Sveučilište J.J.Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Rebeka Čordaš

Internet protokoli za e-mail komunikaciju

Završni rad

Osijek, 2011.

Sveučilište J.J.Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike

Rebeka Čordaš

Internet protokoli za e-mail komunikaciju

Završni rad

Mentor:
doc.dr.sc. Domagoj Matijević

Osijek, 2011.

Sažetak

Protokoli su skup pravila koja određuju kako dva uređaja ili programa međusobno komuniciraju. Oni određuju formate poruka i načine kako ih računala izmjenjuju. SMTP, POP i IMAP su protokoli za e-mail komunikaciju. SMTP protokol služi za slanje poruka od klijenta do servera. POP protokol služi za dohvaćanje poruka od servera skidajući sve podatke sa servera. IMAP protokol također služi za dohvaćanje poruka sa servera, ali pritom ne skida poruke na računalo nego ih čita i obavlja operacije nad njima na udaljenom serveru. Programski jezik Python sadrži biblioteke u kojima su implementirane najbitnije funkcije koje koriste određeni protokoli. U biblioteci smtplib su implementirane funkcije koje koristi SMTP protokol. Biblioteka poplib sadrži funkcije koje koristi POP protokol, a biblioteka imaplib sadrži funkcije koje koristi IMAP protokol.

Ključne riječi

protokol, SMTP, POP, IMAP, e-mail, klijent, server, RFC, Python

Abstract

Protocols are a set of rules that determine how the two devices or applications communicate with each other. They define message formats and how the computers send messages to each other. SMTP, POP and IMAP are protocols for e-mail communication. SMTP protocol is used for sending messages from client to server. POP protocol is used for retrieving messages from server by downloading all messages from server to computer. IMAP protocol is also used for retrieving messages from server, but it doesn't download messages to the computer. It reads messages and performs operations on them on remote server. Python programming language contains libraries where the most important functions that protocols use are implemented. The smtplib library contains most important functions which are used by SMTP protocol. The poplib library contains most important functions which are used by POP protocol, while the imaplib library contains most important functions which are used by IMAP protocol.

Key words

protocol, SMTP, POP, IMAP, e-mail, client, server, RFC, Python

Sadržaj

1 Uvod

Kako se u današnje vrijeme internet sve brže širi i količina podataka koji kruže tom globalnom mrežom se svake godine višestruko povećava potrebni su neki alati kojima će se omogućiti brzo i sigurno slanje podataka između korisnika. Jedni od tih alata su i internet protokoli. U ovom radu ću se baviti internet protokolima za e-mail komunikaciju, tj. SMTP, POP i IMAP protokolima.

Rad je podijeljen na šest dijelova: Uvod, SMTP, POP i IMAP, protokoli za e-mail komunikaciju, Implementacija mail klijenta u Pythonu, Zaključak, Literatura i Prilog - popis slika.

Poglavlje SMTP, POP i IMAP, protokoli za e-mail komunikaciju pobliže pojašnjava sam pojam protokola i kako oni funkcioniraju. Poglavlje je podijeljeno na tri potpoglavlja: SMTP protokol, POP protokol i IMAP protokol. U svakom od tih potpoglavlja je pobliže objašnjen princip rada određenog protokola te kratka povijest razvitka samog protokola kao i njegovih mehanizama za autentifikaciju. U sklopu svakog potpoglavlja postoji jedan odlomak koji govori o protokolu i programskom jeziku Python, tj. na koji je način određeni protokol implementiran u Python te su nabrojane i pojašnjene najbitnije funkcije biblioteke u kojoj je određeni protokol implementiran.

Kao praktični dio ovog rada sam implementirala mail klijent u programskom jeziku Python. Prilikom implementacije koristila sam sva tri gore navedena protokola i funkcije iz Pythonovih biblioteka u kojima su ti protokoli definirani. Poglavlje Implementacija mail klijenta u Pythonu sadrži slike tog mail klijenta, kao i pojašnjenja kako se određeni dio grafičkog sučelja napravio. Nadalje, priložene su i pojašnjene slike najbitnijih dijelova kôda iz kojih se vrlo jasno vidi praktična primjena navedenih protokola.

2 SMTP, POP3 i IMAP, protokoli za e-mail komunikaciju

1969. godine je američka vladina agencija ARPA uspjela spojiti nekoliko računala sa četiri odvojena sveučilišta u jednu mrežu koja je služila za razmjenu podataka. Tu mrežu su oni nazvali ARPANET. Bitan detalj kod kreiranja te mreže je bio da će ako dođe do oštećenja na nekom dijelu mreže komunikacija dalje nastaviti normalno kao da se ništa nije dogodilo jer će podaci pronaći neki drugi put da dođu do odredišta. Taj detalj je bio posebno zanimljiv američkoj vojsci, pa se 1983. godine ARPANET podijelio na civilni (ARPANET) i vojni (MILNET) dio. Ta mreža se kasnije razvila u internet kakav danas poznajemo.
Kako na internetu postoji sve više i više podataka potrebni su nam neki mehanizmi kojima će ti podaci putovati kroz mrežu i sigurno stizati na odredište. Jedni od tih mehanizama su i internet protokoli. Protokoli su skup pravila koja određuju kako dva uređaja ili programa međusobno komuniciraju. Oni određuju formate poruka i načine kako ih računala izmjenjuju. Ugrubo internet protokole možemo podijeliti na transportne protokole i na servisne protokole. Neki od transportnih protokola su: TCP/IP, UDP, SCTP, RSVP itd., a neki od servisnih protokola su: DNS, FTP, HTTP itd. i nama posebno zanimljivi SMTP, POP i IMAP.
SMTP, POP i IMAP su protokoli za e-mail komunikaciju. Oni služe za prijenos naših elektroničkih poruka s jednog računala na drugo. Sva tri protokola su se razvila osamdesetih godina dvadesetog stoljeća. Tada je internet još bio jako mala mreža i svi su bili "prijateljski" nastrojeni, pa su tim protokolima falili mehanizmi za autentifikaciju. Svi su slali poruke na međusobno povjerenje. Svi internet protokoli su definirani u nekom RFC-u. RFC (request for comment) je serija zabilješki o internetu u kojoj se definiraju novi protokoli i mehanizmi. Svaki RFC dobiva svoj broj. Jednom definirani i objavljeni RFC se ne smije mijenjati, pa ako se naprave neke nadogradnje na protokolu one se moraju definirati u potpuno novom RFC-u. Pa je tako SMTP prvi put definiran u RFC 821, POP u RFC 918, a IMAP u RFC 1064, dok su današnje verzije koje mi trenutno koristimo definirane redom u RFC 5321, RFC 2449 i RFC 2060.
U početku su svi internet protokoli bili vrlo jednostavni, pa su i SMTP, POP i IMAP mogli prenositi samo tekstualne podatke, ali kako se povećavala kompleksnost podataka na internetu postalo je jasno da se ti protokoli moraju nadograditi kako bi mogli slati i druge vrste podataka osim tekstualnih. U tu svrhu se razvio MIME (Multipurpose Internet Mail Extensions) standard koji dekodira binarne datoteke kako bi se mogle prenositi putem SMTP, POP i IMAP protokola.
U ovom poglavlju ću vam pokušati približiti SMTP, POP i IMAP protokole za e-mail komunikaciju.

