UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Boštjan Cigan

Izdelava grafičnega uporabniškega vmesnika za FTP strežnik vsftpd

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2011

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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

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

Št. naloge: 00131/2011
Datum: 01.09.2011

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

Kandidat: **BOŠTJAN CIGAN**

Naslov: **IZDELAVA GRAFIČNEGA UPORABNIŠKEGA VMESNIKA ZA FTP STREŽNIK VSFTPD**

THE CONSTRUCTION OF A GRAPHICAL USER INTERFACE FOR THE FTP SERVER VSFTPD

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi predstavite razvoj grafičnega uporabniškega vmesnika za FTP strežnik vsftpd. V okviru tega najprej predstavite značilnosti FTP protokola in strežnika vsftpd. Pri izdelavi vmesnika izhajajte iz zahtev in ustrezno načrtujete arhitekturo rešitve, za prikaz delovanja pa izdelajte tudi ustrezen odjemalec. Na koncu strežnik preverite z obremenitvenim testom in prikažite ugotovitve.

Mentor:

viš. pred. dr. Igor Rožanc

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Boštjan Cigan,

z vpisno številko 63060008,

sem avtor diplomskega dela z naslovom:

Izdelava grafičnega uporabniškega vmesnika za FTP strežnik vsftpd

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 16.09.2011 Podpis avtorja/-ice:

Zahvala

Ob zaključku diplomskega študija bi se rad zahvalil svojemu mentorju viš. pred. dr. Igorju Rožancu za pomoč pri diplomskem delu.

Posebna zahvala gre moji družini, ki so me podpirali in mi omogočili študij.

Za neskončno zalogo čokolad in drugih dobrin se zahvaljujem tudi teti Lilijani in babici Ani.

Zahvaljujem se tudi Anji za vso vzpodbudo in dobro voljo.

Kazalo

Seznam uporabljenih kratic in simbolov

FTP (angl.) File Transfer Protocol; protokol za prenos datoteke
PHP (angl.) PHP Hypertext Preprocessor; skriptni programski jezik
SQL (angl.) Structured Query Language; sestavljen poizvedni jezik
XML (angl.) eXtensible Markup Language; razširljiv označevalni jezik
RFC (angl.) Request for Comments; tekstovni dokumenti, ki opisujejo metode, obnašanja in inovacije interneta
PI (angl.) Protocol Interpreter; tolmač protokola
DTP (angl.) Data Transfer Process; proces prenosa podatkov
TCP (angl.) Transmission Control Protocol; zanesljiv protokol za prenos podatkov v obe smeri
SSL (angl.) Secure Sockets Layer; kriptografski protokoli za varno komunikacijo na internetu
FTPS (angl.) FTP Secure; varni FTP
RSA (angl.) Riverst Shamir Adleman algoritem; algoritem za šifriranje javnih ključev
DSA (angl.) Digital Signature Algorithm; algoritem za šifriranje javnih ključev
HTML (angl.) HyperText Markup Language; markirni jezik HTML
CSS (angl.) Cascading Style Sheets; kaskadne stilske datoteke
PAM (angl.) Pluggable authentication module; mehanizem za avtentifikacijske sheme
TLS (angl.) Transport Layer Security; kriptografski protokoli za varno komunikacijo na internetu (nadgradnja SSL protokola)

Povzetek

Cilj naloge je ustvariti grafični uporabniški vmesnik za FTP strežnik vsftpd. Za lažje razumevanje tematike smo predstavili protokol FTP in njegovo delovanje. Opisali smo zmožnosti in konfiguracijske datoteke strežnika vsftpd. Pred razvojem aplikacije smo naredili zajem zahtev in načrtali arhitekturo sistema. Za upravljanje s FTP strežnikom smo ustvarili strežnik v Javi, ki sprejema ukaze in nanje odgovarja v XML datotekah. Za prikaz vmesnika smo ustvarili odjemalec v programskem jeziku PHP, ki v ozadju generira ukaze na podlagi vnosa uporabnika in jih pošilja na strežnik. Delovanje smo testirali z obremenitvenim testom in rezultate ustrezno predstavili. V zaključku so predstavljene sklepne ugotovitve in ideje za nadalnji razvoj.

Ključne besede:

FTP, vsftpd, grafični uporabniški vmesnik, strežnik, odjemalec, SSL

Abstract

The purpose of thesis is to create a graphical user interface for the FTP server vsftpd. The FTP protocol and its functions are presented for better understanding and the capabilities and configurational files of the vsftpd server as well. Before the application development, planning of the system architecture was performed and the functional and non-functional specifications were presented thorough. For managing the FTP server we created a Java server that accepts commands and responds via XML files. We developed a client in PHP for the user interface that generates commands based on the user input and sends them to the server. A stress test was performed to test the application aswell. The conclusion presents findings and ideas for the continued development of the application.

Key words:

FTP, vsftpd, graphical user interface, server, client, SSL

Poglavje 1

Uvod

V današnjih časih si težko predstavljamo računalnike brez grafičnih uporabniških vmesnikov. Leta 1975 so inženirji podjetja Xerox prvič predstavili koncepte modernih uporabniških vmesnikov. Večina uporabnikov računalnikov se niti ne zaveda, da s klikanjem po vmesnikih ti izvajajo konzolne ukaze (recimo uporaba delete tipke v raziskovalcu sproži ukaz remove z imenom datoteke).

V naši nalogi smo se lotili izdelave grafičnega uporabniškega vmesnika za FTP strežnik vsftpd. V diplomskem delu smo najprej predstavili FTP protokol in se na kratko poglobili v njegovo zgodovino. Opisali smo delovanje, ukaze in sestavo samega protokola. V nadaljevanju smo se poglobili v FTP strežnik vsftpd in njegove ukaze, sestavo konfiguracijskih datotek in zapisnikov. Pregledali smo tudi obstoječe rešitve grafičnih uporabniških vmesnikov.

V tretjem poglavju smo predstavili razvoj naše rešitve, funkcionalne in nefunkcionalne zahteve naše aplikacije. Načrtali smo ustrezne diagrame in opisali razvoj odjemalca, strežnika in samo arhitekturo sistema.

V četrtem poglavju smo analizirali našo aplikacijo in jo testirali pod obremenitvijo. Predstavili pa smo tudi možne izboljšave v nadalnjem razvoju.

V zadnjem poglavju so predstavljene sklepne ugotovitve diplomskega dela.

Poglavje 2

FTP in vsftpd

2.1 Protokol FTP

2.1.1 Zgodovina protokola

File Transfer Protocol[5] (v nadaljevanju FTP) spada med najbolj znane protokole. Avtor je Abhay Bhushan, specifikacije pa je objavil v dokumentu RFC 114, 16. aprila 1971, pred nastankom TCP in IP.

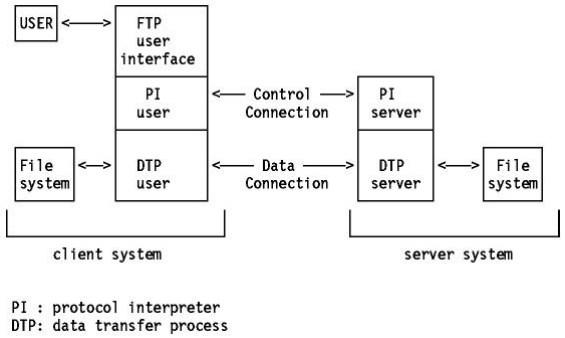
Zasnovan je bil za okolje, v kateremu odjemalci in strežniki med seboj komunicirajo s čimmanj omejitvami. Poleg tega je bilo načrtovano, da deluje preko komunikacijskih kanalov, kjer paketi potujejo neposredno do cilja. Osnovne specifikacije se nahajajo v dokumentu RFC 959 (Oktober 1985), nadgradnje pa so objavljene v dokumentih RFC 2228 (varnostne razširitve) in RFC 2428 (podpora IPv6).

Danes se pretežno uporablja v navezi s spletnimi strežniki, saj lahko uporabniki s pomočjo njega preprosto nalagajo celotne internetne strani na spletne strežnike in s tem posodabljajo vsebino.

2.1.2 Shema protokola

FTP zaradi zagotavljanja zanesljive povezave uporablja TCP kot transportni protokol. Uporabljata se dve povezavi; prva je kontrolna (ang. control connection), druga pa podatkovna (ang. data connection) (uravnava prenos podatkov). Vzpostavitelj kontrolne povezave prevzame funkcije odjemalca, vlogo strežnika pa zagotavlja oddaljen gostitelj.

Na obeh straneh (odjemalec in strežnik) je povezava sestavljena iz tolmača protokola (ang. protocol interpreter - PI) in iz procesa prenosa podatkov (ang. data transfer process - DTP), na odjemalčevi strani pa obstaja tudi uporabniški vmesnik (na sliki 2.1).



Slika 2.1: Shema FTP protokola.

Uporabniški vmesnik komunicira s tolmačem, ki nadzoruje kontrolno povezavo, tolmač pa pošilja odzive kontrolnemu protokolu in upravlja s podatkovno povezavo. Med prenosom datotek s podatki upravljajo DTP-ji.

2.1.3 Načini povezovanja

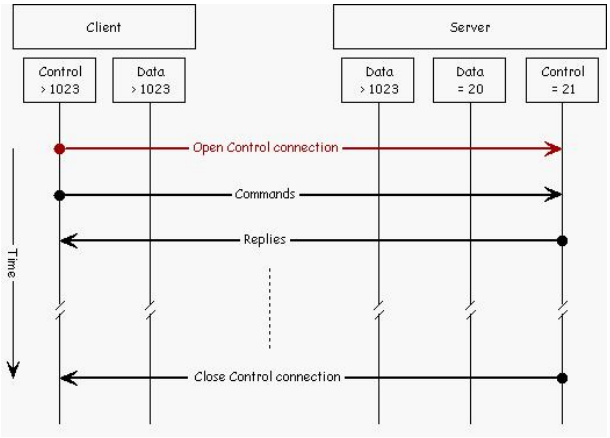
FTP je kompleksen protokol, ker uporablja kontrolno povezavo (primarna) in podatkovno povezavo (sekundarna). Obstajata dva načina vzpostavitve podatkovne povezave:

- normalni oziroma aktivni način (PORT) in
- pasivni način (PASV)

2.1.3.1 Kontrolna povezava

Kontrolna povezava[5] je komunikacijska pot med odjemalčevim tolmačem in strežniškim tolmačem. Odgovorna je za izmenjavo ukazov in odzivov. Povezava deluje podobno kot Telnet protokol.

Pred prenosom datotek na strežnik mora odjemalec vzpostaviti kontrolno povezavo s strežnikom. Odjemalec naredi TCP povezavo iz naključnih vrat N (N večji od 1023) do strežniških vrat s številko 21 (prikaz podatkovnega toka je viden na sliki 2.2).



Slika 2.2: Podatkovni tok kontrolne povezave.

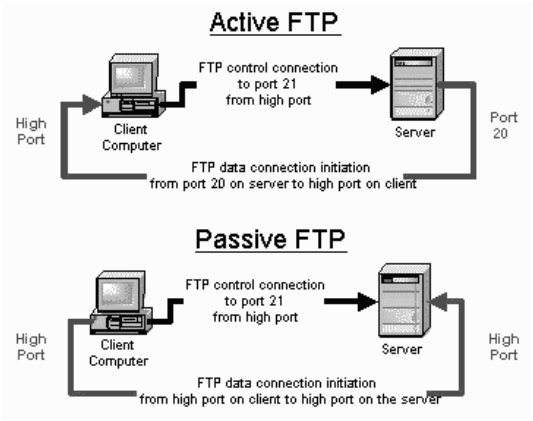
Protokol zahteva, da je kontrolna povezava med prenosom podatkov ves čas aktivna. Podatkovna povezava ne more obstajati brez odprte kontrolne povezave. Zapiranje kontrolne povezave je odgovornost uporabnika, vendar povezavo zapre strežnik na zahtevo uporabnika.[5]

2.1.3.2 Podatkovna povezava

Podatkovna povezava[5] je komunikacijska pot med odjemalčevim DTP-jem (ang. data transfer process) in med strežniškim DTP-jem, ki je namenjena izmenjavi podatkov (datotek, seznamov). Odvisno od izbranega načina FTP se je podatkovna povezava začela na strežniku (aktivni način) ali na odjemalcu (pasivni način).

V aktivnemu načinu (slika 2.3) odjemalec pošlje `PORT` ukaz strežniku, iz katerega strežnik, izve na katerega odjemalca (IP naslov) in številko vrat (nerezervirana vrata večja od 1023) se mora povezati za vzpostavitev podatkovne povezave. Po sprejetju `PORT` ukaza strežnik vzpostavi podatkovno povezavo iz svojih lokalnih vrat 20 na IP naslov in številko vrat, ki jih je izvedel iz `PORT`

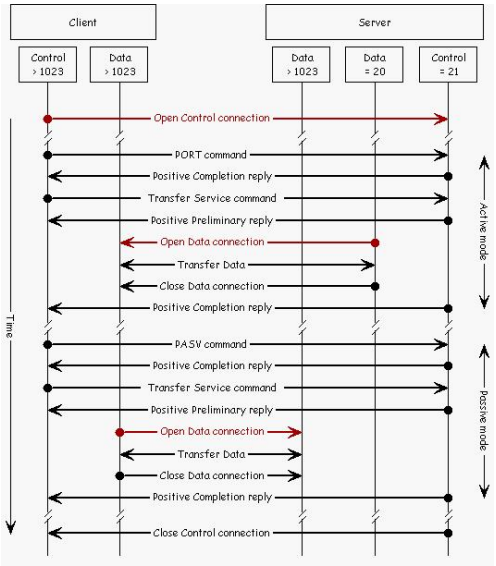
ukaza.



Slika 2.3: Aktivni in pasivni način vzpostavljanja FTP povezave.

