Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Berkovič

Razpletanje sej spletnega strežnika

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Boštjan Slivnik

Ljubljana, 2016

Izvorna koda programa je dostopna na naslovu:
https://github.com/kb2623/spletne-seje

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Spletni strežniki hranijo dnevnik uporabniških sej. Vse seje zapisujejo v eno samo datoteko, zato ni enostavno ugotoviti, kateri zapisi tvorijo posamezne seje. Izdelajte program, ki kar najbolje rekonstruira seje shranjene v posameznemu dvevniku spletnega strežnika.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Klemen Berkovič sem avtor diplomskega dela z naslovom:

Razpletanje sej spletnega strežnika (angl. *Web session reconstruction*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 9. marec 2015 Podpis avtorja:

Zahvaljujem se asistentu, dr. Marku Poženelu za njegovo strokovno pomoč, mentorju, doc. dr. Boštjanu Slivniku za pomoč in vodenje pri opravljanju diplomskega dela, kolegoma, Andreju Hafnerju in Simonu Trebovšku za konkretne primere log datotek spletnega strežnika in kolegici, Tei Berkovič za pomoč in vzpodbudo pri izdelavi pisnega izdelka. Posebna zahvala velja staršem, ki so mi omogočili študij in me ves čas podpirali.

Kazalo

Slike

Tabele

Seznam uporabljenih kratic

kratica	angleško	slovensko
ACID	atomicity, consistency, isolation, durability	atomičnost, doslednost, izolacija, trajnost
API	application programming interface	vmesnik za programiranje aplikacij
AJAX	asynchronous JavaSript and XML	asinhron JavaSript in XML
ARP	active record pattern	vzorec aktivnega zapisa
BCEL	byte code engineering library	knjižnica za manipulacijo vmesne kode
BYOD	bring your own device	prinesi svojo napravo
CLI	command-line interface	vmesnik z ukazno vrstico
CRUD	create, read, update and delete	kreiraj, beri, posodobi in izbriši
ELF	extended log format	razširjen log format
GUI	graphical user interface	grafični uporabniški vmesnik
HTML	hypertext markup language	označevalni jezik za hipertekst
HTTP	hypertext transfer protocol	protokol za prenos hiperteksta
HTTPS	HTTP secure	varni HTTP
HQL	Hibernate query language	Hibernate poizvedovalni jezik
IP	internet protocol	internetni protokol
JAR	Java archive	Java arhiv
JCP	Java community process	proces skupnosti Jave
JDBC	Java database connectivity	povezovanje Jave na podatkovno bazo

kratica	angleško	slovensko
JDK	Java development kit	razvojno orodje za programski jezik Java
JDO	Java data objects	Java podatkovni objekt
JPA	Java persistence API	obstojni Java API
JPQL	Java persistence query language	obstojni Java poizvedovalni jezik
JRE	Java runtime environment	okolje za izvajanje Java programov
JSR	Java specification requests	specifikacija zahtev za programski jezik Java
jOOQ	Java object oriented querying	Java objektno poizvedovanje
RDBMS	relational database management system	sistem za upravljanje z relacijskimi podatkovnimi bazami
RFC	request for comments	zahteva za komentar
RSS	rich site summary	bogat povzetek strani
NCSA	national center for supercomputing applications	nacionalni center za super računalniške aplikacije
ORM	object relational mapping	objektno relacijska preslikava
SQL	structured query language	strukturiran poizvedovalni jezik
TCP	transmission control protocol	protokol za nadzor prenosa
XML	extensible markup language	razširljiv označevalni jezik
W3C	world wide web consortium	konzorcij za svetovni splet

Povzetek

Naslov: Razpletanje sej spletnega strežnika

Dostop do spleta je dandanes mogoč že skoraj povsod, zato se večina namiznih aplikacij seli na splet, kar pripomore k konstantnim posodobitvam spletnih tehnologij. Spletni strežniki so ena izmed pomembnejših spletnih tehnologij, ki olajšuje delo razvijalcem in vzdrževalcem, ter pripomore k boljši uporabniški izkušnji. Zaradi mnogih izboljšav spletnih strežnikov je steznik zmožen opravljati mnogo operacij, med katerimi je tudi beleženje dostopa do virov, ki se nahajajo na strežniku. Diplomska naloga obsega razvoj aplikacije za rekonstrukcijo sej iz log datotek spletnega strežnika. Aplikacija kot vhod pridobi eno ali več tekstovnih log datotek spletnega strežnika. Izhod aplikacije je relacijska podatkovna baza. V izhodni podatkovni bazi so shranjene rekonstruirane seje uporabnikov, ki jih lahko uporabimo za preverjanje delovanja spletnega strežnika ali pa jih uporabim za izboljševanje vsebin na spletnem strežniku. Seje iz izhodne podatkovne baze se lahko uporabljene tudi za različne analize, med katerimi je tudi analiza uporabniškega profila.

Ključne besede: Spletni strežnik, log datoteke, klikotok, osejevanje, podatkovna baza.

Abstract

Title: Web session reconstruction

Internet access is now possible from almost everywhere, so most desktop applications are moving to the web, which contributes to the constant updates of online technologies. Web servers are one of the most important web technologies, which facilitate the work of developers and maintainers, and contributes to a better user experience. Because of the many improvements to web servers, web server is able to perform many operations, including the recording of access to resources located on the server. The thesis comprises the development of applications for the reconstruction of the session in the web server log files. Applications input is one or more text log files from a web server. Applications output is a relational database. The output database stores reconstructed user sessions, which can be used to verify the functioning of the Web server, or use them to improve the content on the web server. Sessions from the output database can also be used for various analyzes, including an analysis of the user profile.

Keywords: Web server, log files, clickstream, sessionization, database.

Poglavje 1

Uvod

Dostop do interneta oziroma svetovnega spleta se hitro širi in posledično narašča tudi število spletnih mest, ki jih lahko obiščemo. Povečuje se število brezplačnih dostopnih točk, ki se uporabljajo za dostop do spleta. Za dostop do spleta se že dolgo ne uporablja več žice, ki predstavlja medij za prenos podatkov od izvora do ponora, temveč ga zamenjuje medij, ki je neviden in ima preprosto namestitev. Medij, ki zamenjuje žico je kar zrak. Po zraku se podatki prenašajo z uporabo radijskih valov, ki so že uporabljeni pri mobilnih telefonih, televizorjih, radiih ... Hitrosti prenosa se konstantno povečujejo. Povečuje se tudi število naprav, ki lahko odstopajo do spleta, tako da internet naprave (ang. *Internet of Things*) niso več sanje, temveč resničnost. Pri spletu poznamo dva tipa naprav. To so odjemalci in strežniki. Odjemalci so preproste naprave, ki pridobivajo podatke s strežnikov ter jih prikažejo. Lastnost prikaza je odvisna od naprave, saj nekatere naprave nimajo tega izhoda. Ker odjemalci podatke samo pridobivajo in ne tudi procesirajo, so lahko odjemalci preproste naprave, recimo je to večina mobilnih naprav, ki morajo varčevati z energijo. Odjemalci pošiljajo zahteve strežniku, ta pa servira zahteve odjemalcev. Strežniki so robustne naprave, ki podpirajo sočasno procesiranje in zahtevne odevale podatkov. Večina strežnikov podpira protokol HTTP, ki je standarden protokol v spletnih strežnikih. Spletne aplikacije delujejo na spletnih strežnikih ter omogočajo kvalitetnejši razvoj aplikacije, kar je privedlo do tega, da se namizne aplikacije transformira v spletne aplikacije. Ena pomembnejših prednosti spletne aplikacije je ta, da uporabniku ni potrebno prenesti celotnega program na svojo napravo oziroma odjemalca, samo zato, da bi jo uporablja oziroma testi-

ral. Ker se celotna aplikacija nahaja le na strežniku, je pri posodobitvi aplikacije potrebno posodobiti samo datoteke, ki se nahajajo na strežniku. Tako spletne aplikacije zagotavljajo, da odjemalci uporabljajo vedno najnovejšo verzijo aplikacije. Spletne aplikacije nam zagotavljajo tudi prenosljivost na različne operacijske sisteme in naprave, ki imajo odjemalca oziroma posrednika s podporo protokolu HTTP. Primer aplikacije, ki dokazuje, da so spletne aplikacije zmožne prodirati na različna področja, je prototip sistema za evidentiranje živine [30], ki ima tudi aplikacijo za naprave Android, vendar pa tudi ta komunicira s spletnim strežnikom, tako da je zagotovljena sinhronizacija podatkov med različnimi napravami in aplikacijami. Primer spletne aplikacije je aplikacija, ki je namenjena evidentiranju dela zaposlenih [35]. Spletna aplikacija je sicer predelava sistema za spletne trgovine, vendar je dober primer aplikacije, ki bi podjetju omogočala politiko BYOD ter s tem olajšala stroške podjetja do zaposlenih, saj bi lahko zaposleni uporabljali svoje naprave.

Na spletu najdemo veliko število spletnih mest, katerih glavna panoga je prodaja različnih izdelkov uporabnikom spletnega mesta. Taka spletna mesta tekmujejo med seboj za nove in že obstoječe uporabnike oziroma kupce, ki predstavljajo njihov glavni tržni delež. Informacije o tipih uporabnikov pripomorejo spletni trgovini k izboljšanju spletnega mesta in s tem večjo verjetnost, da bodo že obstoječi uporabniki ostali še vedno njihovi kupci ter da bo spletno mesto privabilo nove uporabnike. Informacije o tipih uporabnika na spletnem mestu lahko uporabimo tudi na drugih tipih spletnih mest za izboljševanje vsebine spletnega mesta. Informacije o tipih uporabnikov se izdelajo na podlagi sej, ki so jih uporabniki imeli na spletnem mestu. Seje uporabnikov v diplomskem delu izdelujemo na podlagi podatkov, ki se nahajajo v log datotekah spletnih strežnikov. Primaren namen uporabe log datotek spletnega strežnika je pomoč pri preverjanju pravilnosti delovanja implementacije spletnega strežnika. Podatki v log datoteki predstavljajo klikotoke uporabnikov, ki jih v diplomskem delu izkoriščamo za izdelavo sej. Uporabili smo že obstoječe rešitve in metode, ki se uporabljajo za rekonstrukcijo sej iz log datotek spletnega strežnika, med katerimi je uporaba časovne hevristike in markovskih modelov. Vendar pa rekonstrukcija seje z uporabo log datotek spletnega strežnika ne izdela točnih sej, saj klikotoki niso vedno popolni ter lahko vsebujejo zavajajoče podatke, ki nam otežujejo proces rekonstrukcije sej.

Izdelek diplomskega dela kot izhod izdela relacijsko podatkovno bazo, ki vsebuje rekonstruirane seje uporabnikov spletnega mesta. Relacijsko podatkovno bazo smo uporabili, ker zmanjša velikost podatkov v primerjavi z vhodno log datoteko. Relacijska podatkovna baza omogoča lažji pregled podatkov in uporabo različnih poizvedb SQL za prikaz in sortiranje podatkov, kar pa nam vhodna tekstovna log datoteka ne omogoča. Izdelek diplomskega dela se ne omejuje samo na eno relacijsko podatkovno bazo, temveč podira množico relacijskih podatkovnih baz, ki morajo izpolnjevati določene pogoje. Kljub dodatnim pogojem izdelek diplomskega dela podpira večino aktualnih relacijskih podatkovnih baz, med katerimi so privzeto podprte MySQL, Microsoft SQL Server, PostgreSQL, H2, HSQLDB ... V primeru, ko izbrana relacijska podatkovna baza ni prevzeto podprta, so v diplomskem delu navodila, ki vam bodo pomagala rešiti ta problem nekompatibilnosti, saj smo tudi v izdelku morali dodati podporo za relacijsko podatkovno bazo SQLite.

V diplomskem delu je veliko slik, ki prikazujejo razredne diagrame, vendar diagrami vsebujejo nekaj nestandardnih konstruktov razrednega diagrama. Nestandardna konstrukta sta metodi za nastavljanje in pridobivanje vrednosti polj razreda. Metode za nastavljanje so prikazane s predpisom `set...(...:...): void`, kjer `...` predstavlja ime polja oziroma njegov podatkovni tip. Metode za pridobivanje so prikazane s predpisom `get...():...`, kjer velja isto pravilo za simbol `...`, kot pri metodah za nastavljanje.

Poglavje 2

Spletni strežnik

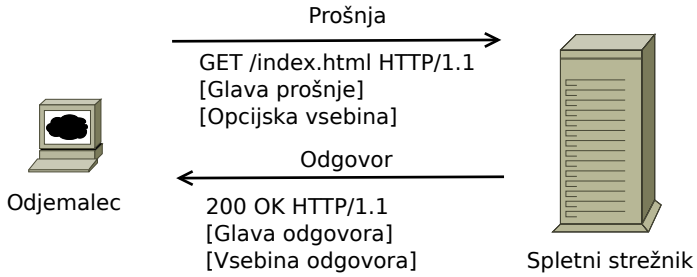
Primarna funkcija spletnega strežnika je shranjevanje, procesiranje in posredovanje spletnih strani odjemalcu. Komunikacija med odjemalcem in strežnikom poteka preko protokola HTTP ali HTTPS. Protokol HTTPS je nadgradnja protokola HTTP, ki se uporablja pri varnih oziroma kodiranih povezavah. Strani, ki jih prejme odjemalec, so zapisane v datoteki HTML, ki vsebujejo slike, stilske predloge in skripte. Napisane so v različnih programskih jezikih, med katerimi je najpogosteje uporabljen JavaScript.

HTTP je protokol brez stanj, ki deluje na aplikacijskem nivoju sklada TCP/IP. Plasti sklada TCP/IP si sledijo v naslednjem zaporedju: aplikacijska, prenosna, omrežna, povezovalna in fizična plast. Plasti med seboj komunicirajo, tako da ena plast izlušči podatke, namenjene izbrani plasti, ter ostanek podatkov pošlje višji ali nižji plasti. Smer pošiljanja obdelanih podatkov je odvisna od smeri komunikacije, ali podatke sprejemamo ali jih pošiljamo. S tako komunikacijo med plastmi sklad TCP/IP zagotavlja modularnost komunikacijskih protokolov.

Protokol HTTP bazira na arhitekturi odjemalec-strežnik, kar omogoča protokolu boljšo povezljivost, lažje prilagajanje, boljšo skalabilnost sistema, boljšo dosegljivost sistema, boljšo podatkovno integriteto ter zaradi redundance sistema zagotavlja, da sistem deluje kljub izpadu oziroma nedelovanju ene izmed njegovih komponent. Protokol HTTP ima tudi možnost pošiljanja podatkov iz odjemalca k strežniku. V ta namen se uporabljajo spletni obrazci (ang. *web form* ali *HTML forms*). Trenutno se v spletnih strežnikih uporabljajo protokoli HTTP/1.0 [34], HTTP/1.1 [24] in HTTP/2.0 [21], ki pa šele prihaja v veljavo.

Protokol HTTP v prenosni plasti sklada TCP/IP uporablja protokol TCP. Protokol TCP skrbi za prenos podatkov med strežnikom in odjemalcem. Protokol TCP zagotavlja odjemalcu prejemanje paketov v istem vrstnem redu, kot so bili ti poslani s spletnega strežnika. Vrata (ang. *port*) 80 so prevzeta vrata, na katerih deluje protokol HTTP. Vrata 443 pa so prevzeta vrata, na katerih deluje protokol HTTPS.

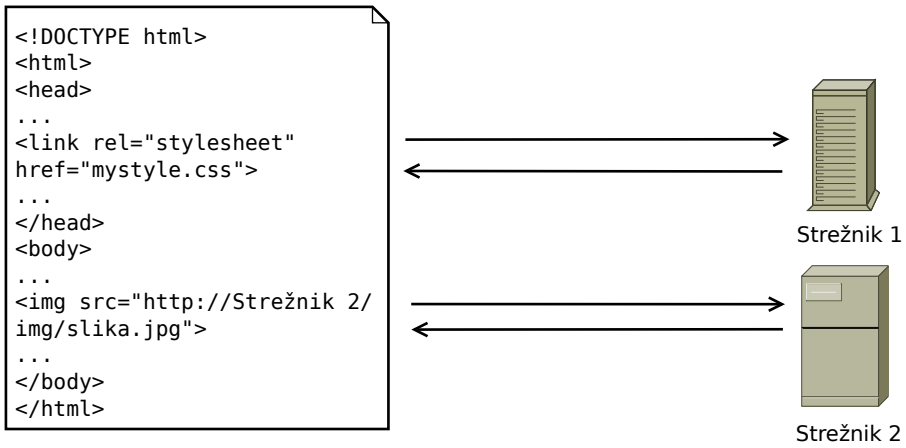
Protokol HTTP v omrežni plasti sklada TCP/IP uporablja protokol IP. Protokol IP se uporablja za usmerjanje podatkov od izvira do ponora. Izvor in ponora morata biti definira s številko IP, ki jih ločuje od drugih naprav na spletu. Številke IP so definirane v protokolu IPv4 [18] ali IPv6 [23].



Slika 2.1: Komunikacija med odjemalcem in strežnikom z uporabo protokola HTTP/1.1.

Slika 2.1 prikazuje osnovno komunikacijo med spletnim strežnikom in odjemalcem. Sliki 2.1 prikazuje samo podatke, ki se uporabljajo pri protokolu HTTP. Komunikacijo vedno začne odjemalec, spletni strežnik pa je v stalni pripravljenosti ter čaka na prošnje oziroma zahteve. Na sliki 2.1 odjemalec pošlje zahtevo po dokumentu `index.html`. Spletni strežnik mu na to zahtevo odgovori s sporočilom, ki vsebuje statusno kodo [27]. V primeru, ko je zahteva legitimna, ima odgovor status 200 in hkrati vsebuje zahtevano vsebino, kot je to prikazano na sliki 2.1. V primeru ne legitimne zahteve pa spletni strežnik odgovori samo s statusno kodo.

