka	Opis
ID	VARCHAR(16)	Primarni identifikator
NAME	VARCHAR(250)	Ime živila
EN_SUM	DECIMAL(10,3)	Skupna vrednost energije
EN_CH	DECIMAL(10,3)	Energija iz ogljikovih hidratov
EN_FAT	DECIMAL(10,3)	Energija iz maščob
EN_PRO	DECIMAL(10,3)	Energija iz beljakovin
EN_ALC	DECIMAL(10,3)	Energija iz alkohola
FAT_SUM	DECIMAL(10,3)	Skupna vrednost maščob
FAT_CHO	DECIMAL(10,3)	Količina holesterola
PRO_SUM	DECIMAL(10,3)	Skupna vrednost beljakovin
CH_SUM	DECIMAL(10,3)	Skupna vrednost ogljikovih hidratov
VIT_A	DECIMAL(10,3)	Količina vitamina A
VIT_B6	DECIMAL(10,3)	Količina vitamina B6
VIT_C	DECIMAL(10,3)	Količina vitamina C
VIT_E	DECIMAL(10,3)	Količina vitamina E
VIT_K	DECIMAL(10,3)	Količina vitamina K
MIN_CA	DECIMAL(10,3)	Količina minerala kalcij
MIN_FE	DECIMAL(10,3)	Količina minerala železo
MIN_MG	DECIMAL(10,3)	Količina minerala mangan
H20	DECIMAL(10,3)	Količina vode
ALC	DECIMAL(10,3)	Količina alkohola
ASH	DECIMAL(10,3)	Količina prahu

LAST_UPDATE	TIMESTAMP	Čas zadnje posodobitve podatkov
TYPE	INT	Tip živila (tuji ključ)
STATUS	INT	Status živila (tuji ključ)

Tabela 3: Zgradba podatkovne tabele *PRODUCTS*.

2. **Tabela *PRODUCTS_TYPE_DIC***
   - namenjena kot imenik za tipe živil,
   - primarni ključ je vsebovan v tabeli *PRODUCTS*.

Ime stolpca	Tip podatka	Opis
ID	INT	Primarni identifikator
LAST_UPDATE	TIMESTAMP	Čas zadnje posodobitve podatkov
NAME	VARCHAR(3)	Ime tipa živila
DSCR	VARCHAR(50)	Opis tipa živila

Tabela 4: Zgradba podatkovne tabele *PRODUCTS_TYPE_DIC*.

3. **Tabela *PRODUCTS_STATUS_DIC***
   - namenjena kot imenik za trenutni status živil,
   - primarni ključ je vsebovan v tabeli *PRODUCTS*.

Ime stolpca	Tip podatka	Opis
ID	INT	Primarni identifikator
LAST_UPDATE	TIMESTAMP	Čas zadnje posodobitve podatkov
NAME	VARCHAR(20)	Ime statusa živila
DSCR	VARCHAR(50)	Opis statusa živila

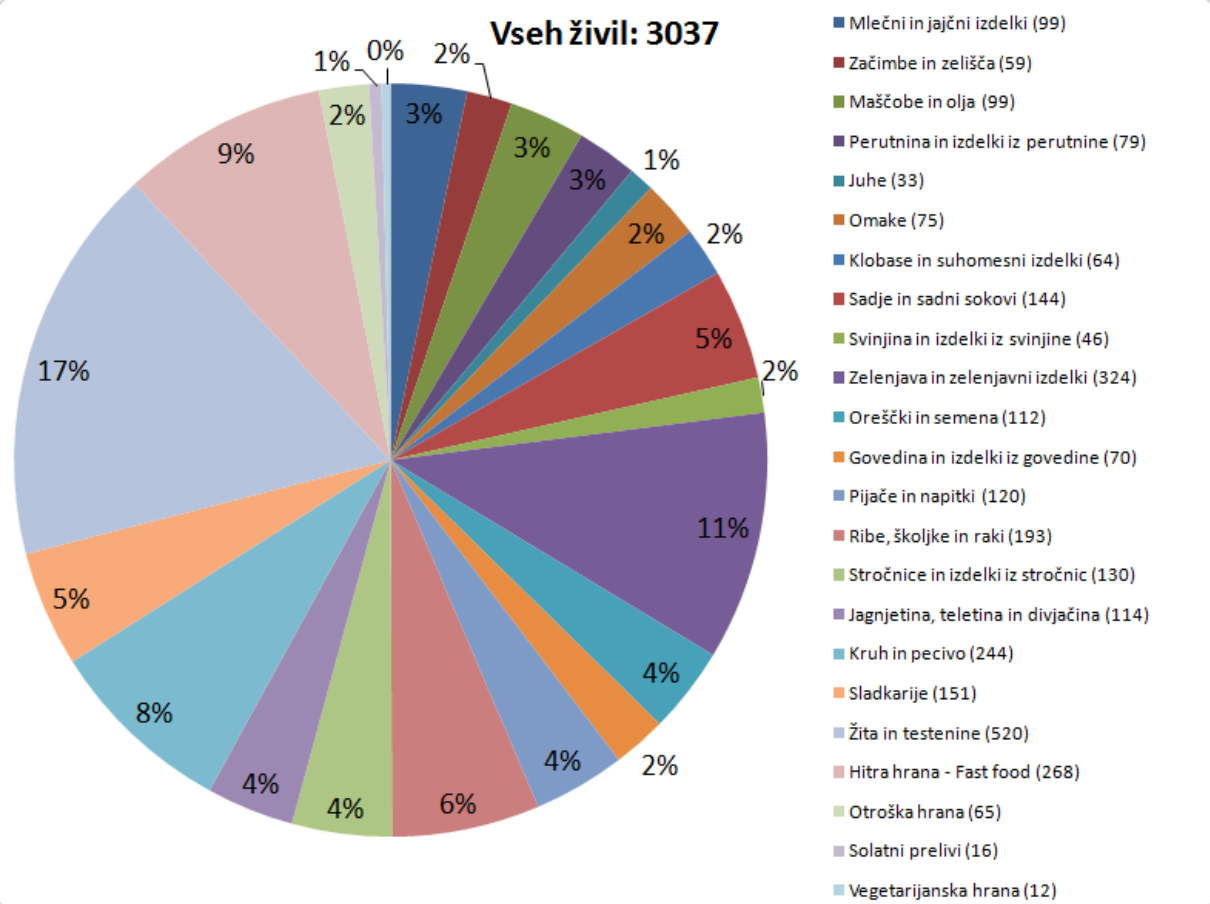
Tabela 5: Zgradba podatkovne tabele *PRODUCTS_STATUS_DIC*.

4. **Tabela *PRODUCTS_UNITS_DIC***
   - namenjena kot imenik za enote živil za posamezni atribut živila,
   - določa enote, v katerih je predstavljena vrednost določenega atributa živila za lažje pretvorbe med enotami.

Ime stolpca	Tip podatka	Opis
ID	VARCHAR(10)	Primarni identifikator
LAST_UPDATE	TIMESTAMP	Čas zadnje posodobitve podatkov
NAME	VARCHAR(5)	Ime enote živila

Tabela 6: Zgradba podatkovne tabele *PRODUCTS_UNITS_DIC*.

