



**Istituto Tecnico Industriale Statale
“Leonardo Da Vinci” – Parma**

Anno Scolastico 2004/2005 5B Informatica Abacus

La posta elettronica



Tesina d'informatica di Andrea Modenini

Insegnanti:

Prof. Alberto Ferrari
Prof. Alberto Paganuzzi
Prof. Ing. Luciano Varani

Firme

INDICE

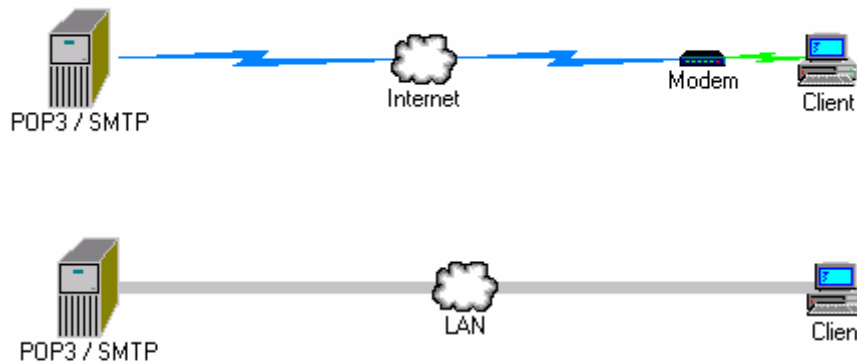
Presentazione	4
Fondamentali	5
<i>L'Email</i>	5
Modello di servizio.....	5
Indirizzi e-mail.....	6
Messaggi e-mail.....	6
Intestazioni.....	7
Corpo e allegati.....	7
Abusi.....	8
<i>I Protocolli</i>	9
Pop3 - Post Office Protocol v.3.....	9
Gli standard RFC	10
Simple Mail Transfer Protocol.....	10
Esempio di comunicazione SMTP	11
La sicurezza e lo spamming del protocollo SMTP	11
Gli standard RFC	12
<i>MIME</i>	13
I Programmi	14
<i>JOutlook</i>	14
Descrizione.....	14
GUI	14
Caratteristiche fondamentali del progetto.....	16
Documentazione Classi più complesse.....	18
Class joutlook.....	18
Class frmMain.....	19
Class EMail.....	21
Class EMailVector	22
Class Account	23
Class AccountVector	24
Class Settings	25
Class ReceiveMail.....	26
Class SendMail.....	26
Class Contatto.....	27
Class frmMail.....	28
Class JOperationBar.....	29
Frazioni di codice importanti	30
Ricezione di un EMail.....	30
Invio di un EMail	30
Impostazione dell'Email nel frmMail	30
l'Email	32
<i>MailNET</i>	35
Descrizione.....	35
GUI	35
Caratteristiche principali del progetto.....	36
Il Linguaggio.....	37
Documentazione delle Classi più complesse.....	37

Class POP3Connection	37
Frazioni di codice importanti	39
Gestione POP3	39
Invio di un EMail	41
Salvataggio di un Cookie	41
Upload di un Allegato	41
Note Finali	42

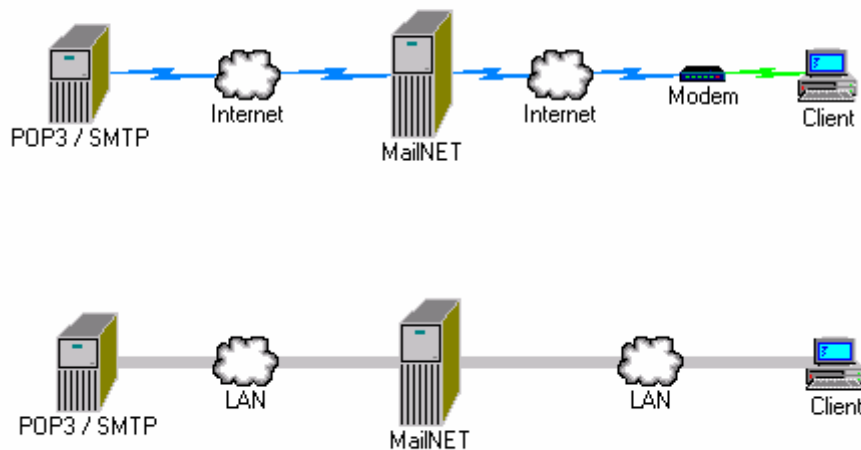
Presentazione

Lo scopo della tesina è la gestione dei principali protocolli di posta elettronica, ovvero POP3 ed SMTP. La tesina è divisa in 2 parti.