Slika 2.2 prikazuje nadaljnje postopke, ki se morajo zgoditi, da se stran `index.html` iz slike 2.1 dokončno prikaže uporabniku. Slika 2.2 ima na levi strani primer datoteke HTML, ki za dokončen prikaz strani potrebuje še stilsko predlogo in sliko. Kot je vidno s slike 2.2, se lahko dodatni resursi nahajajo na različnih spletnih strežnikih. Za pridobivanje obeh resursov uporabimo protokol HTTP,



Slika 2.2: Naslednji korak pri nalaganju spletne strani po prejetju dokumenta HTML.

zato je postopek pridobivanja obeh resursov enak kot na sliki 2.1.

Ena popularnejših rešitev, ki jo podpirajo spletni strežniki, je izdelava dinamičnih spletnih strani, ki jih spletni strežnik izdela s pomočjo dodatnega programskega jezika, npr. PHP, C#, Java ... Večina spletnih strežnikov podpira tudi virtualna gostovanja (ang. *Virtual hosting*), ki omogočajo, da se na enem IP naslovu nahaja več različnih spletnih mest.

Po anketi spletne strani Netcraft [8] o uporabljenosti spletnih strežnikov trenutno prevladuje spletni strežnik Apache, ki mu sledijo spletni strežnik podjetja Microsoft, spletni strežnik nginx in spletni strežnik podjetja Google. V nadaljevanju sledi opis formatov log datotek, najpogosteje uporabljenih spletnih strežnikov.

2.1 Log datoteke

Za učinkovito upravljanje spletnega strežnika so potrebne povratne informacije o aktivnostih in težavah, ki se lahko pojavijo med delovanjem spletnega stražnika. V ta namen spletni strežniki podpirajo beleženje dostopov do virov, ki se nahajajo na spletnem strežniku. Dostopi se beležijo v tekstovne oziroma log datoteke, ki pa imajo vnaprej določen format. Ena vrstica v log datoteki predstavlja zabeležen dostop do zahtevanega vira. Vrstice v log datoteki so urejene po kronološkem zaporedju obdelave zahtev, kar pomeni, da se vsaka nova obdelana zahteva zapiše na

konec log datoteke. Vrstica log datoteke vsebuje različne podatke, ki jih pri večni spletnih strežnikov lahko dodajamo ali odvzemamo. Tipi podatkov oziroma polja, ki so uporabljeni v vrstici log datoteke, so definirani v implementaciji spletnega strežnika. Vendar pa različne implementacije spletnih strežnikov uporabljajo standardna polja, ki imajo definiran vzorec podatkov, kot so številka IP, datum, čas, metoda HTTP ... Polja za beleženje identifikacijskega niza odjemalca imajo predpisan vzorec podatkov v RFC 2616 [25]. Polja za beleženje zahtevanega vira imajo predpisan vzorec podatkov v RFC 1738 [22], za relativne poti pa najdemo vzorec v RFC 1808 [26]. Polje za beleženje piškotka ima predpisan vzorec podatkov v RFC 6265 [19]. Vzorce standardnih polj smo v diplomskem delu razdelili na dva formata log datotek spletnih strežnikov; to sta format NCSA in W3C.

2.1.1 NCSA log format

Format NCSA definira vzorce polj v spletnem strežniku Apache, nginx, Caddy, Lighttpd ... Razlika med spletnim strežnikom Apache in ostalimi navedenimi implementacijami je v nastavitvi formata log datoteke, kjer ima vsaka izmed implementacij drugačen niz, ki se uporablja za imena tipov polj. V diplomskem delu se bomo osredotočili na spletni strežnik Apache, ki deluje na operacijskem sistemu Linux. Pri spletnem strežniku Apache imamo konfiguracijo spletnega strežnika, strnjeno v datoteki `apache2.conf`, v mapi `/etc/apache2`. V primeru, ko se na spletnem strežniku nahaja več spletnih mest, pa konfiguracijo za izbrano spletno stran najdemo v datoteki `default`, ki se nahaja v imeniku `/etc/apache2/stran`. Prevzeti spletni strežnik Apache uporablja v log datotekah format Common Log Format. Datoteka `access.log` je prevzeta datoteka za shranjevanje podatkov izbranega formata log datoteke. Format log datoteke spremenimo tako, da dodamo vrstico z modulom `CustomLog`, ki vsebuje dva argumenta. Prvi argument vsebuje nize imen polj, ki se začnejo z znakom `%`, ter ločilne znake, ki se uporabljajo za ločitev podatkov. Drugi argument predstavlja pot do datoteke, kamor format zapiše podatke. Prvi argument lahko uporabimo kot argument ukazne vrstice naši izdelani aplikaciji za opredelitev polj v log datoteki.

Tabela 2.2 prikazuje nekaj tipov polj, ki se lahko uporabijo v formatu log datoteke. Celoten spisek podprtih tipov polj lahko najdete v dokumentaciji spletenega strežnika Apache [1]. Prvi stolpec v tabeli 2.2 prikazuje niz, ki se uporablja v

ime tipa	opis
%h	Ime oddaljenega gostitelja. Beleži naslove IP, če imamo modul HostnameLookups izklopljen, v nasprotnem primeru beleži imena oddaljenih gostiteljev.
%l	Oddaljen logname [31]. Podatek vrne pošiljatelj, če imamo vklopljen modul IdentityCheck.
%r	Prva vrstica zahteve.
%u	Ime oddaljenega uporabnika, če je bila zahteva avtorizirana.
%t	Čas prejetja zahteve v oliki [18/Jan/2015: 12:21:45 -400]. Zadnji podatke v obliki predstavlja časovni pas od GMT.
%s	Koda statusa.
%b	Velikost odgovora v bajtih, ki ne vključuje glave HTTP. V primeru ko je velikost prenosa enak 0 bajtov, se namesto znaka 0 zapiše znak -.
%{Referer}i	Prejšnja obiskana stran.
%{User-agent}i	Identifikacijski niz odjemalca.

Tabela 2.2: Nekaj tipov polji formata NCSA.

modulu CustomLog za identifikacijo tipa podatka, ki se bo beležil v log datoteko.

Nekaj standardnih formatov ter primer prvega argumenta modula CustomLog za izdelavo tega formata:

Common Log Format

Je eden standardnih formatov, ki se uporablja kot prevzet format log datotek v različnih spletnih strežnikih. Argumenti, ki se uporabijo za izdelavo tega formata so %h %l %u %t %r %s %b.

Combined Log Format

Je drugi izmed standardnih formatov, ki se lahko uporablja kot format log datotek v spletnem strežniku Apache. Format doda formatu Common Log Format še polji napotitelja in niz uporabnikovega posrednika. Argumenti, ki se uporabijo za izdelavo tega formata so %h %l %u %t %r %s %b "\%{Referer}i\" "\%{User-agent}i\".

2.1.2 W3C log format

Format W3C definira vzorce polji v spletnem strežniku IIS, ki je last podjetja Microsoft. Format lahko najdemo tudi pod imenom `Extended Log File Format` [3]. Prevzet ločilni znak formata je presledek. Format ima veliko polj, ki so identična poljem formata NCSA, kot so to številka IP, metoda HTTP ... Razlike med formati so opazne pri vzorcu polja časa, piškotka in niza uporabnikovega posrednika. Polje za beleženje časa obdelave zahteve je predstavljeno s poljem za beleženje časa in s poljem za beleženje datuma. Pri vzorcu piškotka in niza uporabnikovega posrednika je razlika v ločilnem znaku, kjer se namesto presledka uporablja znak `+`. Format W3C v izhodno log datoteko zapiše tudi direktive, ki so poseben konstrukt tega formata. Direktive se v log datoteki začnejo z znakom `#`, ki mu sledi ime direktive, ter se konča z znakom `:`. Po znaku `:` sledi presledek ter podatki direktive. Podatki direktive so med seboj ločeni s presledkom. Najpogosteje uporabljene direktive so:

Version

Različica programske opreme, ki je bila uporabljena za izdelavo log datoteke.

Fields

Imena polji uporabljena v formatu log datoteke.

Software

Programska oprema, ki je bila uporabljena za izdelavo log datoteke.

Date

Datum in čas začetka beleženja v log datoteko.

```
#Version: 1.0
#Date: 12-Jan-1996 00:00:00
#Fields: time cs-method cs-uri
00:34:23 GET /foo/bar.html
12:21:16 GET /foo/bar.html
12:45:52 GET /foo/bar.html
12:57:34 GET /foo/bar.html
```

Slika 2.3: Primer W3C formata.

Slika 2.3 prikazuje preprost primer formata W3C, ki beleži podatke o času, metodi HTTP in resursu, ki ga je zahteval odjemalec. Prve tri vrstice na sliki 2.3 predstavljajo direktive formata W3C, med katerimi je najpomembnejša direktiva `Fields`, ki jo izdelana aplikacija uporablja za definicijo formata log datoteke. Preostale vrstice v sliki 2.3 pa prikazujejo zahteve, ki jih je odjemalec posredoval spletnemu strežniku.

Poglavje 3

Uporabljena programska orodja in tehnologije

Pri razvoju aplikacije smo uporabil več različnih orodji ter tehnologij, ki so nam olajšale razvoj aplikacije in poenostavile programsko kodo. Aplikacija je napisana v programskem jeziku Java. Uporabili smo tehniko ORM za pretvarjanje objektov v obliko, ki se zapiše v podatkovno bazo. Orodje Hibernate je implementacija tehnologije ORM, ki je uporabljena v aplikaciji. Shema podatkovne baze se izdela na podlagi standarda JPA 2.1. Določeni deli aplikacije niso v celoti implementirani, ker so odvisni od log formata datotek, ki jih podamo kot vhod v aplikacijo. Ker se Java aplikacija izvaja v navideznem stroju, lahko navodila za izvajanje podamo med samim izvajanjem naše aplikacije. JDK vsebuje orodje `javac` za prevod programske kode Java v vmesno kodo, ki predstavlja navodila za izvajanje aplikacije na navideznem stroju. Vendar orodja `javac` ne more uporabljati med izvajanjem aplikacije, zato upravljamo orodje Javassist, ki skrbi za izdelavo potrebne vmesne kode. Ker aplikacija vsebuje veliko datotek, ki jih je potrebno prevesti, ter veliko dodatnih knjižnic, smo se pri izdelavi projekta odločili za dodatno orodje, ki služi prevajanju projekta. Gradle je orodje, ki opravlja to nalogo in nam poenostavi delo prevajanja projekta. Za obdelavo vhodnih argumentov ukazne vrstice smo v izdelku diplomskega dela uporabili programski vmesnik Args4j.

3.1 Java 8

Java je objektno usmerjen prenosljiv programski jezik, ki ima podprte generične tipe in sočasno izvajanje. Ima podobno sintakso kot C in C++, vendar ima programski jezik Java določene funkcionalnosti poenostavljene. Java je razdeljena na razvojno okolje oziroma JDK in navidezni stroj oziroma okolje za poganjanje, ki mu pravimo JRE. Razvijalec z razvojnim okoljem prevede razred v vmesno kodo, ki je razumljiva navideznemu stroju. Programski jezik Java omogoča prenosljivost aplikacije na različne operacijske sisteme, ki imajo navidezni stroj za izvajanje vmesne kode. Ker programski jezik Java zagotavlja prenosljivost preko vmesne kode in navideznega stroja, nam kot programerju ni potrebno spreminjati programske kode ali ponovno prevajati programa za delovanje na drugih operacijskih sistemih.

Java ima prevzeto vključeno podporo za odsevni (ang. *Reflection*) API. Odsevni API je prevzeto vključen v programski jezik. Odsevni API nam omogoča pregledovanje in spreminjanje strukture ali obnašanja razreda, ko je program v izvajanju [13]. Odsevni API bazira na razredu, ki ga dobimo kot rezultat prevajanja programske kode, zato nam omogoča preverjanje podatkovnih tipov, poizvedovanje po lastnostih in metodah razreda ter spreminjanje lastnosti primerka razreda.

Programski jezik Java ima tudi pripise (ang. *Annotations*), ki predstavljajo dodatne meta podatke, ki jih lahko programer vključi v programsko kodo. Pripise lahko dodamo razredom, poljem razreda, metodam razreda, parametrom metod in paketom. Do pripisov lahko dostopoma s pomočjo odsevnega API. Programski jezik Java pozna tri tipe pripisov, ki definirajo njihovo dostopnost oziroma vidnost:

Class

Pripis je dodan v class datoteko, vendar tak pripis ni mišljen za uporabo med izvajanjem aplikacije.

Runtime

Pripis je dodan v class datoteko ter je dostopen med izvajanjem programa.

Source

Takšne vrste pripise lahko uporablja prevajalnik za dodatne informacije med prevajanjem.

Java 8 [33] je trenutno najnovejša verzija programskega jezika Java. Izdana je bila 18. marca 2014. Uvedla je nekaj novih metod, ki jih lahko razvijalec uporabi za lažji razvoj aplikacije in lepšo preglednost same programske kode. V diplomskem delu smo uporabili naslednje novosti:

Izboljšani vmesniki

Vmesnikom je bila dodana nova ključna beseda `default`, ki pove prevajalniku, da ima funkcija neko prevzeto delovanje. Prevzeto delovanje metode moramo implementirati že v izbranem vmesniku.

Lambda izrazi

Izrazi so predstavljeni z vmesniki, ki vsebujejo pripis `FunctionalInterface`. Vmesnik, ki predstavlja lambda izraz lahko vsebuje več metoda, vendar samo ena metoda ne sme vsebovati implementacije. Primer vmesnika za lambda izraze je vmesnik `Comparator`.

Izboljšan API za datum in čas

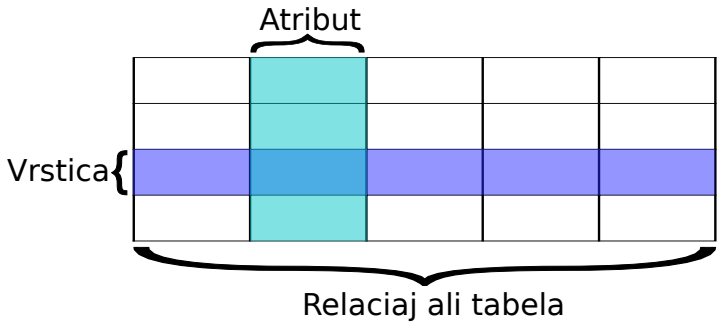
Dodane so bile novi razredi za obdelavo časa in datuma. Razredi se nahaja v paketu `java.util`. API nam omogoča lažje parsanje podatkov o čas, lažjo izdelavo razredov časa, dodane so bile tudi funkcije za obdelavo časa ...

3.2 Relacijske podatkovne baze

Relacijska podatkovna baza je digitalna baza podatkov, ki bazira na relacijskem modelu podatkov. Programskemu orodju za vzdrževanje podatkovne baze pravimo RDBMS. Skoraj vse relacijske podatkovne baze uporabljajo jezik SQL za poizvedovanje in vzdrževanje podatkov [14].

Relacijski podatkovni model organizira podatke v tabele oziroma relacije, ki so sestavljene iz vrstic. Vsaka vrstica vsebuje edinstven ključ, ki opredeljuje vrstico v tabeli. Tabela predstavlja podatkovni tip in vrstice v tabeli predstavljajo primerke tega podatkovnega tipa. Tabele lahko med seboj povezujemo preko primarnih ključev, ki se v povezavi preslikajo v sekundarne ključe.

Slika 3.1 prikazuje primer tabele v relacijski podatkovni bazi. Kot lahko vidimo na sliki, so tudi predstavljeni osnovni termini, ki se uporabljajo pri relacijskih podatkovnih bazah.



Slika 3.1: Terminologija relacijske podatkovne baze.

V nadaljevanju sta opisa dveh podatkovnih baz, ki smo jih uporabili pri razvoju aplikacije. Podatkovna baza SQLite je prevzeta relacijska podatkovna baza, katere gonilnik JDBC je vključen v aplikacijo. Aplikacijo smo tudi testirali na relacijski podatkovni bazi H2, ki deluje hitreje kot podatkovna baze SQLite.

3.2.1 SQLite

SQLite je odprto kodna relacijska podatkovna baza, napisana v programskem jeziku C, ki ne potrebuje strežnika za delovanje in deluje samo v vgrajenem načinu. Podpira tudi večino znanih operacijskih sistemov, med katerimi so Windows, Linux, Android, Mac OS X, iOS, Solaris in drugi. Paket vsebuje tudi klienta CLI, ki se lahko uporablja za administracijo podatkovne baze.

Ima preprost API, ki je dobro dokumentiran, zato je predelava programske kode preprosta. Podatkovna baze SQLite ne potrebuje dodatnih knjižnic za delovanje. Transakcije podpirajo lastnosti ACID. Podatkovna baza se na disku shrani kot ena datoteka. Za delovanje ne potrebuje dodatne konfiguracije [15].

Relacijska podatkovna baza SQLite ima podprtih večino funkcionalnosti jezika SQL, a pri spreminjanju tabel ne moremo dodajati novih odvisnosti, ki jih imajo povezave med tabelami. Prav tako ne pozna stavka SQL za izdelavo zaporedji. V sintakso jezika SQL ima vgrajene dodatne funkcije, med katerimi je tudi funkcija `last_insert_rowid()`, ki vrne vrednost zadnjega vnesenega primarnega ključa.

3.2.2 H2

H2 je odprta kodna relacijska podatkovna baza, napisana v Java programskem jeziku, ki lahko deluje v strežniškem ali vgrajenem načinu. Ker je podatkovna baza napisana v Java programskem jeziku, podpira večino znanih operacijskih sistemov. Paket vsebuje tudi vmesni GUI in CLI, ki se lahko uporabljata za administracijo podatkovne baze [6]. Za razliko od podatkovne baze SQLite ima H2 podprte funkcionalnosti jezika SQL za spreminjanje odvisnosti med tabelami, kot tudi stavek za izdelavo zaporedji.

3.3 ORM

ORM je tehnika za pretvarjanje podatkov instance razreda v vrstici relacijske podatkovne baze. V programskem jeziku Java so orodja, ki implementirajo tehniko ORM: JPA, JDO, ARP, Carbonado, jOOQ, Hibernate ... [10]. V diplomskem delu smo uporabili orodje Hibernate. Metapodatke za preslikavo razredov smo orodju Hibernate podali preko pripisov standard JPA 2.1. Tako smo zagotovili prenosljivost našega podatkovnega modela tudi na orodja, ki implementirajo tehniko ORM, ter uporabljaja pripise standarda JPA 2.1 za izdelavo podatkovnega modela podatkovne baze.