2.1 SMTP protokol

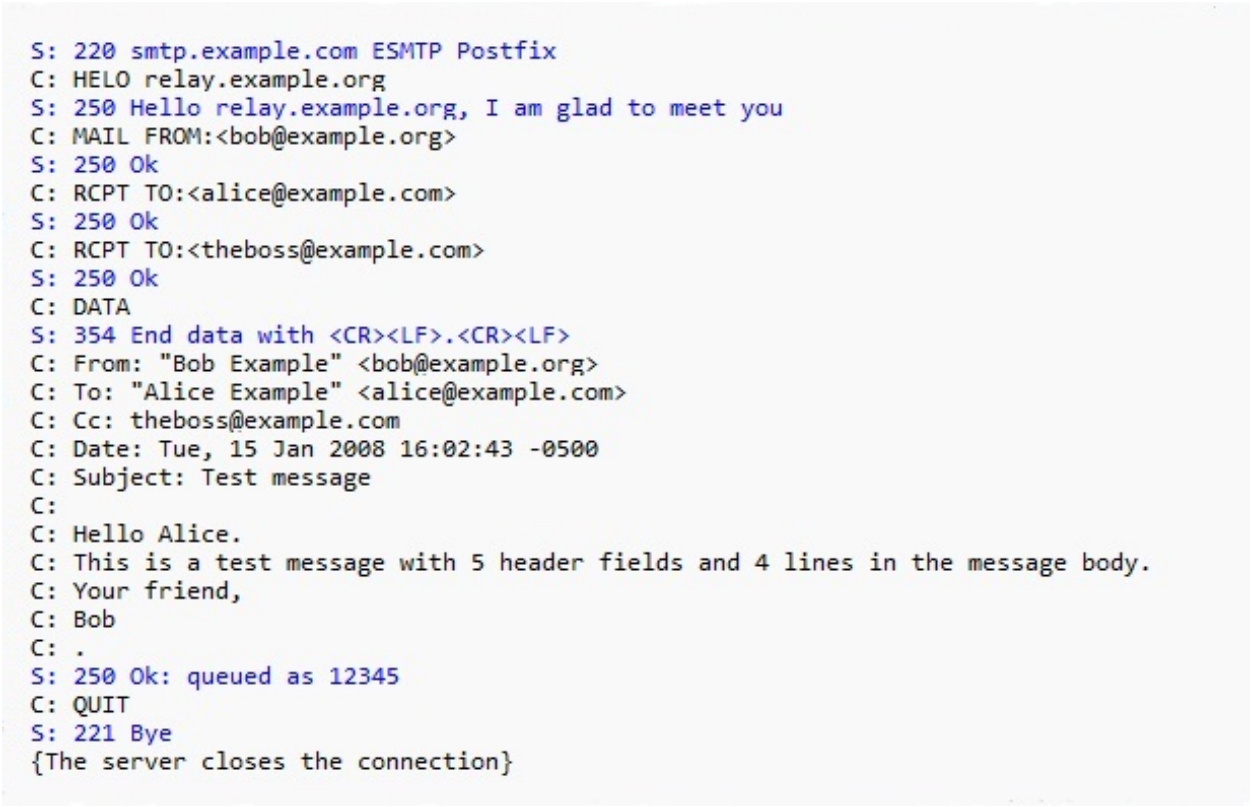
SMTP (Simple Mail Transfer Protocol) je internet standard koji služi za prijenos elektroničke pošte putem mreža zasnovanih na internet protokolima. Prvi puta je definiran u RFC821 1982. godine, a zadnji puta je ažuriran u RFC5321 2008. godine, gdje je dodan i ESMTP, tj. prošireni SMTP. SMTP protokol je određen za prijenos odlazne pošte i za to koristi TCP port 25.
1960 - ih godina su se koristili razni standardi za slanje elektroničke pošte s jednog računala na drugo. Ljudi su komunicirali jedni s drugima koristeći standarde napravljene posebno za operativne sustave koje je koristilo njihovo računalo i mogli su slati poruke samo ljudima koji su koristili isti takav sustav. Kako se sve više i više računala međusobno povezivalo (naročito u mreži američke vlade ARPANET) razvijali su se standardi koji su dozvoljavali slanje elektroničke pošte i korisnicima koji su koristili drukčiji operativni sustav od vas. U procesu razvoja tih standarda 1970-ih godina se razvio i SMTP. On je prvi puta implementiran 1971. u Mail Box protokolu i u SNDMSG programu. U to je vrijeme u ARPANET bilo spojeno samo 50-ak računala. 1973. godine je implementiran i u FTP Mailu i Mail Protocolu. Razvoj SMTP-a se nastavio kroz cijele 1970 - e sve dok se ARPANET 1980. nije pretvorio u moderni internet. Dvije godine nakon toga je Jon Postel objavio SMTP kao RFC821. Sendmail je bio jedan od prvih agenata u koji je implementiran SMTP protokol. Neki od poznatih SMTP servera su i *Postfix, qmail, Novell, GroupWise, Exim, Novell NetMail, Microsoft Exchange Server, Sun Java System Messaging Server.* SMTP protokol je prvo bio baziran samo na čistom ASCII tekstu i nije se dobro nosio sa binarnim datotekama. Kako bi se to popravilo razvijeni su standardi poput MIME koji dekodiraju binarne datoteke kako bi se mogle prenositi putem SMTP-a. Mnogi ljudi su doprinjeli specifikaciji SMTP-a, a među njima su i *Jon Postel, Eric Allman, Dave Crocker, Ned Freed, Randall Gellens, John Klensin* i *Keith Moore.*
SMTP je protokol baziran na tekstu u kojem pošiljatelj šalje poruku primatelju izdajući potrebne naredbe i podatke putem neke pouzdane veze, obično TCP-a. Prilikom razmjene podataka SMTP-om izmjenjuju se poruke SMTP klijenta i odgovarajući odgovori SMTP servera. Razmjena može uključivati nijednu ili više SMTP transakcija. SMTP transakcija se sastoji od tri slijeda naredbi i odgovora. To su:

1. MAIL naredba pomoću koje se određuje povratna adresa, tj. adresa na koju se šalje obavijest ako poruka ne stigne na odredište.

2. RCPT naredba kojom se određuje primatelj poruke. Ova naredba se može izdati više puta, za svakog primatelja po jednom.

3. DATA za slanje poruke teksta. Sastoji se od zaglavlja poruke i teksta poruke koji su odvojeni praznim redom. DATA je zapravo skupina naredbi na koju server odgovara dva puta: prvi put potvrđuje da je spreman primiti tekst, a drugi put (nakon primanja teksta) šalje poruku je li primio ili odbacio cijelu poruku.

Na sljedećoj slici je prikazan tipični primjer slanja poruke putem SMTP protokola na dvije adrese koje imaju istu domenu.



Slika 1. Primjer slanja poruke SMTP-om

Nakon što klijent uspostavi sigurnu vezu sa serverom razgovor započinje pozdravom servera koji šalje svoj FQDN (Fully Qualified Domain Name), a klijent odgovara naredbom HELO kojom se identificira. Koriste se prije opisane naredbe MAIL, RCPT i DATA na koje server odgovara 250 Ok jer je sve prošlo u redu. Ovdje se jasno vidi da je zaglavlje poruke odvojeno od tijela poruke praznim redom. Na kraju klijent poziva naredbu QUIT i završava razgovor. Prvotna specifikacija SMTP-a nije imala sposobnost ovjeriti autentičnost pošiljatelja poruke. U tu svrhu je razvijen SMTP-AUTH definiran u RFC 2554. Taj prošireni SMTP (ESMTP) omogućuje klijentu da uspostavi sigurnu vezu sa serverom i ovjeri autentičnost razmjene podataka. No, ni taj mehanizam nije dovoljan za zaštitu od neželjene pošte, pa postoji Anti-Spam Research Group koja pokušava napraviti bolji i sigurniji SMTP protokol koji će prepoznavati i odbijati neželjenu poštu.

2.1.1 SMTP i Python

Python ima ugrađenu implementaciju SMTP protokola u svoju standardnu biblioteku koja je spremljena pod nazivom `smtplib` i omogućuje osnovne operacije sa SMTP protokolom. Taj modul su napravili The Dragon De Monsyne, Eric S. Raymond, Carey Evans i Gerhard Haering kao modifikaciju biblioteke HTTP lib.
smtplib sadrži klase za iznimke i pogreške, klasu SMTP, klasu SMTP_SSL koja nasljeđuje funkcije iz klase SMTP i dodaje nove vezane za sigurnost, te klasu LMTP (Local Mail Transfer Protocol) koja također nasljeđuje funkcije iz klase SMTP. Klasa SMTP je najbitnija i u njoj su sadržane naredbe, tj. funkcije koje korisnici najčešće koriste. U ovom potpoglavlju ću se osvrnuti na najbitnije funkcije u klasi SMTP, a to su: connect, send, getreply, helo, ehlo, noop, mail, rcpt, data, verify, login, starttls, sendmail, close i quit.
Funkcija connect je definirana sa `connect(self, host='localhost', port = 0)` i ona spaja korisnika na zadani host i port. Ako ime hosta završava sa : i nekim brojem, a port nije definiran, funkcija connect će automatski dio iza dvotočke shvatiti kao port na koji se treba spojiti.
Funkcija send je definirana sa `send(self, str)`. Njezin posao je poslati str (tj. neki string) serveru.
Funkcija getreply je definirana sa `getreply(self)`. Ona dohvaća odgovor servera. Odgovor servera dolazi kao uređeni par. Na prvom mjestu se nalazi kôd odgovora servera, npr. '250' ako je sve u redu ili '500' ako dođe do pogreške u sintaksi. Ako klijent ne može pročitati odgovor ova će vrijednost biti postavljena na -1. Na drugom mjestu odgovora je string koji ide uz određeni kôd. Ako je to dugačak tekst koji se sastoji iz više redova ovdje se on interpretira kao jedan string koji se sastoji iz više redova.
Funkcija helo je definirana sa `helo(self, name=' ')`. Ta funkcija započinje razgovor sa serverom. Kada koristimo funkciju helo možemo iza te naredbe definirati našu domenu da server zna tko smo mi, npr. HELO mailhost2. cf.ac.uk. Server na tu naredbu odgovara svojim "punim imenom", tj. pošalje svoj FQDN.
Funkcija ehlo je definirana sa `ehlo(self, name=' ')`. To je gotovo ista funkcija kao i helo, ali na ovaj zahtjev server odgovara svojim FQDN i uz to šalje klijentu i popis svojih proširenih mogućnosti ako one postoje.
Funkcija noop je definirana sa `noop(self)`. Noop znači no operation i to je funkcija koja ne radi ništa. Ali i ona je vrlo korisna kada npr. trebamo čekati na neku konekciju ili kada želimo da klijent klikne neku tipku za nastavak, sve dok se to ne dogodi ova funkcija održava program na životu, ali ništa drugo ne radi.
Funkcija mail je definirana sa `mail(self,sender,options=[])`. To je SMTP naredba mail koju sam već pojasnila u poglavlju SMTP protokol.
Funkcija rcpt je definirana sa `rcpt(self,recip,options=[])` i to je također kao i mail naredba objašnjeno prije u tekstu. U pythonu ovom naredbom definiramo samo jednog primatelja, pa ju možemo pozvati onoliko puta koliko imamo primatelja.
Funkcija data je definirana sa `data(self,msg)`. Šalje poruku msg serveru i dobiva dva puta odgovor od servera što je pojašnjeno prije u tekstu.
Funkcija verify je definirana sa `verify(self, address)`. Ta funkcija provjerava ispravbost poruke address.