La prima consiste nella programmazione di un client di posta elettronica, un programma che semplifica all'utente la gestione e-mail scaricando e memorizzando in locale i messaggi e gli eventuali allegati. Il programma ha funzionalità simili a Microsoft Outlook Express. La connessione avviene direttamente tra il client ed il Server di posta tramite internet o la rete LAN.



La seconda parte consiste nella programmazione di un'interfaccia Web per la gestione della posta elettronica. La connessione del client avviene sul Server MailNET tramite un browser (via LAN o Internet). Il Server MailNET è connesso a sua volta al Mail Server e fa da intermediario con il client.



nota: il sistema di rete può anche essere misto ovvero Internet – LAN.

Concetti Fondamentali

L'Email

La E-Mail (abbreviazione di Electronic Mail, ovvero posta elettronica) è un servizio internet grazie al quale ogni utente può inviare o ricevere dei messaggi.

È la controparte digitale ed elettronica della posta ordinaria e cartacea. A differenza di quest'ultima, il tempo necessario per raggiungere il destinatario è normalmente di pochi secondi/minuti.



Modello di servizio

Lo scopo del servizio di e-mail è il trasferimento di messaggi da un utente ad un altro.

Ciascun utente può possedere una o più caselle e-mail dove può ricevere ed inviare messaggi. L'accesso alla casella di posta elettronica è normalmente controllato da una password o da altre forme di autenticazione .

La modalità di accesso al servizio è quindi asincrona , ovvero per la trasmissione di un messaggio non è necessario che mittente e destinatario siano contemporaneamente attivi o collegati.

La consegna al destinatario dei messaggi inviata non è garantita. Nel caso un server SMTP non riesca a consegnare un messaggio che ha ricevuto, tenta normalmente di inviare una notifica al mittente per avvisarlo della mancata consegna, ma anche questa notifica è a sua volta un messaggio di e-mail (generato automaticamente dal server), e quindi la sua consegna non è garantita.

Il mittente può anche richiedere una conferma di consegna o di lettura dei messaggi inviati, ma il destinatario è normalmente in grado di decidere se vuole inviare o meno tale conferma. Il

significato della conferma di lettura può essere ambiguo, in quanto aver visualizzato un messaggio per pochi secondi in un client non significa averlo letto.

Indirizzi e-mail

A ciascuna casella è associato un indirizzo e-mail. Ci sono casi particolari in cui ad una casella possono essere associati più indirizzi (es. all'ITIS i miei indirizzi sono *modenini.andrea@itis.pr.it* e *smodenin@itis.pr.it*). Questi hanno la forma nomeutente@dominio, dove nomeutente è un nome scelto dall'utente o dall'amministratore del server, e dominio è un nome DNS.

Modalità di funzionamento del servizio

I componenti fondamentali del sistema di e-mail sono i client (detti in gergo MUA, Mail User Agent), utilizzati per accedere ad una casella di posta elettronica e per inviare messaggi, e i server, che svolgono due funzioni fondamentali: immagazzinare i messaggi per uno o più utenti (detti in gergo MS, Message Store), ricevere i messaggi in arrivo ed in partenza e smistarli (detti in gergo MTA, Mail Transfer Agent).

I protocolli tipicamente impiegati per lo scambio di email sono l' SMTP, usato per l'invio, la ricezione e l'inoltro dei messaggi tra server, e POP e IMAP, usati per la ricezione e consultazione dei messaggi da parte degli utenti.

I client richiedono la configurazione dei server da contattare, e sono quindi adatti principalmente a computer usati regolarmente. È anche molto diffusa la possibilità di consultare una casella e-mail attraverso il web.

In Italia esistono numerosi siti che offrono gratuitamente uno o più indirizzi e-mail. Questi offrono sempre un accesso alla e-mail tramite web, e talvolta solo quello.



Messaggi e-mail

Un messaggio di e-mail è costituito da varie intestazioni (header) e da un corpo del messaggio. Lo standard di riferimento è RFC 822 e RFC 2822.