3.3.1 JPA

JPA je programski vmesnik in standard definiran v programskem jeziku Java. Uporablja se za definicijo podatkovnega modela, ki definira preslikave iz objektnega zapisa v relacijski zapis. Programski vmesnik JPA je v programskem jeziku Java definiran v paketu `javax.presistence`. V tem paketu so definirani pripisi, ki se uporabljajo za definiranje podatkovnega modela. Programski vmesnik JPA definira tudi poizvedovalni jezik JPQL, ki se uporablja za poizvedovanje instanc razredov, ki so shranjeni v podatkovni bazi.

Vsak razred, ki ga želimo preslikati v podatkovno bazo, mora vsebovati polje, ki identificira instanco razreda v podatkovni bazi. Ko izvajajoč program oziroma aplikacija izdela nov primerek razreda, ta primerek oziroma objekt dobi referenco, ki služi identifikaciji tega primerka. Podatkovne baze ne poznajo referenc, poznajo

pa primarne ključe, ki služijo istem namenu kot reference primerkov. Primarni ključ primerka uporabljamo za poizvedovanje, definiranje relacij, brisanje in spreminjanje primerka in s tem podatkov v podatkovni bazi. Standard JPA podpira tudi sestavljene primarne ključe (ang. *Composite Primary Keys*). JPA podpira samodejno nastavljanje primarnih ključev preko pripisa `GeneratedValue`, ki mu lahko podamo strategijo za izdelavo le-tega. Strategije, ki jih ponuja JPA, so:

Auto
: Strategija narekuje orodju ORM, naj samodejno izbere eno izmed spodaj navedenih strategij, ki je primerna za izbrano podatkovno bazo.

Identity
: Strategija narekuje orodju ORM, naj za izbiro primernega ključa uporabi identiteto tabele podatkovne baze, kateri se shranjuje razred oziroma podatkovni tip.

Sequence
: Strategija narekuje orodju ORM, naj za izbiro primarnega ključa uporabi zaporedje, ki je konstrukt podatkovne baze.

Table
: Strategija narekuje orodju ORM, naj za izbiro primarnega ključa uporabi posebno tabelo, ki je namenjena samo hranjenju zadnjega uporabljenega primarnega ključa.

Predhodnik standarda JPA je Hibernate, katerega razvoj se je začel že leta 2001. Standard JPA 1.0 je bil definiran 11. maja 2006 s strani JCP v JSR 220 [28]. Eden pomembnejših mejnikov standarda JPA je bila definicija standarda JPA 2.0, ki ima naslednje nove pomembne lastnosti:

- razširitev funkcij za pretvarjanje objektov v vrstice relacijske podatkovne baze, kjer je bila dodana podpora za sezname,
- dodan je bil kriterij (ang. *Criteria*) API za potrebe poizvedovanja. Pri tem programskem vmesniku programer uporabi kombinacije razredov paketa `javax.persistence.criteria` in s tem izdela poizvedbo v jeziku JPQL.

Standard JPA 2.0 se trenutno uporablja v Batoo JPA, ObjectDB in OpenJPA orodjih. Trenutno najnovejša verzija programskega vmesnika JPA je 2.1, definirana 22. aprila 2013 s strani JCP v JSR 338 [29]. Standard JPA 2.1 ima naslednje pomembne lastnosti, ki so uporabljene v aplikaciji:

- pretvorniki, ki nam olajšajo delo z enum razredi,
- izdelava podatkovne sheme,
- izboljšave jezika JPQL in kriterij API.

Tabela 3.2 prikazuje podatkovne tipe programskega jezika Java, ki jih standard JPA 2.1 zna preslikati v podatkovno bazo. V drugem stolpcu tabele 3.2 so prikazani podatkovni tipi, ki se uporabijo v podatkovni bazi za preslikavo Java tipa. Orodje ORM uporabi najprimernejšega od navedenih za izbrano podatkovno bazo. Zadnji dve vrstici v tabeli 3.2 predstavljata sezname in slovarje, kjer pa ni navedenega tipa podatkovne baze. Tip podatkovne baze za sezname in slovarje je vezan na podatkovni tip, ki ga hranimo v seznamu oziroma slovarju. Zadnji stolpec v tabeli 3.2 nam pove katere, podatkovne tipe Jave lahko uporabimo kot primarne ključe. Sledijo pomembnejši konstrukti standarda JPA, ki so uporabljeni v aplikaciji.

Osnovni podatkovni tipi

Osnovni podatkovni tip je predstavljen z `Basic` predpisom, ki se uporablja pri poljih ali metodah razreda, katere želimo shraniti v podatkovno bazo. Standard JPA avtomatično predvideva, da so polja, ki ne vsebujejo dodatnega pripisa iz paketa `javax.persistence`, osnovni podatkovni tip, zato nam ni potrebno pisati tega pripisa.

Začasen podatkovni tipi

Začasen (ang. *Transient*) podatkovni tip je predpis, ki se uporablja, kadar imamo v razredu polje, ki ga ne želimo zapisati v podatkovno bazo. Predpis je viden med izvajanjem programa ter ga lahko pripišemo poljem in metodam razreda. Začasen pripis moramo obvezno dopisati, saj v nasprotnem primeru shranimo podatek, ki ga nismo želeli shrani.

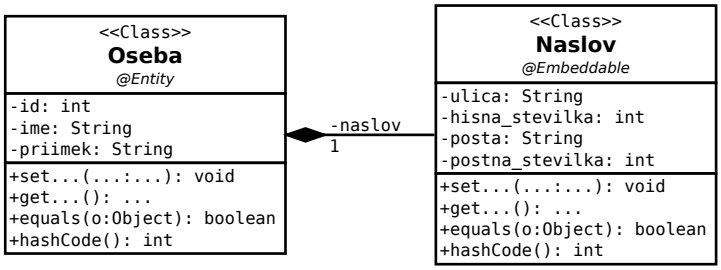
Java tip	tip podatkovne baze	pk
java.lang.Boolean in boolean	BOOLEAN, BIT, SMALLINT, INT, NUMBER	✓
java.lang.Byte in byte	VARBINARY, BINARY, BLOB	✓
java.lang.Character in char	VARCHAR, CHAR, VARCHAR2	✓
java.lang.Double in double	NUMERIC, FLOAT, DOUBLE	
java.lang.Float in float	NUMERIC, FLOAT, DOUBLE	
java.lang.Integer in int	NUMERIC, NUMBER, INT, LONG	✓
java.lang.Long in long	NUMERIC, NUMBER, LONG	✓
java.lang.Short in short	SMALINT, SHORT	✓
java.lang.String	VARCHAR, CHAR, VARCHAR2, CLOB, TEXT	✓
java.lang.Enum in java.lang.Enum[]	NUMERIC, VARCHAR, CHAR	✓
java.time.LocalDate	DATE, TIMESTAMP, DATETIME	
java.time.LocalTime	TIME, TIMESTAMP, DATETIME	
java.util.Serializable	VARBINARY, BINARY, BLOB	
java.util.Collection		
java.util.Map		

Tabela 3.2: Nekaj podatkovnih tipov programskega jezika Java, ki jih standard JPA zna pretvoriti.

Vgrajen podatkovni tip

Vgrajen podatkovni tip je predstavljen z `Embeddable` predpisom, ki se uporablja v razredih, katere želimo združiti z drugim razredom in ne želimo posebne tabele za hranjenje podatkov le tega v podatkovni bazi. Predpis je viden med izvajanjem programa in ga lahko pripišemo samo razredom. Razred, ki ima ta predpis ne potrebuje identifikacijskega polja.

`Embedded` je predpis, ki se pripiše poljem ali metodam razreda, ki vsebujejo podatkovni tip s predpisom `Embeddable`. Tudi ta predpis je viden med izvajanjem programa.



Slika 3.2: Razredni diagram z vgrajenega tip.

Oseba	
id	INTEGER
ime	VARCHAR
priimek	VARCHAR
ulica	VARCHAR
hisna_stevilka	INTEGER
posta	VARCHAR
postna_stevilka	INTEGER

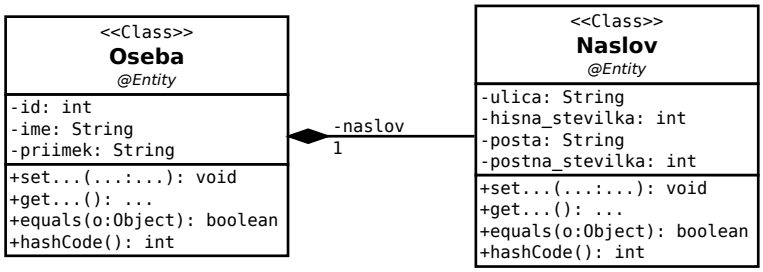
Slika 3.3: Tabela vgrajenega tipa.

Slika 3.2 prikazuje razredni diagram, ki vsebuje vgrajen podatkovni tip. Slika 3.3 prikazuje, kako se razredi na sliki preslikajo v relacijsko podatkovno bazo.

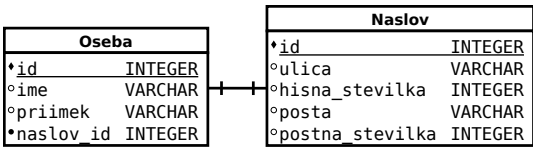
Povezava med dvema objektoma

Povezava med dvema objektoma je predstavljena s predpisom `OneToOne`. Pripis lahko pripišemo polju ali metodi razreda. Pripis je viden med izvajanjem. Razredi ki jih povezujemo, morajo vsebovati identifikacijsko polje oziroma primarni ključ,

saj se primarni ključ preslika v tuj ključ, ki ga uporabljamo za združevanje tabel pri poizvedbah z jezikom SQL in JPQL.



Slika 3.4: Razredni diagram s povezavo ena na ena.



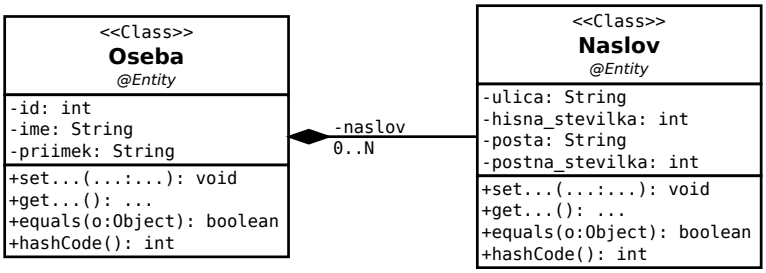
Slika 3.5: Primer preslikave povezave ena na ena.

Slika 3.4 prikazuje razredni diagram, kjer nimamo vgrajenega objekta. Slika 3.5 prikazuje, kako se razredni diagrami na sliki 3.4 preslikajo v relacijsko podatkovno bazo. Kot je vidno na sliki 3.5, tabela `Oseba` pridobi tuj ključ, ki ima odvisnost vezano na primarni ključ tabele `Naslov`.

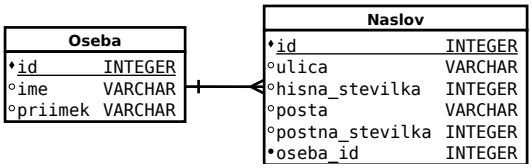
Povezava enega objekta z množico objektov

Za povezovanje enega objekta z množico drugih objektov uporabljamo `OneToMany` pripis, ki se uporablja pri poljih tipa tabela, seznam ali slovar. Pripis lahko pripišemo poljem in metodam razreda. Pripis je viden med izvajanjem. S tem konstruktom standarda JPA rešuje problem zapisa tabel, seznamov in slovarjev v podatkovno bazo.

Slika 3.6 prikazuje razredni diagram, kjer je polje naslov tipa seznam ali slovar in ima pripis `OneToMany`. Slika 3.7 prikazuje preslikavo razrednega diagrama na sliki 3.6 v relacijsko podatkovno bazo. Pri tej preslikavi dobi tabela Naslov dodaten atribut, ki predstavlja tuj ključ, povezan s tabelo `Oseba`. Pri takih povezavah ima lahko ena oseba več naslovov, en naslov lahko pripada samo eni osebi.



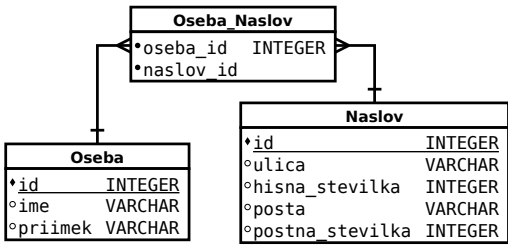
Slika 3.6: Razredni diagram s povezavo ena na mnogo.



Slika 3.7: Primer preslikave povezave ena na mnogo.

Povezava množice objektov z drugo množico objektov

Za povezovanje množice objektov z drugo množico objektov uporabljamo `Many ToMany` pripis, ki se uporablja pri poljih tipa tabela, seznam ali slovar. Pripis pripišemo poljem in metodam razreda. Pripis je viden med izvajanjem.



Slika 3.8: Primer preslikave povezave mnogo na mnogo.

Slika 3.6 prikazuje razredni diagram, kjer je polje naslov tipa tabela, seznam ali slovar in ima pripis `ManyToMany`. Slika 3.8 prikazuje preslikavo razrednega diagrama na sliki 3.6 v relacijsko podatkovno bazo. Pri tej preslikavi imamo dodatno tabelo, ki nam omogoča povezavo različnih oseb z različnimi naslovi.

Preslikava dedovanja med razredi

Pri objektnem programiranju poznamo tehniko dedovanja med razredi, kjer podrazred `B` razširi ali implementira delovanje razreda `A`. V programskem jeziku Java poznamo dve vrsti dedovanja. Prvi tip dedovanja je preko vmesnikov (ang. *interface*), kjer implementiramo delovanje razreda `A` v podrazredu `B`. Drugi tip dedovanja je preko razširjanja (ang. *extends*), kjer razred `B` razširi razred `A` ter mu tako doda nove funkcionalnosti. Vendar pa programski vmesnik JPA pozna le dedovanje preko razširjanja razredov. Pripis, ki se uporablja za identifikacijo dedovanja med razredi je `Inheritance`, ki vsebuje atribut za določanje strategije preslikave dedovanja. Programski vmesnik JPA pozna tri strategije za preslikavo dedovanja, ki so:

Vse v eno tabelo (ang. *single table inheritance*)

Pri tej strategiji se vsi podrazredi razreda `A` preslikajo v isto tabelo, kjer so atributi tabele vsa polja podrazredov. Strategija doda tabeli atribut diskriminator, ki se uporablja za identifikacijo podrazreda. Podatkovni tip diskriminatorja je v programskem vmesniku JPA prevzeti niz, ime atributa v tabeli je prevzeto `DTYPE`. Prevzeta vrednost diskriminatorja za posamezen podrazred je kar ime razreda. Podatkovni tip diskriminatorja in ime atributa lahko spreminjamo preko pripisa `DiscriminatorColumn`, kjer atribut `discriminatorType` predstavlja podatkovni tip in atribut `name` predstavlja ime atributa v podatkovni bazi. Vrednosti diskriminatorjev za posamezen podrazred v hierarhiji dedovanja spreminjamo preko pripisa `Discriminator Value`, kjer atribut `value` predstavlja vrednost, ki se zapiše v atribut diskriminatorja pri zapisu izbranega razreda.

Več tabel z uporabo združevanja (ang. *joined, multiple table inheritance*)

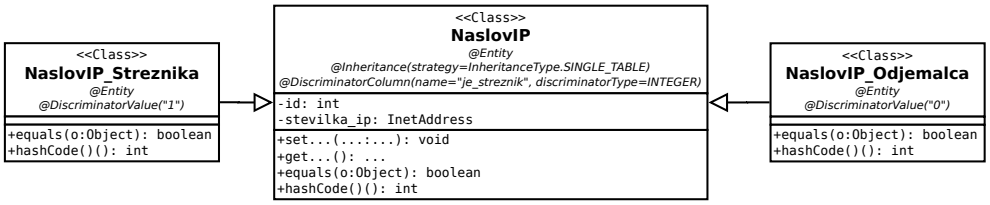
Pri tej strategiji je rezultat preslikave dedovanja med razredi več tabel. Vsak razred v hierarhiji dedovanja se preslika v svojo tabelo, kjer so atributi te tabele samo polja izbranega razreda v hierarhiji. Tabele se med seboj povezujejo preko identifikacijskega polja, ki se nahaja v razredu iz katerega so vsi podrazredi razširjeni. Strategija ima ime `join`, saj moramo vse tabele do izbranega podrazreda med seboj združiti, če želimo iz podatkov v

podatkovni bazi izdelati primerek izbranega podrazreda. Strategija podpira atribut diskriminator, ki je pri določenih implementacijah vmesnika JPA obvezen.

Tabela za vsak podrazred (ang. *table per class inheritance*)

Pri tej strategiji se za vsak razred hierarhije dedovanja, ki ni abstrakten razred, izdela nova tabela. Tabela vsebuje atribute, ki so del izbranega podrazreda, in vse atribute nadrazredov.

V izdelani aplikaciji smo uporabili le strategijo vse v eno tabelo.



Slika 3.9: Razredni diagram za naslove IP.

NaslovIP	
♦id	INTEGER
◦stevilka_ip	VARCHAR
◦je_streznik	INTEGER

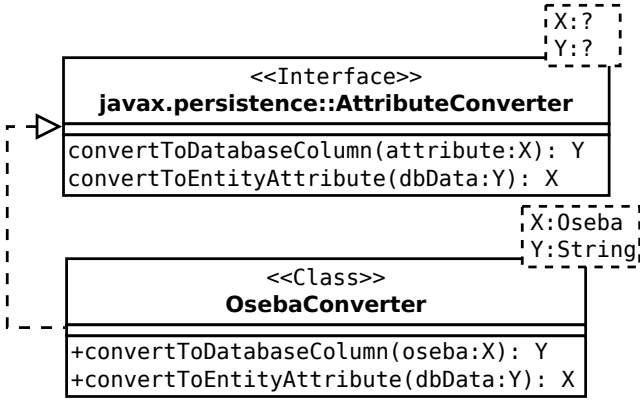
Slika 3.10: Primer preslikave dedovanja z uporavo strategije vse v eno tabelo.