Podatke o živilih smo pridobili s pomočjo spletnega vira [15]. Slika 9 prikazuje trenutno stanje oz. pregled števila živil po vseh tipih v naši podatkovni bazi. Kot lahko vidimo, imamo trenutno na voljo 3037 različnih živil.



Slika 9: Pregled živil v podatkovni bazi po odstotkih.

Kot lahko vidimo, zgornji odstotki tipov živil odražajo temelje zdrave in uravnotežene prehrane, kot jih prikazuje piramida zdrave prehrane. Največje število živil predstavljajo škrobna živila (žita in testenine, kruh, riž), sledijo zelenjava in zelenjavni izdelki ter sadje in sadni sokovi, beljakovinska živila (mlečni in jajčni izdelki, perutnina in izdelki iz perutnine, ribe, školjke, raki itd.) ter na koncu živila z veliko maščob in sladkorjev (maščobe in olja, sladkarije itd.).

Tu velja še omeniti, da naša aplikacija omogoča tudi pregled vseh zgoraj prikazanih tipov živil in njihove podrobnosti, kot to prikazuje slika 10. Tako lahko uporabnik tudi sam pregleduje živila in preuči njihove lastnosti.

Calculate | Products

Select product type: Mlečni in jajčni izdelki Show Values represent 100g of product!

NAME	EN_SUM (kcal)	EN_CH (kcal)	EN_FAT (kcal)	EN_PRO (kcal)	EN_ALC (kcal)	FAT_SUM (g)	FAT_CHO (mg)	PRO_SUM (g)	CH_SUM (g)	VIT_A (iu)	VIT_B6 (mg)	VIT_C (mg)	VIT_E (mg)	VIT_K (mcg)	MIN_CA (mg)	MIN_FE (mg)	MIN_MG (mg)	H2O (g)	ALC (g)	ASH (g)
Maslo, soljeno	717.0	0.2	713.0	3.6	0.0	81.11	215.0	0.85	0.06	2499.0	0.0030	0.0	2.32	7.0	24.0	0.02	2.0	15.87	0.0	2.11
Sir, blue	353.0	9.1	252.6	91.4	0.0	28.74	75.0	21.4	2.34	763.0	0.166	0.0	0.25	2.4	528.0	0.31	23.0	42.41	0.0	5.11
Sir, v kocki (brick)	371.0	10.8	260.9	99.2	0.0	29.68	94.0	23.24	2.79	1080.0	0.065	0.0	0.26	2.5	674.0	0.43	24.0	41.11	0.0	3.18
Sir, brie	334.0	1.7	243.3	88.6	0.0	27.68	100.0	20.75	0.45	592.0	0.235	0.0	0.24	2.3	184.0	0.5	20.0	48.42	0.0	2.7
Sir, camembert	300.0	1.8	213.2	84.5	0.0	24.26	72.0	19.8	0.46	820.0	0.227	0.0	0.21	2.0	388.0	0.33	20.0	51.8	0.0	3.68
Sir, caraway	376.0	11.8	256.7	107.5	0.0	29.2	93.0	25.18	3.06	1054.0	0.074	0.0	0.0	0.0	673.0	0.64	22.0	39.28	0.0	3.28
Sir, cheddar	403.0	5.0	291.3	106.3	0.0	33.14	105.0	24.9	1.28	1002.0	0.074	0.0	0.29	2.8	721.0	0.68	28.0	36.75	0.0	3.93
Sir, cheshire	387.0	18.5	269.0	99.8	0.0	30.6	103.0	23.37	4.78	985.0	0.074	0.0	0.0	0.0	643.0	0.21	21.0	37.65	0.0	3.6
Sir, colby	394.0	9.9	282.2	101.5	0.0	32.11	95.0	23.76	2.57	994.0	0.079	0.0	0.28	2.7	685.0	0.76	26.0	38.2	0.0	3.36
Sir, cottage, iz delno posnetega mleka (2% MM)	90.0	14.0	17.0	58.7	0.0	1.93	8.0	13.74	3.63	78.0	0.076	0.0	0.02	0.2	69.0	0.16	6.0	79.31	0.0	1.39
Sir, cottage, iz posnetega mleka (1% MM)	72.0	10.5	9.0	52.9	0.0	1.02	4.0	12.39	2.72	41.0	0.068	0.0	0.01	0.1	61.0	0.14	5.0	82.48	0.0	1.39
Sir, edamec	357.0	5.5	244.4	106.7	0.0	27.8	89.0	24.99	1.43	825.0	0.076	0.0	0.24	2.3	731.0	0.44	30.0	41.56	0.0	4.22
Sir, feta	264.0	15.8	187.1	60.7	0.0	21.28	89.0	14.21	4.09	422.0	0.424	0.0	0.18	1.8	493.0	0.65	19.0	55.22	0.0	5.2
Sir, fontina	389.0	6.0	273.7	109.3	0.0	31.14	116.0	25.6	1.55	913.0	0.083	0.0	0.27	2.6	550.0	0.23	14.0	37.92	0.0	3.79
Sir, gjetost	466.0	165.1	259.4	41.2	0.0	29.51	94.0	9.65	42.65	1113.0	0.271	0.0	0.0	0.0	400.0	0.52	70.0	13.44	0.0	4.75
Sir, gavda	356.0	8.6	241.2	106.5	0.0	27.44	114.0	24.94	2.22	563.0	0.08	0.0	0.24	2.3	700.0	0.24	29.0	41.46	0.0	3.94
Sir, gruyere	413.0	1.4	284.3	127.3	0.0	32.34	110.0	29.81	0.36	948.0	0.081	0.0	0.28	2.7	1011.0	0.17	36.0	33.19	0.0	4.3
Sir, limburger	327.0	1.9	239.5	85.6	0.0	27.25	90.0	20.05	0.49	1155.0	0.086	0.0	0.23	2.3	497.0	0.13	21.0	48.42	0.0	3.79
Sir, monterey	373.0	2.6	266.2	104.5	0.0	30.28	89.0	24.48	0.68	769.0	0.079	0.0	0.26	2.5	746.0	0.72	27.0	41.01	0.0	3.55
Sir, mozzarella, iz polnomastnega mleka	300.0	8.5	196.5	94.7	0.0	22.35	79.0	22.17	2.19	676.0	0.037	0.0	0.19	2.3	505.0	0.44	20.0	50.01	0.0	3.28
Sir, mozzarella, polnomastna, suha	318.0	9.6	216.6	92.2	0.0	24.64	89.0	21.6	2.47	745.0	0.062	0.0	0.21	2.5	575.0	0.2	21.0	48.38	0.0	2.91
Sir, mozzarella, iz delno posnetega mleka	254.0	10.7	139.9	103.6	0.0	15.92	64.0	24.26	2.77	481.0	0.07	0.0	0.14	1.6	782.0	0.22	23.0	53.78	0.0	3.27
Sir, mozzarella, iz delno posnetega mleka, suha	302.0	14.8	176.1	110.8	0.0	20.03	54.0	25.96	3.83	517.0	0.079	0.0	0.37	1.3	731.0	0.25	26.0	46.46	0.0	3.72

Slika 10: Primer izpisa živil v spletni aplikaciji iz podatkovne baze.