Funkcija login je definirana sa `login(self, user, password)`. Ovom funkcijom se klijent autentificira kada se spaja na SMTP server koji zahtjeva autentifikaciju. Funkcija zahtjeva argumente user, tj. korisničko ime i password, tj. lozinku. Ako prije pozivanja ove funkcije nisu pozvane funkcije HELO ili EHLO ona prvo pokuša uspostaviti ESMTP EHLO vezu sa serverom. Ako autentifikacija ne uspije ova funkcija će vratiti jednu od sljedećih grešaka: *SMTPHeloError*, ako server nije pravilno odgovorio na funkciju helo; *SMTPAuthenticationError*, ako server nije prihvatio unešenu kombinaciju korisničkog imena i lozinke ili *SMTPException*, ako nije pronađena odgovarajuća metoda autentifikacije.
Funkcija starttls je definirana sa `starttls(self, keyfile = None, certfile = None)`. Ova funkcija stavlja SMTP vezu u TLS način rada. Ako prije pozivanja ove funkcije nisu pozvane funkcije HELO ili EHLO ova funkcija će pozvati ESMTP EHLO. Ako server podržava TLS starttls funkcija će šifrirati ostatak SMTP razgovora. Ako funkciji proslijedimo parametre keyfile i certfile mogu se provjeriti identiteti SMTP servera i klijenta.
Funkcija sendmail je definirana sa `sendmail(self, from_addr, to_addrs, msg, mail_options=[],rcpt_options=[])`. Ova funkcija obavlja cijeli prijenos poruke. Njeni argumenti su: from_addr, adresa s koje se šalje poruka; to_addr, lista adresa na koje se šalje poruka (ako je unešen samo jedan string funkcija će ga gledati kao listu s jednom adresom); msg, poruka koja se šalje; mail_options, popis ESMTP opcija (npr. 8bitmime) za mail naredbu; rcpt_options, popis ESMTP opcija za svaku rcpt naredbu. Ako prije poziva ove funkcije nisu pozvane funkcije HELO ili EHLO, ona će prvo pokušati pozvati ESMTP EHLO. Ako to ne uspije, pokušat će HELO naredbu i tada će ESMTP opcije biti zanemarene. Ova funkcija vraća `True` ako je poruka stigla bar na jednu adresu. Poslije toga vraća riječnik u kojem svako polje pripada po jednom primatelju koji nije dobio poruku. Svako polje se sastoji od uređenog para SMTP koda pogreške i odgovarajuće poruke pogreške. Ako su svi primatelji primili poruku funkcija će vratiti prazan riječnik.
Funkcija close je definirana sa `close(self)`. Ona zatvara vezu sa SMTP serverom, dok funkcija quit koja je definirana sa `quit(self)` prekida sve razgovore i zatvara vezu sa serverom.

2.2 POP protokol

POP (Post Office Protocol) je internet standard koji koriste e-mail klijenti kako bi dohvatili poruke sa udaljenog servera, obično putem TCP/IP veze. POP protokol se razvijao kroz nekoliko verzija, ali tek je treća verzija, tj. POP3 postala standard za dohvaćanje poruka. Većina webmail servisa podržava POP3 protokol, a među njima su i Hotmail, Gmail i Yahoo! Mail.
POP1 je definiran u RFC 918, POP2 u RFC 937. POP3 je prvotno definiran u RFC 1081. Aktualna verzija POP3 protokola je definirana u RFC 1939 i nadopunjena u RFC 2449. Početna verzija POP3 protokola je dopuštala samo USER/PASS autentikaciju, dok sadašnja verzija podržava nekoliko metoda autentikacija kako bi osigurao nekoliko razina zaštite e-mail klijentima u pristupanju svojim porukama. Postoji i prijedlog za POP4 protokol koji bi popravio neke trenutne nedostatke POP3 protokola (npr. podržavao bi pristup svim dijelovima pošte, a ne samo dolaznoj pošti), ali od 2003. godine se nije ništa ozbiljnije radilo na implementaciji POP4 protokola.
Općenito, e-mail klijenti koji koriste POP protokol se spoje na server, dohvate poruke, spreme ih na klijentovo računalo, obrišu ih sa servera i odjave se sa servera. POP3 server sluša na portu 110. Nakon ostvarenja veze server zahtjeva kriptiranu komunikaciju i za to koristi STLS ako je podržan ili POP3S koji koristi TLS ili SSL na portu 995 (npr. Gmail). Poruke u sandučiću su označene rednim brojevima ili jedinstvenim identifikacijskim brojevima. Ako su označene rednim brojevima onda se resetiraju prilikom svakog spajanja na server, a ako su označene identifikacijskim brojevima oni ostaju jednaki prilikom svakog spajanja na server i tada smo sigurni da uvijek gledamo istu poruku ako otvorimo poruku s istim brojem. Klijent može označiti broj poruke koju želi obrisati i kada se odjavi sa servera ta poruka se briše iz sandučića.
Neki od mail servera koji koriste POP3 su: *Dovecot, Mailtraq, Nginx, qmail-pop3d, RePOP, Zimbra.*

2.2.1 POP3 i Python

Python u svojoj standardnoj biblioteci ima ugrađen modul `poplib` koji omogućuje korisnicima da lako koriste sve naredbe koje POP protokol dopušta. Taj modul su napravili *David Ascher, Piers Lauder* i *Hector Urtubia.*
poplib modul sadrži klase za pogreške, klasu POP3, te klasu POP3_SSL koja nasljeđuje funkcije iz klase POP3 i dodaje još neke vezane za sigurnost. Klasa POP3 je najbitnija i u njoj su sadržane naredbe, tj. funkcije koje korisnici najčešće koriste. U ovom potpoglavlju ću se osvrnuti na najbitnije funkcije unutar klase POP3 i pobliže pojasniti što koja funkcija radi. Najbitnije funkcije u POP3 klasi su: user, pass_, stat, list, retr, dele, noop, rset, quit, apop, top i uidl.
Funkcija user je definirana sa `user(self, user)`. Zahtjeva korisničko ime od korisnika, a funkcija pass_ koja je definirana sa `pass_(self, pswd)` zahtjeva korisnikovu lozinku i pomoću ta dva podatka se spaja na POP server.
Funkcija stat je definirana sa `stat(self)`. Pozivanjem ove funkcije tražimo status poštanskog sandučića. Odgovor dobijemo u obliku uređenog para u kojem je na prvom mjestu broj

poruka koje se nalaze u našem sandučiću, a na drugom mjestu ukupna veličina svih poruka u byteima.
Funkcija list je definirana sa `list(self, which=None)`. Ova funkcija vraća odgovor u obliku (response, ['mesg_num octets', ...], octets), gdje je response odgovor servera u kojem napiše je li sve proslo u redu te stanje sandučića koje bismo dobili da pozovemo funkciju stat; ['mesg_num octets',...] je lista poruka koja za svaku poruku napiše broj poruke i njezinu veličinu u byteima, a octets je broj okteta (npr. ako u se u sandučiću nalaze 3 poruke taj broj će iznositi 24). Primjer jednog odgovora: `('+OK 3 messages (5675 bytes)', ['1 2395', '2 1626','3 1654'], 24)`.
Funkcija retr je definirana sa `retr(self, which)`. Vraća cijelu poruku pod brojem koji je definiran argumentom which. Ova funkcija šalje odgovor u obliku ['response', ['line', ...], octets] slično kao i funkcija list, jedino je razlika što u drugom članu odgovora dobijemo linije poruke, a ne popis poruka. Kada pozovemo ovu funkciju server će označiti da je poruka pročitana.
Funkcija dele je definirana sa `dele(self, which)`. Ona označava poruku pod brojem koji je definiran argumentom which zha brisanje. Obično se ta poruka obriše tek kada se odjavimo sa servera.
Funkcija noop je definirana sa `noop(self)` i ona isto kao i ta funkcija kod SMTP protokola ne radi ništa, nego samo održava program na životu dok se ne klikne neka tipka na tipkovnici ili dok se ne pozove neka druga funkcija.
Funkcija rset je definirana sa `rset(self)`. Nju koristimo kada želimo "odznačiti" sve poruke koje smo označili za brisanje.
Funkcija quit je definirana sa `quit(self)`. Ovom funkcijom se odjavljujemo sa servera. Server tada napravi sve potrebne promjene (npr. obriše poruke koje smo označili za brisanje i otključa sandučić kako bi se s nekog drugog mjesta opet mogli prijaviti u njega i na kraju prekine vezu sa klijentom.
Funkcija apop je definirana sa `apop(self, user, secret)`. apop je skraćenica za authorized POP. Argumenti ove funkcije su user, tj. korisničko ime da server zna s kime razgovara i secret, tj. tajni kod koji znaju samo klijent i server i tako prenose poruke da ih nitko drugi ne može pročitati.
Funkcija top je definirana sa `top(self, which, howmuch)`. Ova funkcija vrati zaglavlje poruke pod brojem zadanim argumentom which i prvih nekoliko redova, gdje je broj tih redova zadan argumentom howmuch. Odgovor je u obliku ['response', ['line', ...], octets] isto kao i kod funkcije retr.
Funkcija uidl je definirana sa `uidl(self, which=None)`. Vraća listu jedinstvenih identifikacijskih brojeva poruka. Ako u argument which napišemo neki broj onda će ova funkcija vratiti jedinstveni identifikacijski broj poruke koju smo označili. Ako which ostavimo prazan onda ce vratiti listu identifikacijskih brojeva svih poruka koje se nalaze u sandučiću i to u obliku ['response', ['mesgnum uid', ...], octets], gdje srednji član označava tu listu uređenih parova gdje je prvi broj broj poruke, a drugi identifikacijski broj te poruke.

2.3 IMAP protokol

IMAP (Internet Message Access Protocol) je isto kao i POP protokol za dohvaćanje poruka iz sandučića. Definirao ga je Mark Crispin 1986. godine kao suprotnost POP protokolu koji je tada bio u širokoj upotrebi. IMAP protokol pristupa udaljenom poštanskom sandučiću i na tom udaljenom sandučiću izvodi potrebne operacije nad porukama, za razliku od POP protokola koji preuzme sve poruke iz sandučića na računalo i odjavi se. IMAP je prije bio poznat pod nazivima: *Internet Mail Access Protocol, Interactive Mail Access Protocol* i *Interim Mail Access Protocol*.
Izvorni Interim Mail Access Protocol je bio implementiran kao klijent na Xerox Lisp stroju i na TOPS-20 serveru. Do danas se nije sačuvala niti jedna kopija specifikacije tog protokola. Interim protokol je vrlo brzo zamijenjen Interactive Mail Acces Protocol-om (IMAP2). On je prvi puta definiran 1988. godine u RFC 1064, a ažuriran je 1990. u RFC 1176. IMAP2 je prvi uveo zastavice naredbi i odgovora i to je bila prva verzija IMAP protokola koja je javno distribuirana. IMAP3 je izumrla i vrlo rijetka verzija IMAP protokola. Definiran je 1991. u RFC 1203 i nikada nije bio prihvaćen na tržištu. IESG (Internet Engineering Steering Gruop) ga je proglasio "povijesnim protokolom" već 1993. godine. Kada se razvio MIME, IMAP2 je proširen tako da podržava MIME i dodana je funkcija upravljanja datotekama unutar sandučića koja je nedostajala u IMAP2 protokolu. Ta proširena verzija IMAP2 protokola je nazvana IMAP2bis i objavljena je samo kao skica. Nikada nije bila definirana u nekom RFC-u. Početkom 1990-ih u IETF-u (Internet Engineering Task Force) je osnovana IMAP Working Group koja je preuzela odgovornost za stvaranje dizajna IMAP2bis-a. Oni su odlučili preimenovati IMAP2bis u IMAP4, kako se ne bi miješao sa IMAP3 verzijom. Od tada akronim IMAP stoji za Internet Message Access Protocol. IMAP4 je definiran u prosincu 1994. godine u RFC 1730, a nadograđen je u RFC 2060 u prosincu 1996. godine. Postoji vrlo malo implementacija klijenata i servera koji koriste IMAP4 definiran u RFC 1730 zbog njegovog kratkog vijeka.
Neke prednosti IMAP protokola nad POP protokolom:

- prilikom korištenja POP protokola klijent se samo nakratko spoji na server, preuzme poruke i odjavi se, a pri korištenju IMAP protokola klijent je spojen na server dok god to želi i to ubrzava čitanje poruka ako smo pristupili nekom jako velikom sandučiću (brže je vrijeme odgovora na zahtjev jer smo već spojeni, ne moramo se ponovo spajati).