Slika 3.9 prikazuje generalizacijo naslovov IP na naslove, ki predstavljajo strežnike in na naslove, ki predstavljajo odjemalce. Razredni diagram vsebuje tudi podatkovni tip `InetAddress`, ki je del programskega jezika Java, vendar ga standard JPA ne zna avtomatično preslikati v podatkovni tip relacijske podatkovne baze, zato uporabljamo pretvornike atributov, ki so opisani v nadaljevanju. Slika 3.10 prikazuje preslikavo razrednega diagrama na sliki 3.9 v tabelo relacijske podatkovne baze.

Pretvornik atributov

Pretvornik atributov oziroma `AttributeConverter` je programski vmesnik, ki omogoča pretvarjanje vhodnega podatkovnega tipa v izhodni podatkovni tip. Vho-

dni podatkovni tip uporablja programer v programu, izhodni podatkovni tip se pa zapiše v podatkovno bazo. Pretvorniki atributov so uporabni pri razredih, ki niso podprti v standardu JPA. V diplomskem delu so pretvorniki atributov uporabljeni za pretvarjanje razredov `enum` in nekaterih razredov, ki jih standard JPA ne podpira.



Slika 3.11: Razredni diagram implementacije pretvornika atributov.

Slika 3.11 prikazuje metode vmesnika `AttributeConverter`, ki jih mora programer implementirati za pravilno delovanje pretvarjanja podatkovnih tipov. Slika 3.11 prikazuje tudi primer implementacije pretvornika atributov na primeru razreda `Oseba`. Vhodni tip primera je `Oseba`, izhodi tip pa `String`.

3.3.2 Hibernate

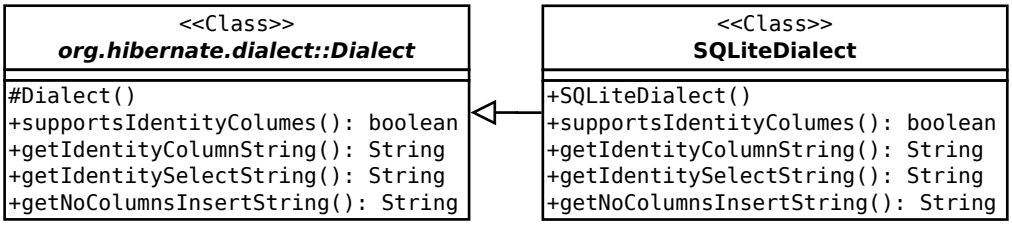
Hibernate je orodje, ki implementira tehniko ORM. S pomočjo datotek, zapisanih v standardu XML, ali pripisov, definiranih v standardu JPA, izdela logiko za pretvarjanje podatkov iz razrednega zapisa v zapis, razumljiv relacijskim podatkovnim bazam. Zapis, razumljiv relacijskim podatkovnim bazam ,je zapisan v jeziku SQL. Hibernate je brezplačno orodje, izdano pod licenco GNU LGPL 2.1. Orodje ima svoj poizvedovalni jezik HQL, iz katerega izhaja jezik JPQL [7].

Orodje ima poseben razred, ki definira sintakso jezika SQL za izbrano relacijsko podatkovno bazo. Temu razredu pravimo `Dialect` in ga najdemo v paketu `org.hibernate.dialect`, kjer se nahajajo dialekti za relacijske podatkovne baze, ki jih orodje prevzeto podpira. V primeru, ko za izbrano relacijsko podatkovno bazo

dialekt ne obstaja, ga lahko programer izdela, tako da razširi razred `Dialect`, ter implementira potrebne metode.

V nadaljevanju je kratka predstavitev dialekta v orodju, ki prikazuje katere metode je potrebno implementirati za pravilno delovanje aplikacije. Opis bazira na dialektu za relacijsko podatkovno bazo SQLite, za katero orodje nima prevzetega dialekta.

Hibernate dialekt



Slika 3.12: Razredni diagram za dialekt relacijske podatkovne baze SQLite.

Slika 3.12 prikazuje poenostavljen razredni diagram z abstraktnim razredom `Dialect` ter implementacijo abstraktnega razreda `SQLiteDialect`. V abstraktnem razredu `Dialect` so zaradi boljše preglednosti izpuščena polja in metode. Izpuščena polja in metode si lahko ogledate na [2]. V razredu `SQLiteDialect` so tudi izpuščena polja in metode. Razred `SQLiteDialect` vsebuje le metode, ki jih mora programer obvezno implementirati v dialektu za izbrano relacijsko podatkovno bazo, če ta podatkovna baza nima prevzete podpore v orodju Hibernate.

Ena od obveznih metod je tudi konstruktor brez argumentov, ki ga orodje Hibernate uporablja preko odsevnega programskega vmesnika za izdelavo objekta. V tem konstruktorju mora programer povedati, kako se podatkovni tipi programskega jezika Java preslika v podatkovni tip izbrane relacijske podatkovne baze. Tukaj si lahko programer pomaga s tablo 3.2. Podatkovni model aplikacije uporablja za izdelavo primarnega ključa strategijo `Identity`, zato mora metoda `supports IdentityColumes` vrniti vrednost `true`. Metoda `getIdentityColumnString` mora vrniti podatkovni tip relacijske podatkovne baze, ki se uporablja za hranjenje primarnega ključa. Metoda `getIdentitySelectString` mora vrniti poizvedbo SQL, ki vrne naslednjo vrednost primarnega ključa. Metoda `getNoColumnsInsert`

`String` se uporabi takrat, ko v relacijsko podatkovno bazo vnašamo vrstico brez vrednosti.

3.4 Javassist

Javassist je orodje za manipulacijo vmesne kode programskega jezika Java. Omogoča izdelavo novih in spreminjanje že obstoječih razredov med izvajanjem programa. Orodje ima v prevzetem paketu vključena dva programska vmesnika. Eden od vmesnikov je namenjen programski kodi, napisani v programskem jeziku Java, drugi pa je namenjen vmesni kodi [9]. Programski vmesnik, namenjen programski kodi, je preprost za uporabo, saj nove in obstoječe razrede izdelujemo in spreminjano na podlagi programske kode. Programski vmesnik, namenjen programski kodi, nima vseh funkcionalnosti, ki jih podpira vmesna koda in programski jezi Java. Pri programskem vmesniku, namenjenem vmesni kodi, imamo na voljo vse funkcionalnosti vmesne kode in programskega jezika Java, vendar je vmesnik težji za uporabo in zahteva poznavanje vmesne kode. Za pomoč pri uporabi vmesnika, namenjenega vmesni kodi, lahko uporabimo orodje `javap`, ki se nahaja v JDK paketu za programski jezik Java.

```
1  ClassPool pool = ClassPool.getDefault();
2  CtClass aClass = pool.makeClass("NewClass");
3  aClass.setModifiers(Modifier.PUBLIC);
4  aClass.getClassFile().setMajorVersion(ClassFile.JAVA_8);
5  aClass.setSuperclass(pool.get(OldAbsClass.class.getName()));
6  {
7    CtField field = CtField.make("private String name", aClass);
8    aClass.addField(field);
9  }
10 {
11   CtMethod method = CtMethod.make(
12   "public " + String.class.getName() + " getName() {
13     return this.name;
14   }",
15   aClass);
16   aClass.addMethod(method);
17 }
18 {
19   CtMethod method = CtMethod.make(
20   "public void setName(" + String.getClass().getName() + " name) {
21     this.name = name;
22   }",
23   aClass);
24   aClass.addMethod(method);
25 }
```

Koda 3.1: Primer izdelava novega razreda.

Koda 3.1 prikazuje preprost primer uporabe programskega vmesnika orodja Javassist. V primeru, prikazanem v kodi 3.1, izdelamo nov razred z imenom `New`

`Class`, ki razširi razred `OldAbsClass`. Razredu dodamo eno polje z imenom `name` in dve metodi, ki služita pridobivanju in nastavljanju tega polja.

```
1 ConstPool constPool = aClass.getClassFile().getConstPool();
2 AnnotationsAttribute attr = new AnnotationsAttribute(constPool,
      AnnotationsAttribute.visibleTag);
3 {
4   Annotation anno = new Annotation(Entity.class.getName(), constPool);
5   attr.addAnnotation(anno);
6 }
```

Koda 3.2: Dodajanje pripisov.

Koda 3.2 prikazuje dodajanje pripisa `Entity` razredu, ki smo ga izdelali s kodo 3.1. Primer prikazuje uporabo programskega vmesnika, namenjenega vmesni kodi.

3.5 Args4j

Args4j je majhen programski vmesnik za programski jezik Java, ki nam omogoča preprosto obdelavo vhodnih argumentov ukazne vrstica za programe, ki uporabljajo predvsem vmesnik CLI [32]. Za hranjenje obdelanih vhodnih argumentov potrebujemo poseben razred, kateremu dodano pripise programskega vmesnika Args4j ali pa uporabimo datoteko XML, katere struktura je definirana v programskem vmesniku Args4j. Pripisa, ki ju uporablja programski vmesnik Args4j, sta:

Option

Pripis lahko pripišemo metodi ali polju razreda ter je viden med izvajanjem programa. Pripis mora obvezno vsebovati argument `name`, sicer pa ima še dodatne argumente, ki jih uporabljamo za definiranje medsebojne odvisnosti med argumenti, medsebojno izključevanje argumentov, obveznost argumenta, dodatna imena argumenta ... Argument, ki je definiran preko tega pripisa, ne more vsebovati več vrednosti.

Argument

Pripis lahko pripišemo metodi ali polju razreda ter je viden med izvajanjem programa. Pripisu ni potrebno podati nič dodatnih argumentov, vendar ima dodatne argumente, ki so podobni argumentom pripisa `Option`. Argument,

ki je definiran preko tega pripisa lahko vsebuje več vrednosti, vendar moramo pripisu podati argument `multiValued` z vrednostjo `true`.

Datoteka XML uporablja ista imena za značke, ki so uporabljena za pripise. Enako velja tudi za imena argumentov značk. Programski vmesnik omogoča preprosto izdelavo pomoči za uporabniški vmesnik CLI. Omogoča tudi izdelavo HTML ali XML dokumentacije za vse argumente ukazne vrstice.

V diplomskem delu smo predelali razred `CmdLineParser`. Ta predstavlja glavni razred za obdelavo vhodnih argumentov v razred `PropertiesCmdParser`, ki lahko preko argumenta `-props` prejme datoteko s končnico `properties`, ta pa vsebuje argumente ukazne vrstice, shranjene v tekstovni datoteki. Tekstovna datoteka ima vnaprej določen format, ki je določen s strani programskega jezika Java [12]. Razred, ki se v izdelani aplikaciji uporablja za hranjenje obdelanih argumentov ukazne vrstice, je `ArgumentParser`.

3.6 Gradle

Gradle je orodje za avtomatizacijo procesa prevajanja projekta. Orodje je zelo podobno orodjem Apache Ant in Apache Maven, vendar za konfiguracijo ne potrebuje datotek, napisanih v jeziku XML, temveč uporablja jezik, ki bazira na domensko specifičnem jeziku Groovy. Orodje je napisno v programskem jeziku Java in Groovy [5]. Orodje podpira 60 različnih programskih jezikov, med katerimi so programski jezik Java, C, C++, Python, Scala ... Orodje podpira vtičnike, ki dodajo novo funkcionalnost prevajanju projekta ali pa dodajo podporo za različna razvojna orodja. Ena pomembnejših lastnosti Gradleja je vključevanje programskih orodji, ki jih nismo razvili sami, preko različnih skladišč, kot so Maven, Ivy in lokalne datoteke [4].

Večina projektov, napisanih v programskem jeziku Java, ima podobna opravila, ki se morajo izvršiti, da izdelamo aplikacijo. Pri Java projektih moramo najprej prevesti programsko kodo projekta. Sledi testiranje prevedene programske kode ter na koncu izdelava datoteke JAR, ki vsebuje prevedeno programsko kodo. Gradle ima vtičnike, ki definirajo osnovna opravila za izdelavo aplikacije, tako da programerju ni potrebno vedno na novo pisati dodatnih ukazov za opravila. Osnovna opravila za izdelavo Java aplikacije se nahajajo v vtičniku `java`.

Opravila lahko programer implementira z uporabo domensko specifičnega jezika, ki ga Gradle uporablja v skriptah za prevajanje.

V izdelku diplomskega dela smo uporabili `java`, `pmd`, `jacoco` in `shadow` vtičnike. Vtičnika `pmd` in `jacoco` dodata opravili za analizo testov in analizo programske kode. Vtičnik `shadow` pa doda opravilo za izdelavo datoteke JAR z vsemi knjižnicami projekta. Ročno smo dopisali opravilo, ki je podobno opravilom iz vtičnika `shadow`. Opravilo smo poimenovali `fatJar`.

Poglavje 4

Seje

Glavni problem diplomskega del je rekonstrukcija seje, ki jo je imel uporabnik na izbranem spletnem strežniku s pomočjo podatkov v log datoteki. Podatki v log datoteki predstavljajo neurejene klikotoke uporabnikov. Klikotok je skupek podatkov, kjer en podatek predstavlja klik uporabnika na element vmesnika GUI, ki pa je lahko del spletne strani, spletne aplikacije ali pa namizne aplikacije. Podatki klikotoka so uporabi za analiziranje uporabnikov, analiziranje aplikacije, testiranje ... Klikotok uporabnika je v log datoteki urejen po kronološkem zaporedju klikov uporabnika ter je pomešan s klikotoki drugih uporabnikov. Klikotok uporabnika v log datoteki v večini primerov ne vsebuje povratnih povezav oziroma strani, ki jih je uporabnik že obiskal v trenutni seji. Novejši odjemalci uporabljajo predpomnilnik (ang. *cache*) za hranjenje obiskanih spletnih strani in njihovih resursov. Zato ni potrebno odjemalcu ponovno pošiljati zahteve za že obiskano spletno stran ali njihovega resurse. Spletne strani in resursi se v predpomnilniku odjemalca hranijo do zaključka delovanja odjemalca ali pa dokler odjemalec ne ugotovi, da mu zmanjkuje prostora v predpomnilniku.

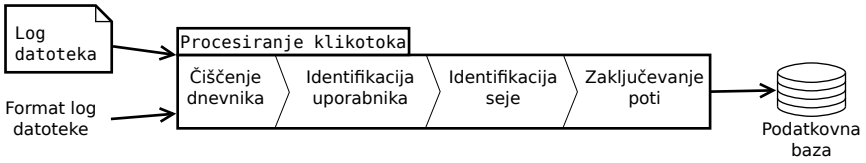
Uporabniki se do izbranega spletnega strežnika povezuje preko protokola HTTP ali HTTPS s pomočjo različnih odjemalcev. Na splošno lahko odjemalce razdelimo na odjemalce, ki jih uporabljajo ljudje, ter robote oziroma pajke (ang. *web crawler*), katerih delovanje je opredeljeno z umetno inteligenco ali programskim jezikom. Odjemalce, ki jih uporabljajo ljudje, lahko delimo še na spletne brskalnike z vmesnikom GUI, spletne brskalnike z vmesnikom CLI, mobilne brskalnike in druge odjemalce, kot so odjemalci RSS (ang. *feed reades*) ali pa odjemalci

elektronske pošte. Robote pa lahko delimo po vlogah, ki jih opravljajo. Vloge, ki jih lahko robot ima, so: zbiranje podatkov o posodobitvah na izbrani spletni strani, testerji povezav (ang. *link checkers*), validatorji (ang. *validators*), platforme v oblaku (ang. *cloud platforms*) in ostale, ki imajo specifične funkcije za analizo uporabnikovega vedenja, kot sta npr. robota !Susie in MetaURI ali pa so del kakšnega orodja za zajem podatkov na spletni strani, recimo robot WebCapture. Vrsto uporabljenega odjemalca v log datoteki identificiramo preko polja uporabnikov posrednik (ang. *user agent*). V diplomskem delu nas zanimajo le zahteve, podane z odjemalci, ki so jih uporabljali ljudje, saj nam zahteve, podane s strani robotov, lahko otežujejo postopek rekonstrukcije sej. Pri večini formatov

```
User-agent: *
Disallow: *.gif
Disallow: /bar.html
Disallow: /foo.php
```

Slika 4.1: Primer datoteke `robots.txt`.

log datotek nimamo podatka o uporabnikovem posredniku, kar nam otežuje odstranjevanje zahtev, podanih s strani robotov, vendar pa večina robotov upošteva protokol za izključevanje robotov (ang. *robots exclusion protocol*) [17], ki narekuje robotom naj z izbranega strežnika najprej pridobijo datoteko `robots.txt`, v kateri so zapisana pravila do katerih spletnih strani ali resursov ne sme robot dostopati. Slika 4.1 prikazuje primer datoteke `robots.txt`, kjer je v prvi vrstici definirano pravilo, ki pove, kdo naj bi spoštoval pravila, navedena v datoteki. Druga vrstica datoteke definira pravilo za izključevanje resursov, ki imajo končnico `gif`. Tretja vrstica datoteke definira pravilo za izključevanje spletne strani `bar.html`, zadnja vrstica datoteke pa definira pravilo za izključevanje spletne strani `foo.php`.



Slika 4.2: Proces rekonstrukcije seje.

Slika 4.2 prikazuje abstrakcijo procesa rekonstrukcije spletnih sej, ki ima štiri glavne procese pri procesiranju podatkov klikotoka. Slika 4.2 prikazuje tudi vhode in izhode procesa rekonstrukcije sej za izdelano aplikacijo.

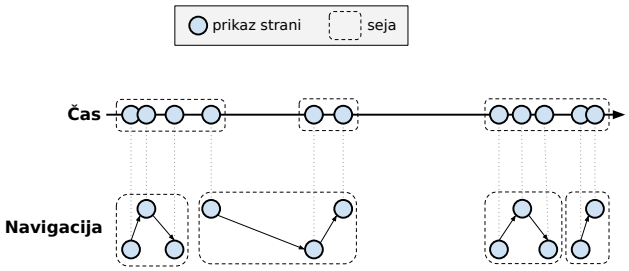
Pri procesu čiščenja dnevnika odstranjujemo zahteve, ki nam otežujejo proces osejevenja. V to skupino spadajo predvsem zahteve robotov, vendar ima izdelana aplikacija možnost, da takih zahtev ne ignorira. V to skupino spadajo tudi zahteve za resurse spletne strani, ki v log datoteki nimajo zahteve za spletno stran. Take zahteve se pojavijo predvsem na začetku log datoteke, vendar pa nekateri roboti proizvedejo takšne zahteve.