4.3.2 Aplikacijski strežnik

Za postavitev in izdelavo aplikacijskega strežnika smo za temeljno orodje uporabili Eclipse in njegovo javansko okolje. Dodatno smo Eclipse razširili s pomočjo dveh glavnih vtičnikov. Za postavitev projekta aplikacije smo uporabili vtičnik Apache Maven, za proženje aplikacije pa vtičnik Apache Tomcat. Razvijanje aplikacije smo pričeli z definiranjem razredov za delo s podatkovno bazo. Tu smo zaradi relativne preprostosti naše podatkovne baze razrede definirali sami. Nato smo pričeli z izdelavo razredov s poslovno logiko, servletov in strani JSP za prikaz naše aplikacije in podatkov v spletnem brskalniku.

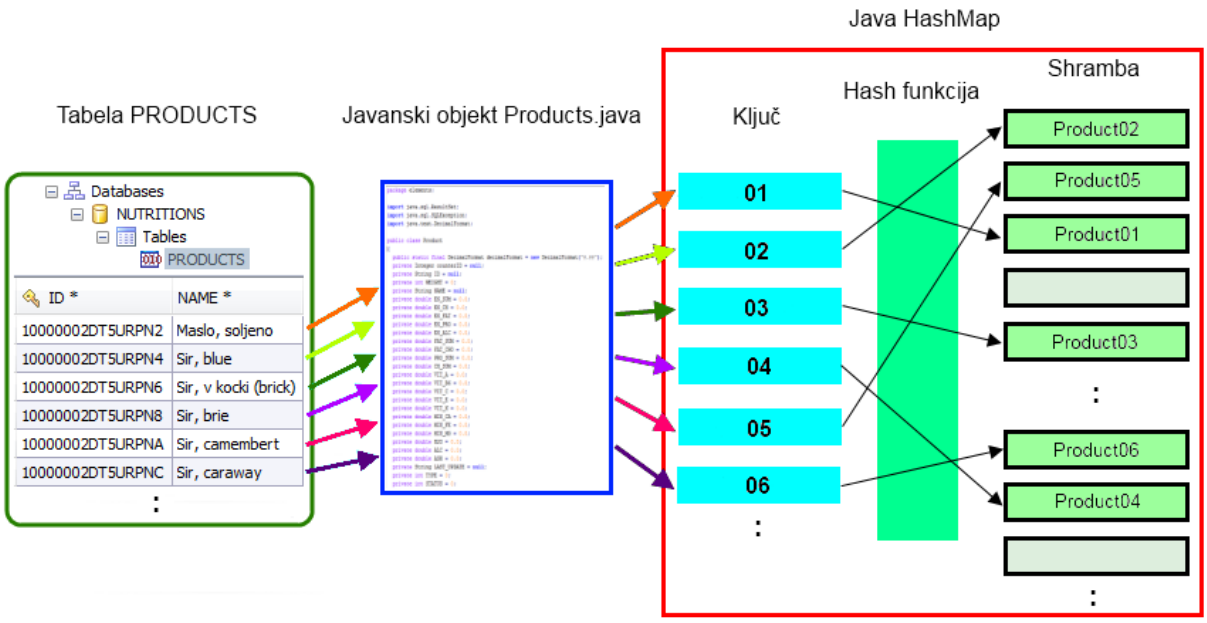
4.3.2.1 Razredi za delo s podatkovno bazo

Kot smo že omenili, so zaradi preprostosti naše podatkovne baze razredi za delo z njo prav tako relativno enostavni. Definirali smo štiri razrede za delo z entitetami, kjer vsaka od njih predstavlja eno podatkovno tabelo v podatkovni bazi. Primer take entitete za podatkovno tabelo *PRODUCTS_TYPE_DIC* prikazuje slika 11.

```
package database;

import java.sql.ResultSet;
import java.sql.SQLException;

public class ProductTypeDic
{
  private Integer ID = 0;
  private String LAST_UPDATE = null;
  private String NAME = null;
  private String DSCR = "";

  public ProductTypeDic(ResultSet resultSet)
                        throws NumberFormatException, SQLException
  {
    ID = Integer.valueOf(resultSet.getInt("ID"));
    LAST_UPDATE = resultSet.getString("LAST_UPDATE");
    NAME = resultSet.getString("NAME");
    DSCR = resultSet.getString("DSCR");
  }

  public int getID()
  {
    return ID;
  }

  public String getLAST_UPDATE()
  {
    return LAST_UPDATE;
  }

  public String getNAME()
  {
    return NAME;
  }

  public String getDSCR()
  {
    return DSCR;
  }
}
```

Slika 11: Entitetni razred za podatkovno tabelo *PRODUCTS_TYPE_DIC*.

Razredi za delo s podatkovno bazo omogočajo prenos podatkov iz podatkovne baze v programski spomin. To se zgodi ob zagonu aplikacijskega strežnika, v katerem nato operiramo z omenjenimi podatki. S tem seveda pridobimo boljšo zmogljivost sistema in aplikacije, saj nam ni potrebno ob vsaki poizvedbi uporabnika dostopati do podatkovne baze. Podatki o živilih se preslikajo v naprej definiran javanski razred *Product.java*, ki predstavlja naš interni objekt v aplikacijskem strežniku. Te objekte o živilih hranimo v javanski zgoščeni tabeli (HashMap), ki lahko hrani podatkovne objekte različnih tipov. Gre za zbirko, kjer vsak objekt, ki predstavlja eno živilo, shranimo na podlagi unikatnega identifikatorja oz. ključa, s pomočjo katerega do objekta živila kasneje v razredih s poslovno logiko tudi dostopamo (slika 12).



Slika 12: Preslikava živil iz podatkovne tabele *PRODUCTS* v Java HashMap.

4.3.2.2 Razredi s poslovno logiko

Razredi s poslovno logiko predstavljajo jedro aplikacije. Z njihovo pomočjo pridobimo različne predloge obrokov glede na uporabnikove zahteve. Razredi komunicirajo na eni strani z razredi za delo s podatkovno bazo in na drugi strani z javanskimi servleti, ki posredujejo uporabnikove zahteve za izračun obrokov. Kot smo že omenili, predstavljajo razredom s poslovno logiko objekti Java HashMap glavni vir podatkov. Tu se podatki črpajo iz več različnih zbirk HashMap, na eni strani iz zbirke o vseh živilih in na drugi npr. iz zbirke HashMap, ki vsebuje imena vseh različnih tipov živil (pridobljena iz podatkovne tabele *PRODUCTS_TYPE_DIC*). Ti so še posebej pomembni kasneje na straneh JSP za prikaz podatkov v spletnem brskalniku.

Operacije znotraj razredov s poslovno logiko smo zaradi lažje implementacije razdelili na več smiselnih korakov, s katerimi pridobimo predloge obrokov.