Intestazioni

Le intestazioni sono informazioni di servizio che servono a controllare l'invio del messaggio, o a tener traccia delle manipolazioni che subisce. Ciascuna intestazione è costituita da una riga di testo, con un nome seguito dal carattere ':' e dal corrispondente valore.

Alcune di queste vengono definite direttamente dall'utente. Tra le principali si possono citare:

Subject: (**Oggetto:**) dovrebbe contenere una breve descrizione dell'oggetto del messaggio. È considerata buona educazione utilizzare questo campo per aiutare il destinatario a capire il contenuto del messaggio.

From: (**Da:**) indica l'indirizzo e-mail del mittente

To: (**A:**) quello del destinatario

Cc: (**Carbon Copy, Copia Carbone:**) quelli dei destinatari in copia conoscenza

Quando il destinatario riceve un messaggio, può normalmente vedere chi sono gli altri destinatari e i loro indirizzi. Questo non sempre può essere opportuno, per ragioni di privacy e di sicurezza . In particolare, se si invia un messaggio ad un gran numero di persone che non necessariamente si conoscono tra di loro, costoro non necessariamente saranno d'accordo che il loro indirizzo, ed il fatto che hanno ricevuto quel messaggio, sia reso noto ad estranei. Inoltre, molti worm si propagano per e-mail, e utilizzano gli indirizzi presenti nei messaggi per diffondersi. Inviare un messaggio con gli indirizzi dei destinatari in chiaro significa quindi esporre tutti i destinatari ad un ulteriore rischio di contagio se uno di loro viene contagiato.

Per ovviare a questo problema, è consigliabile utilizzare l'intestazione speciale **Bcc:** (nota anche come **CCN:**), che indica che il messaggio deve essere consegnato ai destinatari che seguono, ma che il loro indirizzo non deve apparire nel messaggio stesso (copia conoscenza nascosta).

Altre intestazioni vengono aggiunte dai programmi che manipolano il messaggio, la più importante è **Received:** , che viene aggiunta da ciascun server SMTP che manipola il messaggio, indicando da quale indirizzo IP il messaggio è stato ricevuto, a che ora, e altre informazioni utili a tracciarne il percorso. Altre intestazioni segnalano ad esempio che il messaggio è stato valutato da qualche tipo di filtro automatico antivirus o anti spam , e la valutazione espressa dal filtro.

Corpo e allegati

Il corpo del messaggio era originalmente composto di testo semplice.

In seguito è stata introdotta la possibilità di inserire dei file in un messaggio e-mail (**allegati**), ad esempio per inviare immagini o documenti. Per fare questo si utilizza la codifica MIME (o la più desueta uuencode).

Gli allegati vengono utilizzati anche per comporre un messaggio e-mail in html , per rappresentare la formattazione del testo. Questa pratica non è molto apprezzata dai puristi di Internet , in quanto aumenta la dimensione dei messaggi, e non tutti i client sono in grado di interpretare l'html.

Molti server impongono limiti massimi alla dimensione del messaggio da trasmettere, che devono essere presi in considerazione se si inviano grossi file come allegati.

È considerata cattiva educazione anche la pratica di inviare messaggi ad un grande numero di destinatari inserendo grandi allegati, soprattutto se gli allegati sono in formati proprietari che non tutti i destinatari potrebbero poter leggere, come per esempio Microsoft Word .

Abusi

Il principale utilizzo improprio dell'e-mail è lo spam , l'invio massiccio a molti utenti di messaggi indesiderati, in genere di natura pubblicitaria-commerciale. Secondo alcune fonti, l'incidenza di questi messaggi raggiungerebbe i due terzi del traffico totale di posta elettronica.

Un altro fenomeno negativo è costituito dalle catene di sant'Antonio , messaggi che circolano a volte per mesi o anni portando informazioni allarmanti o promesse di facili guadagni.

Esiste inoltre la possibilità di falsificare il nome e l'indirizzo del mittente visualizzati nel programma client del destinatario, inducendo l'utente a ritenere attendibile un messaggio del tutto falso. Questo meccanismo è sfruttato dagli worm che si replicano per posta elettronica, allo scopo di creare confusione tra gli utenti colpiti e distogliere l'attenzione dai sistemi realmente infetti, oppure per indurre ad installare allegati infetti.