Proces identifikacije uporabnikov nam otežuje protokol HTTP, kajti nima stanj. Ker protokol HTTP nima stanj, so v protokol dodali mehanizem za upravljanje stanj, ki mu pravimo piškotek (ang. *Cookie*) [20]. Mehanizem doda dve novi polji v glavo protokola HTTP, ki se uporabljata za upravljanje podatka o piškotku. Piškotek kot tak se v protokolu HTTP direktno ne uporablja, saj naj bi protokol zagotavljal anonimnost. Protokol HTTP skrbi le za prenos in upravljanje podatkov piškotka. Podatki piškotka so namenjeni aplikacijam, ki uporabljajo protokol HTTP za identifikacijo uporabnika in sledenje uporabniškim aktivnostim. Piškotek je eden glavnih podatkov v formatu log datoteke, ki ga lahko uporabljamo za identifikacijo uporabnikov. Vendar tega podatka nimamo vedno na razpolago v formatu log datoteke. Takrat si pomagamo s kombinacijo različnih tipov polj za identifikacijo uporabnika. Ta polja so:

- klientova številka IP,
- piškotek,
- ime oddaljenega gostitelja,
- oddaljen logname,
- oddaljen uporabnik,
- uporabniški posrednik.

V seznam bi lahko vključili tudi vrata, ki so bila uporabljena za pošiljanje odgovora, a se bi morali omejiti samo na protokol HTTP/2, ki je uvedel to novost, da se morajo vsi resursi poslati preko istih vrat, ki so bila uporabljena za prenos spletne strani.

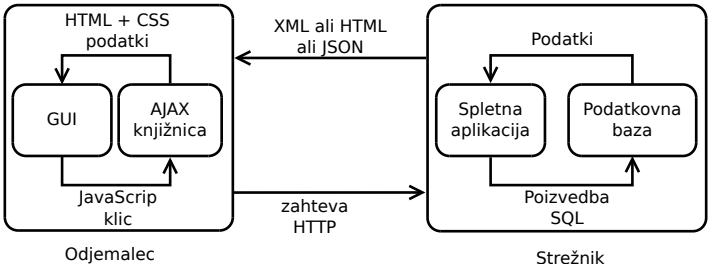
Seja je zaporedje zahtev, kjer si zahteve sledijo v nekem časovnem intervalu ali pa si sledijo glede na namen, ki ga je imel uporabnik. Nameni uporabnika so opredeljeni glede na vrsto spletnih strani, ki jih ima spletni strežnik, ki ga je uporabnik uporabljal. V primeru uporabe spletne knjižnice je lahko uporabnikov namen iskanje knjige. V primeru uporabe spletne trgovine pa je lahko uporabnikov namen iskanje izdelka. Zaporedje zahtev ustvari en uporabnik s sprehodom po spletnem strežniku. Metodi, ki uporablja časovne intervale za identifikacijo sej, pravimo časovna hevristika. Metodi, ki uporablja namen uporabnika, pravimo navigacijska hevristika. Slika 4.3 prikazuje primer razpletanja sej s počjo časovne in navigacijske hevristike.



Slika 4.3: Primerjava procesa rekonstrukcije seje za časovno in navigacijsko hevristiko [16].

V procesu zaključevanja poti je potrebno dodati manjkajoče zahteve za spletne strani, ki jih zaradi predpomnilnika odjemalca odjemalec ni ponovno zahteval. Vhod tega proces je seja, ki ne upošteva sprehoda po strani z več okni ali zavihki odjemalca, zato moramo v tem procesu upoštevati tudi to slabost procesa identifikacije sej. Izhod tega procesa je več sej, kjer upoštevamo dodatno prepletenost vhodne seje, ter razdelimo vhodno sejo na več podsej. Če je dolžina vhodne seje enaka n, potem je možno število podsej enako B_n. Vrednost B_n predstavlja Bellovo število [37], ki narekuje, na koliko različnih načinov lahko razdelimo n elementov v m različnih množicah, kjer nobena množica ni prazna. Večina novejših spletnih mest so spletne aplikacije, ki za prenos podatkov do odjemalca uporabljajo tehnologijo AJAX, ki nam otežuje proces zaključevanja poti. Tehnologija AJAX deluje tako, da v prikazano spletno stran na strani odjemalca preko skriptnega jezika JavaSript vključi dodatne podatke v spletno stran, ki jih je odjemalec prejel od

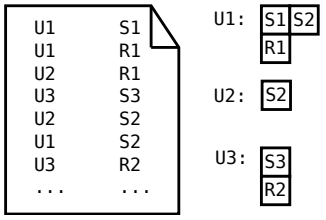
spletnega strežnika. Primer delovanja tehnologije AJAX je prikazan na sliki 4.4,



Slika 4.4: Komunikacija z uporabo tehnologije AJAX.

kjer uporabnik pregleduje podatke. Zaradi velikosti podatkov uporabnikov odjemalec ne pridobi celotnih podatkov v eni zahtevi, temveč se podatki razdelijo na več razdelkov. Problemi se pojavljajo pri ugotavljanju takih zahtev. V primeru, ko ima format log datoteke polje napotitelja, take zahteve lahko prepoznamo. Ko v formatu log datoteke nimamo polja napotitelja, pa takšnih zahtev ne moremo prepoznati. Za pravilno rekonstrukcijo in zaključevanje manjkajočih povezav lahko uporabljamo stohastične modele, kot sta markovski model ali skriti markovski model ter njune različice, ki uporabljajo več predhodnih stanj modela za napovedovanje. Markovski model se uporablja pri izračunu verjetnosti prehoda med dvema stranema. Vhod v markovski model je stohastična matrika, ki jo poznamo tudi pod imeni verjetnost matrika, prehodna matrika, matrika zamenjav ali markova matrika. Matrika je lahko vhodni argument programa, toda je v tem primeru rešitev omejena samo na eno spletno mesto. Stohastične matrike se lahko naučimo tekom delovanja programa. Uporabimo lahko metodo preiskovanja prostora stanj, kjer stanja predstavljajo spletne strani na spletnem mestu, vendar pa ta metoda potrebuje vnaprej opredeljena stanja, ki jih lahko zasede seja uporabnika. Slabost metode preiskovanja stanj je tudi časovna potratnost pri pregledovanju prostora stanj, ki jih ima lahko seja uporabnika. Pospešitev metode pregledovanja stanj je uporaba dobre hevristične funkcije. Ena izmed metod, ki bi se lahko s konkretno predelavo uporabljala za rekonstrukcijo sej, je gručenje uporabnikov. Metoda potrebuje dodatne podatke, ki so specifični za spletno mesto.

Na levi strani slike 4.5 vidimo primer log datoteke, ki vsebuje klikotok treh uporabnikov. Prvi stolpec v log datoteki predstavlja identifikacijo uporabnika.



Slika 4.5: Primer razpletanja sej.

Drugi stolpec v log datoteki predstavlja zahtevano spletno stran ali resurs spletne strani. Na desni strani slike 4.5 je viden rezultat razpletanja sej, kjer imamo tri uporabnike in tri spletne strani, ki imajo tudi resurse. Viden je tudi postopek čiščenja, kjer prve zahteve uporabnika $U2$ ne uvrstimo v sejo.

Eno boljših del tega področja je doktorska disertacija as. dr. Marka Poženela [36], ki je izdelal proces rekonstrukcije sej z uporabo metode pregledovanja stanj in markovskimi modeli. Vendar pa, kot je že on omenil, je popolna rekonstrukcija sej še vedno problem kljub metodam, ki jih je uporabil. Probleme pri rekonstrukciji sej nam predstavljajo tudi proxy strežniki, odjemalci Tor in I2P omrežja ...

4.1 Časovna in navigacijska hevristika

Tipe časovne in navigacijske hevristike smo črpali iz [39]. Časovna hevristika ima dve strategiji, ki se lahko uporabljata za razpletanje sej, kot sta:

Časovna hevristika z uporabo maksimalne dolžine seje

Časovna dolžina seje ne sme presegati praga α. Naj bo t_0 časovni žig prve zahteve v izdelani seji β. Nova zahteva s časovnim žigom t pripada seji β, če velja $t - t_0 \leqslant \alpha$. Prva zahteva s časovnim žigom, večjim kot $t_0 + \alpha$, postane nova seja δ.

Časovna hevristika z uporabo časa razmišljanja (ang. *dwell time*)

Časovni žig zahteve za novo stran ne sme presegati praga α. Naj bo t_n časovni žig zadnje zahteve v seji β. Nova zahteva s časovnim žigom t pripada seji β v primeru, ko drži $t_n - t \leqslant \alpha$, v nasprotnem primeru dodamo zahtevo s časovnim žigom t v novo sejo δ.

Navigacijska hevristika ima več strategij, vendar je v diplomskem delu opisana samo ena navigacijska hevristka, saj vse ostale navigacijske hevristike potrebuje več vhodnih podatkov za delovanje, kot je to na primer struktura spletnega mesta, opisi možnih opravil na spletnem mestu ... Pri navigacijski hevristiki, ki je opisana v diplomskem delu, si lahko pomagamo s poljem napotitelja (ang. *referrer*), ki nam pove, s katere spretne strani je prišel uporabnik do nove spletne strani.

Navigacijska hevristika z uporabo polja napotitelja

Naj bosta p in q dve zahtevi za spletno stran, kjer zahteva p že pripada seji β. Zahteva q se doda seji β, če je polje napotitelja v zahtevi q enako zahtevani spletni strani v zahtevi p. V nasprotnem primeru se izdela nova seja δ, ter se zahteva q doda seji δ.

Vendar pa imamo pri uporabi navigacijske hevristike z uporabo polja napotitelja isti problem, kot pri polju piškotka, saj večina log formatov nima tega polja.

Poglavje 5

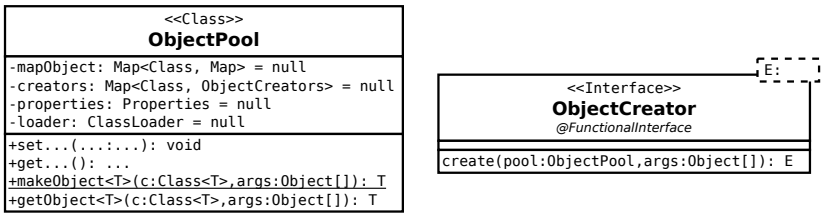
Implementacija rekonstrukcije sej

Za procesiranje klikotoka, ki se nahaja v log datoteki spletnega strežnika, smo morali razviti različne rešitve, ki niso del programskega jezika Java ter za te procese ne obstaja nobeno že izdelano orodje. Prvi pomembnejši del, ki smo ga morali razviti je parser, ki je zmožen obdelati različne tipe formatov log datotek. Parser omogoča tudi preprosto dodajanje novih tipov polj formatov log datotek. Za identifikacijo tipa polj log formata smo uporabili imena, ki so definirana v implementaciji spletnega strežnika Apache in standardu formata ELF. Razredi, odgovorni za hranjenje podatkov vrstice in za shranjevanje v relacijsko podatkovno bazo, se nahajajo v paketu `org.sessionization.parser.fields`. V tem paketu so razredi, ki se uporabljajo pri obeh tipih formatov log datotek, vendar pa imata formata tudi specifična polja, ki se lahko uporabljajo samo v izbranem formatu. Te razrede pa najdemo za log format NCSA v paketu `org.sessionization.parser.fields.ncsa` in za format W3C v paketu `org.sessionization.parser.fields.w3c`. Razviti smo tudi dinamične razrede, ki se uporabljajo za hranjenje podatkov o sejah ter omogočajo lažjo obdelavo podatkov. Dinamične razrede uporabljamo predvsem zaradi različnim formatov log datotek, saj pred poganjanjem programa ne vemo natančno, kakšen je format log datoteke. Razvili smo tudi dodatne podatkovne strukture, ki pripomorejo k hitrejšem delovanju programa. Ena pomembnejših je drevo Patricia, ki ga uporabljamo pri procesiranju klikotoka. Večina razvitih

podatkovnih struktur zagotavlja tudi sinhronizacijo podatkov pri sočasnem izvajanju. Aplikacija vsebuje tudi skladišče za objekte, s čimer poskušamo zmanjšati delovanje smetarja, ki skrbi za brisanje neuporabnih objektov. Skladišče objektov je namenjeno predvsem orodju Hibernate, saj orodju ne moremo povedati, da so vrstice, ki imajo iste vrednosti atributov, enake ter naj zato ne shrani objekta, ampak naj samo nastavi vrednost identifikacijskega polja v objektu. Ta problem bi lahko rešili s poizvedovanjem v jeziku HQL oziroma JPQL, vendar so hitrosti poizvedb na nekaterih relacijskih podatkovnih bazah časovno potratne. Delo celotne aplikacije smo razdelili na tri tipe niti. Prvi tip niti uporablja parser ter tako pridobiva podatke iz log datoteke. Drug tip niti so niti, ki izvedejo proces čiščenja dnevnika in proces identifikacije sej. Zadnji tip so niti, ki služijo shranjevanju objektov v relacijsko podatkovno bazo.

5.1 Skladišče objektov

Vzorec skladišča objektov (ang. *object pool design pattern*) [38] pripomore k pohitritvi nekaterih programov. Njegova efektivnost se opazi v situacijah, kjer je cena izdelave novega objekta visoka. Programski jezik Java uporablja ta vzorec pri razredih, ki se uporabljajo za nadziranje delovanja niti. Pri aplikaciji smo razvili skladišče objektov predvsem zaradi orodja Hibernate. Izdelano skladišče objektov podpira tudi sočasno izvajanje.



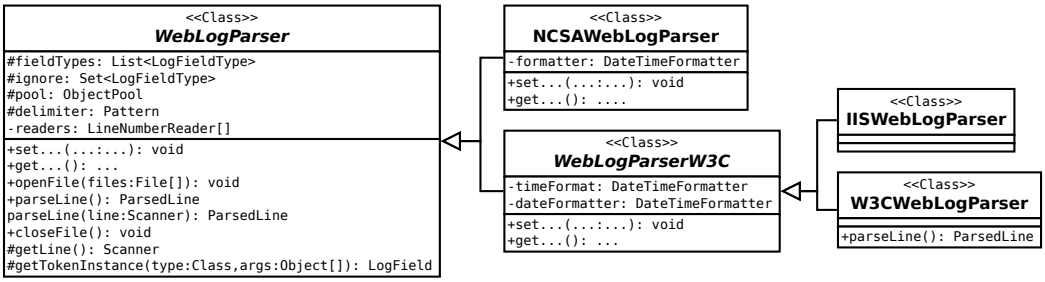
Slika 5.1: Razredni diagram skladišča objektov.

Slika 5.1 prikazuje razred `ObjectPool` ter funkcijski vmesnik `ObjectCreatro`. Funkcijski vmesnik se uporablja za izdelavo objektov, ki v razredu vsebujejo združevanje z razredi, ki jih želimo hraniti v našem skladišču. Razred `ObjectPool` implementira vzorec skladišča objektov, vendar pa je implementacija splošnejša od imple-

mentacije navede v knjigi. Statična metoda `makeObject` omogoča izdelavo novih primerkov razreda. Tip razreda, za katerega želimo izdelati primerek, navedemo v argumentu `c`. Argument `args` predstavlja vhodne argumente konstruktorja, ki ga želimo uporabiti za izdelavo primerka razreda. Metoda `getObject` uporablja polje `mapObject`, kjer se nahajajo že izdelani primerki. Metoda najprej preveri ali smo objekt že izdelali. Če objekt že obstaja, potem vrne najden objekt, v nasprotnem primeru pa ga izdela s pomočjo metode `makeObject`, ga shrani v polje `mapObject` ter vrne.

5.2 Parser

V izdelku diplomskega dela imamo tri tipe parserjev. Prvi tip parserja se uporablja za obdelavo formata NCSA, drugi pa se uporablja za obdelavo formata W3C in tretji za obdelavo formata IIS, ki je podoben log formatu W3C. Razlika med log formatoma W3C in IIS je v tem, da IIS nima direktiv ter uporablja drugačen ločilni znak za delitev polj. Prvega smo poimenovali `NCSAWebLogParser`, drugega pa `W3CWebLogParser` in zadnjega kot `IISWebLogParser`.



Slika 5.2: Razredni diagram parserja.

Abstraktni razred `WebLogParser` na sliki 5.2 vsebuje skupne lastnosti vseh parserjev, ki jih ima izdelana aplikacija. Abstraktni razred `WebLogParser` implementira tudi vmesnika `Iterable` in `AutoCloseable`, ki nista vidna na sliki 5.2. Vmesnik `Iterable` doda možnost uporabe `foreach` zank ter uporabo metode `for Each`, ki je novost programskega jezika Java 8. Vmesnik `AutoCloseable` omogoča uporabo kontrolnega stavka `try` z resursom, kjer se resurs avtomatično zapre oziroma uniči, ko ga več ne potrebujemo.

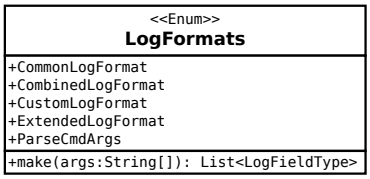
Polje `readers` abstraktnega razreda `WebLogParser` predstavlja vhodne log datoteke, iz katerih pridobivamo podatke. Tabelo vhodnih log datotek smo uporabili, ker smo podprli možnost, ki jo ponuja spletni strežnik Apache za hranjenje log formata v več različnih datotekah. Pomembnejša funkcija abstraktnega razreda `Web LogParser` je metoda `parseLine`, ki skrbi za pridobivanje obdelanih podatkov iz log datoteke spletnega strežnika. Sprememba metode `parseLine` je bila potrebna le pri parseju za format W3C, kjer moramo upoštevati direktive, ki jih ima format log datoteke. V abstraktnem razredu `WebLogParser` vsebuje tudi različico `parse Line` metode, ki prejme argument `line` tipa `Scanner`, ki predstavlja vrstico iz log datoteke. Argument je tipa `Scanner`, ker ta tip podpira branje podatkov s pomočjo regularnih izrazov. Polje `fieldTypes` abstraktnega razreda `WebLogParser` predstavlja format log datoteke. Format log datoteke je predstavljen s tipi polj, ki jih vsebujejo vhodne log datoteke. Pri log formatu tipa NCSA in IIS je potrebno tipe polj nastaviti pred uporabo metode `parseLine`, kar pa ne drži za log format W3C, kjer uporabimo direktivo `Fields`. S pomočjo te datoteke nastavimo tipe polj vhodne log datoteke. Polja log formata lahko tudi ignoriramo. Polja, ki jih želimo ignorirati, se nahajajo v polju `ignore` abstraktnega razreda `WebLogParser`. V abstraktnem razredu `WebLogParser` imamo tudi polje `delimiter` tipa `Pattern`. Polje definira tip ločilnega znaka med podatki v log datoteki.