- prilikom korištenja POP protokola samo jedan klijent može gledati sandučić, dok kod IMAP protokola više klijenata odjednom može pristupiti istom sandučiću i postoje mehanizmi koji omogućavaju klijentima da primijete promjene koje prave drugi spojeni klijenti.

- obično se poruke prenose internetom u MIME formatu i tako imaju oblik stabla kojem listovi predstavljaju određene tipove podataka koji se nalaze u poruci (tekst, html, slika itd.). IMAP4 omogućuje čitanje tih dijelova posebno, dok POP3 skine cijelu poruku i onda se koriste neke druge naredbe koje parsiraju poruku.

- IMAP koristi zastavice koje omogućuju klijentima da vide u kojem su stanju poruke (označene za brisanje, nove, pročitane itd.), pa olakšavaju klijentima upravljanje svojim

porukama, a POP to ne podržava. IMAP4 omogućuje i definiranje vlastitih zastavica (npr. omiljene poruke). Jedan od servisa koji omogućuje kreiranje vlastitih zastavica je i Gmail.

- korisnici koji koriste IMAP4 mogu stvarati, preimenovati ili brisati datoteke u svom sandučiću i premještati poruke između tih datoteka.

- IMAP omogućuje klijentima pretraživanje poruka u sandučiću po raznim kriterijima bez da skidaju sve poruke na računalo i onda ih pretražuju kao što je to slučaj kod POP protokola.

2.3.1 IMAP i Python

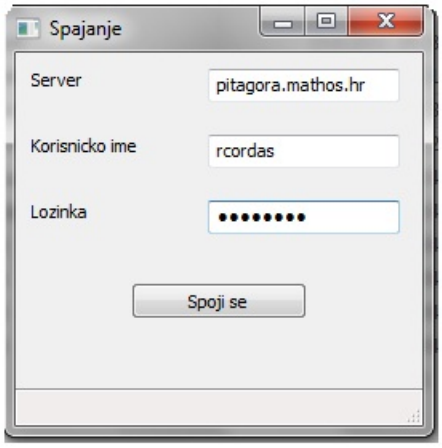
Python u svojoj standardnoj biblioteci ima ugrađen modul `imaplib` koji omogućuje korisnicima lako korištenje svih naredbi koje IMAP protokol dopušta. Taj modul su izradili i dopunili: *Piers Lauder, Donn Cave, Anthony Baxter, Tino Lange, Andreas Zeidler, Rick Holbert* i *Tomas Lindroos*.
imaplib modul sadrži klase IMAP4, error, abort, readonly te klase IMAP4_SSL i IMAP4_stream koje nasljeđuju sve funkcije klase IMAP4 i dodaju još neke. IMAP4 je najbitnija klasa ovog modula i u ovom potpoglavlju ću se osvrnuti na najbitnije funkcije iz te klase, a to su: open, read, readline, send, shutdown, recent, response, append, capability, authenticate, close, copy, create, delete, expunge, fetch, list, login, logout, noop, partial, rename, search, select, status i store.
Funkcija open je definirana sa `open(self, host = '', port = IMAP4_PORT)`. Stvara vezu sa udaljenim serverom imena host i na portu koji je definiran atributom port. Ako nijedan od ta dva argumenta nije definiran ova funkcija će za njihove vrijednosti uzeti localhost i standardni IMAP4 port. Ova veza će se koristiti za radnje read, readline, send i shutdown.
Funkcija read je definirana sa `read(self, size)`. Ona uzima argument size i očitava toliku količinu podataka sa servera koja je zadana argumentom size. Size je veličina podataka u byteima.
Funkcija readline je definirana sa `readline(self)`. Čita jedan red podataka sa udaljenog servera.
Funkcija send je definirana sa `send(self, data)`. Ova funkcija šalje podatke koji su predani funkciji u argumentu data na udaljeni server na koji smo se prije spojili.
Funkcija shutdown je definirana sa `shutdown(self)`. Ova funkcija ne prekida vezu sa serverom već samo zatvara sve kanale za prijenos podataka koji su otvoreni funkcijom open.
Funkcija recent je definirana sa `recent(self)`. Vraća najnovije odgovore ako postoje, a ako ne onda ova funkcija pozove naredbu NOOP kako bi osvježila server. Odgovor je u obliku `(typ, [data]) = <instance>.recent()`, ako nema poruka onda je vrijednost argumenta data None, a u suprotnom je data zapravo lista najzadnjih pozvanih naredbi i odgovora.
Funkcija response je zadana sa `response(self, code)`. Ova funkcija vraća podatke za odgovor kojem je kod zadan argumentom code ako oni postoje, u suprotnom vrati vrijednost None.
Funkcija append je zadana sa `append(self, mailbox, flags, date_time, message)`.

Ovom funkcijom dodajemo poruku u zadani sandučić. Argument mailbox određuje datoteku u sanučiću u koju ćemo dodati poruku; argument flags nam dopušta da odaberemo zastavice koje želimo (npr. poruka je pročitana, označena za brisanje itd.); argument date_time upisuje datum i vrijeme dodavanja poruke i message je sama poruka koju dodajemo u datoteku.
Funkcija capability je definirana sa `capability(self)`. Ovom funkcijom zahtjevamo od servera da nam pošalje skup svih mogućih naredbi koje on podržava.
Funkcija authenticate je zadana sa `authenticate(self, mechanism, authobject)`. Ova funkcija zahtjeva obradu odgovora. Argumentom mechanism određujemo koji će se mehanizam autentikacije koristiti. Taj mehanizam mora biti iz popisa mogućnosti servera koje dobijemo ako pozovemo funkciju capability. Authobject mora biti neka od naredbi za koju mi zapravo možemo provesti autentifikaciju odgovora.
Funkcija close je definirana sa `close(self)`. Ovom funkcijom zatvaramo trenutno označenu datoteku u sandučiću. Kada pozovemo ovu naredbu sve poruke koje su označene za brisanje se brišu. Preporuča se pozivanje ove funkcije prije pozivanja funkcije logout.
Funkcija copy je definirana sa `copy(self, message_set, new_mailbox)`. Kopira skup poruka koje su definirane argumentom message_set u datoteku u sandučiću koja je definirana argumentom new_mailbox.
Funkcija create je definirana sa `create(self, mailbox)`. Ovom funkcijom stvaramo novu datoteku u sandučiću koja će biti spremljena pod nazivom koji je dan argumentom mailbox. Npr. ako hoćemo napraviti novu datoteku u sandučiću u koju ćemo spremati najdraže poruke možemo napisati create(favorites).
Funkcija delete je definirana sa `delete(self, mailbox)`. Ova funkcija služi za brisanje cijele datoteke iz sandučića. Ime te datoteke je dano argumentom mailbox.
Funkcija expunge je definirana sa `expunge(self)`. Kada pozovemo ovu funkciju trajno brišemo sve poruke iz datoteke u kojoj se nalazimo koje su označene za brisanje. Za svaku na ovaj način obrisanu poruku se stvori odgovor 'expunge'.
Funkcija fetch je definirana sa `fetch(self, message_set, message_parts)`. Ova funkcija dohvaća dijelove poruke. Odgovor dobivamo u obliku (typ, [data, ...]) = <instance>.fetch(message_set, message_parts), gdje bi argument message_parts trebao biti string, tj. skup znakova kojim ćemo opisati koji dio poruke želimo dohvatiti npr. '(UID BODY[TEXT])'. U tom slučaju bismo dohvatili jedinstveni identifikacijski broj poruke i tekstualni dio poruke.
Funkcija list je definirana sa `list(self, directory='""', pattern='*')`. Pozivom ove funkcije dobijemo izlistanje svih datoteka koje imamo u sandučiću u obliku liste. Npr. jedan sandučić može sadržavati datoteke inbox, sent mail, outbox i deleted.
Funkcija login je definirana sa `login(self, user, password)`. Ova funkcija nas spaja sa serverom. Prosljeđujemo joj korisničko ime putem argumenta user i našu lozinku putem argumenta password. A funkcija logout koja je definirana sa `logout(self)` prekida sve veze sa serverom.
Funkcija noop je definirana sa `noop(self)` i isto kao i kod SMTP i POP protokola to je funkcija koja ne radi ništa, ali u nekim slučajevima može biti korisna.
Funkcija partial je definirana sa `partial(self, message_num, message_part, start, length)`. To je zapravo modificirana funkcija fetch. Ona dohvaća dio poruke s brojem koji je definiran argumentom message_num, samo što ovdje još moramo definirati koja će biti

početna točka od koje ćemo početi čitati poruku (argument start) i koju ćemo količinu podataka dohvatiti (argument length).
Funkcija rename je definirana sa `rename(self, oldmailbox, newmailbox)`. To je funkcija koja služi za preimenovanje datoteka u poštanskom sandučiću. Ovoj funkciji prosljeđujemo dva stringa, prvi je dan argumentom oldmailbox i određuje datoteku kojoj želimo promijeniti ime, a drugi je dan argumentom newmailbox i to je zapravo novo ime koje želimo dati označenoj datoteci.
Funkcija search je definirana sa `search(self, charset, *criteria)`. Ovo je funkcija za pretraživanje sandučića. Odgovor dobijemo u obliku (typ, [data]) = <instance>.search(charset, criterion, ...), gdje je data lista brojeva poruka u sandučiću koje zadovoljavaju zadane kriterije pretraživanja.
Funkcija select je definirana sa `select(self, mailbox='INBOX', readonly=False)`.
Ovom funkcijom označavamo datoteku u sandučiću s kojom želimo raditi. Ako ne upišemo argumente funkcija će shvatiti kao da smo odabrali inbox, tj. datoteku sa pristiglim porukama.
Funkcija status je definirana sa `status(self, mailbox, names)`. Kao što kod POP protokola možemo zatražiti stanje našeg sandučića, tj. broj poruka koje se nalaze u njemu i njihovu ukupnu veličinu, tako i ovdje ovom funkcijom možemo tražiti stanje, ali ne cijelog sandučića nego jedne datoteke u njemu koja je određena argumentom names.
Funkcija store je definirana sa `store(self, message_set, command, flags)`. Inače, kada mijenjamo zastavice na porukama, tj. na neki način obilježavamo da je na pojedinim porukama došlo do promjene, te promjene se spreme tek kada se odjavimo sa servera. Ali ako želimo da se na svim ili samo na nekim porukama (to je određeno argumentom message_set) promjene spreme baš kada mi to želimo možemo pozvati ovu funkciju.