1. **Izločanje živil, ki imajo vrednosti nad zahtevanimi parametri uporabnika**

Slika 13 prikazuje prvi korak do pridobitve predlogov obrokov. Gre za izločanje živil, ki imajo vrednosti beljakovin, ogljikovih hidratov ali maščob večje od zahtevanih vrednosti s strani uporabnika za posamezen obrok.

```
FOR each product in all products storage list
    productProteins = product proteins sum / product weight * minimal meal product part weight
    productCarboh = product carboh sum / product weight * minimal meal product part weight
    productFat = product fat sum / product weight * minimal meal product part weight

    IF productProteins > user meal proteins criteria OR
       productCarboh > user meal carboh criteria OR
       productFat > user meal fat criteria
    THEN
        product is not suitable for calculation
        CONTINUE
    ENDIF
    store product in suitable products storage for calculation
ENDFOR
```

Slika 13: Izločanje živil z vrednostmi nad zahtevanimi parametri uporabnika.

V tem koraku se sprehodimo preko vseh živil v naši zbirki in za vsako posamezno živilo izračunamo njegovo količino beljakovin, ogljikovih hidratov in maščob. Ker pa moramo pridobiti npr. količino beljakovin v živilu za njegovo najmanjšo enoto, torej 20 g živila, to količino beljakovin v živilu delimo s 100 g (ker so vrednosti živila v naši podatkovni bazi predstavljena za 100 g živila) ter pridobljeno vrednost množimo z našo najmanjšo enoto živila (20 g). Če taka končna vrednost beljakovin, ogljikovih hidratov ali maščob za 20 g trenutnega živila presega vrednost količine beljakovin, ogljikovih hidratov ali maščob v obroku, zahtevanih s strani uporabnika, takega živila ne vključimo v zbirko živil, s pomočjo katere kasneje napravimo izračun predlogov obrokov.

2. Naključno razvrščanje zbirke živil za izračun predlogov obrokov

Sledi drugi korak, kjer naključno razvrstimo zbirko živil, na podlagi katere bomo izračunali predloge obrokov za uporabnika.
Za naključno razvrščanje zbirke živil smo uporabili kar javansko metodo *shuffle* iz zbirke Java Collections, ki je prisotna vse od verzije Java 5.0 dalje. Metoda naključno razporedi živila s tem, da vsako premakne na naključno izbrano pozicijo. Pri tem imajo vsa živila enako verjetnost za izbor in premik v zbirki. S pomočjo omenjenega naključnega razvrščanja živil se uporabniku ob vsaki poizvedbi z enakimi vhodnimi parametri generirajo novi predlogi obrokov, sestavljeni iz povsem različnih enot živil.

3. Izračun števila različnih kombinacij obrokov iz zbirke živil

Pri izračunu števila različnih kombinacij obrokov iz trenutne zbirke živil smo si pomagali z že obstoječimi algoritmi za izračun števila različnih kombinacij znotraj določenega nabora števil. Ko govorimo o številu različnih kombinacij obrokov, dejansko govorimo o številu gradenj

dinamičnih dreves preiskovalnega prostora iz poglavja 3.2.1. Poglejmo si prikazani algoritem na naslednjem primeru:

- Imamo 10 živil v zbirki: $\boldsymbol{a = 10}$
- Želimo imeti 3 živila v vsakem obroku: $\boldsymbol{b = 3}$
- Formula: $\boldsymbol{R = \frac{a!}{b!(a-b)!}}$
- Izračun: $\boldsymbol{R = \frac{a!}{b!(a-b)!} = \frac{10!}{3!(10-3)!} = \frac{10!}{3!\,7!} = 120}$ različnih kombinacij obrokov

Tu velja omeniti, da opisani izračun za različne kombinacije obrokov upošteva samo eno enoto živila, npr. 20 g živila. Če bi upoštevali še posamezne enote živila (kot to počnemo pri izgradnji dinamičnega drevesa preiskovalnega prostora), bi se rezultat izračuna drastično povečal.

V našem primeru smo se odločili, da na koncu zaradi omenjene časovne zahtevnosti procesiranja vseh možnih kombinacij živil znotraj zbirke živil algoritem omejimo tako, da mu določimo največje število preiskanih kombinacij obrokov oz. dinamičnih dreves. To smo določili na preprost način, in sicer smo število živil v naši zbirki delili s številom živil v posameznem obroku. Tako npr. za 3037 živil v zbirki in za npr. 3 živila v obroku dobimo število 1012, ki določa največje število preiskanih kombinacij obrokov.

4. Procesiranje različnih kombinacij živil za izračun obrokov

Naslednji korak predstavlja procesiranje vseh različnih kombinacij živil za izračun obrokov (slika 14).

```
combinationsCounter = 1

remember start time by milliseconds

WHILE combinationsCounter != max processed combinations
    get unique random products
    process combination for unique random products
    INCREMENT combinationsCounter
ENDWHILE

remember end time by milliseconds

processedTimeSeconds = (end time - start time) / 1000
```

Slika 14: Procesiranje vseh različnih kombinacij živil.

Ta korak za vsako procesirano kombinacijo pridobi ustrezna (še ne uporabljena v katerikoli shranjeni predlogi obrokov) živila, jih posreduje naprej za izračun predlogov obrokov in na koncu izpiše porabljeni čas za procesiranje vseh omejenih kombinacij živil.
Procesiranje vsake posamezne kombinacije živil (kot prikazuje slika 15) oz. delovanje algoritma smo že podrobno predstavili v poglavju 3.2. Tu bomo samo na kratko opisali psevdokodo algoritma za izračun predlogov obrokov.

```
continueProcess = true
bestMealsForTree = sort meals in current dynamic tree to get next processed one

WHILE continueProcess
    currentBestMeal = bestMealsForTree[0]
    productIndex = get current product index (dynamic tree level index) for currentBestMeal
    productParts = create product parts for product in currentBestMeal at productIndex position

    FOR productPart in productsParts
        newMeal = currentBestMeal with all its current products
        add productPart to newMeal

        IF newMeal suits user meal parameters
        THEN
            IF newMeal number of products in meal (defined by user) is not achieved
            THEN
                add newMeal to bestMealsForTree
            ENDIF
            continueProcess = false
        ENDIF
    ENDFOR

    IF bestMealsForTree is not empty
    THEN
        remove currentBestMeal from bestMealsForTree
    ENDIF

    IF bestMealsForTree is empty
    THEN
        continueProcess = false
    ENDIF

ENDWHILE
```

Slika 15: Procesiranje posamezne kombinacije živil.

Algoritem najprej izvede razvrščanje trenutnih obrokov in s tem pridobi obrok, ki je trenutno najboljši oz. ga je najbolj smiselno razvijati naprej. Nato pridobi nivo trenutnega živila v obroku (nivo v dinamičnem drevesu) in izračuna njegove enote. Te enote živila dodaja k novemu obroku (zgrajenemu iz živil trenutno najboljšega obroka) in sproti preverja, če obrok še ustreza vrednostim, določenim s strani uporabnika. Prav tako algoritem preveri, če smo slučajno že prekoračili dovoljeno število živil v posameznem obroku. Če je trenutni novi obrok ustrezen, se zabeleži v trenutno zbirko najboljših. Če imamo na koncu vsaj en novi obrok v zbirki najboljših, se stari obrok (predhodnik novih obrokov) odstrani iz zbirke.