V abstraktnem razredu `WebLogParser` imamo tudi polje `pool`, ki hrani objekt tipa `ObjectPool`. Ta objekt uporabljamo pri transformaciji podatkov log datoteke v objekt, ki se uporablja v aplikaciji za obdelavo sej. Objekt se uporablja v metodi `getTokenInstance`. Vrednost polja je lahko tudi `null`, kar pomni, da se bo za iste podate log datoteke ob klicu metode `getTokenInstance` ustvaril nov objekt, ki bo imel drugačno referenco. V nasprotnem primeru pa skušamo z razredom tipa `ObjectPool` podati vedno isti objekt za iste podatke log datoteke.

5.2.1 Nastavitev tipov polj log datoteke

Za obdelavo argumenta niza imen polj modula `CustomeLog` spletnega strežnika Apache in za obdelavo direktive `Fields` log formata ELF uporabljamo razred, viden na sliki 5.3.

Za izdelavo tipov polj, ki se uporabljajo pri formatu `Common Log Format`, uporabimo element `CommonLogFormat`. Pri uporabi tega elementa je lahko vho-

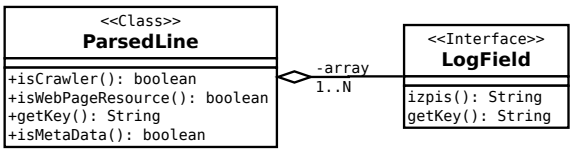


Slika 5.3: Razred za izdelavo tipov polj formata log datoteke.

dni argument metode `make` enak `null`. Za izdelavo tipov polj, ki se uporabljajo pri formatu `Combined Log Format`, uporabimo element `CombinedLogFormat`. Pri uporabi tega elementa je lahko vhodni argument metode `make` enak `null`. Pri log formatu NCSA lahko tudi imena polj navedemo tako, da uporabimo element `CustomLogFormat`, kjer metodi `make` podamo tabelo imen polj log formata. Za obdelavo direktive `Fields` se uporablja element `ExtendedLogFormat`, kjer je argument metode `make` tabela podatkov direktive. Dodali smo tudi elemente `ParseCmd Args`, kjer lahko uporabimo imena tipov polj, ki se uporabljata v obeh formatih.

5.2.2 Obdelava vrstice log datoteke

Parser s pomočjo metode `parseLine` vrne objekt tipa `ParsedLine`, katerega razredni diagram je viden na sliki 5.4. Razred `ParsedLine` nam omogoča prepoznavanje robotov, identifikacijo uporabnikov, prepoznavanje direktiv in prepoznavanje tipa zahteve.

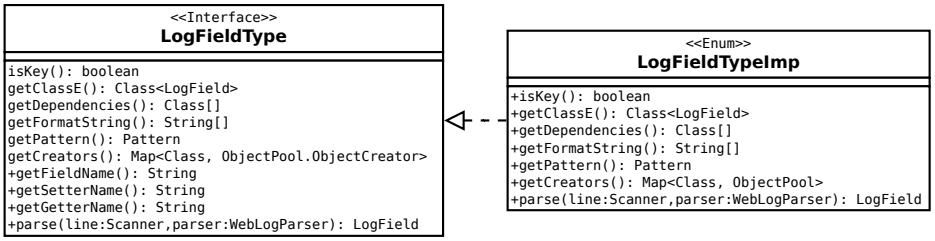


Slika 5.4: Razreni diagram za obdelano vrstico.

Razred `ParsedLine` ima polje, ki vsebuje podatke obdelane vrstice log datoteke. Polje je tabela, ki hrani podatke tipa `LogField`, zato morajo razredi za hranjenje podatkov obdelane vrstice implementirati vmesnik `LogField`. Vsi razredi, ki se nahajajo v paketu `org.sessionization.parser.fields` in v njegovih podpaketih, implementirajo ta vmesnik. Vmesnik zahteva od programerja imple-

mentacijo dveh metod. Metoda `izpis` se uporablja za preprost izpis podatkov in metoda `getKey` vrne niz, če polje predstavlja del niza za identifikacijo uporabnika.

Metode razreda `ParsedLine` so namenjene procesu čiščenja dnevnika in identifikaciji uporabnikov. Za prepoznavanje robotov uporabljamo metodo `isCrawler`. Za identifikacijo uporabnikov uporabljamo metodo `getKey`, ki vrne niz s kombinacijo polj za identifikacijo uporabnika. Za prepoznavanje vrstic z direktivami uporabljamo metodo `isMetaData`. Metoda `isWebPageResource` nam omogoča prepoznavanje zahteve, ki prestavlja resurs spletne strani.



Slika 5.5: Razreni diagram za obdelavo podatkov vrstice log datoteke.

Ker smo metodo `parseLine` implementirali že v abstraktnem razredu `WebLog Parser`, smo si pri transformaciji podatkov pomagali z vmesnikom, prikazanim na sliki 5.6. Podatke log datoteke procesiramo s pomočjo metode `parse`. Metoda uporablja metodo `getPattern`, ki poda vzorec podatkov, ki jih metoda mora pridobiti iz vhodne vrstice log datoteke. Vhodna vrstica je predstavljena z argumentom `line` tipa `Scanner`. Metoda kot enega od argumentov prejme parser, ki se uporablja za pridobivanje objektov iz skladišča objektov. Vmesnik vsebuje tudi metode, ki pa niso potrebne pri obdelavi vrstice log datoteke, temveč se uporabljajo za izdelavo dinamičnih razredov, za izdelavo objekta v skladišču objektov in za podajanje odvisnosti orodju Hibernate. Vmesnik vsebuje implementacije določenih metod, vendar pa smo morali za določena polja log formatov implementacijo spremeniti, kar pa je vidno v razredu `LogFieldTypeImp`.

Tipe polj log formata smo implementirali s pomočjo razreda tipa `enum`, vidnega na sliki 5.6. Razred tipa `enum` smo uporabili z namenom preprostega dodajanja novih tipov polj, ki jih nismo v aplikaciji zajeli. Ta razred se uporablja v razredu `LogFormats`, kjer s pomočjo sprehoda čez vse elemente razreda `LogFieldTypeImp` pridobimo vsa polja, ki jih je zmožna aplikacije obdelati. Koda 5.1 v vrstici 5

prikazuje `for` zanko za sprehajanje po elementih razreda tipa `enum`.

```
1  private Map<String, LogFieldType> enumMaper;
2
3  LogFormats(int n) {
4    Map<String, LogFieldType> map = new RadixTreeMap<>();
5    for (LogFieldType type : EnumSet.allOf(LogFieldTypeImp.class)) {
6      for (String s : type.getFormatString()) {
7        switch (s.isEmpty() ? 3 : n) {
8          case 0: // log format NCSA
9            if (s.charAt(0) == '%') map.put(s, type);
10           break;
11         case 1: // log format W3C
12           if (s.charAt(0) != '%') map.put(s, type);
13           break;
14         case 2:
15           map.put(s, type);
16           break;
17       }
18     }
19   }
20   enumMaper = map;
21 }
```

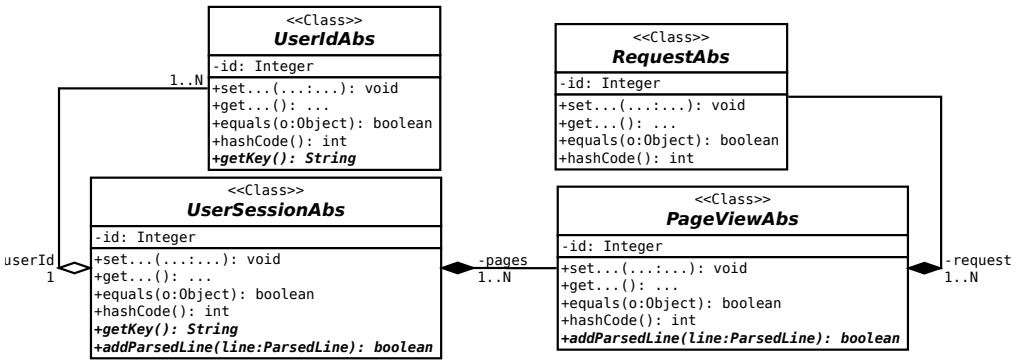
Koda 5.1: Sprehod po elementih razreda tipa `enum`.

Nov tip polja log formata dodamo tako, da razredu `LogFieldTypeImp` dodano nov element, ki bo predstavljal nov tip polja log datoteke. Konstruktorju novega elementa podamo tri argumente: 1. argument predstavlja ime tipa, ki se uporablja v implementaciji spletnega strežnika, 2. argument predstavlja razred, ki se bo uporabljal za hranjenje tega podatka in 3. argument prestavlja vzorec za pridobivanje podatkov, ki ga uporabi razred `Scanner`. V primeru, ko novo polje predstavlja identifikacijo uporabnika, spremenimo tudi metodo `isKey`, tako da vrne vrednost `true`. Če nov razred potrebuje dodatne razrede za delovanje, jih podamo v implementaciji metode `getDependencies`. Med odvisnosti razreda spadajo pretvorniki podatkovnih tipov za programski vmesnik JPA, notranji razredi ... V primeru, ko ima nov tip notranje razrede ter te razrede želimo shraniti v skladišče objektov, moramo implementirati tudi metodo `getCreators`.

5.3 Dinamični razredi

Eden glavnih problem izdelave programa za rekonstrukcijo sej, ki podpira več različnih formatov log datotek spletnih strežnikov, so raznoliki podatki, ki se lahko pojavijo v vrstici formata log datoteke. Podatek moramo tudi znati hraniti ter pri procesiranju podatkov uporabiti pravilno funkcijo za njihovo obdelavo. Programski jezik Java nam preko razreda `ClassLoader` omogoča nalaganje dodatnih razredov,

ki jih lahko pridobimo iz različnih virov, kot so lokalne datoteke, ki prestavljajo prevedeno programsko kodo, lokalne datoteke v arhivi `jar` ali pa jih pridobimo iz oddaljenega vira preko omrežja. Z orodji kot so Javassist, ASM ali BCEL, lahko razrede izdelamo med delovanjem programa, ter jih imamo shranjene le v delovnem pomnilniku in jih ob končanju programa pozabimo. V razviti aplikaciji smo uporabili orodje Javassist. Za uporabo dinamičnih razredov smo uporabiti dedovanje preko abstraktnih razredov, saj smo določene funkcionalnosti dinamičnih razredov implementirali že v abstraktnih razredih.



Slika 5.6: Razreni diagram iz katerih izhajajo dinamični razredi.

V aplikaciji smo razvili štiri dinamične razrede, katerih abstraktni razredi so vidni na sliki 5.6. Imena dinamičnih razredov imajo ista imena, vendar brez pripone abs.

UserSessionAbs

Razred hrani sejo uporabnika. Razred je zmožen identifikacije uporabnika preko abstraktne metode `getKey`, ki jo znamo implementira šele ko poznamo format log datoteke. Za dodajanje novih zahtev seji uporabljamo abstraktno metodo `addParsedLine`, ki pa jo tudi znamo implementirati šele, ko nam je znan format log datoteke. V primeru, ko v formatu nimamo podatkov o identifikaciji uporabnika, polja `userId` ne dodamo v implementacijo razreda. Isto pravilo velja za podatke, ki se ne uporabljajo za identifikacijo uporabnika, toda takrat se ne doda polje `pages` v implementacijo razreda. V polju `pages` shranjujemo prikaze spletnih strani, ki si jih je ogledal uporabnik. Razred vsebuje tudi implementacije metod za pridobivanje vrednosti

razredov tipa `UserIdAbs` in `PageViewAbs`, ki vrneta vrednost `null`. Vendar pa sta metodi za nastavljanje obeh polj odvisni od formata log datotek, zato njuni implementaciji nista v abstraktnem razredu. Implementacijo razreda izdelamo z uporabo razreda `UserSessionDump`.

UserIdAbs

Razred hrani identifikacijo uporabnika. Ta razred lahko identificira več objektov tipa `UserSessionAbs`. Identifikacijo uporabnika pridobimo z uporabo abstraktne metode `getKey`, katero implementiramo šele, ko nam je znan format log datoteke. Abstraktni razred vsebuje tudi vse metode za pridobivanje vrednosti razredov, ki predstavljajo identifikacijo uporabnikov. Ko poznamo format log datoteke, pa določene metode za pridobivanje prikrijemo z metodami v implementaciji tega razreda. Implementacija tega razreda vsebuje tudi metode za nastavljanje objektov, ki so v log formatu odgovorni za identifikacijo uporabnika. Implementacijo razreda izdelamo z uporabo razreda `UserIdDump`.

PageViewAbs

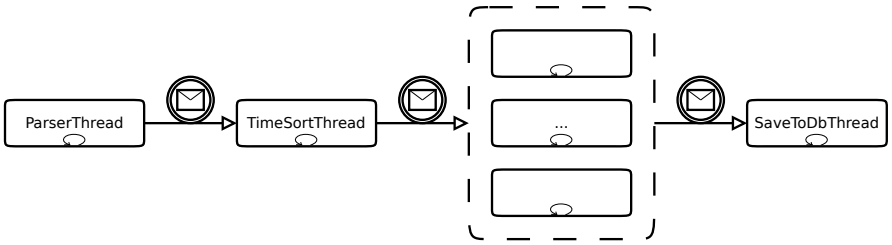
Razred prestavlja prikaz spletne strani. Prikaz spletne strani je skupek zahtev, kjer imamo eno zahtevo za spletno stran, ostale zahteve pa predstavljajo resurse te spletne strani. Večino delov tega razreda smo implementirali v abstraktnem razredu. Razred vsebuje polje `requests`, ki hrani zahteve odjemalca. Abstraktno metodo implementiramo šele v implementacije razreda, ker pred izvajanjem programa nimamo potrebnega konstruktorja za izdelavo implementacije razreda `RequestAbs`. Implementacijo razreda izdelamo z uporabo razreda `PageViewDump`.

RequestAbs

Razred predstavlja zahteve odjemalca. V tem razredu hranimo podatke, ki se ne uporabljajo za identifikacijo uporabnika. Razred vsebuje tudi implementacije metod za pridobivanje vrednosti razredov, ki pa se v odvisnosti od formata log datoteke prikrijejo v implementaciji razreda. Implementacija tega razreda doda metode za nastavljanje polj. Implementacijo razreda izdelamo z uporabo razreda `RequestDump`.

5.4 Niti

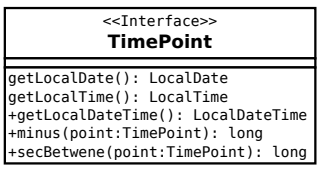
Programski jezik Java podpira paradigmo sočasnosti preko uporabe niti. Niti omogočajo deljenje dela na več vzporedno delujočih delavcev oziroma niti. Problem razpletanja sej smo v diplomskem delu razdeli na tri tipe niti. Niti med seboj komunicirajo preko prioritetnih vrst, ki omogočajo blokiranje niti. `Blocking Queue` je vmesnik programskega jezika Java, ki podpira tak tip prioritetnih vrst. V izdelani aplikaciji smo uporabili razred `ArrayBlockingQueue`, ki predstavlja implementacijo vmesnika `BlockingQueue`.



Slika 5.7: Komunikacija med nitmi.

Slika 5.7 prikazuje komunikacijo in smer komunikacije med nitmi. Slika 5.7 vsebuje štiri tipe niti, kjer četrti tip niti predstavlja nadgradnjo izdelane aplikacije, ki implementira proces zaključevanja. Med nitma za časovno hevristiko in shranjevanje v relacijsko podatkovno bazo lahko dodamo tudi procese, ki jih nismo predvideli v diplomskem delu. Za zaključevanje dela niti smo uporabili metodo signaliziranja preko posebnega statičnega polja, ki jih vsebujejo razredi niti.

5.4.1 Časovna hevistika



Slika 5.8: Vmesnik za časovno hevristiko.

Metodo časovne hevristike smo implementirali preko niti, ki je predstavljena z razredom `TimeSortThread`. Za izračun pretečenega časa med dvema zahtevama smo razvili dodaten vmesnik. Slika 5.8 prikazuje vmesnik, ki predstavlja enega glavnih delov implementacije časovne hevristike. V vmesniku smo implementirali glavne metode, ki se uporabljajo za izračun časovne razlike med dvema zahtevama. Metodi `minus` in `secBetwene` vrneta časovno razliko v sekundah, ki so pretekle med zahtevama. Razlika med metodama je ta, da metoda `minus` lahko vrne negativno vrednost. Negativno vrednost pridobimo, ko je čas, podan kot argument, manjši od časa, nad katerim izvajamo operacijo odštevanja. Implementacija metod `getLocalDate` in `getLocalTime` je odvisna od formata log datoteke.