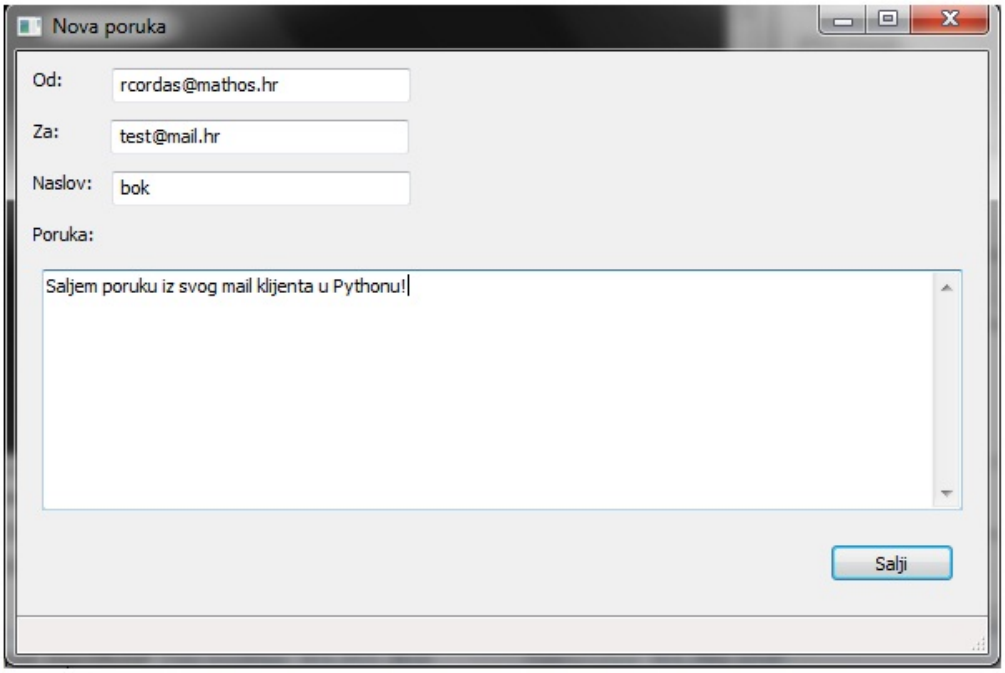
3 Implementacija mail klijenta u Pythonu

U sklopu praktičnog dijela ovoga rada implementirala sam e-mail klijent u programskom jeziku Python. Kako bi mail klijent što bolje izgledao i bio što lakši za upotrebu napravila sam grafičko sučelje i u tu svrhu sam koristila wxPython. wxPython je alat za pravljenje grafičkog sučelja u Pythonu. Implementiran je u Python kao proširenje kako bi olakšao korisnicima pravljenje grafičkog sučelja. Nakon instalacije wxPython proširenja možemo uključiti wx biblioteku koja nam omogućuje stvaranje prozora, gumbova, polja za upisivanje teksta itd. Na sljedećim slikama možete vidjeti kako izgleda grafičko sučelje napravljeno u wxPythonu.