V pasivnemu načinu (vidno na sliki 2.3) odjemalec pošlje ukaz `PASV` strežniku, ki od strežnika zahteva naj čaka na povezavo in posluša na določenih podatkovnih vratih (ki niso njegova privzeta na 20). Če strežnik podpira pasivni način, pošlje nazaj odziv z IP naslovom in številko vrat (nerezervirana vrata večja od 1023), na katerih strežnik posluša. Odjemalec potem vzpostavi podatkovno povezavo iz naključnih lokalnih nerezerviranih vrat (večja od 1023) na IP naslov in številko vrat, ki jih je izvedel iz odziva na ukaz `PASV`.[5]

Podatkovna povezava[5] se bo vzpostavila le po prejemu odzivov na ukaze `LIST`, `RETR`, `STOR`, zato se mora način povezave določiti pred pošiljanjem ukazov. Podatkovni ukazi dobijo več odzivov; prvi je pozitiven uvodni odgovor, drugi pa pozitivni zaključni odgovor (prikaz podatkovnega toka je viden na sliki 2.4).



Slika 2.4: Podatkovni tok podatkovne povezave.

2.1.4 FTP ukazi

Komunikacija med odjemalcem in strežnikom poteka s pomočjo vnaprej določenih ukazov. Najbolj uporabljani ukazi so:[4]

- `USER` - uporablja se za pošiljanje uporabniškega imena strežniku,
- `PASS` (password) - uporablja se za pošiljanje gesla strežniku,
- `PASV` (passive) - prehod v pasivni način delovanja,
- `STOR` (store) - shrani datoteko na strežnik,
- `DELE` (delete) - izbriši datoteko iz strežnika,
- `MKD` (make directory) - ustvari direktorij na strežniku,
- `RMD` (remove directory) - izbriši direktorij na strežniku,
- `RNFR` (rename from) - določi datoteko ali direktorij, ki se bo preimenoval,
- `RNTO` (rename to) - določi, v kaj se bo datoteka ali direktorij iz ukaza RNFR preimenovala.

Ostali ukazi in njihove razlage so pojasnjeni v dodatku A.

2.2 FTP strežnik vsftpd

Very secure FTP daemon (v nadaljevanju vsftpd) je FTP strežnik za Unix sisteme (tudi za Linux).[6] Podpira IPv6, SSL in FTPS. Vsftpd je privzeti strežnik v namestitvi operacijskih sistemov Ubuntu[20], CentOS[21], Fedora[23], NimbleX[22] in RHEL[24].

Čeprav vsftpd ne zasede veliko prostora, ponuja vse, kar ponujajo večje rešitve, vendar z večjo varnostjo. Nekatere izmed zmožnosti so:

- podpora za IPv6,
- enkripcija skozi SSL,
- upravljanje s pasovno širino,
- podpora virtualnim uporabnikom in
- možnost nastavljanja nastavitev za vsakega uporabnika.

2.2.1 Uporaba

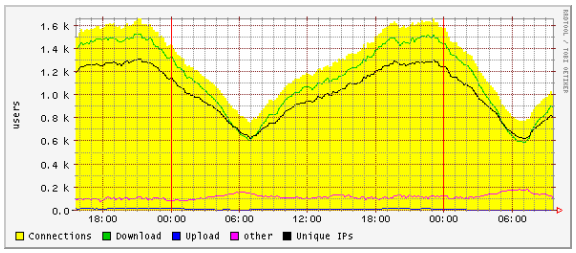
Zaradi same zmogljivosti, varnosti in stabilnosti vsftpd za svoj primarni strežnik uporablja več znanih spletnih strani kot so stran Redhat, Suse, Gimp, Kde in podobne.[6] Seznam večjih strani:

- ftp.redhat.com,
- ftp.suse.com,
- ftp.debian.org,
- ftp.freebsd.org,
- ftp.gnu.org,
- ftp.gnome.org,
- ftp.kde.org,
- ftp.kernel.org,

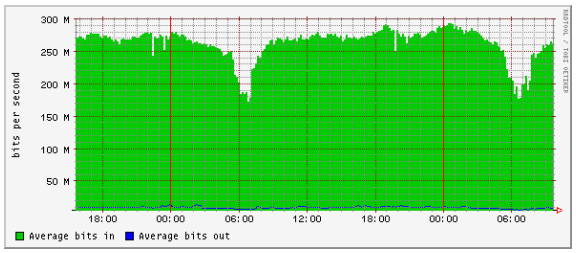
- rpmfind.net,
- ftp.linux.org.uk,
- ftp.gimp.org.

2.2.2 Test zmogljivosti

Testi zmogljivosti so pokazali, da strežnik vsftpd zmore naenkrat streči več kot 1500 uporabnikom in da je zmožen prenesti do tri terabajte ali več podatkov v enemu dnevu. Slika 2.5 prikazuje število trenutno povezanih uporabnikov po urah in njihove akcije, slika 2.6 pa povprečno število prenesenih bitov na sekundo.



Slika 2.5: Število trenutno povezanih uporabnikov na vsftpd in njihove akcije.



Slika 2.6: Število prenesenih bitov na sekundo v vsftpd.

2.2.3 Namestitev in konfiguracija strežnika

Namestitev strežnika je zelo enostavna. Na operacijskemu sistemu Ubuntu lahko uporabimo kar ukaz:

```
1 bostjan@ZeroNETPC3:~$ sudo apt-get install vsftpd
```

Če želimo namestitev opraviti sami, si pa lahko paket prenesemo iz uradne strani.

```
1 bostjan@ZeroNETPC3:~$ wget https://security.appspot.
    com/downloads/vsftpd-2.3.4.tar.gz
2 bostjan@ZeroNETPC3:~$ tar -xvzf vsftpd-2.3.4.tar.gz
3 bostjan@ZeroNETPC3:~$ ./configure && make && make
    install
```

Po namestitvi strežnik zaženemo na sledeči način:

```
1 bostjan@ZeroNETPC3:~$ sudo /etc/init.d/vsftpd start
```

2.2.3.1 Konfiguracijska datoteka

Pred dejanskim zagonom vsftpd, program interpretira ukaze iz konfiguracijske datoteke. Ta se običajno nahaja v `/etc/vsftpd.conf`.

Sestava datoteke je zelo preprosta. Vsaka vrstica lahko vsebuje komentar ali ukaz. Komentarji so označeni z # in so med izvajanjem prezrti. Ukazi so sestavljeni iz dveh delov; `ime_ukaza=vrednost`. Med znakom = in vrednostjo ukaza ne sme biti presledka, ker program sicer pri izvajanju javi napako.

Ukazi so sestavljeni iz treh sklopov:

- izbira tipa boolean (nastavljene na true ali false),
- numerični izbor in
- nastavljanje na nize.

Vsi ukazi, ki so zapisani v glavni datoteki, veljajo globalno; to pomeni, da te nastavitve veljajo za vse uporabnike (tudi virtualne če so omogočeni), razen če se nastavitve v posameznih konfiguracijskih datotekah uporabnikov napišejo na novo.

Program prihaja s predhodno nastavljeno datoteko, ki vsebuje nastavitve za lokalne uporabnike (vsi Unix uporabniki, ki bodo dodani v skupino FTP, se bodo lahko prijavljali) in ima vključeno anonimno prijavljanje. Celotna datoteka na začetku namestitve izgleda takole:

```
1  # Example config file /etc/vsftpd.conf
2  write_enable=YES
3  dirmessage_enable=YES
4  ftpd_banner="Welcome␣to␣thepatch␣FTP␣service."
5  local_enable=YES
6  local_umask=022
7  chroot_local_user=YES
8  anonymous_enable=YES
9  anon_upload_enable=YES
10 anon_umask=022
11 anon_mkdir_write_enable=YES
12 anon_other_write_enable=YES
13 syslog_enable=YES
14 connect_from_port_20=YES
15 ascii_upload_enable=YES
16 ascii_download_enable=YES
17 pam_service_name=vsftpd
18 listen=no
```

Za naše potrebe se bomo lotili le razlage najpomembnejših ukazov, ki so potrebni za delovanje strežnika. Ostali ukazi in njihove razlage so navedene v dodatku B.

Z vrstico 18 strežniku povemo, da ne upravlja sam s prihajajočimi povezavami. Vrstica 8 pove, da dovoljujemo prijave anonimnim uporabnikom (privzeta sta ftp in anonymous), vrstica 2 dovoljuje vsem uporabljanje ukazov `STOR`, `DELE`, `RNFR`, `RNTO`, `MKD`, `RMD`, `APPE` in `SITE`. Vrstica 13 omogoči pisanje zapisnikov strežnika (beleženje akcij vseh uporabnikov na strežniku).

2.2.3.2 Konfiguracijske datoteke uporabnikov

Če želimo za vsakega uporabnika nastaviti drugačne nastavitve, nam vsftpd to omogoča z ukazom `user_config_dir`, ki programu pove, kje se nahajajo posamezne konfiguracijske datoteke za vsakega uporabnika.

Potem lahko v mapi, ki smo jo določili, ustvarimo datoteko (brez končnice) z imenom uporabnika. V tej datoteki lahko na novo definiramo ali pa dodamo

nova pravila za tega določenega uporabnika. Če bi hoteli ustvariti uporabnika, ki ima domačo mapo v `/home/ftp/uporabnik` in nima pravice do zapisovanja datotek, bi v datoteko zapisali:

```
1 # Primer uporabnika, ki se nahaja v direktoriju /home
     /ftp/uporabnik in nima pravice do zapisovanja
2 local_root=/home/ftp/uporabnik
3 write_enable=NO
```

Za omejevanje pravic uporabniku lahko uporabimo že standardizirane FTP ukaze, ki jih napišemo v vrstico `cmds_denied`. Če želimo uporabniku preprečiti preimenovanje datotek na strežniku, v njegovo konfiguracijsko datoteko zapišemo `cmds_denied=RNTO,RNFR`. Obraten ukaz, ki uporabniku dovoljuje omejen nabor ukazov, se imenuje `cmds_allowed`. Primer uporabnika, ki lahko prenaša datoteke iz strežnika, ne sme jih pa nalagati in brisati:

```
1 # Primer uporabnika, ki se nahaja v direktoriju /home
     /ftp/uporabnik in nima pravice do nalaganja in
     brisanja datotek
2 local_root=/home/ftp/uporabnik
3 cmds_denied=DELE,RMD,STOR,MKD
```

Za blokiranje uporabnikov nam uporabnika ni potrebno izbrisati, temveč lahko v konfiguracijsko datoteko le zapišemo `cmds_denied=USER,PASS`, kar odjemalcu prepoveduje uporabo `USER` in `PASS` ukaza, ki sta potrebna za prijavo na FTP strežnik.

Med zanimivejša ukaza sodita tudi `deny_file` in `hide_file`. Prvi uporabniku preprečuje nalaganje določenih datotek, drugi pa skriva datoteke pred uporabnikom.

Za preprečevanje nalaganja datotek MP3 lahko v konfiguracijsko datoteko dodamo vrstico `deny_file=*.mp3`, za skrivanje pa `hide_file=*.mp3`. Če želimo preprečiti več končnic, jih ločimo z vejico; `*.mp3,*.jpg`.

2.2.3.3 Zapisniki strežnika

Strežnik vse akcije uporabnikov zapisuje v datoteko, ki jo definiramo v konfiguracijski datoteki. Izsek iz datoteke izgleda takole:

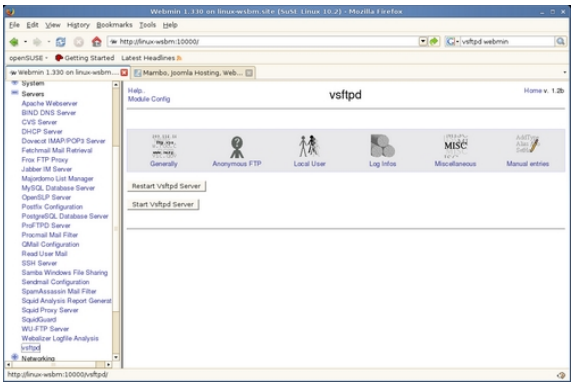
```
1  Sun Apr 24 16:37:15 2011 [pid 2] CONNECT: Client "
     127.0.0.1"
2  Sun Apr 24 16:37:15 2011 [pid 1] [zerocool] OK LOGIN:
      Client "127.0.0.1"
3  Sun Apr 24 16:37:52 2011 [pid 2] CONNECT: Client "
     127.0.0.1"
4  Sun Apr 24 16:37:52 2011 [pid 1] [zerocool] FAIL
     LOGIN: Client "127.0.0.1"
5  Sun Apr 24 16:38:06 2011 [pid 2] CONNECT: Client "
     127.0.0.1"
6  Sun Apr 24 16:38:06 2011 [pid 1] [zerocool] OK LOGIN:
      Client "127.0.0.1"
7  Sun Apr 24 16:38:15 2011 [pid 3] [zerocool] OK RENAME
     : Client "127.0.0.1", "/0612201011.jpg␣/061220111.
     jpg"
8  Sun Apr 24 16:38:28 2011 [pid 3] [zerocool] OK MKDIR:
      Client "127.0.0.1", "/New␣directory"
9  Sun Apr 24 16:38:41 2011 [pid 3] [zerocool] OK RENAME
     : Client "127.0.0.1", "/New␣directory␣/New␣director
     "
10 Sun Apr 24 16:38:57 2011 [pid 3] [zerocool] OK RENAME
     : Client "127.0.0.1", "/061220111.jpg␣/New␣director
     /061220111.jpg"
11 Sun Apr 24 16:39:19 2011 [pid 2] CONNECT: Client "
     127.0.0.1"
12 Sun Apr 24 16:39:19 2011 [pid 1] [zerocool] OK LOGIN:
      Client "127.0.0.1"
13 Sun Apr 24 16:39:19 2011 [pid 3] [zerocool] OK UPLOAD
     : Client "127.0.0.1", "/Read␣Me.txt", 3624 bytes,
     235.25Kbyte/sec
14 Sun Apr 24 16:39:35 2011 [pid 3] [zerocool] OK
     DOWNLOAD: Client "127.0.0.1", "/Read␣Me.txt", 3624
     bytes, 120.94Kbyte/sec
15 Sun Apr 24 16:37:15 2011 [pid 2] CONNECT: Client "
     127.0.0.1"
16 Sun Jul 31 04:19:08 2011 [pid 3] [zerocool] OK DELETE
     : Client "127.0.0.1", "/Read␣Me.txt"
```

Vsaka vrstica vsebuje datum, pid številko, uporabniško ime in IP naslov odjemalca. Pred vsakim ukazom imamo tudi besedici `FAIL` (zahteva ni bila uspešna) in `OK` (zahteva je bila uspešna). Pri ukazu `RENAME` dobimo še dve dodatna podatka, ime datoteke, ki jo bomo preimenovali in novo ime datoteke. Ukaza `UPLOAD` in `DOWNLOAD` imata podano ime, velikost in hitrost nalaganja ali prenašanja datoteke, `MKDIR` vsebuje ime ustvarjene mape, `DELETE` pa ime datoteke, ki smo jo izbrisali.