Algoritem 5.1 Časovno hevristika z uporabo čas razmišljanja.

```
1: function procesLine(file, len)
2:     for all l in file do
3:         if !isMetaData(l) then
4:             s ← getSession(l)
5:             if s = null and !isResouce(l) and !isRobot(l) then
6:                 s ← makeSession(l)
7:                 addSession(s)
8:             else if s ≠ null then
9:                 if isResouce(l) then
10:                    addToSession(s, l)
11:                else if minus(l, s) ≤ len then
12:                    addToSession(s, l)
13:                else
14:                    finishSession(s)
15:                    s ← makeSession(l)
16:                end if
17:            end if
18:            if s ≠ null then
19:                offer(s)
20:            end if
21:            for all s in queue() and minus(l, s) > len do
22:                finishSession(s)
23:            end for
24:        end if
25:    end for
26:    for all s in queue() do
27:        finishSession(s)
28:    end for
29: end function
```

Algoritem 5.1 prikazuje psevdo kodo implementacije časovne hevristike. Vhodi algoritma sta datoteka, predstavljena s spremenljivko $file$, in maksimalni čas, predstavljen s spremenljivko len. Maksimalni čas predstavlja čas, ki lahko preteče med dvema zahtevama, preden zahtevo v obdelavi uvrstimo v novo sejo. Algoritem uporablja tudi tabelo za hranjenje sej in prioritetno vrsto, urejeno po času seje, kjer je prvi element seja z najmanjšim časom. V vrstici 21 je začetek `for` zanke, ki zaključi seje, ki presegajo maksimalen čas zahteve v obdelavi. S tem zmanjšamo

prostorsko zahtevnost algoritma. Ko obdelamo vse vrstice log datoteke, se na tabeli za hranjenje sej te še vedno nahajajo, zato imamo v vrstici 26 `for` zanko, ki zaključi preostale seje. Čas, ki se uporabi za izračun časa razmišljanja je po navadi čas zadnje zahteve za resurs spletne strani.

Algoritem 5.1 upošteva tudi proces čiščenja dnevnika. Zahteve, ki predstavljajo resurse spletne stani in nimajo pred tem zahteve za spletno stran, algoritem ne upošteva. Ignoriramo tudi zahteve, podane s strani robota. Za proces čiščenja dnevnika sta pomembni vrstici 5 in 9.

5.4.2 Shranjevanje v relacijsko podatkovno bazo

Za shranjevanje podatkov v relacijsko podatkovno bazo je zadolžena nit, predstavljena z razredom `SaveToDbThread`. Ker je aplikacija zmožna shranjevati podatke tudi v relacijsko podatkovno bazo, ki smo jo pred tem že uporabljali, smo morali dodati nov vmesnik, ki skrbi za pravilno nastavljanje identifikacijskih polj razredov aplikacije za hranjene podatkov vrstice log datoteke.

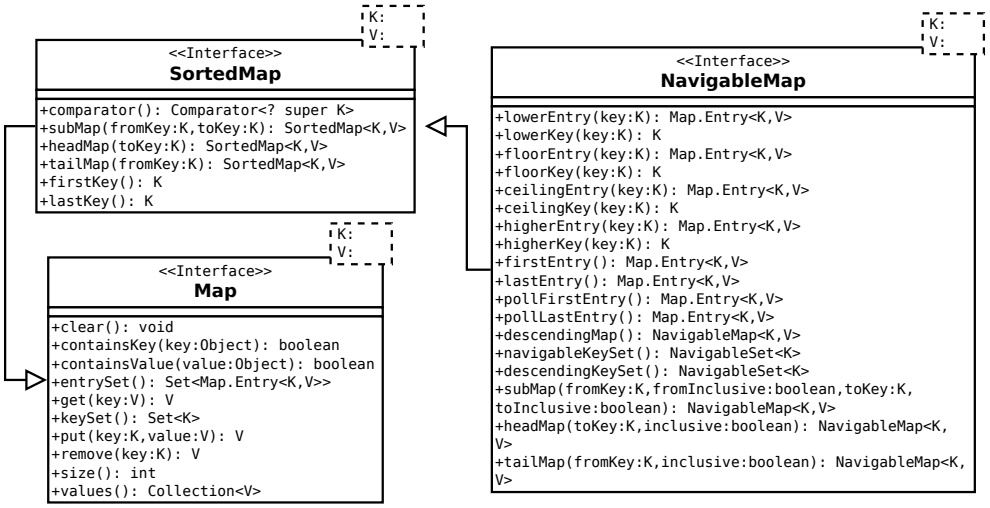
Slika 5.9: Vmesnik za nastavljanje identifikacijskega polja.

Slika 5.9 prikazuje vmesnik, ki ga morajo implementirati razredi za hranjene dodatkov obdelane vrstice log datoteke. Metoda `setDbId` je v implementacijah odgovorna za nastavljanje identifikacijskih polj s pomočjo razreda `Session`, ki ga uporabljamo za izdelavo poizvedb v jeziku HQL in JPQL. Metoda ko rezultat vrne vrednost, ki se nastavi identifikacijskemu polju. V primeru, ko ima objekt že nastavljeno identifikacijsko polje, metoda ne izvede poizvedbe, ampak samo vrne vrednost polja.

5.5 Podatkovne strukture

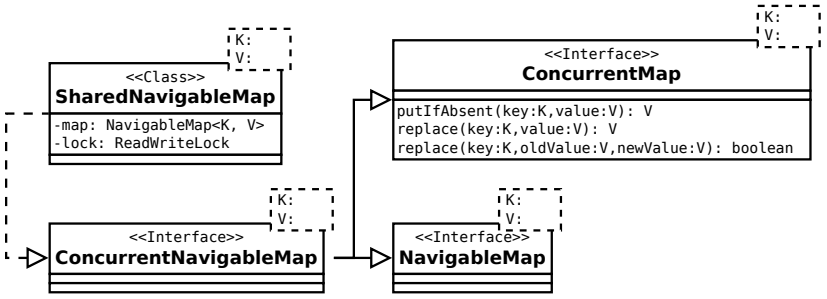
Programski jezik Java ima veliko podatkovnih struktur, ki jih lahko uporabljamo. V diplomskem delu smo dodatno razvili podatkovne strukture, ki so pripomo-

gle k boljši učinkovitosti delovanja aplikacije. Izboljšano učinkovitost podatkovne strukture opazimo pri porabi pomnilnika ter pri hitrosti delovanja. Nove podatkovne strukture so tipa slovar in prioritetna vrsta. Izdelali smo tudi podatkovno strukturo, ki ima lastnosti slovarja in prioritetne vrste. Vse dodatno izdelane podatkovne strukture se nahajajo v paketu `org.datastruct`.



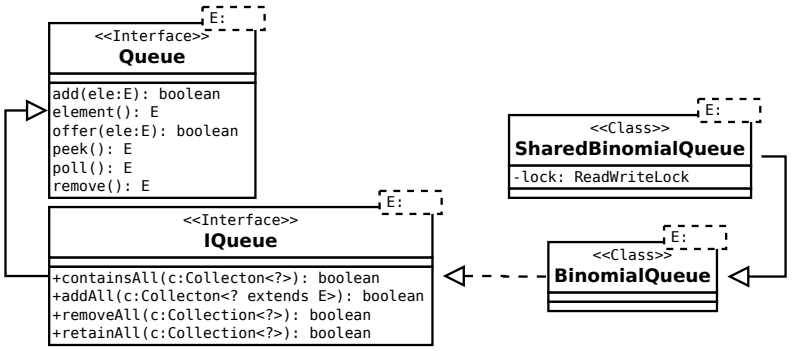
Slika 5.10: Vmesniki, uporabljeni v slovarjih.

Implementacije slovarjev, ki smo jih dodatno izdelali, so: drevo AVL, preskočen seznam in drevo Patricia. Slika 5.10 prikazuje dedovanje med vmesniki, ki predstavljajo sezname. V aplikaciji smo v prej navedenih strukturah implementirali vmesnih `NavigableMap`. Vse metode vmesnikov za slovarje smo implementirali v iterativnem načinu.



Slika 5.11: Vmesniki za vzporedno delovanje slovarjev.

Slika 5.11 prikazuje implementacijo sočasnih slovarjev. V aplikaciji smo zagotovili razred `SharedNavigableMap`, ki se uporablja za izdelavo sočasnih podatkovnih struktur tipa `NavigableMap`. Razred `SharedNavigableMap` vsebuje konstruktor, ki prejme slovar tipa `NavigableMap`. Podan slovar se uporablja v metodah razreda `SharedNavigableMap`, kjer uporabljamo ključavnico tipa `ReadWriteLock`. Ključavnica nam omogoča sočasno branje podatkov strukture. Pri operacijah spreminjanja podatkov pa ključavnica blokira vse, ki bi želeli brati oziroma spreminjati podatke, vse dokler se ne sprosti ključavnica za spreminjanje podatkov.



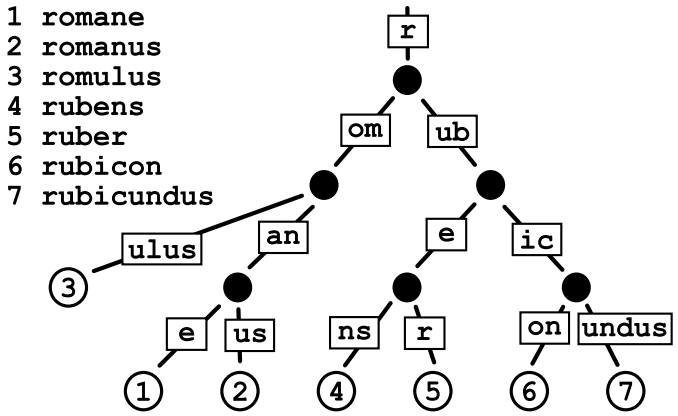
Slika 5.12: Implementacija binomske kopice.

Slika 5.12 prikazuje dedovanje med vmesniki za podatkovne strukture tipa vrsta. Prioritetna vrsta, ki smo jo razvili v aplikacije, je binomska kopica. Binomska kopica je zmožna delovati pravilno tudi v sočasnih programih. To smo zagotovi preko razreda `SharedBinomialQueue`. Vmesnik `Queue` implementira tudi vmesnike `Collection` in `Iterable`. Določene metode vmesnika `Collection` smo implementirali že v vmesniku `IQueue`.

5.5.1 Drevo Patricia

Podatkovna struktura spada med številska drevesa. Za iskanje elementov lahko uporablja podatke, ki predstavljajo sekvenco oziroma zaporedje podatkov istega tipa. Drevo je zmožno tudi stiskanje poti. Aplikacijo drevesa najdemo v omrežnih napravah, kjer s stiskanjem poti zmanjša velikost podatkov in omogoča hitrejše iskanje. Drevo se lahko uporablja tudi pri urejevalnikih besedila za preverjanje pravilnosti besed v besedilu. Najdemo ga tudi v različnih razvojnih okoljih, kjer pro-

gramerju ponuja opcije za zaključevanje besed, npr. imena spremenljiv, ključnih besed ...



Slika 5.13: Primer drevesa Patricia [11].

Slika 5.13 prikazuje primer drevesa Patricia, ki vsebuje sedem vrednosti. Na sliki 5.13 je vidna tudi tehnika stiskanja poti, ki je značilna za to drevo. V aplikaciji smo razvili drevo Patricia, ki za podatkovni tip ključa sprejme razred `String`. Drevo smo implementira preko vmesnika `NavigableMap`. V aplikaciji uporabljamo to drevo kot primarno podatkovno strukturo za hranjene sej uporabnikov, kjer je ključ drevesa kombinacija polj za identifikacijo uporabnika.

Poglavje 6

Uporaba aplikacije

V tem poglavju so prikazani osnovni primeri uporabe izdelane aplikacije. Prikazan je osnovni primer za uporabo aplikacije z uporabo prevzete relacijske podatkovne baze SQLite. Prikazan je tudi primer uporabe aplikacije z uporabo relacijske podatkovne baze H2. Primer, ki uporablja relacijsko podatkovno bazo H2, se lahko uporablja kot primer za druge relacijske podatkovne baze, ki jih v diplomskem delu nismo omenili. Prikazano je tudi kako moramo navesti dialekt, če orodje Hibernate nima prevzetega dialekta za izbrano relacijsko podatkovno bazo.

Argumente lahko navedemo tudi v datoteki, ki ustreza sintaksi datoteke z lastnostmi za programski jezik Java. Datoteka naj bi imela končnico `.properties`. Sintaksa vrednosti, ki se nahaja v tej datoteki je `ključ=vrednost`, `ključ = vrednost`, `ključ:vrednost` ali `ključ vrednost`. Datoteka omogoča tudi zapis vrstic, ki se ignorirajo ob obdelavi te datoteke. Takim vrsticam pravimo tudi komentarji in vrstice se začnejo z znakom `#`. Vrednosti so lahko tudi večvrstične, kjer pa moramo na koncu vrstice, ki se nadaljuje v naslednji vrstici, dodati znak \. Proces rekonstrukcije sej je ponovljiv proces, ki ga izvedemo s pomočjo izdelane aplikacije. Izdelano relacijsko podatkovno bazo lahko uporabimo tudi za hranjenje podatkov več različnih log datotek, ki imajo isti format log datoteke. S pomočjo datotek z lastnostmi zagotovimo prenosljivost vhodnih argumentov programa, med katerimi je tudi relacijska podatkovna baza.

Prikazanih je tudi nekaj poizvedb v jeziku SQL, ki se lahko uporabljajo v izhodni relacijski podatkovni bazi. Poizvedbe v jeziku SQL so mišljene za uporabo pri standardnih formatih log datotek.

6.1 Vhodni argumenti

V tem podpoglavju so prikazane opcije vodnih argumentov ukazne vrstice in ključi datoteke z lastnostmi. Nekateri argumenti so obvezni in jih moramo podati ob vsakem zagonu aplikacije. Nekateri izmed njih imajo tudi medsebojne odvisnosti.

-h

Izpis pomoči aplikacije.

-props

Pot do datoteke z lastnostmi, ki predstavljajo argumente vhodne vrstice, shranjene v datoteki.

-c oz. crawlers

Z uporabo te opcije povemo aplikaciji naj ignorira spletne robote. Opcija ne potrebuje dodatnih argumentov.

-dbcp oz. database.connection.pool_size

Maksimalno število povezav za povezovanje do relacijske podatkovne baze.

-dbddl oz.database.ddl

S to opcijo povemo programu naj izdela novo shemo relacijske podatkovne baze ali pa naj uporabi že obstoječo shemo.

-dbdic oz. database.dialect.class

Celotno ime dialekta, ki ga orodje Hibernate uporabi za izbrano relacijsko podatkovno bazo. Prevzeti dialekti se nahajajo v paketu `org.hibernate.dialect` orodja Hibernate.

-dbdi oz. database.dialect

Pot do datoteke z vmesno kodo ali arhiva `jar`, ki vsebuje dialekt za izbrano relacijsko podatkovno bazo. V primeru, ko navedemo to opcijo, moramo vnesti opcijo `-dbdi`.

-dbdrc oz. database.driver.class

Celotno ime gonilnika JDBC, ki ga orodje Hibernate uporabi za izbrano relacijsko podatkovno bazo.

-dbdr oz. database.driver

Pot do gonilnika JDBC relacijske podatkovne baze, ki se nahaja v arhivu `jar`. V primeru, ko navedemo to opcijo, moramo vnesti opcijo `-dbdrc`.

-dbun oz. database.username

Uporabniško ime uporabnika, ki se bo uporabil za pisanje v relacijsko podatkovno bazo.

-dbpw oz. database.password

Geslo izbranega uporabnika.

-dbsq oz. database.sql.show

S to opcijo povemo orodju Hibernate naj izpisuje poizvedbe v jeziku SQL, ki jih je uporabil.

-dbsqf oz. database.sql.show.format

S to opcijo povemo orodju Hibernate naj uporabi formatiranje poizvedb v jeziku SQL. V primeru, ko navedemo to opcijo, moramo vnesti opcijo `-dbsq`.

-dburl oz. database.url

Pod do datoteke izbrane relacijske podatkovne baze, ki se lahko nahaja na lokalni napravi ali pa na oddaljenem strežniku. V primeru, ko ne uporabimo prevzete relacijske podatkovne baze, moramo to opcijo obvezno vnesti.

-dbs oz. database.schema

Ime sheme, ki bo uporabljena za izdelavo oziroma posodabljanje podatkov relacijske podatkovne baze.

-fd oz. format.date

Format datuma log datoteke. Ta podatek uporabljamo za obdelavo formata datuma v obeh formatih log datotek.

-ft oz. format.time

Format časa log datoteke. Ta podatke uporabljamo samo pri formatu W3C.

-fl oz. format.log

Opcija predstavlja format vhodnih log datotek. Podatke je obvezen. Za format `Common Log Format` lahko uporabimo niz `COMMON`. Za format `Combined`

`Log Format` lahko uporabimo niz `COMBINED`. Za format `Extended Log Format` lahko uporabimo niz `EXTENDED`, kjer aplikacija po obdelavi prvih nekaj vrstic dobi format log datoteke preko direktive `Fields`. Ker lahko log format NCSA prilagajamo svojim željam in potrebam, tej opciji podamo niz, ki smo ga uporabili pri izdelavi formata log datoteka.

-flo oz. format.locale

Opcija predstavlja področne nastavitve, ki predstavljajo določeno geografsko, politično in kulturno območje. V programskem jeziku Java so te nastavitve prikazane z razredom `Locale`. V aplikaciji uporabljamo ta razred za obdelavo formata datuma, saj datum v formatu NCSA vsebuje mesec, predstavljen z nizom črk in samo s številko.

-pp oz. objectpool.properties

Pot do datoteke z lastnostmi za skladišče objektov.

-sqps oz. session.queue.parsedline.size

Velikost vrste med nitima za obdelavo vrstic in nitjo za časovno hevristiko.

-sqss oz. session.queue.session.size

Velikost vrste med nitima za časovno hervistiko in nitjo za shranjevanje podatkov v relacijsko podatkovno bazo.

-st oz. session.time

Najdaljši čas razmišljanja uporabnika na spletni strani. Vrednost mora biti podana v sekundah.

-li oz. log.ignore

Polja, ki jih bomo med obdelavo ignorirali. Vrednost opcije mora biti podana v enem nizu.

Zadnji izmed argumentov ukazne vrstice je ena ali več log datotek. Vhodne log datoteke navedemo s pomočjo poti, na kateri se nahaja datoteka.