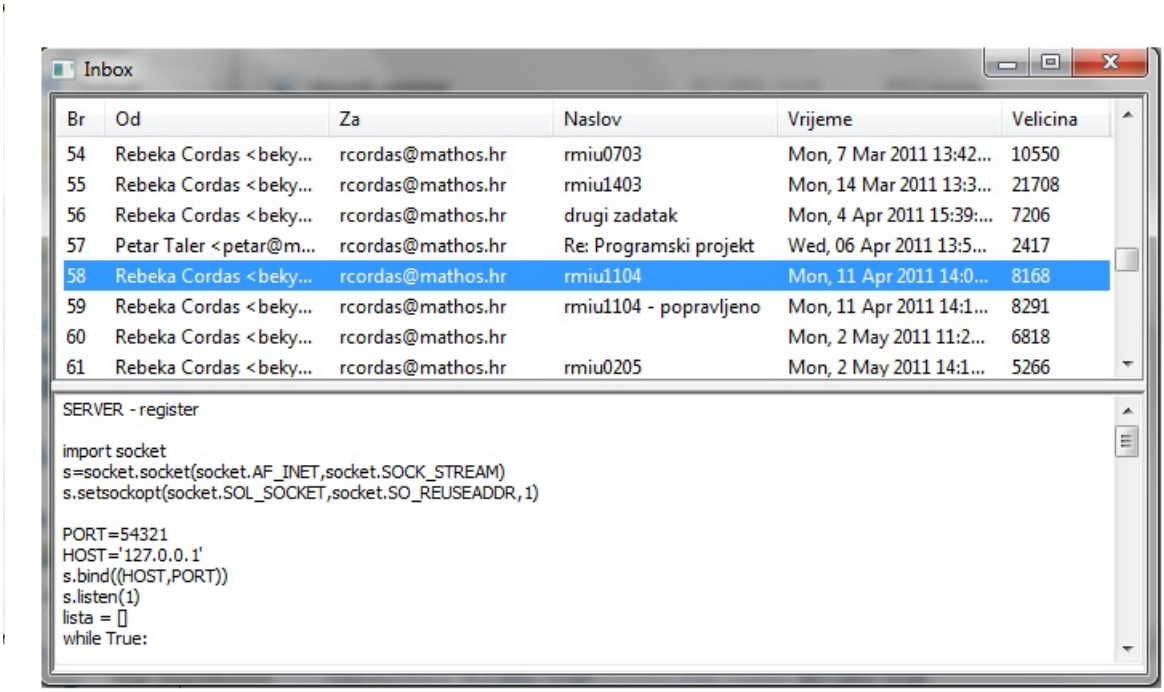


Slika 2. Prozor za spajanje na server

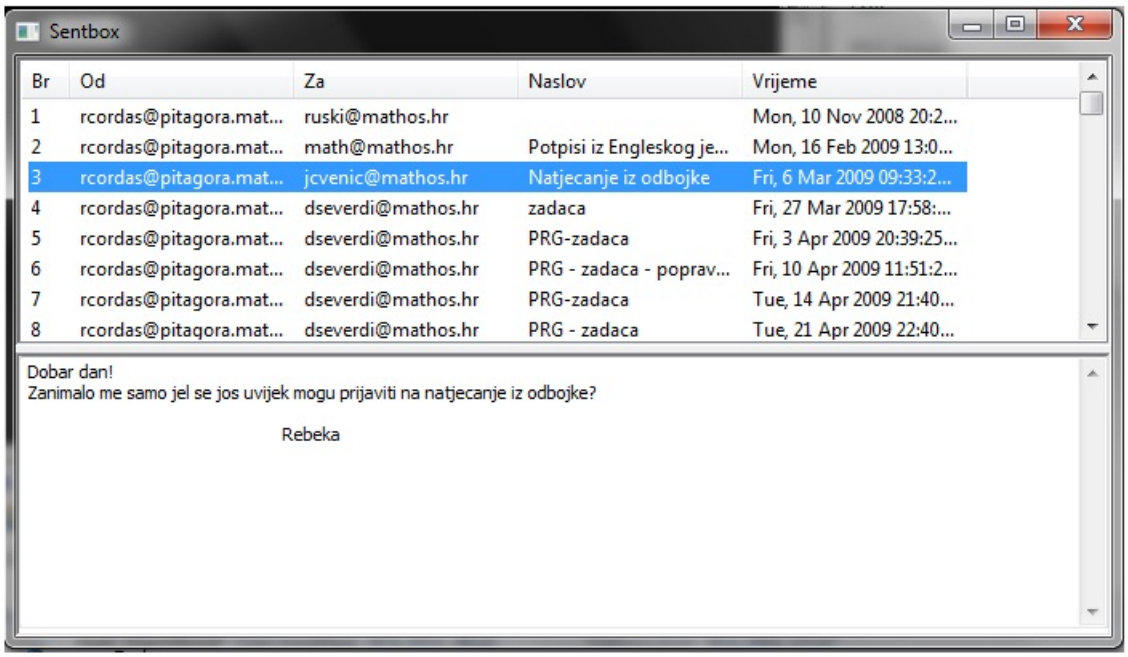
Slika 3. Izbornik



Slika 4. Prozor za slanje nove poruke



Slika 5. Popis primljenih poruka

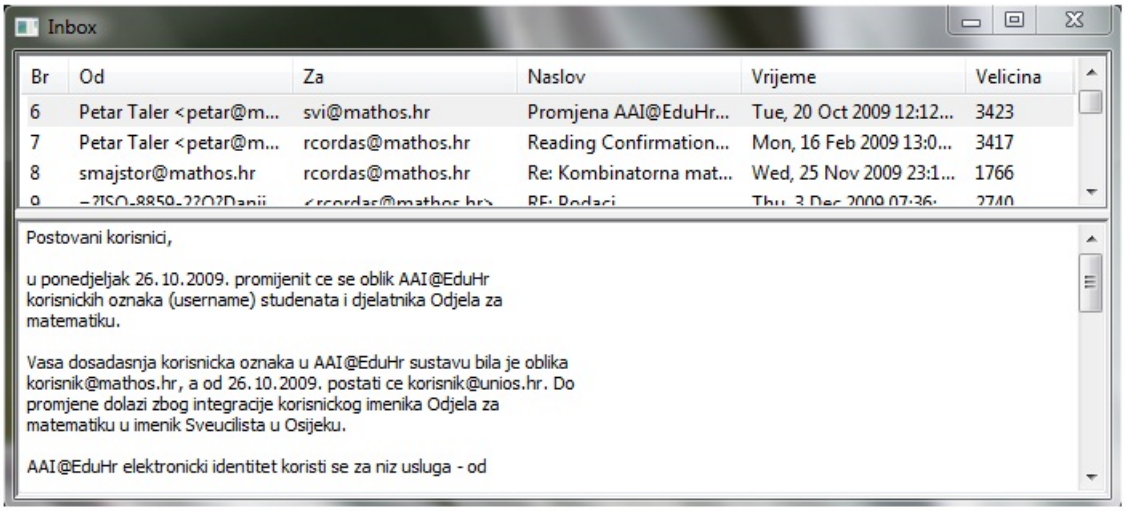


Slika 6. Popis poslanih poruka

Na slici 2 se vidi prozor koji se otvori kada se program pokrene. U taj prozor upisujemo podatke u polja korisničko ime i lozinka i spajamo se na server koji upišemo u polje server. Klikom na gumb 'Spoji' spajamo se na server. Ako smo nekim slučajem unijeli pogrešne podatke u status baru će se ispisati poruka 'neuspjela autentifikacija' i polja za upis će se očistiti i biti spremna za unos novih podataka. Neke od naredbi iz wx biblioteke koje su korištene u ovom dijelu programa su `wx.Frame`, `CreateStatusBar`, `wx.Panel`, `wx.BoxSizer`, `wx.StaticText`, `wx.TextCtrl`, `wx.Button`.
wx.Frame je naredba kojom napravimo prozor u koji ćemo slagati sve ostale objekte. Na dnu prozora možemo stvoriti statusbar, tj. traku u kojoj će se ispisivati obavijesti i to postižemo naredbom CreateStatusBar. Naredbom wx.Panel stvaramo panel koji možemo shvatiti kao neku 'radnu plohu' na koju možemo slagati 'kutije' i gumbe koje stvaramo naredbama wx.SetBoxSizer i wx.Button. Kutije mogu biti orijentirane horizontalno i vertikalno. U prozoru na slici 2 postoji jedan panel koji zauzima cijeli prozor i na njega je postavljena jedna vertikalna kutija u kojoj su jedna ispod druge poslagane tri horizontalne kutije i na kraju jedan gumb. Kao što na panel možemo postavljati objekte tako i u kutije možemo slagati pojedine objekte. U ovom slučaju smo za to koristili naredbe wx.StaticText i wx.TextCtrl. Naredbom wx.StaticText smo dobili teskt Server, Korisnicko ime i Lozinka. Dakle, to je naredba kojom dobijemo jedan redak teksta gdje želimo i najčešće se koristi u ovakvim situacijama kada trebamo opisati što treba unijeti u određeno polje. Polja za unos teksta pokraj redaka Server, Korisnicko ime i Lozinka su nastala pozivom naredbe wx.TextCtrl. Dakle, to je naredba kojom pravimo polje u koje je moguć unos teksta s kojim ćemo mi kasnije nešto raditi. wx.TextCtrl ima puno dodatnih opcija. Jednu od njih sam ovdje iskoristila prilikom upisivanja teksta u polje Lozinka. Ako unutar zagrada nakon

wx.TextCtrl upišemo style=wx.TE_PASSWORD prilikom upisivanja teksta u to polje neće se prikazivati slova i brojevi nego kružići koji čuvaju tajnost vaše lozinke. Klikom na gumb 'Spoji' zapravo pozivamo posebno definiranu funkciju koja spaja korisnika na server. Ova operacija je omogućeno zbog naredbe Bind koja povezuje određen gumb sa nekom radnjom. Na slici 3 je prikazan prozor koji se otvori ako spajanje uspije. To je zapravo jedan izbornik koji nam omogućava snalaženje u našem poštanskom sandučiću. Klikom na gumb 'Nova poruka' otvorit će se novi prozor koji omogućava slanje poruke, gumb 'Pristigle poruke' nam otvara popis svih primljenih poruka koje imamo u sandučiću, gumb 'Poslane poruke' otvara popis svih poruka koje je korisnik poslao i klikom na gumb 'Odjava' prekidaju se sve veze sa serverom i gasi se program. U stvaranju ovog prozora su korištene naredbe `wx.Frame`, `wx.Panel`, `wx.Button` te `wx.Boxsizer`. Kao i u prethodnom slučaju postoji jedan panel koji zauzima cijeli prozor, na njega je postavljena jedna vertikalna kutija i na nju su poslagane četiri horizontalne kutije u koje su postavljeni gumbovi. Gumbovi su funkcijom Bind vezani za nove funkcije koje obavljaju daljnje radnje.
Slika 4 prikazuje prozor koji dobijemo klikom na gumb 'Nova poruka' u izborniku sa slike 3. SMTP protokol zahtjeva podatak o pošiljatelju poruke, pa zbog toga ovdje postoji polje pod nazivom 'Od:'. Sljedeća polja su uobičajena polja koja se traže prilikom slanja e-mail poruke putem bilo kojeg servisa. To su polja 'Za', 'Naslov' i 'Poruka'. Na kraju stoji gumb 'Salji'. I ovdje su kao i ranije korištene naredbe `wx.Frame`, `wx.StaticText`, `wx.Textctrl`, `wx.BoxSizer`, `wx.Button` i `wx.Panel`. Na panel je postavljena jedna vertikalna kutija u kojoj su jedna ispod druge poslagane horizontalne kutije na kojima se nalaze polja napravljena naredbama wx.StaticText i wx.TextCtrl, te gumb 'Salji'. I ovdje je korištena jedna jako korisna opcija funkcije wx.Textctrl. Naime, ako u zagrade iza naredbe wx.TextCtrl upišemo style=wx.TE_MULTILINE dobit ćemo polje u koje je moguće upisati tekst od više redaka i na desnoj strani se pojavljuje traka za pomicanje, tj. scrollbar koja nam omogućuje pomicanje napisanog teksta gore - dolje ako upisani tekst prelazi granice prozora. Na taj način možemo upisivati dugačak tekst koji će i dalje biti pregledan. Klikom na gumb 'Salji' poruka se šalje na upisanu adresu i javlja se obavijest da je poruka poslana.
Na slikama 5 i 6 su prikazani popisi primljenih i poslanih poruka. Ta dva prozora su napravljena na isti način pa ću opisati samo jedan. Glavni prozor je podijeljen na dva dijela. U gornjem dijelu se nalazi lista primljenih (poslanih) poruka. Lista je podijeljena u 6 (odnosno 5) stupaca. U tim stupcima se nalaze podaci o porukama, a to su: redni broj poruke, tko je poslao poruku, tko je primatelj poruke, naslov primljene (odnosno poslane) poruke, datum kada je poruka primljena (poslana), te veličina primljene poruke u byteima (ovog podatka nema u poslanim porukama). Klikom na jedan od podataka u listi u donjem dijelu prozora se prikaže tekst odabrane poruke i na desnoj strani tog dijela se pojavljuje scrollbar, tj. traka za pomicanje ako poruka prijeđe granice tog dijela prozora. Prilikom stvaranja ovog prozora korištene su naredbe `wx.Frame`, `wx.SplitterWindow`, `wx.Panel`, `wx.TextCtrl`, `wx.ListCtrl`, `wx.BoxSizer`.
U stvaranju ovih prozora je korištena jedna nova naredba wx.SplitterWindow koja omogućuje korisniku da podijeli prozor na dva dijela. Možemo odabrati želimo li podijeliti prozor na dva vertikalna ili dva horizontalna dijela. U ovom slučaju su bila potrebna dva horizontalna prozora, pa je pozvana naredba SplitHorizontally. Nakon podjele prozora u svaki od novonastalih prozora je postavljen po jedan panel na koji su poslagani ostali objekti. Na

gornjem dijelu se pojavljuje nova naredba, a to je wx.ListCtrl. Tom naredbom stvaramo listu koju možemo po želji uređivati. Ovdje smo tu listu rastavili na šest stupaca i svakom stupcu smo dali određeno ime. Podatke u listu upisujemo red po red koristeći naredbe InsertStringItem i SetStringItem. Prvom naredbom unosimo podatak u prvi stupac u retku. Prosljeđujemo joj broj retka i podatak koji se upisuje na to mjesto, a nakon toga pozivamo funkciju SetStringItem koja u sljedeće stupce u tom retku upisuje elemente koje joj proslijedimo. Kada kliknemo na jedan element (redak) liste poziva se posebno definirana funkcija koja utječe na donji dio prozora, tj. u donjem dijelu prikazuje tekst odabrane poruke. Na donjem dijelu prozora nalazi se objekt definiran funkcijom wx.TextCtrl koji je smješten na panel. I ovdje je korištena jedna opcija koju ima naredba wx.TextCtrl. Kako se ovdje radi o čitanju poruka ne bi bilo zgodno da se sadržaj otvorene poruke može mijenjati po volji, pa je odabrana opcija da se poruka može samo čitati, tj. da je read-only i to se postiglo upisivanjem style=wx.TE_READONLY u zagrade iza naredbe wx.TextCtrl. Kao i kod prozora za slanje nove poruke i ovdje je korištena opcija wx.TE_MULTILINE, te se s desne strane pojavljuje traka za pomicanje gore - dolje ako poruka prelazi granice prozora. Postoji još jedna korisna naredba kod SplitterWindow funkcije koju sam ovdje iskoristila. Naime, kako korisnik po volji može povećavati i smanjivati oba prozora ne bi izgledalo lijepo kada bi se bilo koji od prozora mogao smanjiti do kraja tako da se u njemu ništa ne vidi. Upravo zbog toga postoji naredba SetMinimumPaneSize kojom možemo odrediti najmanju veličinu do koje se neki prozor može smanjiti. U zagradu iza ove naredbe se upisuje broj piksela kojim se određuje ta veličina. U ovom prozoru sam odredila da je to 100 piksela. To se može vidjeti na sljedećoj slici.