5. **Razvrščanje najboljših predlogov obrokov za končni izbor**

Zadnji korak, ki ga izvedemo, je razvrščanje najboljših predlogov obrokov za končni izbor, ki se ponudi uporabniku (slika 16).

```
bestMealsOverall = storage of best meals overall
bestMealForTree = sort meals for current dynamic tree and get best one

IF bestMealsOverall is not empty
THEN
    FOR currentBestMeal in bestMealsOverall list
        IF bestMealForTree is better than currentBestMeal
        THEN
            replace bestMealForTree with currentBestMeal in bestMealsOverall
            BREAK
        ENDIF
    ENDFOR
ELSE
    add bestMealForTree to bestMealsOverall
ENDIF

IF bestMealsOverall size is larger than user number of meals parameter
THEN
    shrink bestMealsOverall to user number of meals parameter
ENDIF
```

Slika 16: Razvrščanje najboljših predlogov obrokov.

Tu najprej razvrstimo predloge obrokov, ki smo jih pridobili iz procesiranja ene kombinacije živil oz. pri pregledu enega dinamičnega drevesa preiskovalnega prostora. Nato pregledamo trenutno končno zbirko predlogov, in če ugotovimo, da je najboljši predlog obroka za trenutno kombinacijo boljši od enega izmed končnih predlogov, najboljšega za trenutno kombinacijo uvrstimo v končni izbor. S tem se vsi slabši predlogi obrokov v končnem izboru pomaknejo za eno mesto navzdol v zbirki. Na koncu še odstranimo vse odvečne predloge obrokov v končnem izboru, če le-ta presega število predlogov, ki jih je uporabnik zahteval.

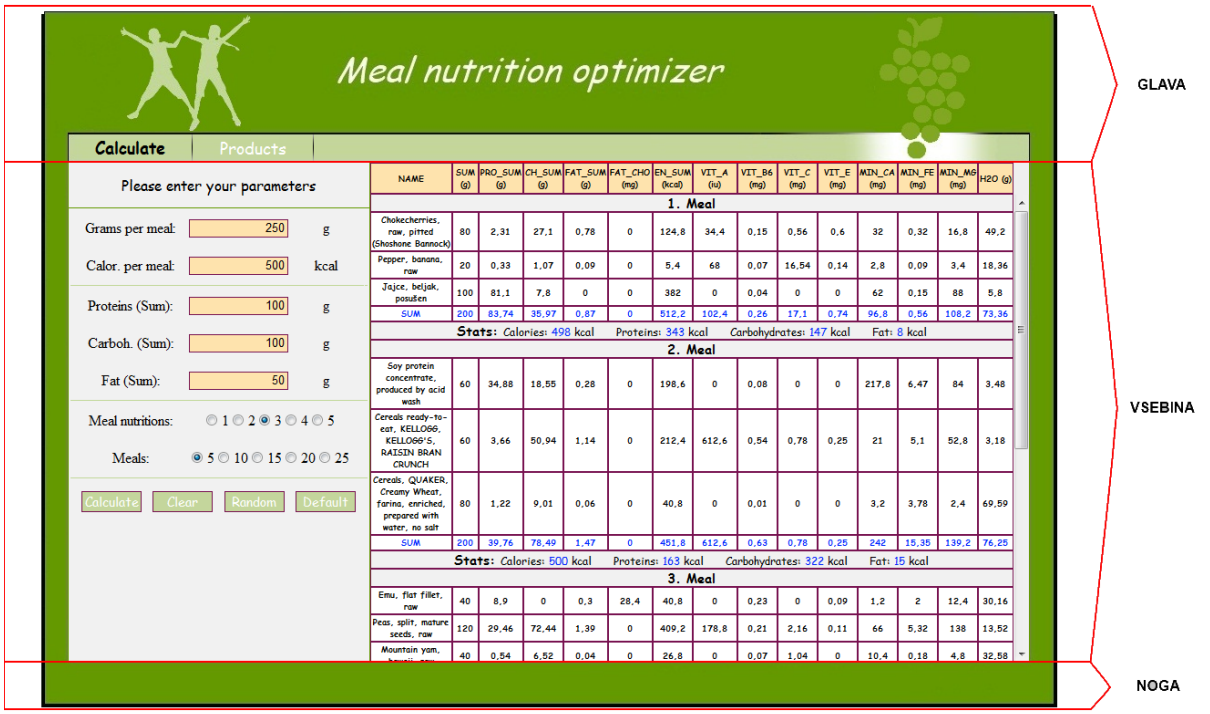
4.3.2.3 Servlets

Kot smo že omenili, smo v naši aplikaciji uporabili HTTP servlet, in sicer za dodajanje dinamične vsebine na strežnik. Servlet služi za komunikacijo med stranmi JSP in razredi s poslovno logiko. V naši aplikaciji smo uporabili en sam javanski razred s servleti, imenovan *MainServlet.java*, v katerem smo uporabili servlet metodo *doPost()*, ki omogoča posredovanje celotnega vnosnega obrazca in njenih vhodnih vrednosti s strani JSP. Omenjeni razred se uporablja tako pri pridobitvi predlogov obrokov kot tudi pri izpisu vseh živil glede na tip

živila (kot to lahko vidimo na sliki 10). V omenjenem razredu opravimo tudi preverjanja vhodnih parametrov s strani uporabnika, kot npr., ali je vnesena teža obroka večja od vsaj 20 g. Če ni, se uporabniku vrne ustrezno sporočilo, da mora minimalna teža obroka znašati vsaj 20 g. V našem primeru servlet skrbi tudi za sporočila uporabniku, kot npr., da za vnesene parametre ni bila najdena zadostna količina živil za tvorjenje vsaj enega predloga obroka. Poleg ustreznega sporočila se vrne stranem JSP tudi nekaj ključnih atributov. Z njihovo pomočjo nastavimo vnosna polja na vrednosti, ki jih je vnesel uporabnik, preden jih je potrdil in s tem zahteval izračun predlogov obrokov.

4.3.2.4 JSP, CSS in JavaScript

Strani JSP nekako predstavljajo uporabniški vmesnik naše aplikacije. Odvisne so od komunikacije s servleti in skupaj s podlogami CSS skrbijo za ustrezen prikaz podatkov v spletnem brskalniku. Imamo dve glavni strani JSP, prva je uporabljena pri zaslonu za izračun predlogov obrokov, druga pa pri izpisu vseh živil glede na tip iz naše podatkovne baze z živili (slika 10). Sta pa obe glavni strani JSP sestavljeni iz treh strani JSP oz. segmentov (slika 17), ki so uporabljeni pri izdelavi večine današnjih sodobnih spletnih strani.



Slika 17: Zgradba glavne strani JSP v aplikaciji.

Ti segmenti so:

- **glava** (angl. *Header*),
- **vsebina** (angl. *Content*),
- **noga** (angl. *Footer*).

S pomočjo omenjenih treh gradnikov je izdelava in dodajanje novih spletnih strani JSP enostavno, saj strani JSP, ki predstavljata glavo in nogo, preprosto vključimo v novo stran z novo vsebino. S tem imamo avtomatično zagotovljeno glavno idejo prikaza spletne aplikacije in nam tako ni treba kopirati izvorne kode na straneh JSP. Primer takšnega nepotrebnega kopiranja bi bila npr. nova opcija v meniju spletne aplikacije, ki jo v našem primeru dodamo samo v glavo in je avtomatično vključena na vseh straneh JSP.