2.2.4 Obstoječi grafični uporabniški vmesniki

Pred izdelovanjem lastnega vmesnika smo preverili že nekaj obstoječih rešitev. Med obstoječe rešitve spadajo modul za webmin[13], vsftpd nas gui[15] in kvsftpdmanager[14].

Webmin modul ponuja le urejanje konfiguracijskih datotek, ne ponuja pa direktnega dodajanja in urejanja obstoječih uporabnikov, urejanja pravic in branja zapisnikov.

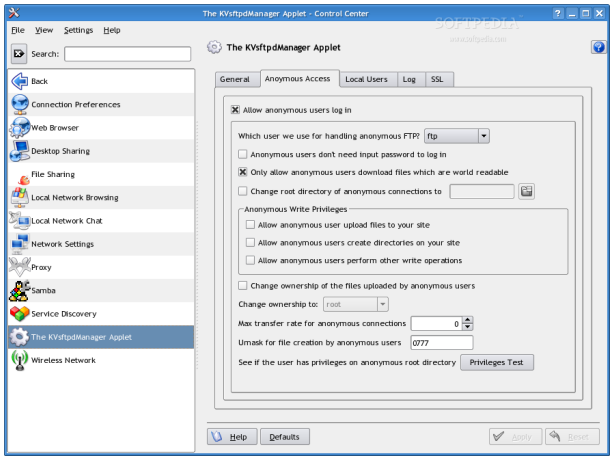


Slika 2.7: Webmin modul vsftpd.

Vsftpd nas gui in njegova celotna dokumentacija sta napisana v francoskem jeziku (kar omogoča težavnosti pri namestitvi), projekt pa ne podpira urejanja uporabniških pravic in branja zapisnikov.

Kvsftpdmanager podobno kot webmin modul ne ponuja direktnega dodajanja in urejanja obstoječih uporabnikov, ponuja le urejanje konfiguracijskih datotek.

Pri pregledu obstoječih rešitev smo ugotovili, da te ne ustrezajo našim potrebam. Zato smo se lotili razvoja lastne rešitve.



Slika 2.8: Kvsftpd manager.

Poglavje 3

Razvoj aplikacije

V poglavju bomo najprej predstavili orodja, ki smo jih uporabili pri razvoju. V nadaljevanju bomo naredili zajem zahtev, napisali funkcionalne in nefunkcionalne zahteve aplikacije ter predstavili samo arhitekturo sistema. Na koncu poglavja smo predstavili izvedbo našega načrta.

3.1 Orodja

3.1.1 Netbeans

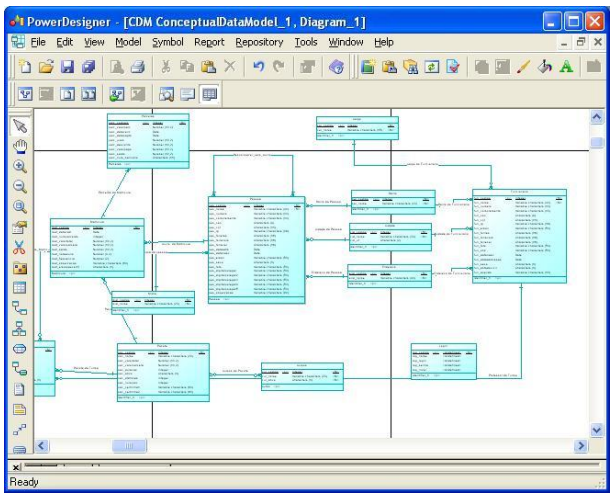
Netbeans[16] je integrirano razvojno okolje, ki omogoča izdelovanje aplikacij v jezikih Java, Javascript, PHP, Python, Groovy, C, C++, Scala, Clojure in drugih. Razvit je bil v programskemu jeziku Java, zato lahko teče kjerkoli, kjer je nameščen JVM (virtualni stroj Java). Za hitrejši razvoj programerjem omogoča:

- avtomatsko generiranje kode,
- avtomatsko dopolnjevanje kode,
- preverjanje sintaktičnih napak,
- takojšen dostop do dokumentacije.

3.1.2 Sybase PowerDesigner

PowerDesigner[8] je orodje, ki se uporablja pri modeliranju sistemov. Razvilo ga je podjetje Sybase, njegov tržni delež pa dosega 39%. Na voljo je za

operacijski sistem Microsoft Windows, možna pa je tudi uporaba v razvojnem orodju Eclipse (kot vtičnik).



Slika 3.1: Orodje Sybase PowerDesigner.

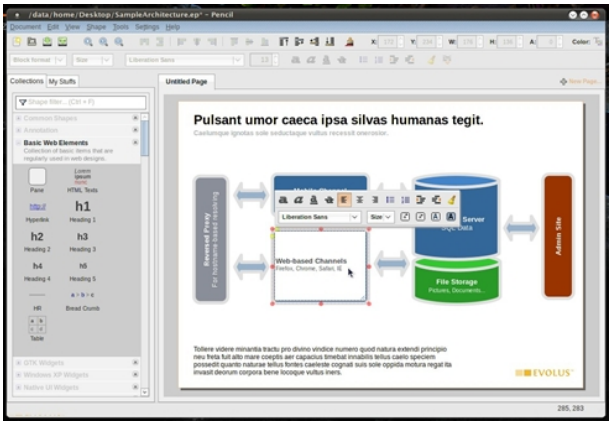
V razvoju smo orodje uporabili pri načrtovanju diagrama primerov uporabe in diagrama hierarhije funkcij.

3.1.3 Pencil

Pencil[7] je odprtokodno orodje (licenca GPL), namenjeno izdelovanju prototipov uporabniških vmesnikov in izdelavi različnih diagramov. Na voljo je v samostojni različici ali kot dodatku za brskalnik Firefox. Trenutno je na voljo v verziji 1.2 in omogoča:

- izdelavo diagramov in prototipov uporabniških vmesnikov,
- izvoz v HTML, PNG, PDF, OpenOffice dokumente,
- dodajanje zunanjih objektov.

V razvoju smo orodje uporabili pri načrtovanju uporabniškega vmesnika, izdelavo prototipov in XML diagramov.



Slika 3.2: Orodje Pencil.

3.2 Zajem zahtev

Z analizo obstoječih rešitev smo ugotovili, da večina aplikacij, ki omogoča delo s strežnikom vsftpd, ne obstaja več ali pa ponujajo premalo opcij uporabniku. Zato smo se odločili za izdelavo lastne rešitve, ki večino dela postori sama, hkrati pa z ustrezno dokumentacijo drugim programerjem omogoča izdelavo vmesnikov v drugih programskih jezikih.

3.2.1 Predpogoji

Za uporabo aplikacije vsftpdG je potrebna izpolnitev naslednjih predpogojev:

- nameščen operacijski sistem Ubuntu Server ali drugi,
- za poganjanje strežnika nameščena Java 1.5 (ali novejša) in podatkovna baza MySQL 5.1 (ali novejša),
- za poganjanje odjemalca nameščen spletni strežnik Apache ali Nginx s PHP podporo (5.1 ali višja),
- poznavanje osnovne uporabe terminala in
- poznavanje osnovne uporabe routerja.

3.2.2 Funkcionalne zahteve

Aplikacija vsftpdG bo uporabljala princip strežnik-odjemalec, zato so naloge tudi ustrezno razdeljene.

3.2.2.1 Strežnik

Strežnik predstavlja osrčje programa, ki sprejema ukaze odjemalca in izvaja ustrezne akcije.

1. **Prijava uporabnika**

   - preveri, če uporabniško ime obstaja,
   - preveri, če se geslo ujema z geslom v podatkovni bazi,
   - preveri, če je uporabnik presegel število nepravilnih prijav (omejitev so tri prijave na trideset minut),
   - pridobi vlogo uporabnika (administrator, uporabnik),
   - preveri če je uporabnik blokiran,
   - vzpostavi sejo.

2. **Urejanje uporabnikov**

   - pridobivanje seznama uporabnikov,
   - brisanje uporabnikov,
   - urejanje uporabniških pravic.

3. **Ustvarjanje statistike FTP strežnika**

   - branje dnevnikov FTP strežnika,
   - zapisovanje vseh uspešno in neuspešno izvedenih ukazov strežnika v podatkovno bazo.

4. **Vzdrževanje vzpostavljenih sej**
   Brišejo se seje, pri katerih je čas od preteka zadnje aktivnosti večji od dvajset minut.

5. **Urejanje IP naslovov**

- blokiranje dostopa do FTP strežnika določenemu IP naslovu,
- deblokiranje dostopa do FTP strežnika določenemu IP naslovu.

6. **Odziv strežnika odjemalcu**
   Vsi odzivi strežnika se vračajo v formatu XML.
7. **Avtentifikacija odjemalca**
   - če odjemalec teče na strežniku in ne kot samostojna aplikacija, se mora odjemalec registrirati s pomočjo api ključa,
   - generiranje api ključa.

3.2.2.2 Odjemalec

Odjemalec je grafični uporabniški vmesnik, ki omogoča prijave uporabnikom in v ozadju izdeluje ukaze in jih pošilja strežniku v izvedbo.

1. **Prijava uporabnika**
   - pošlji podatke za prijavo na strežnik,
   - preberi odziv strežnika.
2. **Avtentifikacija odjemalca**
   - če odjemalec ne deluje kot samostojna aplikacija, se mora registrirati s pomočjo api ključa.
3. **Nadaljevanje seje uporabnika**
   - pošlji podatke strežniku za nadaljevanje seje,
   - preberi odziv strežnika.
4. **Pošiljanje ukazov na strežnik**
   - pošiljanje ukaza za dodajanje, urejanje, brisanje uporabnika,
   - pošiljanje ukaza za zaustavitev strežnika,
   - pošiljanje ukaza za blokiranje ali deblokiranje uporabnikov.
5. **Branje odzivov strežnika**
   - branje XML odzivov,
   - predstavitev podatkov v uporabniku razumljivi obliki.

3.2.2.3 Ostalo

1. **Avtentifikacija FTP uporabnikov**
    - za avtentifikacijo FTP uporabnikov vsftpd ustrezno priredimo za uporabo virtualnih uporabnikov,
    - virtualni uporabniki se hranijo v podatkovni bazi MySQL.

3.2.3 Nefunkcionalne zahteve

Varnost

Zaradi ustvarjanja FTP računov se zahteva visoka varnost. Potrebno je beleženje vseh akcij uporabnikov. Administrativni vmesnik omogoča dodajanje, urejanje in brisanje uporabnikov, zato mora biti ustrezno zaščiten pred vdori.

Dosegljivost in odzivnost

Zaradi ustvarjanja novih računov mora aplikacija biti vedno dosegljiva in odzivna, možna so le odstopanja v vrednosti enega procenta ali manj (ponovno zaganjanje strežnika in redna servisiranja).

Sočasni dostop

FTP strežnik vsftpd je zmožen sočasno servisirati 1500 uporabnikom ali več, zato mora naša aplikacija tudi ustrezno servisirati toliko uporabnikom ali več.

3.2.4 Identifikacija uporabnikov

Program bo sočasno uporabljalo več uporabnikov, glede nivoja dostopa pa ločimo le dva, administratorja in uporabnika. Potrebna je izdelava sistema za upravljanje privilegijev.

Administrator

Administrator ima vse pravice do sistema. Lahko dodaja, ureja, briše in blokira uporabnike ter njihove pravice. Ima vpogled v trenutno vzpostavljene seje in vse dnevnike FTP strežnika. Dodeljeno mu je blokiranje in deblokiranje IP naslovov ter prižiganje in ugašanje FTP strežnika, omogočeno pa mu je tudi

nalaganje datotek.

Uporabnik

Uporabnik lahko pogleda svoje dnevnike, nalaga datoteke v sistem in spremeni svojo geslo za dostop.

3.3 Izvedba rešitve

Samo izvedbo rešitve smo razdelili na štiri sklope. V prvem sklopu smo načrtovali, v drugem sklopu smo izdelali strežnik, v tretjem smo izdelali odjemalec in GUV, v četrtem pa smo ustvarili namestitveno skripto za poenostavitev same namestitve programa.

3.3.1 Načrtovanje

3.3.1.1 Arhitektura aplikacije

Aplikacija bo delovala na principu strežnik-odjemalec (osnovna shema sistema je prikazana na sliki 3.3). Strežnik bo namenjen sprejemanju in izvajanju ukazov, ki jih bo pošiljal odjemalec (podobno kot telnet). Prednosti takšne arhitekture so:

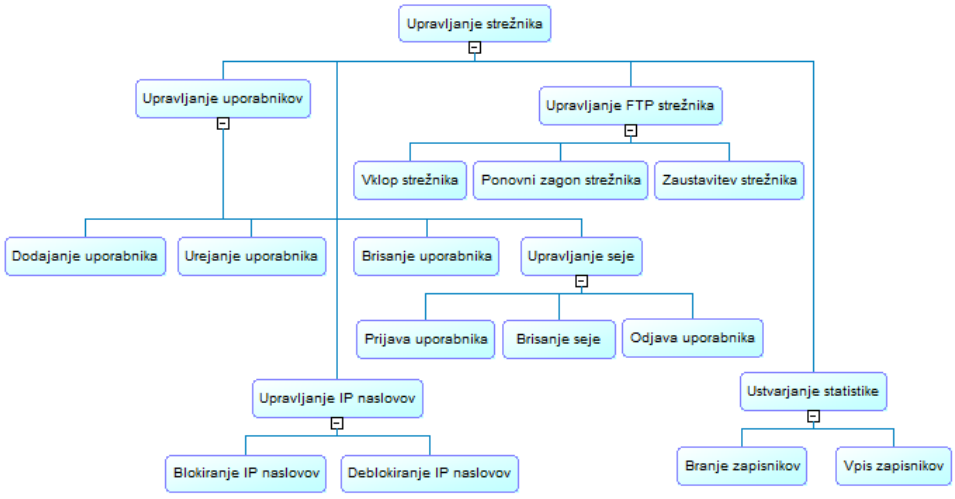
- avtonomija podatkovne baze,
- večja varnost in
- neodvisnost grafičnega uporabniškega vmesnika.