6.2 Primeri uporabe

Spodnji primer uporabe programa prikazuje osnovni primer uporabe. V tem primeru uporabimo relacijsko podatkovno bazo SQLite, kjer je izhodna datoteka rela-

cijske podatkovne baze `sqliteDB`. Log format vhodne log datoteke je `Common Log Format`. S podanimi argumenti aplikacija prevzeto izdela tudi tabele, potrebne za hranjenje podatkov v relacijski podatkovni bazi.

```
$ java -jar SpletneSeje.jar -lf COMMON logCommon
```

Spodnji primer prikazuje uporabo datoteke z lastnostmi, kjer nastavimo tip formata in ime izhodne datoteke relacijske podatkovne baze SQLite.

```
$ java -jar SpletneSeje.jar -props db.properties
   logCommon
```

```
format.log = COMMON
database.url = jdbc:sqlite:baza
```

Slika 6.1: Primer datoteke z lastnostmi `db.properties`.

Naslednji primer prikazuje primer uporabe relacijske podatkovne baze H2. Izhodna datoteka je poimenovana `h2db`. Format log datoteke, ki ga obdelujemo, je `Combined Log Format`. Aplikaciji smo morali v tem primeru podati tudi pot do gonilnika JDBC ter njegovo polno ime. Orodje Hibernate ima za relacijsko podatkovno bazo tudi prevzet dialekt, katerega polno ime je `org.hibernate.dialect.H2Dialect`. Za pisanje podatkov v podatkovno bazo uporabljamo uporabnika z uporabniškim imenom `SA`. S podanimi argumenti aplikacija prevzeto izdela tudi tabele, potrebne za hranjenje podatkov v relacijski podatkovni bazi.

```
$ java -jar SpletneSeje.jar -lf COMBINED -dbdr lib/h2
   -1.4.190.jar -dbdrc org.h2.Driver -dbdic org.
   hibernate.dialect.H2Dialect -dburl jdbc:h2:./h2db -
   dbun SA access_log1
```

Spodnji primer prikazuje uporabo datoteke z lastnostmi za relacijsko podatkovno bazo H2. V primeru vidimo kako lahko uporabimo že izdelano datoteko relacijske podatkovne baze za dodajanje novih sej. V tem primeru prikazuje ponovljivost procesa rekonstrukcije sej.

```
$ java -jar SpletneSeje.jar -props h2.properties
   access_log2
```

Spodnji primer prikazuje uporabo relacijske podatkovne baze HSQLDB. Relacijska podatkovna baza deluje na oddaljenem strežniku, katerega naslov je `server.somedomain.com`. Do strežnika se povezujemo preko protokola HTTPS. V primeru vidimo tudi kako imajo vhodni argumenti večjo moč od argumentov v datoteki z lastnostmi. V primeru je prikazana uporaba deljenega zapisa formata log datoteke na več datotek. Vhodne datoteke moramo v tem primeru navesti v pravilnem

```
format.log = COMBINED
database.driver = lib/h2-1.4.190.jar
database.driver.class = org.h2.Driver
database.dialect.class = org.hibernate.dialect.H2Dialect
database.connection.pool_size = 2
database.url = jdbc:h2:~/IdeaProjects/spletne-seje/h2db
database.ddl = Update
#database.sql.show = true
#database.sql.show.format = true
database.username = SA
```

Slika 6.2: Primer datoteke z latnostmi `h2.properties`.

vrstnem redu, kot je to navedeno v nizu za izdelavo log formata `Combined Log Format`.

```
$ java -jar SpletneSeje.jar -props hsql.properties -
   dburl jdbc:hsqldb:https:dbserver.somedomain.com
   access_log_p1 access_log_p2 access_log_p3
```

```
format.log = COMBINED
database.driver = lib/hsqldb.jar
database.driver.class = org.hsqldb.jdbc.JDBCDriver
database.dialect.class = org.hibernate.dialect.HSQLDialect
database.connection.pool_size = 2
database.url = jdbc:hsqldb:file:./hsqldb
#database.sql.show = true
#database.sql.show.format = true
database.schema = PUBLIC
database.username = SA
```

Slika 6.3: Primer datoteke z latnostmi `hsql.properties`.

6.3 Primeri poizvedb SQL

Primeri navedenih poizvedb v jeziku SQL se lahko uporabljajo pri obeh standardnih formatih log datotek. Vendar pa bi jih lahko uporabili tudi v drugih formatih log datotek, če ti formati vsebujejo polja, ki so del obeh standardnih formatov log datotek.

```
1 select ADDRESS.ID, ADDRESS.IS_SERVER as SERVER_ADDRESS, ADDRESS.ADDRESS from
      ADDRESS
2   order by ADDRESS.IS_SERVER asc, ADDRESS.ADDRESS asc;
```

Koda 6.1: Poizvedba za prikaz številk IP.

Koda 6.1 prikazuje poizvedbo v jeziku SQL, ki prikaže vse številke IP. Rezultat poizvedbe vsebuje dva tipa številk IP, ki sta številka IP strežnika in številka IP odjemalcev. Vrstica 2 prikazuje urejanje po tipu številke IP, potem pa še po vrednosti številke IP.

```
1 select URISTEAM.ID, URISTEAM.FILE as REQUESTED_FILE_WIHT_PREFIX from
      URISTEAM
2   where URISTEAM.FILE like '/hlace%';
```

Koda 6.2: Poizvedba vseh spetnih strani in njihovih resursov.

Koda 6.2 prikazuje preprost primer izpisa vseh strani, ki so jih zahtevali uporabniki v njihovih sejah. V vrstici 2 je prikazan pogoj, kjer smo zahtevali le spletne strani in resurse, ki se začnejo z nizom `/hlace`.

```
1 select USERSESSION.ID as SESSIONID, ADDRESS.ADDRESS as IP, URISTEAM.FILE,
      URIQUERYPAIR.VALUE as QUERYVALUE, URIQUERYKEY.NAME as QUERYKEY, REQUEST.
      DATE, REQUEST.TIME, REQUEST.SIZE_OF_RESPONSE, PROTOCOL.PROTOCOL,
      PROTOCOL.VERSION, REQUEST.STATUS_CODE
2   from USERID
3     join ADDRESS on USERID.ADDRESS_ID = ADDRESS.id
4     join USERSESSION on USERID.ID = USERSESSION.USERID_ID
5     join USERSESSION_PAGEVIEW on USERSESSION.ID = USERSESSION_PAGEVIEW.
      USERSESSION_ID
6     join PAGEVIEW on USERSESSION_PAGEVIEW.PAGES_ID = PAGEVIEW.ID
7     join PAGEVIEW_REQUEST on PAGEVIEW.ID = PAGEVIEW_REQUEST.PAGEVIEW_ID
8     join REQUEST on PAGEVIEW_REQUEST.REQUESTS_ID = REQUEST.ID
9     join PROTOCOL on REQUEST.PROTOCOL_ID = PROTOCOL.ID
10    join URISTEAMQUERY on REQUEST.STEAMQUERY_ID = URISTEAMQUERY.ID
11    join URISTEAM on URISTEAMQUERY.URISTEAM_ID = URISTEAM.ID
12    join URIQUERY on URISTEAMQUERY.QUERY_ID = URIQUERY.ID
13    join URIQUERY_URIQUERYPAIR on URIQUERY.ID = URIQUERY_URIQUERYPAIR.
      URIQUERY_ID
14    join URIQUERYPAIR on URIQUERY_URIQUERYPAIR.PAIRS_ID = URIQUERYPAIR.ID
15    join URIQUERYKEY on URIQUERYPAIR.KEY_ID = URIQUERYKEY.ID
16  where ADDRESS.ADDRESS like '193.77.14.19'
17  order by USERSESSION.ID asc, REQUEST.DATE asc, REQUEST.TIME asc;
```

Koda 6.3: Poizvedba za prikaz seje uporabnika po števili IP.

Koda 6.3 prikazuje primer poizvedbe v jeziku SQL, ki se uporablja za pregled sej izbranega uporabnika. V vrstici 16 je prikazan pogoj, kjer smo uporabili številko IP uporabnika, o katerem smo želeli pridobiti vse seje.

Poglavje 7

Sklepne ugotovitve

V diplomskem delu smo izdelali aplikacijo, ki je zmožna razpletanja sej iz različnih formatov log datotek. Izdelana aplikacija kot glavne vhodne parametre prejme eno ali več log datotek spletnega strežnika in uporabljen format v log datotekah. Izhod aplikacije je relacijska podatkovna baza, ki hrani rekonstruirane seje uporabnikov. Izhodna relacijska podatkovna baza nam omogoča lepši pregled podatkov in zmanjša velikost podatkov v primerjavi z vhodnimi log datotekami. Izdelali smo tudi parser, ki omogoča obdelavo različnih tipov log formatov. Zaradi ubrane implementacije parserja smo omogočili preprosto dodajanje novih tipov polj, ki jih nismo zajeli v izdelano verzijo aplikacije. Izdelali smo tudi štiri dinamične razrede, ki pripomorejo k lažji obdelavi podatkov sej. Ker današnje naprave in operacijski sistemi omogočajo izvajanje programa na večih nitih, smo v aplikaciji podprli tudi uporabo tega konstrukta. Z uporabo večnitnega delovanja smo omogočili metodam razpletanja sej hitrejše delovanje, saj so lahko določeni procesi časovno potrati ter ustavljajo delovanje procesov, ki bi se lahko izvajali. Primer časovno potratnega procesa v izdelani aplikaciji je lahko shranjevanje v relacijsko podatkovno bazo. To smo opazili tudi pri uporabi naše aplikacije z uporabo prevzete relacijske podatkovne baze SQLite, vendar pa pri uporabi relacijske podatkovne baze H2 nismo imeli podobnih problemov. V diplomskem delu smo razvili tudi veliko podatkovnih struktur, med katerimi je pomembnejša implementacija drevesa Patricia. Izdelane podatkovne strukture so nam olajšale in pohitrile delovanje aplikacije. Izdelane podatkovne strukture omogočajo tudi sočasno delovanje, kjer smo morali biti pozorni na pravilno sinhronizacijo podatkov v podatkovni strukturi. V

izdelani aplikaciji smo podprli uporabo več različnih relacijskih podatkovnih baz, za katere ima orodje Hibernate podprte dialekte. V diplomskem delu je predstavljeno kako zagotoviti dialekt za relacijsko podatkovno bazo, ki nima dialekta v paketu orodja Hibernate.

7.1 Nadaljnji razvoj

Izdelek diplomskega dela ima še veliko nedokončanih funkcionalnosti, ki so:

Proces zaključevanja poti

Procesa zaključevanja poti v aplikaciji nismo implementirali, smo pa podali dobro ogrodje z uporabo dinamičnih razredov in delovanjem na večih nitih. Proces lahko implementiramo z različnimi metodami, kot so markovski modeli, skriti markovski modeli, metodo s pregledovanjem stanj ... Za proces zaključevanja bi kot avtor aplikacije predlagal uporabo markovskih modelov ali skritih markovskih modelov. Sicer primernejši od obeh bi bili skriti markovski modeli, ker so podatki klikotoka včasih nepopolni in zavajajoči. Pri uporabi obeh markovskih modelov bi bilo potrebno razviti primerno metodo učenja nad podatki klikotoka.

Analizator tipov polji

Z implementacijo te funkcionalnosti bi uporabnika razbremenili pisanja imen tipov polji, ki se nahajajo v log datoteki. Ta rešitev je uporabna predvsem za log formata NCSA in IIS, ki nimata dodatnih polji, recimo direktive formata ELF. S to funkcionalnostjo bi podprli tudi log datoteke drugih spletnih strežnikov, ki imajo podobne vzorce tipov polj, ampak imajo polja definirana z drugačnimi imeni. Primer takega spletnega strežnika je spletni strežnik nginx, ki ima iste vzorce podatkov kot spletni strežnik Apache, vendar uporablja drugačna imena polj.

Izboljšava prepoznavanja robotov

V trenutni verziji aplikacije prepoznavamo robote le preko zahteve za datoteko `robots.txt`. Robote lahko prepoznavamo tudi preko niza, ki predstavlja uporabnikovega posrednika. Kot izboljšavo predlagamo uporabo dodatnih podatkovnih baz, kjer imamo zapisane vse nize uporabniških posre-

dnikov, ki predstavljajo robote. Najučinkovitejša rešitev bi bila uporaba spletnega servisa, ki bi nam pomagal pri identifikaciji robotov. Spletni servis ima, v primerjavi z lokalno podatkovno bazo ima vedno posodobljene podatke o robotih, katerih število narašča. Tudi nizi uporabniških posrednikov robotov, ki že obstajajo, se lahko spremenijo, ter nam stari podatki v lokalni podatkovni bazi ne identificirajo takega robota.

Tukaj smo zajeli samo nekatere bistvenejše probleme oziroma izboljšave, ki bi jih bilo potrebno še dodatno izdelati. To niso edine izboljšave, ki jih aplikacija lahko prejme. Med neomenjenimi so tudi izdelava uporabniškega vmesnika, transformacija aplikacije v spletno aplikacijo, implementacija splošnejših metod z uporabo odsevnega programskega vmesnika ...

Literatura

[1] Apache module mod_log_config. [Online]. Dosegljivo: http://httpd.apache.org/docs/2.4/mod/mod_log_config.html#formats. [Dostopano 10. 2. 2016].

[2] Class dialect. [Online]. Dosegljivo: http://docs.jboss.org/hibernate/orm/5.0/javadocs/org/hibernate/dialect/Dialect.html. [Dostopano 10. 2. 2016].

[3] Extended log file format. [Online]. Dosegljivo: https://www.w3.org/TR/WD-logfile.html. [Dostopano 10. 2. 2016].

[4] Gradle. [Online]. Dosegljivo: http://gradle.org/. [Dostopano 10. 2. 2016].

[5] Gradle wikipedia. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Gradle. [Dostopano 10. 2. 2016].

[6] H2 database engine. [Online]. Dosegljivo: http://www.h2database.com/html/main.html. [Dostopano 10. 2. 2016].

[7] Hibernate (framework). [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Hibernate_framework. [Dostopano 10. 2. 2016].

[8] January 2016 web server survey. [Online]. Dosegljivo: http://news.netcraft.com/archives/2016/01/26/january-2016-web-server-survey.html. [Dostopano 10. 2. 2016].

[9] Javassist. [Online]. Dosegljivo: http://jboss-javassist.github.io/javassist/. [Dostopano 10. 2. 2016].

[10] Object-relational mapping. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Object-relational_mapping. [Dostopano 10. 2. 2016].

[11] Patricia trie. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/File:Patricia_trie.svg. [Dostopano 10. 2. 2016].

[12] Properties. [Online]. Dosegljivo: https://docs.oracle.com/javase/tutorial/essential/environment/properties.html. [Dostopano 10. 2. 2016].

[13] Reflection (computer programming). [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Reflection_(computer_programming). [Dostopano 10. 2. 2016].

[14] Relational databse. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Relational_database. [Dostopano 10. 2. 2016].

[15] Sqlite. [Online]. Dosegljivo: https://www.sqlite.org. [Dostopano 10. 2. 2016].

[16] Time vs. navigation orientation (session reconstruction). [Online]. Dosegljivo: https://en.wikipedia.org/wiki/File:Time_vs._Navigation_orientation_%28Session_reconstruction%29.svg. [Dostopano 10. 2. 2016].

[17] The web robots pages. [Online]. Dosegljivo: http://www.robotstxt.org/. [Dostopano 10. 2. 2016].

[18] *Internet Protocol*. Number 791 in Request for Comments. RFC Editor, September 1981.

[19] Adam Barth. *HTTP State Management Mechanism*, pages 8–9. Number 6265 in Request for Comments. RFC Editor, April 2011.

[20] Adam Barth. *HTTP State Management Mechanism*. Number 6265 in Request for Comments. RFC Editor, April 2011.

[21] Mike Belshe, Roberto Peon, and Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. Number 7540 in Request for Comments. RFC Editor, Maj 2015.

[22] Tim Berners-Lee, Dr. Larry Masinter, and Mark P. McCahill. *Uniform Resource Locators (URL)*. Number 1738 in Request for Comments. RFC Editor, December 1994.

[23] Dr. Steve E. Deering and Robert M. Hinden. *Internet Protocol Version 6 (IPv6) Addressing Architecture.* Number 3513 in Request for Comments. RFC Editor, April 2003.

[24] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* Number 2616 in Request for Comments. RFC Editor, Junij 1999.

[25] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, page 145. Number 2616 in Request for Comments. RFC Editor, Junij 1999.

[26] Roy T. Fielding. *Relative Uniform Resource Locators.* Number 1808 in Request for Comments. RFC Editor, Junij 1995.

[27] Roy T. Fielding and Julian F. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* Number 7231 in Request for Comments. RFC Editor, Junij 2014.

[28] EJB 3.0 Expert Group. *JSR 220: Enterprise JavaBeans, Version 3.0.* Maj 2006.

[29] Java Persistence 2.1 Expert Group. *JSR 338: Java Persistence API, Version 2.1.* April 2013.

[30] Andrej Hafner. *Prototip sistema za evidentiranje živine.* September 2015.

[31] Major Michael C. St. Johns. *Identification Protocol.* Number 1413 in Request for Comments. RFC Editor, Februar 1993.

[32] Kohsuke Kawaguchi. Args4j. [Online]. Dosegljivo: http://args4j.kohsuke.org/. [Dostopano 10. 2. 2016].

[33] Tim Lindholm, Frank Yellin, Gilad Bracha, and Alex Buckley. *The Java Virtual Machine Specification, Java SE 8 Edition.* Addison-Wesley Professional, 1st edition, 2014.

[34] Henrik Frystyk Nielsen, Roy T. Fielding, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.0.* Number 1945 in Request for Comments. RFC Editor, Maj 1996.

[35] Rok Plevel. *Predelava Magento sistema za upravljanje vsebin spletne trgovine v sistem za evidenco dela zaposlenih*. September 2015.

[36] Marko Poženel. *Inteligentno razpletanje sej pri izgradnji spletnega podatkovnega skladišča*. Avgust 2010.

[37] Gian-Carlo Rota. The number of partitions of a set. *The American Mathematical Monthly*, 71(5):498–504, 1964.

[38] Alexander Shvets. *Design Patterns Explained Simply*, pages 80–84. sourcemaking.com, 2015.

[39] Myra Spiliopoulou, Bamshad Mobasher, Bettina Berendt, and Miki Nakagawa. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *INFORMS J. on Computing*, 15(2):171–190, April 2003.