Poleg že omenjenih podlog CSS, s katerimi določimo postavitev posameznih gradnikov v aplikaciji, pa strani JSP vsebujejo še celo kopico funkcij JavaScript. Z njimi rešujemo predvsem enostavne probleme in dogodke na strani JSP, kot npr. pritisk na gumb *Clear* (slika 17), s katerim zbrišemo vnosna polja na strani in nastavimo privzete vrednosti. S funkcijami JavaScript si pomagamo tudi pri branju vrednosti iz izbirnih tipk (angl. *Radio button*). Pri tem gre za uporabo preproste funkcije JavaScript, ki na podlagi identifikacije izbirne tipke pridobi vrednost, ki jo je uporabnik izbral. Funkcijo seveda uporabljamo na strani JSP, in to pri vsaki uporabnikovi spremembi pri izbirnih tipkah (slika 17). Poleg omenjene funkcije uporabljamo jezik JavaScript tudi za nastavljanje privzetih in naključnih vrednosti (gumba *Default* in *Random* s slike 17) v vnosna polja, če uporabnik ne želi vnesti svojih kriterijev za izračun predlogov obrokov.

4.3.3 Spletni brskalniki

Kompatibilnost, delovanje in izgled spletne aplikacije v vseh modernih spletnih brskalnikih je najverjetneje največji izziv vseh razvijalcev spletnih strani in aplikacij. Tu gre za neke vrste modularnost spletnih aplikacij glede na vrsto spletnega brskalnika, in sicer na podoben način, kot to govorimo za modularnost aplikacije z vidika njene platformne neodvisnosti, tj. neodvisnosti od operacijskega sistema. Seveda pa je ta problem pogojen predvsem s kompleksnostjo izdelave spletne aplikacije. V našem primeru smo imeli relativno lahko nalogo pri reševanju omenjenega problema, saj je naša spletna aplikacija izdelana modularno in dokaj enostavno. Tu je še posebej pomagala uporaba podlog CSS, ki se med spletnimi brskalniki ne razlikujejo preveč. Za morebitne razlike v prikazovanju spletne aplikacije v spletnih brskalnikih s pomočjo podlog CSS lahko na današnjem spletu dobimo koristne informacije, in sicer, kje v podlogah CSS prihaja do različnih odzivanj in posledično

prikazovanj dela spletne strani med spletnimi brskalniki. Na podlagi teh informacij lahko uporabimo kakšno drugo tehniko za praktično enak rezultat oz. prikaz dela spletne strani, ki pa se med različnimi brskalniki ne spreminja. Seveda pa na današnjem tržišču obstaja tudi nekaj zunanjih knjižnic, kot je npr. JQuery, katerih uporaba med drugim lahko rešuje tudi omenjene probleme.

Aplikacijo smo razvijali s pomočjo spletnega brskalnika *Google Chrome*, prav tako pa smo vsak zaključen sklop funkcionalnosti aplikacije testirali z brskalnikoma *Mozilla Firefox* in *Internet Explorer*, kjer smo naleteli na največ odstopanj glede na prva dva, tako v izgledu spletne aplikacije kot tudi pri sami funkcionalnosti. Seveda pa je potrebno aplikacijo vedno testirati in po potrebi nadgrajevati v novih verzijah spletnih brskalnikov, če želimo zagotoviti njeno konsistenco, prav tako pa se stalno nadgrajujejo tehnologije za izdelavo in proženje spletnih aplikacij.

4.4 Testiranje aplikacije

Testiranje aplikacije in predvsem testiranje posameznih komponent aplikacije ter izvorne kode smo izvajali s programskim ogrodjem JUnit, ki smo ga predstavili v poglavju 4.1.3.3. Izdelali smo več testnih razredov glede na določeno funkcionalnost, ki smo jo želeli testirati. Tu nam je bilo še posebej v pomoč programsko orodje Apache Maven (opisano v poglavju 4.1.5), s katerim se ob vsakem prevodu programske kode naše aplikacije izvedejo tudi vsi testi, ki smo jih izdelali. V primeru, da je kateri od testov neuspešen, se naša aplikacija ne prevede in s tem ne zapakira za distribucijo. To je še posebej uporabno takrat, ko nadgrajujemo aplikacijo in s tem spreminjamo obstoječo izvorno kodo. Tako imamo vedno zagotovljene teste za določene sklope aplikacije in z lahkoto ugotovimo programerske napake, ki smo jih napravili v sklopu spreminjanja izvorne kode. Poglejmo si primer takšnega testiranja, kot to prikazuje slika 18.

```
@Test
public void testCompare()
{
  setFirstProduct();
  setSecondProduct();
  ProductsComparable comparable = new ProductsComparable();

  // Products are equal good
  Assert.assertEquals(0, comparable.compare(this.first, this.second));
  // Product 1 is better on sum difference
  this.first.setPRO_SUM(80);
  Assert.assertEquals(-1, comparable.compare(this.first, this.second));
  // Product 2 is better on sum difference
  this.second.setPRO_SUM(70);
  Assert.assertEquals(1, comparable.compare(this.first, this.second));

  // Products are equal good on sum difference, let the vitamins sum decide
  setFirstProduct();
  setSecondProduct();
  // Product 1 is better on vitamins sum
  this.first.setVIT_A(10);
  Assert.assertEquals(-1, comparable.compare(this.first, this.second));
  // Product 2 is better on vitamins sum
  this.second.setVIT_C(15);
  Assert.assertEquals(1, comparable.compare(this.first, this.second));
```

Slika 18: Primer testiranja sklopa aplikacije.

V prikazanem primeru testiranja določenega sklopa aplikacije oz. izvorne kode testiramo pravilnost primerjanja atributov dveh živil oz. pravilnost naše hevristične funkcije iz poglavja 3.2.3. Slika prikazuje le testa prvih dveh kriterijev hevristične funkcije. V prvem kriteriju primerjamo živili na podlagi vsote razlik med beljakovinami, ogljikovimi hidrati in maščobami. Kot lahko vidimo, najprej primerjamo dve povsem enaki živili oz. živili z

enakimi vrednostmi atributov, nato prvemu živilu spremenimo skupno število beljakovin in ga s tem »izboljšamo«. Sedaj pričakujemo rezultat, da je prvo živilo »boljše« v primerjavi z drugim. Podobno to naredimo tudi pri drugem kriteriju, kjer primerjamo živili na podlagi količine vsebovanih vitaminov. Test izvajamo tako, da spreminjamo količini dveh različnih vitaminov, saj naša hevristična funkcija deluje na podlagi vsote vseh vitaminov v živilu.

Na prikazani način smo praktično testirali večino sklopov naše aplikacije in s tem zagotovili njeno pravilno delovanje pri izračunih predlogov obrokov.