Za arhitekturo strežnika in odjemalca smo se odločili zaradi omejitev programskega jezika PHP. PHP omogoča izvajanje sistemskih ukazov, vendar le na ravni Apache strežnika (če se npr. korenska mapa strežnika nahaja na `/var/www`, lahko PHP manipulira le z datotekami, ki se nahajajo v tej mapi in njenih podmapah). Če bi PHP-ju omogočili dostop do celotnega sistema, bi s tem odprli veliko potencialnih groženj in varnostnih lukenj, saj lahko kdorkoli, ki lahko nalaga skripte na strežnik, naloži tudi zlonamerno kodo. S to arhitekturo tudi uspešno onemogočimo SQL vrivanje, saj odjemalec nima neposrednega dostopa do podatkovne baze.

Slika 3.3: Arhitektura sistema.

3.3.1.2 Diagram hierarhije funkcij

Diagram hierarhije funkcij[2] nam služi za prikaz razdelitve funkcionalnosti. Vsaka hierarhična struktura se začne na vrhu z eno samo vseobsegajočo enoto (koren strukture) in se nadaljuje vse do listov, ki predstavljajo elementarne funkcionalnosti. Na sliki 3.4 je prikazan diagram hierarhije funkcij našega sistema.



Slika 3.4: Diagram hierarhije funkcij sistema.

3.3.1.3 Diagram primerov uporabe

Z modelom primerov uporabe dokumentiramo obnašanje obravnavanega sistema. Ta vsebuje funkcije sistema (primeri uporabe), njegovo okolje (akterje) in odnose med primeri uporabe in akterji (diagram primerov uporabe). Najvažnejša vloga modela je komunikacija; omogoča izmenjavo mnenj o funkcionalnosti in obnašanju sistema med strankami oziroma končnimi uporabniki in tistimi, ki ga načrtujejo.[2] Na sliki 3.5 je prikazan diagram primerov uporabe sistema.

Slika 3.5: Diagram primerov uporabe sistema.

Uporabnik

Uporabnik je glavni akter našega sistema. Na voljo ima naslednje funkcionalnosti:

- prijavo v sistem - uporabnik, ki se nahaja v sistemu, mora za dostop poznati svoje uporabniško ime in geslo, ki ju je določil administrator,
- odjavo iz sistema - uporabnik se lahko kadarkoli odjavi iz sistema,
- ogled svoje statistike - uporabnik lahko pregleduje svojo statistiko (nedavno naložene datoteke, hitrost prenosa ipd.),
- menjavo gesla - uporabnik lahko zamenja svoje geslo,
- nadaljevanje seje - uporabnik lahko nadaljuje že obstoječo sejo, če se IP naslov ujema in seja ni potekla.

Administrator

Administrator ima pooblastila za vse operacije sistema. Poleg primerov uporabe navadnega uporabnika ima še naslednje možnosti:

- ogled statistike vseh uporabnikov - administrator lahko pogleda statistiko vseh uporabnikov,
- dodajanje uporabnika - administrator lahko doda uporabnika (če ta še ne obstaja),
- urejanje uporabnika - administrator lahko ureja uporabnika (pravice ipd.),
- brisanje uporabnikov - administrator lahko izbriše uporabnika in določi, če njegove datoteke ostanejo na sistemu,
- blokiranje uporabnika,
- prižig in ugašanje ftp strežnika,
- blokiranje ip naslovov,

- urejanje sej.

Strežnik

Strežnik izvaja vse operacije administratorja in hkrati dela tudi:

- prebiranje in obdelava zapisnikov - branje zapisnikov strežnika in zapis v podatkovno bazo,
- vzdrževanje sej - brisanje sej, ki niso aktivne,
- prebiranje in obdelava ukazov.

3.3.1.4 Podatkovni model

Podatkovni model je povezana zbirka konceptov, namenjenih opisovanju in manipulaciji s podatki[9]. V modelu smo definirali osnovne in podporne entitete. Podatkovni model na sliki 3.6 prikazuje entitetno relacijski diagram (v nadaljevanju ERD) podatkovne baze.

Osnovne entitete

ERD vsebuje naslednje entitete:

1. **Uporabniki** (`USERS`)
   - hrani seznam uporabnikov,
   - gesla v šifrirani obliki,
   - privilegije uporabnika in status uporabniškega računa.
2. **Seje** (`AUTHENTICATIONS`)
   - hrani trenutno vzpostavljene seje v sistemu,
   - čas prijave,
   - IP naslov, iz katerega je prišla zahteva,
   - število poskusov prijave,
   - čas zadnje aktivnosti uporabnika.

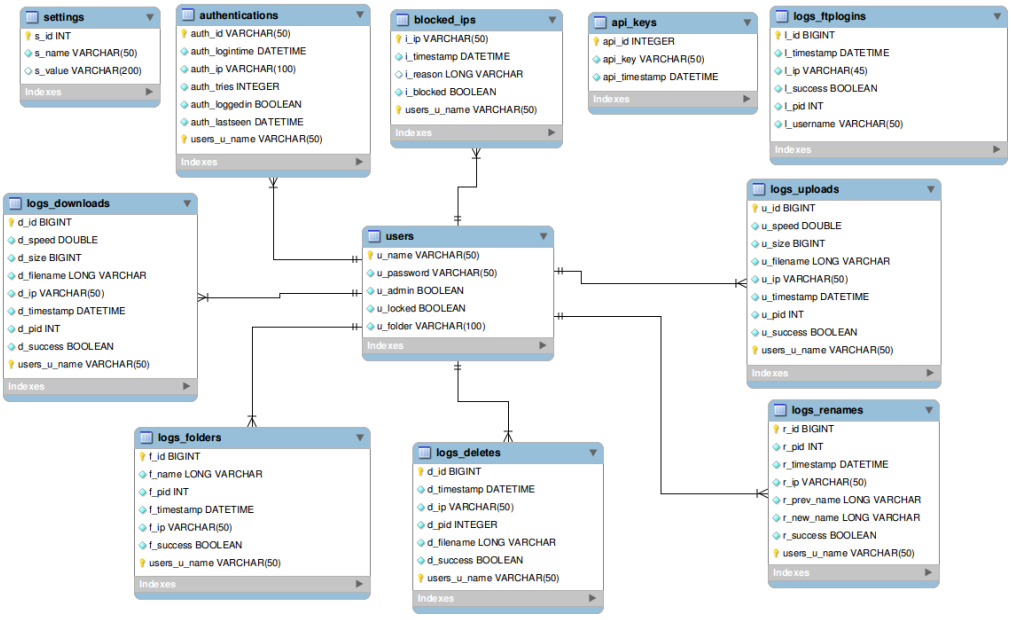
3. **Blokirani IP naslovi** (`BLOCKED_IPS`)
    - hrani blokiran IP naslov,
    - status blokade,
    - razlog in čas blokade.
4. **Zapisnik prijav** (`LOGS_FTPLOGINS`)
    - hrani čas prijave,
    - IP, iz katerega je prišla prijava,
    - uspešnost prijave, pid procesa in uporabniško ime.
5. **Zapisnik naloženih dokumentov** (`LOGS_UPLOADS`)
    - hrani hitrost nalaganja, velikost naloženega dokumenta,
    - ime dokumenta,
    - IP naslov,
    - čas nalaganja,
    - uspešnost nalganja in pid procesa.
6. **Zapisnik preimenovanih dokumentov** (`LOGS_RENAMES`)
    - hrani pid procesa, uspešnost preimenovanja,
    - čas zahteve,
    - IP naslov,
    - predhodno in novo ime datoteke.
7. **Zapisnik izbrisanih dokumentov** (`LOGS_DELETES`)
    - hrani čas izbrisa, pid procesa,
    - ime izbrisane datoteke,
    - uspešnost procesa in IP naslov iz katerega prihaja zahteva.
8. **Zapisnik ustvarjenih map** (`LOGS_FOLDERS`)
    - hrani ime ustvarjene mape,
    - čas, uspešnost in IP naslov zahteve,
    - pid procesa.

9. **Zapisnik naloženih datotek** (`LOGS_DOWNLOADS`)
    - hrani ime in velikost naložene datoteke,
    - hitrost nalaganja,
    - čas in uspešnost nalaganja datoteke,
    - IP naslov in pid procesa zahteve.

Podporne entitete

Poleg osnovnih entitet so za pravilno delovanje aplikacije potrebne še naslednje podporne entitete:

1. **Nastavitve** (`SETTINGS`) V entiteti se hranijo vse nastavitve aplikacije.
2. **API ključi** (`API_KEYS`) V entiteti se hranijo API ključi za avtentifikacijo odjemalcev

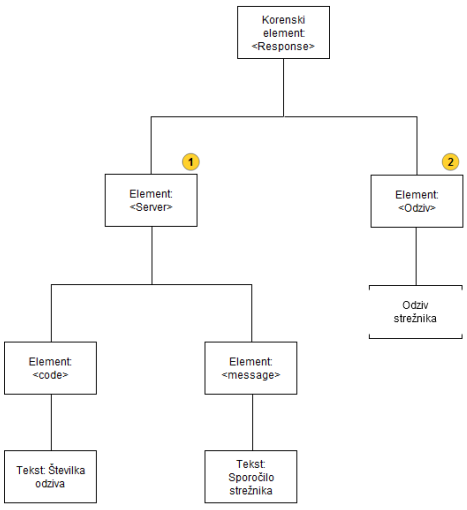


Slika 3.6: Podatkovni model vsftpdG.

3.3.1.5 Diagrami XML odzivov

Strežnik bo odjemalcu odgovarjal v formatu XML. Osnoven diagram odziva strežnika je prikazan na sliki 3.7. XML datoteka je sestavljena iz korenskega elementa `Response`. Njegova otroka sta `Server` in `Odziv`. Element `Server` vsebuje dva otroka, v elementu `code` je zapisana unikatna številka (npr. številka 101 pomeni uspešno prijavo), element `message` pa vsebuje sporočilo. Element `Odziv` vsebuje dodatne podatke, pojavi pa se le pri pridobivanju podatkov (seznam uporabnikov, statistika, podatki uporabnika).

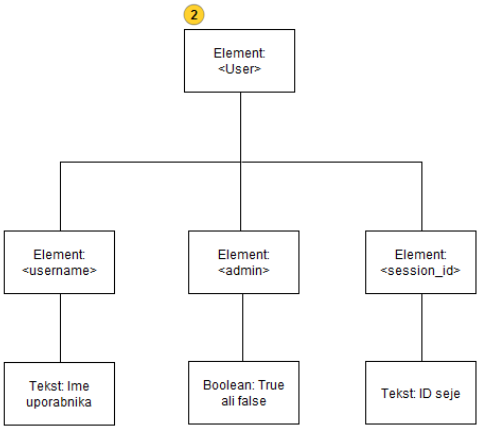
Na sliki 3.8 je prikazan XML diagram za uspešno prijavo uporabnika. Element `User` v tem primeru postane element `Odziv` na sliki 3.7, njegovi otroci pa so `username` (uporabniško ime), `admin` (boolean spremenljivka če je uporabnik administrator) in `session_id` (identifikacijska številka vzpostavljene seje).



Slika 3.7: Diagram XML odziva strežnika.

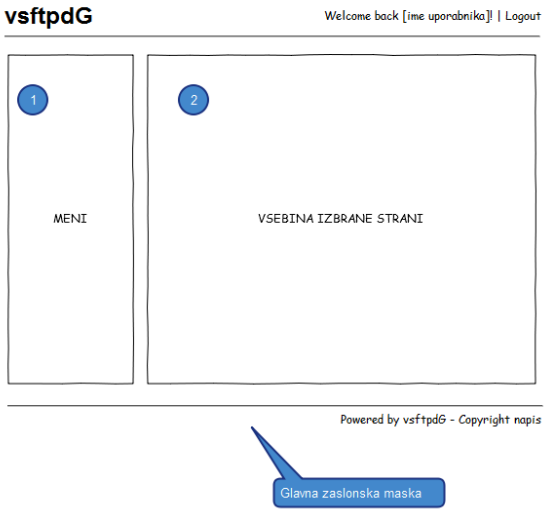
3.3.1.6 Prototip uporabniškega vmesnika

Izdelava papirnatega prototipa uporabniškega vmesnika nam lahko prihrani veliko časa pri razvoju aplikacije. Papirnati protip je lažje spreminjati (ni investicije v kodiranje, ostane načrt, drugo lahko odvržemo), skiciranje pa je tudi hitrejše od programiranja.[3] Na sliki 3.9 je prikazana glavna zaslonska

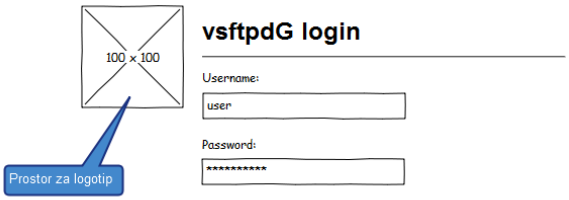


Slika 3.8: Diagram XML odziva strežnika na uspešno prijavo.

maska naše aplikacije, na sliki 3.10 pa je prikazan prototip prijavnega zaslona.



Slika 3.9: Prototip glavne zaslonske maske odjemalca vsftpdG.



Slika 3.10: Prijava v sistem.

3.3.2 Izdelava strežnika

Strežnik predstavlja samo osrčje aplikacije, saj je odgovoren za vse aktivnosti, ki se dogajajo na sistemu.