Slika 7. Najmanja dozvoljena veličina gornjeg prozora

Prilikom implementacije mail klijenta sam koristila sva tri prije navedena protokola uključivanjem biblioteka smtplib, poplib i imaplib. U nastavku teksta ću pojasniti što rade neke bitne funkcije u programu i priložiti slike najbitnijih dijelova mog kôda.
Kao što sam već prije napomenula kod opisivanja slike 2 klikom na gumb Spoji se poziva posebno definirana funkcija koja zapravo obavlja posao spajanja na server. Tu funkciju sam nazvala NaSpoji. Funkcija NaSpoji prilikom poziva prvo uzima vrijednosti iz polja pod nazivom Server, Korisnicko ime i Lozinka naredbom GetValue koje će joj biti potrebne prilikom spajanja. Nakon toga stvara tri objekta: `po = poplib.POP3_SSL(server)`, `p = smtplib.SMTP(server)`, `m = imaplib.IMAP4(server)` klasa smtplib, poplib i imaplib na kojima se obavljaju sve operacije prilikom slanja i primanja poruka. Server je argument koji funkcija uzima iz polja nazvanog Server. Tada funkcija prosljeđuje objektima po, p i m korisničko ime i lozinku i ta tri objekta se spajaju na server funkcijama koje su definirane u klasama smtplib, poplib i imaplib. To se vidi na sljedećoj slici.

```
def NaSpoji(self, event):
    ...
        server = self.server.GetValue()
        user = self.login.GetValue()
        password = self.password.GetValue()
    ...
        po = poplib.POP3_SSL(server)
        p = smtplib.SMTP(server)
        m = imaplib.IMAP4(server)
        try:
            po.user(user)
            po.pass_(password)
            p.login(user, password)
            m.login(user, password)
            self.statusbar.SetStatusText('User connected')
    ...
        except (poplib.error_proto) or (smtplib.SMTPException) or (imaplib.error):
            self.statusbar.SetStatusText('Login failed')
            self.server.Clear()
            self.login.Clear()
            self.password.Clear()
            self.Show()
```

Slika 8. Kôd funkcije NaSpoji

Sa slike je vidljivo i da u pythonu možemo hvatati iznimke, tj. funkcija se prvo pokuša spojiti na server, a ako ne uspije onda javlja grešku 'Login failed'.
Sljedeća bitna funkcija je funkcija NaOdjava koja se aktivira kada u izborniku (Slika 3) kliknemo gumb odjava. Nakon pozivanja ova funkcija na svakom od tri objekta po, p i m poziva funkciju iz biblioteka smtplib, poplib i imaplib koje su zadužene za prekidanje veza sa serverom. Sljedeća slika prikazuje odjavu sa servera.

```
def NaOdjava(self, event):
    po.quit()
    p.quit()
    m.logout()
    self.Close()
    sys.exit()
```

Slika 9. Kôd funkcije NaOdjava

Kada napišemo novu poruku u prozoru za slanje nove poruke (Slika 4) i kliknemo gumb Salji pozivamo funkciju koju sam nazvala NaSalji. Ova funkcija prvo uzima vrijednosti iz polja Od, Za, Naslov i Poruka funkcijom GetValue. Nakon toga od vrijednosti iz polja Od, Za i Naslov stvara zaglavlje iza kojeg je jedan prazan red. Tada na to zaglavlje nadoda tekst poruke i tako stvara poruku koja će se slati. Naime, kako sam već prije napomenula u dijelu 2.1, kada šaljemo poruku putem SMTP protokola potrebno je odvojiti zaglavlje i tekst praznim redom jer je to forma poruke kakvu SMTP prepoznaje. Na kraju funkcija na objektu po poziva funkciju sendmail koja je definirana u biblioteci smtplib. Funkciji sendmail se prosljeđuju pošiljatelj, primatelj i sama poruka koju ona tada šalje na odgovarajuću adresu. Ako je poruka uspješno poslana na traci za obavijesti, tj. statusbaru, će se ispisati da je poruka uspješno poslana. U suprotnom će se aktivirati jedna od iznimaka i u traci za obavijesti će se ispisati obavijest da poruka nije poslana. Sve to je vidljivo na sljedećoj slici.

```
def NaSalji(self,event):
    ...
        od = self.tc0.GetValue()
        za = self.tc1.GetValue()
        naslov = self.tc2.GetValue()
        poruka = self.tc3.GetValue()
        zaglavlje = 'From: %s\r\nTo: %s\r\nSubject: %s\r\n\r\n' % (od, za, naslov)
        mail = zaglavlje + poruka
        try:
            self.m.sendmail(od, za, mail)
            self.statusbar1.SetStatusText('Poruka poslana')
        except (socket.gaierror) or (socket.error) or (socket.herror) or (smtplib.SMTPException)
            self.statusbar1.SetStatusText('Poruka nije poslana')
```

Slika 10. Kôd funkcije NaSalji

Sljedeća bitna funkcija je funkcija OnSelect koja se aktivira klikom na neki elemet liste primljenih ili poslanih poruka. Funkcija OnSelect se nalazi unutar klase kojom su napravljeni popisi primljenih i poslanih poruka. Prilikom kreiranja lista u kojima se vide podaci o pošiljatelju, naslovu, veličini i vremenu slanja poruke sam koristila neke funkcije koje su ugrađene u module poplib i email. email je modul koji pomaže u parsiranju 'sirovih' e-mail poruka. Da bi se bolje vidjelo o čemu se radi na sljedećoj slici je primjer jednog 'sirovog' odgovora.

```
89.172.254.16', '      (SquirrelMail authenticated user mirela)', '      by
www.mathos.hr with HTTP;', '      Thu, 2 Jul 2009 23:10:15 +0200 (CEST)',
'Message-ID: <6b4b6cffd305500ee973e61b1dccfd36.squirrel@www.mathos.hr>',
'In-Reply-To: <8eae5b1eea9e285547df6cb172d3ece4.squirrel@www.mathos.hr>',
'References: <8eae5b1eea9e285547df6cb172d3ece4.squirrel@www.mathos.hr>',
'Date: Thu, 2 Jul 2009 23:10:15 +0200 (CEST)', 'Subject: Re: pitanje', 'From:
mirela@mathos.hr', 'To: rcordas@mathos.hr', 'User-Agent: SquirrelMail/1.4.15',
'MIME-Version: 1.0', 'Content-Type: text/plain;charset=iso-8859-2', 'Content-
Transfer-Encoding: 8bit', 'X-Priority: 3 (Normal)', 'Importance: Normal', '', 'Pao mi je na
pamet sljedeci primjer:', 'Funkcija f:R->(R+), f(x)=x^2, g:(R+)->(R), g(x)
=drugikorijen(x), f nije', 'injekcija, g nije surjekcija, kompozicija fg:(R+)->R+, fg(x)
=x je', 'bijekcija.', '', 'Pozdrav,', 'Mirela', '', '> Dobra ve\xe8er!', '> Imam samo jedno
pitanje...Nije mi jasna jedna stvar.', '> Kada pravim kompoziciju dviju funkcija i moram
provjeriti je li ona', '> injekcija, surjekcija i bijekcija. Moze li kompozicija biti bijekcija
ako', '> to ove dvije funkcije (od kojih je sastavljena kompozicija) nisu?', '> I ima li
smisla praviti kompoziciju i provjeravati je li ona bijekcija ako', '> funkcije od kojih je
kompozicija sastavljena nisu bijekcije?', '> Hvala!', '>
Rebeka', '>', '']
```

Slika 11. 'Sirova' e-mail poruka

Ovakav odgovor se dobije pozivom funkcije top. Ova funkcija je definirana u modulu poplib i ona služi za dohvaćanje poruke pod brojem koji je zadan argumentom koji se upisuje u zagradu iza funkcije top (objašnjeno u 2.2.1). Funckija vraća tri vrijednosti: response, to je neki kôd i odgovarajući tekst koji ide uz taj kôd (npr. 250 Ok); lines, to je zapravo odgovor koji se vidi na slici 11 i octets, tj. broj okteta koje poruka zauzima. Kod ove funkcije je dobro to što ne označava odabranu poruku kao pročitanu kada ju pozovemo, pa se upravo zbog toga koristi prilikom dohvaćanja zaglavlja poruke. Naravno da nije nimalo pregledno ako korisniku ispišemo ovako dobiveni odgovor. Upravo zbog toga je modul email vrlo koristan. Pozivom naredbe email.message_from_string('\n'.join(lines)) cijeli odgovor spojimo u jedan veliki string koji kasnije možemo na neki način pretraživati i izdvajati dijelove koji su nam potrebni. Tako je sa sljedeće slike vidljivo da sada jednostavno napišemo npr. poruka['From'] i program izdvoji pošiljatelja označene poruke.

```
odgovor, lista, octet_count = po.list()
for podatak in lista:
    broj, velicina = podatak.split()
    response, lines, octets = po.top(broj, 0)
    poruka = email.message_from_string('\n'.join(lines))
    brojac = self.list.InsertStringItem(i, str(broj))
    self.list.SetStringItem(brojac,1,str(poruka['From']))
    self.list.SetStringItem(brojac,2,str(poruka['To']))
    self.list.SetStringItem(brojac,3,str(poruka['Subject']))
    self.list.SetStringItem(brojac,4,str(poruka['Date']))
    self.list.SetStringItem(brojac,5,str(velicina))
    i=i+1
```