4.5 Ovrednotenje rešitve

V sklopu ovrednotenja naše rešitve bomo pregledali ocene nekaterih različnih predlogov obrokov. Pri tem smo se odločili za ocenitev predlogov s strani tretje osebe z znanjem in izkušnjami na področju zdrave in športne prehrane. Podali smo predloge obrokov, ki smo jih pridobili s pomočjo različnih vhodnih kriterijev. Napravili smo spletno anketo za ovrednotenje predlogov obrokov na podlagi izbranih splošnih kriterijev, ki smo jih določili sami. Kot smo že v zahvali omenili, smo za primer ocenitve oz. za izpolnitev spletne ankete prosili osebnega trenerja Bojana Justina.

V spletno anketo smo vključili tri različne predloge obrokov in osem kriterijev za ocenitev. V nadaljevanju si bomo ogledali omenjene predloge, kriterije in stopnje za ocenitev, na koncu pregledali ocene po kriterijih ter izračunali njihovo povprečje.

4.5.1 Predlogi obrokov za ocenitev

1. **Predlog obroka številka 1**

Uporabnikovi kriteriji

Grams per meal:	300	g
Calor. per meal:	500	kcal
Proteins (Sum):	50	g
Carboh. (Sum):	80	g
Fat (Sum):	20	g
Meal nutritions:	○ 1 ○ 2 ◉ 3 ○ 4 ○ 5	

Rezultat predloga obroka

NAME	SUM (g)	PRO_SUM (g)	CH_SUM (g)	FAT_SUM (g)	FAT_CHO (mg)	EN_SUM (kcal)	VIT_A (iu)	VIT_B6 (mg)	VIT_C (mg)	VIT_E (mg)	MIN_CA (mg)	MIN_FE (mg)	MIN_MG (mg)	H2O (g)
Divjačina, elk, surovo	160	36,72	0	2,32	88	177,6	0	0	0	0	6,4	4,42	36,8	119,01
Sauce, NESTLE, CHEF-MATE Hot Dog Chili Sauce, ready-to-serve	20	0,85	2,93	0,76	1,4	22	134,4	0,02	0,02	0	6,2	0,31	4	14,99
Makaroni, iz polnozrnate pšenične moke, surovi	80	11,7	60,02	1,12	0	278,4	0	0,18	0	0	32	2,9	114,4	5,87
SUM	260	49,28	62,95	4,2	89,4	478	134,4	0,2	0,02	0	44,6	7,63	155,2	139,87

Stats: Calories: 500 kcal Proteins: 203 kcal Carbohydrates: 258 kcal Fat: 39 kcal

Slika 19: Predlog obroka številka 1 za ocenitev.

Slika 19 prikazuje predlog obroka številka 1 za ocenitev. Kot lahko vidimo je predlog sestavljen iz treh živil, od katerih največji delež (po teži) predstavlja prvo živilo, sledi delež tretjega in nato delež drugega živila.

2. Predlog obroka številka 2

Uporabnikovi kriteriji

Grams per meal:	400	g
Calor. per meal:	500	kcal
Proteins (Sum):	60	g
Carboh. (Sum):	30	g
Fat (Sum):	15	g
Meal nutritions:	○ 1 ○ 2 ◉ 3 ○ 4 ○ 5	

Rezultat predloga obroka

NAME	SUM (g)	PRO_SUM (g)	CH_SUM (g)	FAT_SUM (g)	FAT_CHO (mg)	EN_SUM (kcal)	VIT_A (iu)	VIT_B6 (mg)	VIT_C (mg)	VIT_E (mg)	MIN_CA (mg)	MIN_FE (mg)	MIN_MG (mg)	H2O (g)
Cereals ready-to-eat, QUAKER, QUAKER 100% Natural Cereal with oats, honey, and raisins	40	4,13	26,3	7,16	0,8	186	1,2	1,14	1,08	1,21	44	1	43,2	1,7
Sir, Cottage Cheese Natur Meggle	120	12	3,6	4,8	0	114	0	0	0	0	0	0	0	0
Ribe, tuna, sveža, rumenoplavuta, surova	180	42,08	0	1,71	81	194,4	108	1,62	1,8	0,9	28,8	1,31	90	127,78
SUM	340	58,21	29,9	13,67	81,8	494,4	109,2	2,76	2,88	2,11	72,8	2,31	133,2	129,49

Stats: Calories: 490 kcal Proteins: 239 kcal Carbohydrates: 123 kcal Fat: 128 kcal

Slika 20: Predlog obroka številka 2 za ocenitev.

Slika 20 prikazuje predlog obroka številka 2 za ocenitev. Kot lahko vidimo, je predlog, prav tako kot predlog številka 1, sestavljen iz treh živil, od katerih največji delež (po teži) predstavlja tretje živilo, sledi delež drugega in nato delež prvega živila.

3. Predlog obroka številka 3

Uporabnikovi kriteriji

Grams per meal:	200	g
Calor. per meal:	300	kcal
Proteins (Sum):	40	g
Carboh. (Sum):	60	g
Fat (Sum):	10	g
Meal nutritions:	○ 1 ◉ 2 ○ 3 ○ 4 ○ 5	

Rezultat predloga obroka

NAME	SUM (g)	PRO_SUM (g)	CH_SUM (g)	FAT_SUM (g)	FAT_CHO (mg)	EN_SUM (kcal)	VIT_A (iu)	VIT_B6 (mg)	VIT_C (mg)	VIT_E (mg)	MIN_CA (mg)	MIN_FE (mg)	MIN_MG (mg)	H2O (g)
Riž, beli, kratkozrnati, kuhan	120	2,83	34,48	0,23	0	156	0	0,07	0	0	1,2	1,75	9,6	82,24
Leča, surova	40	10,32	24,03	0,42	0	141,2	15,6	0,22	1,76	0,2	22,4	3,02	48,8	4,16
SUM	160	13,15	58,51	0,65	0	297,2	15,6	0,29	1,76	0,2	23,6	4,77	58,4	86,4

Stats: Calories: 300 kcal Proteins: 54 kcal Carbohydrates: 240 kcal Fat: 6 kcal

Slika 21: Predlog obroka številka 3 za ocenitev.

Slika 21 prikazuje predlog obroka številka 3 za ocenitev. Tokrat je predlog sestavljen le iz dveh živil, od katerih največji delež (po teži) predstavlja prvo živilo.

4.5.2 Kriteriji in stopnje za ocenitev

Slika 22: Kriteriji in stopnje za ocenitev predlogov obrokov.

Slika 22 prikazuje kriterije in stopnje za ocenitev predlogov obrokov. Kot smo že omenili in kot lahko vidimo, smo uporabili osem različnih kriterijev s petimi stopnjami za ocenitev posameznega predloga obroka. Kriterije smo določili sami, večino od njih smo definirali na podlagi trenutno podprtih vhodnih uporabnikovih kriterijev in na osnovnih kriterijih za ovrednotenje obrokov (kot je npr. kriterij kakovost hranil v obroku).

4.5.3 Ocene predlogov obrokov in njihovo povprečje

Tabela 7 prikazuje predloge obrokov in kriterije za ocenitev, dejanske ocene s strani strokovnjaka ter izračun njihovega povprečja in skupnega povprečja preko vseh kriterijev.