3.3.2.1 Branje konfiguracije

Pred samim zagonom se preberejo nastavitve. Nastavitve za dostop do podatkovne baze se nahajajo v datoteki `settings.txt`. Ta je sestavljena na podoben način kot konfiguracijske datoteke strežnika vsftpd, znak # označuje komentar, ostali parametri pa so:

- `mysql_user` - uporabniško ime, s katerim se povežemo na podatkovno bazo,
- `mysql_pass` - geslo, s katerim dostopamo do podatkovne baze,
- `mysql_db` - ime baze, ki jo bomo uporabljali,
- `mysql_server` - kje se podatkovna baza nahaja (lokalno ali IP naslov),
- `mysql_port` - številka vrat, na katerih teče MySQL strežnik (default ali številka vrat),
- `cer_store` - ključ za shranjevanje (certifikat) in
- `cer_key` - ključ certifikata.

Vsi parametri so obvezni za zagon strežnika, brez njih se javi napaka. V nadaljevanju strežnik uporabi te podatke za povezavo s podatkovno bazo MySQL. Iz njih prebere še ostale parametetre programa, ki so:

- `vsftpdg_user_folder` - pot do direktorija, kjer so shranjeni virtualni uporabniki,
- `vsftpd_config_file` - pot do konfiguracijske datoteke strežnika vsftpd,
- `remote_login` - vrednost true ali false; povemo, če je dovoljeno prijavljene iz odjemalcev, ki se ne nahajajo v lokalni mreži,
- `port` - številka vrat na katerih naj teče strežnik,
- `automatic_ip_blocking` - vrednost true ali false (avtomatsko blokiranje IP naslovov),
- `automatic_user_blocking` - vrednost true ali false (avtomatsko blokiranje uporabnikov),
- `vsftpd_log_file` - pot do vsftpd dnevnika,
- `log_minutes` - vrednost v minutah (na koliko časa se osvežuje bralnik dnevnikov),
- `session_timeout` - vrednost v minutah (koliko časa je lahko seja vzpostavljena brez prenašanja podatkov) in
- `session_clean` - vrednost v minutah (na koliko časa se zažene vzdrževalec sej).

Za branje konfiguracijskih podatkov smo izdelali razred `ConfigRead`, ki vrne podatkovno strukturo `HashMap<String, String>`, ki vsebuje vse podatke. Pri branju smo uporabljali javanski razred `Scanner`.

3.3.2.2 Razred `DB`

Za potrebe manipulacije s podatkovno bazo smo ustvarili razred `DB`. Izsek izvorne kode je viden spodaj:

```
1 public class DB {
2
3     private Connection connection = null;
4     // Isto za ostale spremenljivke
5     // ...
6
```

```
7       public DB(String user, String password, String
           host, String port, String db) {
8           this.user = user;
9           // Isto za ostale spremenljivke
10          // ...
11      }
12
13      public void getConnection() {
14
15          String connectionURL = "jdbc:mysql://" + host
                + ":" + port + "/" + db + "?user=" + user
              + "&password=" + password;
16
17          try {
18              connection = DriverManager.getConnection(
                   connectionURL);
19          } catch (SQLException ex) {
20              System.out.println("Wrong␣username,␣
                   password␣or␣mysql␣database␣name.");
21              System.out.println("Program␣exit␣...");
22              System.exit(0);
23          }
24
25      }
26
27      public void closeConnection() {
28          try {
29              connection.close();
30          } catch (SQLException ex) {
31              Logger.getLogger(DB.class.getName()).log(
                   Level.SEVERE, null, ex);
32          }
33      }
34
35      // Ostale funkcije razreda DB
36      // ...
37  }
```

Razred kot parameter prejme ime uporabnika, geslo do podatkovne baze, lokacijo baze, številko vrat in ime podatkovne baze, ki jo bomo uporabljali. S pomočjo funkcije `getConnection` ustvarimo povezavo z bazo (za te namene je uporabljena zunanja JAR knjižnica). Niz `connectionURL` sestavlja naslov, na katerega se moramo povezati, povezavo pa ustvarimo z ukazom `getConnection` (parameter je pa niz `connectionURL`). Funkcija `closeConnection` zapre povezavo s podatkovno bazo.

3.3.2.3 Ustvarjanje povezave z odjemalci

Ker mora biti povezava med odjemalcem in strežnikom varna, bo povezava vzpostavljena v TLS načinu. Pred ustvarjanjem povezave je potrebno prebrati certifikat. Za ta namen smo napisali sledečo kodo:

```
1         // Prebiranje konfiguracije streznika
2         // ...
3
4         SSLContext context = SSLContext.getInstance("
              TLS");
5         KeyManagerFactory kmf = KeyManagerFactory.
              getInstance("SunX509");
6         KeyStore ks = KeyStore.getInstance("JKS");
7
8         char[] store_password = configValues.get("
              cer_store").toCharArray();
9         char[] key_password = configValues.get("
              cer_key").toCharArray();
10
11        FileInputStream fin = new FileInputStream(new
              File("vsftpdg"));
12        ks.load(fin, store_password);
13        kmf.init(ks, key_password);
14        context.init(kmf.getKeyManagers(), null, null
              );
15
16        SSLServerSocketFactory factory = context.
              getServerSocketFactory();
17        SSLServerSocket listener = (SSLServerSocket)
              factory.createServerSocket(Integer.parseInt
              (configValues.get("port")));
```

V prvi vrstici nakažemo, da naj se uporablja protokol TLS. S spremenljivko `fin` odpremo datoteko, v kateri se nahaja certifikat. Z ukazom `ks.load` naložimo certifikat v pomnilnik, ukaz `kmf.init` pa inicializira naš certifikat. Z zadnjo vrstico ustvarimo strežnik na podani številki vrat.

V spodnjemu izseku kode je prikazano osnovno delovanje strežnika. V neskončni zanki sprejemamo zahteve po vzpostavitvi povezave. V spremenljivko `address` zapišemo IP naslov odjemalca. V naslednjemu if stavku najprej preverimo, če je parameter za oddaljen dostop nastavljen na negativno vrednost. Če oddaljen dostop ni dovoljen, najprej preverimo, če je IP naslov odjemalca lokalen. Če je preverjanje uspešno, vzpostavimo povezavo in ustvarimo nit

z razredom `ClientConnection`. Če je oddaljen dostop omogočen, preprosto ustvarimo nit z razredom `ClientConnection`.

```
1         while(serverRunning) {
2
3             server = (SSLSocket) listener.accept();
4             InetAddress address = server.
                 getInetAddress();
5
6             // If remote login isn't enabled, check
                 if the address is localhost
7             if(configValues.get("remote_login").
                 equals("false")) {
8               if(address.isLoopbackAddress()) {
9                   ClientConnection connection = new
                        ClientConnection(server,
                       configValues);
10                  Thread t = new Thread(connection)
                        ;
11                  t.start();
12              }
13            }
14
15            else {
16                ClientConnection connection = new
                     ClientConnection(server,
                     configValues);
17                Thread t = new Thread(connection);
18                t.start();
19            }
20        }
```

3.3.2.4 Razred `XMLErrorBuilder`

Razred `XMLErrorBuilder` smo ustvarili za izdelovanje XML sporočil odjemalcu, ki vsebujejo podatke o napakah. V spodnjemu izseku izvorne kode je prikazan celoten razred `XMLErrorBuilder`. Konstruktor razreda kot argument prejme niz `errorCode` in niz `errorMessage`. V funkciji `buildResponse` ustvarimo XML dokument s pomočjo javanske knjižnice `DocumentBuilderFactory`.

```
1 public class XMLErrorBuilder {
2
3     private String errorCode;
4     private String errorMessage;
5
```

```
6       public XMLErrorBuilder(String errorCode, String
           errorMessage) {
7           this.errorCode = errorCode;
8           this.errorMessage = errorMessage;
9       }
10
11      public String buildResponse() {
12
13          String response_string = null;
14          DocumentBuilderFactory dbf =
               DocumentBuilderFactory.newInstance();
15          DocumentBuilder build = dbf.
               newDocumentBuilder();
16          Document doc =  build.newDocument();
17
18          Element response = doc.createElement("
               Response");
19          doc.appendChild(response);
20
21          Element server = doc.createElement("Server");
22          response.appendChild(server);
23
24          Element code = doc.createElement("code");
25          code.appendChild(doc.createTextNode(errorCode
               ));
26          server.appendChild(code);
27
28          Element message = doc.createElement("message"
               );
29          message.appendChild(doc.createTextNode(
               errorMessage));
30          server.appendChild(message);
31
32          TransformerFactory tFact = TransformerFactory
               .newInstance();
33          Transformer trans = tFact.newTransformer();
34
35          StringWriter writer = new StringWriter();
36          StreamResult result = new StreamResult(writer
               );
37          DOMSource source = new DOMSource(doc);
38
39          trans.transform(source, result);
40
41          response_string = writer.toString();
42          writer.close();
43          return response_string;
```

```
44
45      }
46
47  }
```

Če konstruktorju razreda kot parameter podamo `101` in `Wrong username or password` dobimo sledečo XML datoteko:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?
       >
2  <Response>
3          <Server>
4                  <code>101</code>
5                  <message>Wrong username or password.<
                       /message>
6          </Server>
7  </Response>
```

3.3.2.5 Razred `ClientConnection`

Razred `ClientConnection` je odgovoren za tolmačenje ukazov, ki jih pošilja odjemalec.

```
1              server.setSoTimeout(5*1000);
2
3              while (!(command = in.readLine()).equals(
                   "quit")) {
4
5                  if(command.equals("exit")) break;
6                  // If command is accepted, refresh
                       the last input time
7                  if(command.length() > 0) {
8                      server.setSoTimeout(5*60000);
9                      // Preberi ukaz in poslji odziv
10                     // ...
11                 }
12             }
```

V zgornjemu izseku izvorne kode v prvi vrstici nastavimo čas prekinitve povezave. Če strežnik po petih sekundah ne prejme nobenega ukaza, se povezava avtomatsko prekine. V primeru poslanega ukaza, se čas prekinitve nastavi na pet minut. V `while` zanki prebiramo ukaze odjemalca in jih ustrezno interpretiramo.

Pred pošiljanjem ukazov se mora odjemalec prijaviti. To lahko stori z ukazom `login -u ime_uporabnika -p geslo -t trenutni_cas`. Za ustrezno prebiranje argumentov smo napisali sledečo kodo:

```
1         String[] split = command.split("␣");
2         HashMap<String, String> command_parameters =
             new HashMap<String, String>();
3
4         for(int i=0; i<split.length-2; i+=2) {
5             if((i+1 < split.length) && (i+2 < split.
                 length)) {
6                 command_parameters.put(split[i+1],
                     split[i+2]);
7             }
8         }
```

S pomočjo prve vrstice ukaz ustrezno razbijemo po presledkih in ga zapišemo v tabelo nizov. Potem se z zanko sprehodimo po tabeli in v `HashMap<String, String>` dodamo parametre ukaza. Zaradi večje varnosti se nizu geslo doda trenutni čas, vse skupaj pa se zakodira v SHA512. Ob uspešni ali neuspešni prijavi nam strežnik vrne XML odziv. Ob uspešni prijavi strežnik vrne sledeč odziv:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"
      ?>
2  <Response>
3          <Server>
4                  <code>102</code>
5                  <message>Login successfull.</message>
6          </Server>
7          <User>
8                  <username>zerocool</username>
9                  <admin>true</admin>
10                 <session_id>e4cba237756c6a5bc1658416
11                 7ab62e5106e664f01a29c4944b</
                      session_id>
12         </User>
13 </Response>
```

Po uspešni prijavi se spremenljivka `authenticated` nastavi na `true`, razred `ClientConnection` ob pošiljanju nadalnjih ukazov ustvari konstruktor `CommandParser`, ki prebira ukaze in pošilja XML odzive.

```
1     if(!authenticated} {
2           // do stuff
3     }
4     else {
5         // Send XML respones
6         CommandParser cmdP = new CommandParser(
              command, admin, configValues, username);
7         out.println(cmdP.getCommandResponse());
8         DB dbConnection = new DB(configValues.get("
              mysql_user"), configValues.get("mysql_pass"
              ), configValues.get("mysql_server"),
              configValues.get("mysql_port"),
              configValues.get("mysql_db"));
9         dbConnection.getConnection();
10        try {
11            dbConnection.updateLastSeen(username);
12        } catch (SQLException ex) {
13            Logger.getLogger(ClientConnection.class.
                  getName()).log(Level.SEVERE, null, ex);
14        }
15        dbConnection.closeConnection();
16    }
```

S pomočjo konstruktorja kličemo funkcijo `getCommandResponse`, ki vrne XML odziv na ukaz. Po poslanemu odzivu razred `ClientConnection` v podatkovni bazi posodobi informacijo o tem, kdaj je bil uporabnik nazadnje viden.

3.3.2.6 Razred `CommandParser`

Razred `CommandParser` je odgovoren za tolmačenje ukazov in njihovo izvajanje. Pred izvajanjem ukaza razred preveri, če je podan ukaz na seznamu dovoljenih ukazov. Za vsak ukaz tudi preveri, če so podani vsi obvezni parametri, v nasprotnem primeru vrne XML z opisom napake.

Dodajanje, urejanje in brisanje uporabnika

Uporabnika dodamo s pošiljanjem ukaza `add`. Parametri ukaza so:

- `-u` - ime uporabnika, ki ga dodajamo,
- `-p` - geslo uporabnika,

- `-f` - mapa uporabnika,
- `-deny` - spisek prepovedanih FTP ukazov ločenih z dvema dvopičjema,
- `-admin` - nastavljen na true ali false (če je uporabnik administrator);
- `-blocked` - nastavljen na true ali false (če je uporabnik blokiran) in
- `-denyfile` - spisek prepovedanih datotek ločenih z dvema dvopičjema.