Slika 12. Dohvaćanje podataka iz zaglavlja poruke

Na sličan način dohvaćamo i tekst poruke kada kliknemo na nju. To je definirano u funkciji On Select. Ova funkcija prvo dohvati redni broj poruke koju smo označili. Taj broj prosljeđuje funkciji retr koja je definirana u modulu poplib i tom funkcijom dohvaćamo označenu poruku. Funkcija retr je zapravo ista funkcija kao i top i vraća iste vrijednosti, ali pozivom funkcije retr se označena poruka označava kao pročitana. Nakon toga na isti način pozivom funkcije email.message_from_string('\n'.join(lines)) parsiramo našu poruku, ali u ovom dijelu nam nisu bitni podaci iz zaglavlja nego sam tekst poruke. Zbog toga koristimo funkciju walk. Ta funkcija nam omogućuje da 'prošećemo' kroz poruku i uzmemo dijelove koji su nam bitni. Funkcijom get_content_type() određujemo koju vrstu podataka želimo dohvatiti (u ovom slučaju je to obični tekst) i tada funkcijom get_payload() dohvaćamo odabranu vrstu podataka iz dijela u kojem se trenutno nalazimo. Sve to možete vidjeti na sljedećoj slici.

```
def OnSelect(self, event):
        item = event.GetItem()
        br = item.GetText()
        response, lines, octets = po.retr(br)
        poruka = email.message_from_string('\n'.join(lines))
        ...
        for part in poruka.walk():
            if part.get_content_type() == 'text/plain':
                text = part.get_payload()
                ...
```

Slika 13. Kôd funkcije OnSelect

Kod liste poslanih poruka se kôd ipak malo razlikuje jer je u tom slučaju korišten IMAP protokol i modul imaplib, te se naredbe razlikuju od naredbi u modulu poplib. U imaplib modulu postoji naredba fetch kojoj se mogu proslijediti točno oni dijelovi poruke koje želimo dohvatiti i u tom slučaju nije potrebno parsirati poruku naredbama iz modula email, pa je posao dohvaćanja zaglavlja i teksta poruke uvelike olakšan. Naredba fetch vraća dvije vrijednosti. Prva je odgovor koji se sastoji od uređenog para kôda i odgovarajućeg teksta, a druga je dio poruke koji smo tražili. Ta vrijednost isto dolazi u 'sirovom' obliku, pa je potrebno izdvojiti baš onaj dio koji nam je stvarno potreban. Ako napišemo fetch(i,'(BODY.PEEK[HEADER.FIELDS (FROM)])') dobit ćemo odgovor u obliku koji se vidi na sljedećoj slici.

```
[('1 (BODY[HEADER.FIELDS (FROM)] {36}', 'From: rcordas@pitagora.mathos.hr\r\n\r\n'), ')']
```

Slika 14. 'Sirovi' odgovor dobiven funkcijom fetch

Dakle, odgovor se sastoji od riječnika u kojem je jedan element uređeni par u kojem se na prvom mjestu nalazi tražena naredba, a na drugom mjestu traženi podaci, a drugi element riječnika je zagrada. Kako korisnik ne bi morao tražiti potreban podatak potrebno je izolirati i ispisati samo npr. pošiljatelja poruke. Ako smo odgovor funkcije fetch označili sa 'od' ako napišemo od[0][1] dobit ćemo nulti element riječnika i prvi element tog uređenog para, tj. dobit ćemo tekst 'From: adresa pošiljatelja'. Analogno se postupa i za sve ostale elemente zaglavlja. Na sljedećoj slici je cijeli kôd pomoću kojeg dobijemo sve podatke iz zaglavlja koji nam trebaju.

```
br = m.select("INBOX.Sent")
broj = int(br[1][0])
for i in range (1,broj+1):
    r1, od = m.fetch(i,'(BODY.PEEK[HEADER.FIELDS (FROM)])')
    r2, za = m.fetch(i,'(BODY.PEEK[HEADER.FIELDS (TO)])')
    r3, naslov = m.fetch(i,'(BODY.PEEK[HEADER.FIELDS (SUBJECT)])')
    r4, datum = m.fetch(i,'(BODY.PEEK[HEADER.FIELDS (DATE)])')
    od1 = od[0][1].strip()
    za1 = za[0][1].strip()
    naslov1 = naslov[0][1].strip()
    datum1 = datum[0][1].strip()
    za2 = za1.split('To:')[1].strip()
    od2 = od1.split('From:')[1].strip()
    naslov2 = naslov1.split('Subject:')[1].strip()
    datum2 = datum1.split('Date:')[1].strip()
```

Slika 15. Dohvaćanje podataka iz zaglavlja poruke funkcijom fetch

I ovdje postoji funkcija slična funkciji OnSelect i nazvana je OnSelect1. U biti je to ista funkcija, samo što za dohvaćanje teksta poruke ne koristi funkcije modula poplib nego modula imaplib i zbog toga nema parsiranja poruke funkcijama iz modula email, pa je stvar opet malo olakšana. Funkcija OnSelect1 prvo dohvati broj označene poruke i tada funkcijom fetch(bro,'(BODY[TEXT])') dohvaća tekst poruke pod rednim brojem koji je dan argumentom bro. Uočimo da postoji razlika u dohvaćanju dijelova zaglavlja i teksta poruke. Kada smo dohvaćali zaglavlje koristili smo naredbu BODY.PEEK, a prilikom dohvaćanja teksta naredbu BODY. Sufiks PEEK zapravo označava da ćemo samo zaviriti u poruku i u tom slučaju se neće podići zastavica koja označava da je poruka pročitana. Dakle, u neku ruku to je ekvivalent funkciji top iz poplib modula. Naredba BODY pogleda poruku i postavi zastavicu tako da se vidi da je poruka pročitana, pa bi to bio ekvivalent funkciji retr iz poplib modula. Kada pozovemo naredbu fetch opet dobivamo 'sirovi' odgovor čiju strukturu možete vidjeti na slici 16.

```
('OK', [('13 (BODY[TEXT] {483}', 'Dobra ve\xe8er!\r\nImam samo jedno pitanje...Nije mi jasna jedna
stvar.\r\nKada pravim kompoziciju dviju funkcija i moram provjeriti je li ona\r\ninjekcija, surjekcija i
bijekcija. Moze li kompozicija biti bijekcija ako\r\nto ove dvije funkcije (od kojih je sastavljena kompozicija)
nisu?\r\nI ima li smisla praviti kompoziciju i provjeravati je li ona bijekcija ako\r\nfunkcije od kojih je
kompozicija sastavljena nisu bijekcije?\r\nHvala!\r\n                                    Rebeka\r\n'), ')'])
```

Slika 16. 'Sirovi' tekst poruke dobiven funkcijom fetch

Sa slike je vidljivo da je odgovor uređeni par. Na prvom mjestu se nalazi odgovor, u ovom slučaju je to 'Ok' jer je sve prošlo u redu, a na drugom mjestu je rječnik. U rječniku se opet nalaze dva elementa. Prvi je uređeni par, a drugi zagrada. U uređenom paru na prvom mjestu stoji broj odabrane poruke i naredba koja je pozvana, a na drugom mjestu je sam tekst poruke. Ako smo odgovor funkcije fetch(bro,'(BODY[TEXT])') označili sa poruka, tada ćemo naredbom poruka[1][0][1] dobiti prvi element uređenog para (rječnik), zatim nulti element riječnika (uređeni par naredbe i teksta poruke) i na kraju prvi element tog uređenog para (sam tekst poruke). Sve to se vidi u kôdu funkcije OnSelect1 sa slike 17.

```
def OnSelect1(self, event):
        ...
        item = event.GetItem()
        bro = item.GetText()
        poruka = m.fetch(bro,'(BODY[TEXT])')
        text = poruka[1][0][1]
        ...
```

Slika 17. Kôd funkcije OnSelect1

4 Zaključak

Širenjem interneta i povećavanjem obujma podataka koji kruže internetom razvila se potreba za određenim alatima koji će pouzdano prenositi željene podatke s jednog mjesta na drugo. U tu svhu su nastali protokoli.

U ovom radu sam se bazirala na protokolima koji služe za siguran i učinkovit prijenos e-mail poruka od servera do klijenta i obratno jer u današnje vrijeme e-mail komunikacija postaje najbrži i najučinkovitiji oblik komuniciranja i razmjene podataka. Cilj mi je bio približiti vam sam način rada tih protokola kao i sigurnost kojom se poruke prenose tim protokolima.

Iz praktičnog dijela ovoga rada se jasno vidi da se ovi protokoli jednostavno mogu implementirati, a zapravo su vrlo moćno sredstvo za prijenos podataka. U budućnosti će se internet protokoli sve više usavršavati i prijenos podataka putem tih protokola će biti sve pouzdaniji i učinkovitiji. Vjerujem da je samo pitanje vremena kada će komunikacija putem interneta gotovo u potpunosti potisnuti sve ostale oblike komunikacije upravo zbog tog usavršavanja internet protokola.

Zahvaljujem mentoru doc.dr.sc.Domagoju Matijeviću na svoj pomoći oko pisanja ovog završnog rada i pravljenja praktičnog dijela u Pythonu.

5 Literatura

[1] B. Rhodes, J.Goerzen, Foundations of Python Network Programming: The comprehensive guide to building network applications with Python, second edition, Apress, SAD, 2010.

[2] IMAP protokol. U Wikipediji. Dostupno na:
http://en.wikipedia.org/wiki/Internet_Message_Access_Protocol (28.06.2011.)

[3] POP protokol. U Wikipediji. Dostupno na:
http://en.wikipedia.org/wiki/Post_Office_Protocol (28.06.2011.)

[4] RFC 821 - SMTP protokol
http://tools.ietf.org/html/rfc821 (28.06.2011.)

[5] RFC 1939 - POP3 protokol
http://tools.ietf.org/html/rfc1939 (28.06.2011.)

[6] RFC 2060 - IMAP4 protokol
http://tools.ietf.org/html/rfc2060 (28.06.2011.)

[7] SMTP protokol. U Wikipediji. Dostupno na:
http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol (28.06.2011.)

[8] URL izvori:
http://www.hmailserver.com/documentation/latest/?page=information_imap
(30.06.2011.)
http://www.emailaddressmanager.com/tips/protocol.html (30.06.2011.)

6 Prilog - popis slika