	Pr. obr. št. 1	Pr. obr. št. 2	Pr. obr. št. 3	POVPREČJE
Spl. kakovost hranil v obr.	3	4	2	3,00
Ocena sk. kalorij obr.	2	3	2	2,33
Razm. med b., o. h. in m.	4	4	1	3,00
Ocena sk. kol. beljakovin	3	5	2	3,33
Ocena sk. kol. oglj. hidr.	4	3	4	3,67
Ocena sk. kol. maščob	3	4	2	3,00
Upor. obr. za športnika	4	2	4	3,33
Upor. obr. za hujšanje	2	4	1	2,33
POVPREČJE POVPREČJA				3,00

Tabela 7: Ocene predlogov obrokov in njihovo povprečje.

Pred pregledom in analizo zgornjih ocen moramo najprej poudariti, da smo že pri samih vhodnih uporabnikovih kriterijih v aplikaciji uporabili zelo smiselne in v realnosti uporabne vrednosti in razmerja med lastnostmi želenega obroka. Temu primerno zato nismo pričakovali strogo negativnih ocen za večino kriterijev pri posameznem predlogu obroka.

Če pogledamo povprečno vrednost na nivoju posameznega kriterija, lahko vidimo, da je vrednost najnižja pri kriterijih *Ocena skupnih kalorij obroka* in *Uporabnost obroka za hujšanje.* Tu povprečje znaša 2,33, kar pomeni, da se povprečna ocena za ta dva omenjena kriterija nagiba k drugi najslabši stopnji za ocenitev tj. oceni slabo. V nasprotnem primeru pa je vrednost povprečja najvišja pri kriteriju *Ocena skupne količine ogljikovih hidratov*, in sicer znaša 3,67. Ta druga ocena za omenjeni kriterij se tako približuje drugi najboljši stopnji za ocenitev, tj. oceni zelo dobro. Tu tako lahko zatrdimo, da ocena za podane predloge obrokov med posameznimi kriteriji na splošno niha v rangu ene stopnje za ocenitev, kar je dobro.

Končna ocena, kot jo prikazuje podatek *POVPREČJE POVPREČJA*, tj. vrednost povprečja na podlagi povprečnih vrednosti za posamezne kriterije, pa znaša 3,00. Gre za oceno srednje stopnje na naši lestvici za ocenitev predlogov obrokov, s čimer lahko potrdimo, da so podane predloge obrokov po kakovosti na splošno dobre in da aplikacija ter s tem algoritem za optimizacijo sestave športnih obrokov glede na vsebovanost hranil deluje zadovoljivo.

5 Zaključek

V diplomski nalogi smo izdelali spletno aplikacijo za optimizacijo sestave športnih obrokov glede na vsebovanost hranil. Aplikacija s pomočjo zbirke živil in njihovih lastnosti sestavi različne predloge obrokov. Pri tem ima ključno vlogo uporabnik, ki s svojimi kriteriji določi lastnosti in sestavo ter število želenih predlogov obrokov. Če uporabniku ne ustreza noben izmed podanih predlogov, lahko izračun vedno ponovi in pridobi povsem nove predloge, sestavljene iz novih živil.

V sklopu diplomske naloge smo analizirali priporočila ter principe zdrave in uravnotežene prehrane, na podlagi katerih smo rešitev zasnovali in izdelali. Podrobno smo opisali algoritem za svojo rešitev. Navedli in opisali smo uporabljena programska orodja in tehnologije. Predstavili smo celotno zgradbo aplikacije od izdelave podatkovne baze do uporabe aplikacije v spletnih brskalnikih. Na koncu smo izdelali spletno anketo, kjer smo za nekatere rezultate svoje rešitve, torej predloge obrokov, pridobili primere ocenitve, jih analizirali in s tem podali ovrednotenje rešitve.

Sestava jedilnika in obrokov predstavlja razmeroma kompleksen problem, pogojen z različnimi dejavniki. Glavna kritika naše rešitve bi bila omejitev števila preiskanih kombinacij med živili. To smo morali storiti zaradi časovne prezahtevnosti preiskovanja vseh možnih kombinacij med živili. Idej za izboljšave rešitve nam zaradi omenjene kompleksnosti problema gotovo ne bo zmanjkalo. Glavno kritiko, ki smo jo podali zgoraj, bi lahko rešili s tem, da bi vsem živilom v podatkovni bazi določili nekakšne prioritete in z njimi določili, kako pomembno je živilo za vključitev v obrok. Tako bi izredno omejili nabor na glavna, najpomembnejša, in ostala živila. S tem bi preiskali vse možne kombinacije in pridobili najboljše predloge obrokov glede na določene zahteve uporabnika. Prav tako vidimo izboljšave v nadgradnji rešitve na določene vrste obrokov, kot so zajtrk, kosilo in večerja, ter na morebitno razširitev vhodnih kriterijev. Tako bi uporabnikom, ki morajo paziti na določene sestavine v obrokih, omogočili še lažjo uporabo naše rešitve pri vnosu kriterijev in jim s tem olajšali podrobno analizo posameznih predlogov obrokov po izračunu.

Za morebitno uporabo aplikacije na trgu bi verjetno morali svojo rešitev razširiti tudi z vidika uporabe glede na posameznega uporabnika. S tem bi bila aplikacija bolj »prijazna« do uporabnika na način, kot ga danes omogoča večina boljših spletnih aplikacij. Rešitev bi nadgradili z odstranjevanjem in dodajanjem živil med obroki na zahtevo, s čimer bi si uporabnik lahko sam sestavil jedilnik z živili po svojih željah in ga na koncu seveda podrobno analiziral.

Kazalo tabel

Kazalo slik

Viri in literatura

[1] (2012) Uravnotežena prehrana. Dostopno na: http://www.mojirecepti.com/prehrana-zdravje/uravnotezena-prehrana.html

[2] (2012) Prehrana pri sladkorni bolezni. Dostopno na: http://www.vpd.si/sl/sladkorna-bolezen/Prehrana

[3] Igor Kononenko, *Strojno učenje*, Ljubljana 2005, pogl. 5.3.

[4] (2009) Kaj je MySQL? Dostopno na:http://www.e-uspeh.com/kaj-je-mysql.html

[5] (2012) Toad (software). Dostopno na:http://en.wikipedia.org/wiki/Toad_(software)

[6] (2012)Spletno mesto Java EE. Dostopno na:http://download.oracle.com/javaee

[7] Dustin R. Callaway, *Inside Servlets*, New Jersey, 2001, pogl. 4.

[8] Jim Farley, William Crawford & David Flanagan, *Java Enterprise in a Nutshell*, California: Sebastopol, 2002, pogl. 6.

[9] (2011) JUnit zgodovina. Dostopno na:http://www.martinfowler.com/bliki/Xunit.html

[10] (2012) JavaScript. Dostopno na:http://sl.wikipedia.org/wiki/JavaScript

[11] (2012) CSS. Dostopno na:http://sl.wikipedia.org/wiki/CSS

[12] (2012) Apache Maven. Dostopno na:http://en.wikipedia.org/wiki/Apache_Maven

[13] (2012) Apache Tomcat. Dostopno na:http://en.wikipedia.org/wiki/Apache_Tomcat

[14] (2012) Eclipse. Dostopno na:http://en.wikipedia.org/wiki/Eclipse_(software)

[15] (2012) Hranilne vrednosti živil. Dostopno na: http://www.cenim.se/zivila.php