Za dodajanje uporabnika smo v razredu `DB` dodali metodo `adduser`. Pred dodajanjem se ustrezno preveri, če uporabnik že obstaja.

```
1     public boolean addUser(String username, String
         password, String folder, String[] deny, String
         admin, String blocked, String[] filename_perm,
         String configFolder) throws SQLException,
         FileNotFoundException {
2       boolean userCreated = false;
3
4       // Vstavi uporabnika v podatkovno bazo
5       // ...
6
7       // Ustvari direktorij uporabnika, chown
           direktorija
8       // ...
9
10      fileWriter.println("local_root="+folder);
11
12      if(deny != null) {
13          StringBuilder sb = new StringBuilder("
               cmds_denied=");
14          for(int i=0; i<deny.length; i++) {
15              sb.append(deny[i]);
16              if(i<deny.length-1) sb.append(",");
17          }
18          fileWriter.println(sb.toString());
19      }
20
21      // Ustvari seznam prepovedanih datotek (
           podobno kot zgoraj)
22      // ...
23
24      fileWriter.close();
25
```

```
26          if(chOwn.length() < 1 || createDir.length() <
                1) {
27              userCreated = true;
28          }
29
30          return userCreated;
31      }
```

V zgornjemu izseku kode lahko vidimo, kako strežnik sam zgenerira konfiguracijsko datoteko in postori delo namesto uporabnika.

Urejanje uporabnika poteka na podoben način. Za urejanje uporabnika na strežnik pošljemo ukaz `edit`, parametri so pa isti kot pri ukazu `add`.

Pri brisanju uporabnika pošljemo ukaz `delete`. Ukaz sprejme parametra:

- `-u` - uporabniško ime, ki ga brišemo in
- `-fd` - nastavljen na true ali false (ali se mapa uporabnika izbriše).

Za namen brisanja uporabnika smo v razredu `DB` napisali funkcijo `deleteuser`:

```
1       public boolean deleteUser(...) {
2
3           // Pridobi direktorij uporabnika iz
                podatkovne baze
4           // ...
5           if(folder_delete) {
6               if(folder != null) {
7                   String bash_response = MainFunctions.
                        runBashCommand("rm␣-R␣"+folder);
8               }
9
10              rs.close();
11          }
12
13          String bash_response = MainFunctions.
                runBashCommand("rm␣"+configFolderPath+"/"+
                username);
14
15          // Dodatni SQL stavki za brisanje zapisnikov
                uporabnika in brisanje uporabnika iz
                podatkovne baze ...
16          // ...
17
```

```
18          PreparedStatement ps = null;
19          // Brisanje uporabnika iz podatkovne baze
20          String query_user_delete = "delete from users
               where u_name = ?";
21          ps = connection.prepareStatement(
               query_user_delete);
22          ps.setString(1, username);
23          ps.executeUpdate();
24
25          if(bash_response.length() < 1) return true;
26          else return false;
27
28      }
```

Če je parameter `folder_delete` nastavljen na `true`, iz podatkovne baze najprej preberemo, kje ima uporabnik domači direktorij. Potem kličemo funkcijo `runBashCommand` in izvedemo ukaz v lupini (`rm -R potdomape`). S tem ukazom izbrišemo vse naložene datoteke uporabnika. Po brisanju map je potrebno uporabnika izbrisati še iz podatkovne baze, to storimo s preprosto poizvedbo `delete from users where u_name = ?`. Manjkajoče parametre napolnimo z ukazom `ps.setString(1, username)`.

Blokiranje IP naslovov

IP naslov blokiramo s pomočjo ukaza `blockip`. Ukaz sprejme naslednje parametre:

- `-ip` - IP naslov, ki ga blokiramo,
- `-reason` - razlog blokiranja naslova (niz) in
- `-block` - nastavljen na true ali false (če bo IP naslov blokiran).

Za namen blokade smo v razred `DB` dodali funkcijo `blacklistIP`:

```
1       public String blacklistIP(String port, String
           username, String IP, String block, String
           reason) throws SQLException {
2
3           // Preveri ce je IP naslov ze blokiran
4           // ...
5
6           boolean alreadyBlocked = false;
7
```

```
8           if(Boolean.valueOf(block) == true) {
9                   String blockIP = MainFunctions.
                       runBashCommand("iptables␣-A␣INPUT␣-
                       s␣"+IP+"␣-p␣tcp␣--destination-port␣
                       "+port+"␣-j␣DROP");
10                  xml_error = new XMLErrorBuilder(
                       Responses.BLACKLIST_IP_SUCCESS,
                       Messages.BLACKLIST_IP_SUCCESS);
11          }
12          else if(alreadyBlocked && Boolean.valueOf(
               block) == true) {
13              xml_error = new XMLErrorBuilder(Responses
                   .BLACKLIST_IP_FAIL, Messages.
                   BLACKLIST_IP_FAIL_ALREADY_BLACKLISTED);
14          }
15          else {
16              String blockIP = MainFunctions.
                   runBashCommand("iptables␣-D␣INPUT␣-s␣"+
                   IP+"␣-p␣tcp␣--destination-port␣"+port+"
                   ␣-j␣DROP");
17              xml_error = new XMLErrorBuilder(Responses
                   .UNBLACKLIST_IP_SUCCESS, Messages.
                   UNBLACKLIST_IP_SUCCESS);
18          }
19
20          response = xml_error.buildResponse();
21
22          // Vstavi prenovljene podatke v podatkovno
               bazo
23          // ...
24
25          return response;
26      }
```

Pred blokiranjem naslova najprej preverimo, če je naslov že blokiran. Za potrebe blokiranja IP naslova uporabimo sistemski ukaz `iptables`, ki je požarni zid sistema Linux. Kot parameter podamo IP naslov in vrata našega strežnika.

Nadaljevanje seje in API ključ

Odjemalcem, ki niso sposobni vzdrževati konsistenčne povezave s strežnikom, je omogočen tudi ukaz `session`. Ukaz je namenjen nadaljevanju že prejšnje vzpostavljene seje. Kot parameter lahko dobi:

- `-id` - identifikacijska številka vzpostavljene seje in

- `-ip` - IP naslov iz katerega je bila vzpostavljena (parameter možen le ob vzpostavitvi API načina delovanja).

Za nadaljevanje seje in preverjanje nje smo v razred `DB` dodali funkcijo `getAuthentication`:

```
1     public UserLoginData getAuthentication(String
         session_id, String IP) throws SQLException {
2
3         UserLoginData uld = null;
4         PreparedStatement ps = null;
5         ResultSet rs = null;
6
7         // Pridobi sejo iz podatkovne baze, ce
             obstaja ...
8         // ...
9
10        // V ResultSet rs je zapisan rezultat
             poizvedbe select * from authentications
             where auth_id = session_id
11
12        while(rs.next()) {
13            IP_a = rs.getString("auth_ip");
14            username = rs.getString("users_u_name");
15            loggedIn = rs.getBoolean("auth_loggedin")
                 ;
16        }
17
18        if(username != null) {
19            query = "select␣*␣from␣users␣where␣u_name
                 ␣=␣?";
20            ps = connection.prepareStatement(query);
21            ps.setString(1, username);
22            rs = ps.executeQuery();
23            while(rs.next()) {
24                isAdmin = rs.getBoolean("u_admin");
25                isLocked = rs.getBoolean("u_locked");
26            }
27
28            if(IP_a != null) {
29                if(IP_a.equals(IP)) {
30                    uld = new UserLoginData(username,
                         isAdmin, isLocked, loggedIn,
                        false, "");
31                }
32            }
```

```
33
34          }
35
36          rs.close();
37          ps.close();
38
39          return uld;
40
41      }
```

Funkcija kot parameter prejme identifikacijsko številko seje in IP naslov odjemalca. Če se IP naslov odjemalca ujema z IP naslovom v podatkovni bazi, funkcija vrne podatke uporabnika v konstruktorju `UserLoginData`, drugače pa vrne `null`.

Če odjemalec teče na strežniku (primeri takšnega delovanja so odjemalci napisani v programskih jezikih PHP, JSP ipd.), je možna vzpostavitev povezave s pomočjo API ključa. API ključ omogoča dodajanje dodatnih parametrov ukazu `session` (IP naslov). Za vzpostavitev API povezave pošljemo ukaz `api stevilka_kljuca`. Številka ključa se ustvari ob namestitvi aplikacije in je unikatna za vsak strežnik.

3.3.2.7 Vzdrževanje sej in branje zapisnikov

Za vzdrževanje sej in branje zapisnikov smo ob zagonu strežnika ustvarili dva procesa:

```
1           Timer logRead = new Timer();
2           logRead.scheduleAtFixedRate(new LogReaderTask
               (configValues), 0, Integer.parseInt(
               configValues.get("log_minutes"))*60000);
3
4           Timer sessionCleaner = new Timer();
5           sessionCleaner.scheduleAtFixedRate(new
               SessionClearerTask(configValues), 0,
               Integer.parseInt(configValues.get("
               session_clean"))*60000);
```

Razred `LogReaderTask` je odgovoren za branje zapisnikov, razred `SessionClearerTask` pa za vzdrževanje seje. S pomočjo javanskega razreda `Timer` lahko ustvarimo naloge, ki se bodo ponavljale v časovnih intervalih. S pomočjo metode `scheduleAtFixedRate` nastavimo, katera naloga naj se izvaja in

časovni interval.

Razred `SessionClearerTask`

Po prijavi uporabnikov ni vedno nujno, da uporabnik klikne na gumb za odjavo (lahko tudi samo zapre brskalnik). Lahko se tudi zgodi napaka na strežniku (zaustavitev ipd.). Za ta namen moramo ustvariti razred, ki bo poskrbel za take primere. V razredu `SessionClearerTask` kličemo metodo `clearSessions`, ki se nahaja v razredu `DB`:

```
1     public void clearSessions(int timeout) throws
         SQLException {
2
3         PreparedStatement ps = null;
4         ResultSet rs = null;
5
6         // Poglej trenuten cas, ustvari zacetni cas
             seje (trenuten cas - cas omejitve seje)
7         // ...
8
9         String query = "select␣auth_id␣from␣
             authentications␣where␣auth_tries␣<␣4␣and␣
             auth_id␣not␣in␣("
10                + "select␣auth_id␣from␣
                     authentications␣where␣auth_tries␣<␣
                     4␣and␣auth_lastseen␣>=␣str_to_date
                     (?,␣'%Y-%m-%d␣%T')␣and␣"
11                + "auth_lastseen␣<=␣str_to_date(?,␣'%
                     Y-%m-%d␣%T'))";
12
13        ps = connection.prepareStatement(query);
14        ps.setString(1, timeStartFormatedDate);
15        ps.setString(2, currentFormatedDate);
16        rs = ps.executeQuery();
17
18        while(rs.next()) {
19            String id = rs.getString("auth_id");
20            if(id != null) {
21                removeAuthenticationID(id);
22            }
23        }
24
25        ps.close();
26        rs.close();
27
28    }
```

Pri odstranjevanju sej moramo najprej odstraniti seje, kjer je bilo število poskusov prijav manjše kot štiri, pri tem pa moramo tudi pogledati, če se seja šteje med poteklo (seja je potekla, ko uporabnik že več časa ni poslal ukaza; privzeta vrednost je dvajset minut). To naredimo z SQL stavkom, ki je napisan v spremenljivki `query`. Rezultate poizvedbe zapišemo v spremenljivko `rs`. Potem se sprehodimo v zanki (dokler ima `rs` še kakšen element) in v zanki kličemo metodo `removeAuthenticationID` s podano identifikacijsko številko.

Pri brisanju sej pa moramo pogledati tudi blokirane uporabnike. Če je bil uporabnik blokiran in se ni prijavil po poteku časa, je potrebno pobrisati tudi te zapise. Zato moramo v funkcijo `clearSessions` dodati:

```
1         query = "select␣auth_id␣from␣authentications␣
             where␣unix_timestamp(now())-unix_timestamp(
             auth_logintime)␣"
2                 + ">=␣30*60␣and␣auth_tries␣>␣3␣";
3
4         ps = connection.prepareStatement(query);
5         rs = ps.executeQuery();
6
7         while(rs.next()) {
8             String id = rs.getString("auth_id");
9             if(id != null) {
10                removeAuthenticationID(id);
11            }
12        }
```

V spremenljivko `query` zapišemo SQL poizvedbo, ki iz baze prebere vse seje, katerih čas zadnje prijave je večji od trideset minut. Potem se sprehodimo v zanki (dokler ima `rs` še kakšen element) in v zanki kličemo metodo `removeAuthenticationID` s podano identifikacijsko številko.

Razred `LogReaderTask`

Razred `LogReaderTask` je odgovoren za poganjanje bralnika zapisnikov. V metodi `run` smo zapisali:

```
1     @Override
2     public void run() {
3         String md5FileCheck = MainFunctions.
             runBashCommand("md5sum␣"+configValues.get("
             vsftpd_log_file"));
```

```
4           String md5Comparison = null;
5           String md5Current = md5FileCheck.substring(0,
                md5FileCheck.indexOf("␣"));
6
7           long currentLineNumber = 0;
8
9           File logReadParameters = new File("logreader.
               log");
10          if(logReadParameters.exists()) {
11              // Preberi md5 nazadnje pregledane log
                   datoteke in vrstico, pri kateri je
                   bralnik ostal
12              // ...
13          }
14
15          boolean runLogger = true;
16
17          if(md5Comparison != null) {
18              if(md5Current.equals(md5Comparison))
                   runLogger = false;
19          }
20
21          if(runLogger) {
22              LogParser logParser = new LogParser(
                   configValues, currentLineNumber);
23              long lineNumbers = logParser.
                   executeParser();
24              logParser.writeParametersToFile(
                   lineNumbers, md5Current);
25          }
26      }
```

Pred branjem zapisnikov moramo najprej preveriti, če se je datoteka spremenila od zadnjega branja (tako odstranimo nepotrebno branje). To lahko naredimo z lupinskim ukazom `md5sum`. Če se je bralnik že kdaj zagnal, so parametri zadnjega branja zapisani v datoteki `logreader.log`. Če se oba parametra ujemata, se spremenljivka `runLogger` nastavi na `false` (kar pomeni, da se datoteka od prejšnjega branja ni spremenila). Če preverjanje pogoja `runLogger` uspe, se bralnik zažene z inicializacijo konstruktorja `LogParser` in klicem njegove funkcije `executeParser`. Ko se branje konča, posodobimo parametra `md5` in `line` v datoteki `logreader.log`.

3.3.3 Izdelava odjemalca

3.3.3.1 Izdelava HTML5 predloge

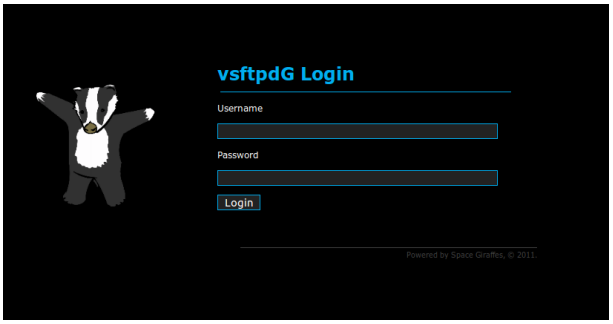
Pred programiranjem smo morali narediti osnoven izgled strani. Za ta namen smo uporabili HTML5[17] in CSS3[18]. Na sliki 3.12 je prikazan prijavni zaslon, na sliki 3.11 pa glavna zaslonska maska. Da smo dosegli takšen izgled, smo uporabili `div` elemente. Izgled v HTML5 dokumentih je definiran s CSS datotekami. Izsek kode iz naše datoteke:

```
1        #main {
2                width: 980px;
3                margin-left: auto;
4                margin-right: auto;
5                margin-top: 100px;
6        }
7        #topbar_vsftpdg {
8                float: left;
9                width: 120px;
10       }
11       body {
12               background-color: #000;
13       }
14       body,td,th {
15               font-family: Verdana, Geneva, sans-
                    serif;
16               font-size: .75em;
17               color: #FFF;
18               line-height: 1.5em;
19       }
20
21       // Ostali stili elementov
22       // ...
```

Za zamik prikaza elementov smo uporabili atribute `margin-right`, `margin-top`, `margin-left` in `margin-bottom`. Ti atributi ustrezno povedo, za koliko točk naj bo odmaknjena predstavitev elementa od robov okna. Z oznako `font-family` smo povedali, katera pisava naj se uporabi, `color` pa označuje barvo pisave. Za menjavo stilov smo uporabili zunanjo skripto[1] in ob pritisku na barvo klicali funkcijo `javascript:chooseStyle('ime_teme', 60)`.

[1] Styleswitch, odprtokodna skripta za menjavanje stilov[12]

Slika 3.11: Glavna stran vsftpdG odjemalca.



Slika 3.12: Prijavni zaslon vsftpdG odjemalca.

3.3.3.2 Izdelava razreda za povezavo s strežnikom

Za potrebe izvajanja ukazov in prejemanja odzivov smo morali najprej napisati razred, ki se bo povezal s strežnikom in izvajal naše ukaze. Izsek kode:

```
1 <?php
2
3 class ServerConnection {
4
5     private $port;
6     private $address;
7     private $connection;
8     private $success;
9
```

```
10      public function __construct($port, $address) {
11          $this->port = $port;
12          $this->address = $address;
13      }
14
15      public function establishConnection() {
16          $this->connection = stream_socket_client("tls
               ://$this->address:$this->port", $errno,
               $errstr, 30, STREAM_CLIENT_CONNECT |
               STREAM_CLIENT_PERSISTENT);
17          if(!$this->connection) {
18              $this->success = false;
19          }
20          else {
21              $this->success = true;
22          }
23          return $this->success;
24      }
25
26      public function getResponse($request) {
27          $response = "No connection established.";
28          if($this->success) {
29              fputs($this->connection, $request."\n");
30              $response = fgets($this->connection);
31          }
32          return $response;
33      }
34
35      // Ostale funkcije razreda ...
36      // ...
37
38  }
39  ?>
```

V funkciji `__construct` smo definirali konstruktor našega razreda. Ta kot parameter prejme številko vrat (`$port`) in naslov, na katerem se nahaja strežnik (`$address`). Funkcija `establishConnection` vzpostavi povezavo s strežnikom (beseda ssl pred naslovom strežnika označuje, da bomo vzpostavljali zaščiteno povezavo). Za vzpostavljanje povezave smo uporabili PHPjevo funkcijo `stream_socket_client`, ki kot parameter prejme naslov strežnika s številko vrat, časovno omejitev in tip vzpostavljene povezave. Funkcija `getResponse` pridobi odziv strežnika na ukaz (podan s parametrom `$request`). Za pošiljanje smo uporabili funkcijo `fputs`, za prebiranje odziva pa funkcijo `fgets`. Pri pošiljanju velja omeniti, da je potrebno na koncu ukaza dodati znak za novo vrstico. Če to pozabimo, se pošiljanje zacikla, ker strežnik bere ukaze po

vrsticah.

3.3.3.3 Branje XML odzivov strežnika

Za ustrezno predstavitev podatkov uporabniku je bilo potrebno ustvariti funkcije, ki bodo prebirale odzive strežnika. Za ta namen smo napisali razred `XMLServerParser`:

```
1  <?php
2
3  class XMLServerParse {
4
5      private $xml;
6
7      public function __construct($xml) {
8          $this->xml = simplexml_load_string($xml);
9      }
10
11     public function getResponseCode() {
12         $code = (string) $this->xml->Server[0]->code;
13         return $code;
14     }
15
16     public function getResponseMessage() {
17         $message = (string) $this->xml->Server[0]->
               message;
18         return $message;
19     }
20
21     // Ostale funkcije razreda
22     // ...
23
24 }
25 ?>
```

Razred kot parameter prejme XML niz, ki se ob instanci razreda pretvori s pomočjo funkcije `simplexml_load_string`. Funkcija `getResponseCode` vrne številko kode odziva strežnika, funkcija `getResponseMessage` pa sporočilo odziva strežnika.

Za prebiranje vseh različnih tipov odzivov strežnika smo v ta razred dodajali posamezne funkcije. Za branje odziva na prijavo smo napisali funkcijo `getLoginData`, ki vrne instanco razreda `UserLogin` z vsemi podatki, ki jih

potrebujemo.

```
1     public function getLoginData() {
2        $username = (string) $this->xml->User[0]->
            username;
3        $admin = (string) $this->xml->User[0]->admin;
4        $session_id = (string) $this->xml->User[0]->
            session_id;
5        $code = $this->getResponseCode();
6        $message = $this->getResponseMessage();
7        return new UserLogin($code, $message,
            $username, $session_id, $admin);
8     }
```

3.3.4 Izdelava skripte za namestitev

Za lažjo namestitev in čimhitrejšo postavitev FTP strežnika smo napisali skripto za namestitev. Namestitev poteka na sledeči način:

- kličemo zagon strežnika s parametrom install (strežnik ustrezno preveri, če je namestitev že predhodno uspela),
- pregledamo, če ima uporabnik vse nameščene pakete (mysql-client, mysql-server, mysql-pamd[19], vsftpd),
- uporabnika vprašamo za MySQL uporabniško ime, geslo, podatkovno bazo, lokacijo in številko vrat,
- za generiranje certifikata uporabnika vprašamo vse potrebne podatke in
- zgeneriramo konfiguracijsko datoteko, pam.d datoteko, postavimo podatkovno bazo in program skopiramo v podan direktorij.

Za generiranje certifikata smo uporabili javansko orodje `keytool`. Ukaz smo klicali z naslednjimi parametri:

```
1 keytool -keystore vsftpdg -keypass 123456 -storepass
     lollol -genkey -keyalg RSA -alias vsftpdg -dname
2 "CN=Bostjan␣Cigan,␣OU=FRI,␣O=Space␣Giraffes,␣L=
     Ljubljana,␣S=Slovenia,␣C=SL"
```

Vrednosti v zgornjemu ukazu smo nadomestili z vrednostmi, ki nam jih je podal uporabnik in ukaz izvedli v lupini. Za preverjanje nameščenih ukazov smo uporabili lupinski ukaz `dpkg` s podanim parametrom `-s` (ime paketa). Če se v izhodu ukaza nahaja vrstica `Status: install ok installed`, je paket nameščen.

Poglavje 4

Analiza aplikacije

4.1 Obremenitveni test strežnika

Za potrebe testiranja našega strežnika smo izdelali namenski PHP odjemalec, ki na strežnik pošilja večje število zahtev in beleži čas izvajanja. Po izvajanju smo zgenerirali ukaz za risanje grafa v Mathematici[10].

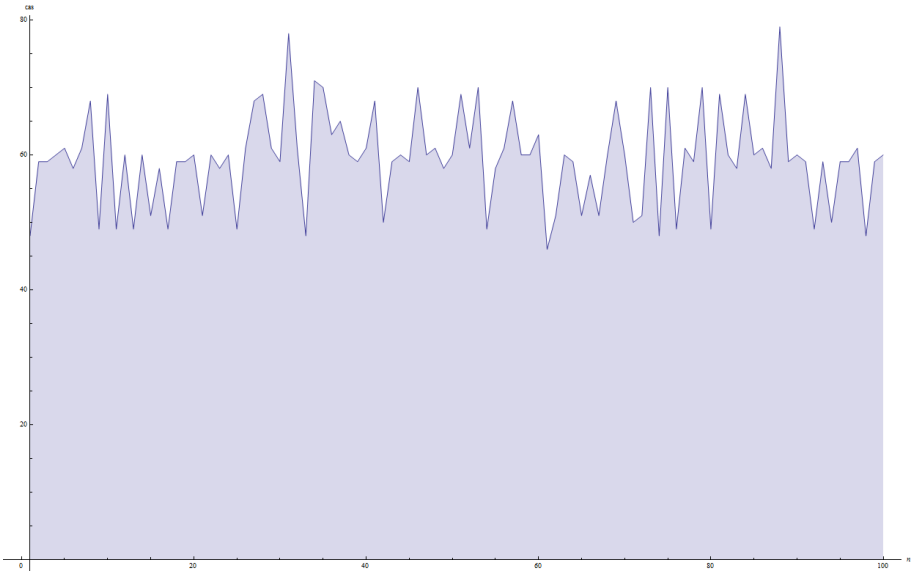
Test z enim uporabnikom

Sistem smo najprej testirali z enim uporabnikom. Izvedli smo prijavo in poslali sto različnih zahtevkov na strežnik. Za vsakega smo izmerili čas in narisali graf (slika 4.1).

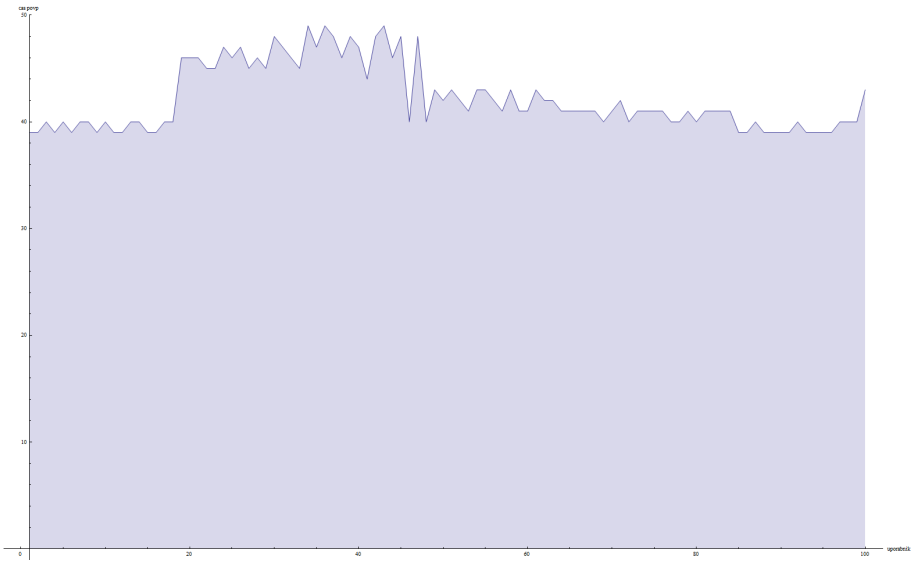
Po testu smo ugotovili, da je bila minimalna hitrost izvajanja zahteve 46 milisekund, maksimalna pa 79 milisekund. Povprečen čas izvajanja zahteve je bil 59.38 milisekund, kar dosega naše cilje glede hitrosti.

Test s stotimi uporabniki

Ker so meritve za enega uporabnika nerealne, smo test izvedli s stotimi uporabniki, ki so hkrati prijavljeni v sistem, vsak izmed njih pa izvede sto ukazov. Na sliki 4.2 je prikazan povprečen čas izvajanja zahteve vsakega prijavljenega uporabnika. Najboljši uporabnik je dosegel 39 milisekund, najslabši pa 49 milisekund, povprečen čas izvajanja zahtev med stotimi uporabniki je 42 milisekund. Kljub visokim obremenitvam je naš sistem ostal stabilen.



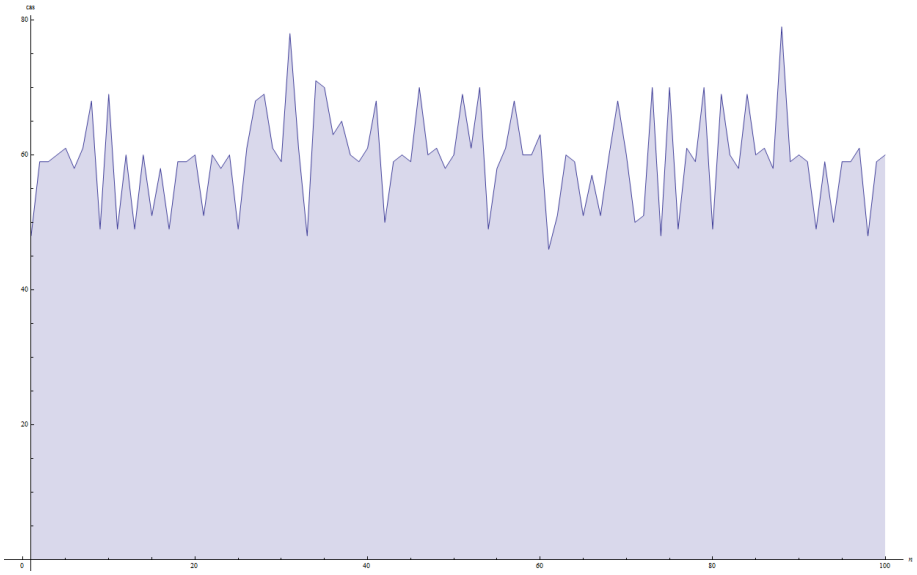
Slika 4.1: Graf hitrosti testiranja enega uporabnika.



Slika 4.2: Graf hitrosti testiranja stotih uporabnikov.

Na sliki 4.3 je prikazan povprečen čas vsake zahteve naključnega uporabnika med stotimi v sistemu. Maksimalen čas izvajanja zahteve je bil 70 milisekund,

minimalen pa 37 milisekund. Povprečen čas izvajanja zahteve je 43.51 milisekunde.



Slika 4.3: Graf hitrosti testiranja enega uporabnika med stotimi uporabniki.

Iz zgornjega testiranja lahko ugotovimo, da naša aplikacija dosega zahteve po hitrosti in je hkrati zmožna servisirati večim uporabnikom brez nepotrebnih zastojev.

4.2 Težave pri izvedbi

Pri izvedbi naše aplikacije smo naleteli na nekaj težav:

- **shranjevanje imen uporabnikov v podatkovni bazi** - ker vsftpd ne podpira shranjevanja v podatkovni bazi, podpira pa uporabo knjižnice pam.d[19], smo morali ustrezno prirediti konfiguracijske datoteke,
- **izgubljene seje** - problem izgubljenih sej se pojavi, če se uporabnik pozabi odjaviti, zato je bilo potrebno ustvariti razred, ki bo seje po neaktivnosti pobrisal,

- **certifikat strežnika** - ker mora biti povezava med strežnikom in odjemalcem varna, smo pri povezavi uporabili certifikat, da pa uporabniku to ne bi povzročalo težav, smo proces izdelave certifikata avtomatizirali v namestitveni skripti,
- **konsistenčna povezava v phpju** - pri vzpostavljanju povezave s strežnikom je pri uporabi konsistenčne povezave prihajalo do težav; če se povezava ni ustrezno zaprla, jo je PHP dodelil drugemu uporabniku - problem smo rešili z izdelavo mehanizma nadaljevanja seje pri strežniku in
- **problem časovnega žiga** - mehanizem časovnega žiga (oziroma timestamp) nam je sredi razvoja predstavil problem (Y2K38[1]), saj smo vse čase v podatkovni bazi označili s strukturo `timestamp`; problem smo rešili z uporabo strukture `datetime`.

4.3 Možne izboljšave

Zaradi narave diplomskega dela in časovne omejitve smo veliko izboljšav naše aplikacije prihranili za prihodnje verzije:

- **omejevanje prenosa** - omejevanje podatkovnega prenosa uporabnikom,
- **omejevanje prostora** - vsftpd sam po sebi ne podpira omejevanje diskovnega prostora, s strežniškim delom naše aplikacije pa lahko dodamo to možnost,
- **nastavitve strežnika** - uporabniku želimo omogočiti nastavljanje parametrov strežnika tudi na daljavo (ne samo ob namestitvi aplikacije),
- **predstavitev statistike** - boljša predstavitev statistike z izdelavo grafov in tabel,
- **odjemalci** - izdelava odjemalcev za druge platforme (Android, Windows Mobile, iOS),
- **nalaganje preko odjemalca** - uporabniku želimo omogočiti nalaganje datotek tudi preko odjemalca (ne samo preko namenskih aplikacij kot je npr. Filezilla),

[1] Y2K38 - Problem prekoračitve tipa integer leta 2038[11]

- **blokiranje zahtev** - želimo izdelati sistem za avtomatsko blokiranje IP naslovov, ki konstantno pošiljajo neveljavne zahteve (trenutno je ta problem rešen s časovno omejitvijo) in
- **razširitev koncepta** - če bo testiranje aplikacije uspešno, bomo poskušali sam koncept razširiti (izdelavo strežnika, ki bo upravljal tudi s spletnimi strežniki kot so Apache, nginx, Tomcat in podobnimi).

Poglavje 5

Zaključek

5.1 Sklepne ugotovitve

Pri izdelavi diplomske naloge smo načrtovali in implementirali grafični uporabniški vmesnik za FTP strežnik vsftpd. Predstavili smo pri razvoju uporabljena orodja in tehnologije ter na koncu izvedli testiranje aplikacije.

Pri izvedbi smo pridobili veliko novih izkušenj iz uporabe Unix sistemov. Pri nadgrajevanju in nadalnjem razvoju ne bo težav, saj smo se držali pravil in standardov kodiranja. Izvorna koda je pregledna in lahko razumljiva. Vse cilje, ki smo si jih med razvojem zastavili, smo uspešno izpolnili.

Aplikacija je trenutno v fazi testiranja in bo kmalu na voljo vsem uporabnikom. Trenutno se tudi dela na novi verziji odjemalca, ki bo ponujala več funkcionalnosti.

Pri samem razvoju nismo imeli večjih težav, kar je rezultat dobrega načrtovanja in učinkovitega raziskovalnega dela. Izkušnje, ki smo jih pridobili, nam bodo v veliko pomoč pri morebitnih novih projektih.

Dodatek A

Seznam FTP ukazov

ABOR

Sintaksa: `ABOR`

Prekine trenuten prenos datoteke.

CWD

Sintaksa: `CWD ime_direktorija`

Spremeni direktorij, v katerem se trenutno nahajamo.

DELE

Sintaksa: `DELE ime_datoteke`

Izbriši dano datoteko, ki se nahaja v strežniku.

LIST

Sintaksa: `LIST ime_direktorija`

Če se ime datoteke nanaša na direktorij, pošlje informacijo o vsaki datoteki, ki se nahaja v direktoriju. Pred uporabo ukaza je potrebno izvesti ukaz PORT ali PASV.

MDTM

Sintaksa: `MDTM ime_datoteke`

Vrne čas zadnjega urejanja dane datoteke v formatu *YYYYMMDDhhmmss*.

MKD

Sintaksa: `MKD ime_direktorija`

Ustvari direktorij na strežniku.

NLST

Sintaksa: `NLST ime_direktorija`

Vrne seznam datotek v danem direktoriju (brez dodatnih informacij), predhodno mora biti izveden ukaz PORT ali PASV.

PASS

Sintaksa: `PASS geslo`

Ukaz se pošlje strežniku za dokončanje prijavnega procesa (po poslanemu ukazu USER).

PASV

Sintaksa: `PASV`

Sporoči strežniku naj preklopi v pasivni način delovanja.

PORT

Sintaksa: `PORT a1,a2,a3,a4,p1,p2`

Določi gostitelja in vrata, na katera naj se strežnik poveže ob naslednjem prenosu. Prvi štirje parametri predstavljajo IP naslov (*a1.a2.a3.a4*), iz parametrov *p1* in *p2* pa se izračuna številka vrat (*p1*256+p2*).

PWD

Sintaksa: `PWD`

Vrne direktorij, v katerem se trenutno nahajamo.

QUIT

Sintaksa: `QUIT`

Prekinite trenutno povezavo.

RETR

Sintaksa: `RETR ime_datoteke`

Prične prenos podane datoteke iz strežnika.

RMD

Sintaksa: `RMD ime_direktorija`

Izbriše podan direktorij na strežniku.

RNFR

Sintaksa: `RNFR ime_datoteke`

Uporablja se za preimenovanje datoteke; s parametrom povemo, katera datoteka bo preimenovana. Ukazu sledi ukaz RNTO, ki določi novo ime datoteke.

RNTO

Sintaksa: `RNTO novo_ime_datoteke`

Uporablja se pri preimenovanju datoteke za ukazom RNFR in določa novo ime datoteke.

SIZE

Sintaksa: `SIZE ime_datoteke`

Vrne velikost dane datoteke v decimalnem številu.

STOR

Sintaksa: `STOR ime_datoteke`

Začne prenos datoteke na strežnik. Pred ukazom se mora izvesti ukaz PORT ali PASV, ki strežniku sporočata, od kod naj prejema podatke.

TYPE

Sintaksa: `TYPE tip_znakov`

Nastavi tip podatkov, ki se bodo prenašali. Tip je lahko:

- A - ASCII tekst
- E - EBCDIC tekst
- I - slika (binarni podatki)
- L - lokalni format

USER

Sintaksa: `USER uporabniško_ime`

Ukaz omogoči začetek prijave in pošlje uporabniško ime.

Dodatek B

Seznam najpogostejših konfiguracijskih opcij vsftpd

B.1 Opcije tipa boolean

anonymous_enable
Če je ukaz nastavljen na yes, so anonimne prijave na strežnik dovoljene.

anon_upload_enable
Za delovanje te nastavitve mora biti na yes nastavljena tudi opcija *write_enable*. Opcija označuje ali imajo anonimni uporabniki pravico do nalaganja datotek.

write_enable
Določa ali so dovoljeni ukazi za pisanje na strežniku (STOR, DELE, RNFR, RNTO, MKD, RMD, APPE, SITE).

delete_failed_uploads
Označuje, če se neuspešno naložene datoteke brišejo iz strežnika.

dirlist_enable
Določa, če se prikaže seznam datotek ob prijavi na strežnik.

local_enable
Določa ali imajo lokalni uporabniki pravico do prijave na strežnik (uporabniki shranjeni v /etc/passwd).

port_enable

Označuje, če je dovoljen ukaz PORT.

B.2 Numerične opcije

max_clients
Določa maksimalno število hkrati prijavljenih uporabnikov.

local_max_rate
Maksimalno število prenosa podatkov v bajtih na sekundo.

max_login_failsrate
Maksimalno število neveljavnih prijav.

accept_timeout
Določa, koliko sekund naj največ čaka strežnik, preden odjemalec pošlje zahtevo za pasivno povezavo.

data_connection_timeout
Določa, koliko sekund naj strežnik čaka pri neuspelih prenosih ali prenosih, ki so se ustavili.

anon_max_rate
Določa maksimalno število prenosa podatkov v bajtih na sekundo za anonimne uporabnike.

delay_successful_login
Določa zamik v sekundah pred dovoljeno uspešno prijavo.

delay_failed_login
Določa zamik v sekundah pred javljanjem neuspešne prijave.

B.3 Opcije z nizi

ftp_username
Določa ime anonimnega FTP uporabnika.

anon_root
Označuje pot do direktorija anonimnih uporabnikov.

cmds_allowed
Določa seznam dovoljenih ukazov na FTP strežniku.
Primer: cmds_allowed=PASV,RETR,QUIT

cmds_denied
Določa seznam prepovedanih ukazov na FTP strežniku (sintaksa je ista kot pri cmds_allowed).

dsa_cert_file
Označuje pot do datoteke, kjer se nahaja DSA certifikat za SSL povezave.

dsa_private_key_file
Pot do privatnega ključa DSA za SSL povezavo. Če opcija ni podana, strežnik predvideva, da se privatni ključ nahaja v certifikatu.

ftpd_banner
Prikazno sporočilo strežnika ob prijavi (če opcija ni navedena, se prikaže privzeto vsftpd sporočilo).

deny_file
Označuje seznam prepovedanih datotek (omejevanje nalaganja).
Primer: deny_file={*.mp3,*.jpg}

hide_file
Označuje, katere datoteke naj se skrijejo na strežniku. Sintaksa je identična deny_file.

rsa_cert_file
Označuje pot do datoteke, kjer se nahaja RSA certifikat za SSL povezave.

rsa_private_key_file
Pot do privatnega ključa RSA za SSL povezavo. Če opcija ni podana, strežnik predvideva, da se privatni ključ nahaja v certifikatu.

user_config_dir
Določa direktorij, kjer so shranjeni virtualni uporabniki in njihova pravila.

user_sub_token
Določa oznako za virtualnega uporabnika. Če npr. kot oznako določimo $USER, lahko v ukazu user_config_dir določimo vrednost /etc/vsftpd/$USER.

vsftpd_log_file
Označuje pot do zapisnika strežnika in ime datoteke.

Slike

Literatura

[1] J. B. Knudsen, *Java Cryptography*, Združene države Amerike: O'Reilly, 1998, pogl. 6.

[2] T. Pender, *UML Bible*, Združene države Amerike: Wiley Publishing, 2003, pogl. 12.

[3] C. Snyder, *Paper Prototyping The Fast and Easy Way to Design and Refine User Interfaces*, Združene države Amerike: Morgan Kaufmann, 2003, str. 12.

[4] (2011) Seznam FTP ukazov. Dostopno na: `http://www.nsftools.com/tips/RawFTP.htm`.

[5] (2011) FTP protokol. Dostopno na: `http://www.isaserver.org/articles/how_the_ftp_protocol_challenges_firewall_security.html`.

[6] (2011) vsftpd Wikipedia. Dostopno na: `http://en.wikipedia.org/wiki/Vsftpd`.

[7] (2011) Orodje Pencil. Dostopno na: `http://pencil.evolus.vn/en-US/Home.aspx`.

[8] (2011) Orodje Sybase PowerDesigner Wikipedia. Dostopno na: `http://en.wikipedia.org/wiki/PowerDesigner`.

[9] (2011) Podatkovni modeli. Dostopno na: `http://colos1.fri.uni-lj.si/ERI/RACUNALNISTVO/PODATKOVNE_BAZE/podatkovni_model.html`.

[10] (2011) Mathematica. Dostopno na: `http://www.wolfram.com/mathematica/`.

[11] (2011) Y2K38 problem Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Year_2038_problem.

[12] (2011) Skripta styleswitch. Dostopno na:
http://www.dynamicdrive.com.

[13] (2011) Orodje Webmin Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Webmin.

[14] (2011) Orodje kvsftpdmanager. Dostopno na:
http://sourceforge.net/projects/kvsftpdmanager.

[15] (2011) Orodje Vsftpd Nas Gui. Dostopno na:
http://sourceforge.net/projects/vsftpdnasgui.

[16] (2011) Orodje Netbeans Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Netbeans.

[17] (2011) HTML5 Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Html5.

[18] (2011) CSS3 Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Cascading_Style_Sheets.

[19] (2011) Pam Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Pluggable_authentication_module.

[20] (2011) Ubuntu Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Ubuntu_(operating_system).

[21] (2011) CentOS Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Centos.

[22] (2011) NimbleX Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/NimbleX.

[23] (2011) Fedora Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Fedora_(operating_system).

[24] (2011) RHel spletna stran. Dostopno na:
http://www.redhat.com/rhel.