"On the Internet, nobody knows you're a dog."

<<on the Internet, nobody know you're a dog>>

I Protocolli

Un protocollo è un insieme di regole che definisce il formato dei messaggi scambiati e consentono a due entità di comunicare tra di loro e di comprendere a comunicazione.

Nell'ambito delle telecomunicazioni , due o più macchine o host (computer , telefono , stampante , ecc...) possono comunicare tra loro rispettando norme che sono dette protocolli di rete . L'aderenza ai protocolli garantisce che due macchine possano comunicare correttamente, anche se sono state realizzate indipendentemente.

I protocolli utilizzati nel campo della posta elettronica oggi sono il POP3, SMTP, IMAP e HTTP. Per la tesi vengono utilizzati i primi due.

Pop3 - Post Office Protocol v.3

Il POP3, descritto in almeno 20 RFC diverse (1081, 1225 e 1460 solo per citare le più significative) è il protocollo complementare dell'SMTP. Se infatti l'SMTP si occupa della spedizione delle e-mail, il POP3 fa l'esatto contrario, ovvero fornisce una serie di comandi per la fase di ricezione. E' per merito dei server POP3 che è possibile ricevere le e-mail "on demand". Una volta inoltrato dal server SMTP del mittente, infatti, un messaggio viene memorizzato dal server POP3 del destinatario, che solitamente è una macchina sempre accesa e destinata prevalentemente a ricevere la posta in arrivo. L'utente può collegarsi in un qualsiasi momento con il proprio server POP3 e trasferire in locale tutti i messaggi destinati a lui, tenendone o meno una copia sul server. Il demone POP3 è solitamente in ascolto sulla porta 110 TCP, alla quale il client deve accedere per poter controllare la sua mailbox. I procedimenti principali utilizzati dal protocollo per avviare il trasferimento sono i seguenti: il client si identifica, inserendo il proprio username. La stringa da inviare deve essere ad esempio: "USER smodenin ". Il server risponde con un +OK ed un codice che indica la corretta ricezione ed elaborazione della stringa trasmessa; una volta fornito il proprio username, l'utente deve farsi riconoscere mediante una password. La stringa da inviare è del tipo "PASS 5info2". Il server risponde inviando il solito +OK, seguito però dal numero di messaggi presenti nella mailbox; il client può quindi iniziare la ricezione dei messaggi, con la stringa "RETR numeromessaggio". Ad esempio: "RETR 1" provoca l'invio dal server al client del primo messaggio (in ordine temporale basato sulla spedizione) presente nella casella di posta; nonostante venga trasmesso al legittimo destinatario, il messaggio rimane memorizzato anche nell'hard disk del server. Per eliminarlo il client può usare la forma: "DELE numeromessaggio"; per chiudere la connessione, come nel caso dell'SMTP, è sufficiente l'invio della stringa: "QUIT".

Il protocollo POP3 è stato ormai sostituito dal più recente IMAP4. Questo nuovo protocollo non ha comunque introdotto sostanziali migliorie al POP3, al punto che, se fosse possibile pesare il numero di software che sfruttano uno solo dei due protocolli, l'ago della bilancia continuerebbe a pendere inesorabilmente verso il POP3.

Gli standard RFC

RFC 1939 - "Post Office Protocol - Version 3"

RFC 2449 - "POP3 Extension Mechanism"

RFC 1734 - "POP3 AUTHentication command"

RFC 2222 - "Simple Authentication and Security Layer (SASL)"

RFC 3206 - "The SYS and AUTH POP Response Codes"

Simple Mail Transfer Protocol

Simple Mail Transfer Protocol (SMTP) è il protocollo standard per la trasmissione via internet di e-mail . In italiano si potrebbe tradurre come "Protocollo elementare di trasferimento postale".

È un protocollo relativamente semplice, testuale , nel quale vengono specificati uno o più destinatari di un messaggio, verificata la loro esistenza, e infine il messaggio viene trasferito. E' abbastanza facile verificare come funziona un server SMTP mediante un client telnet . L'SMTP usa il protocollo di trasmissione TCP e, per accedervi, la porta 25. Per associare il server SMTP a un dato nome di dominio (DNS), si usa un record denominato MX (Mail Exchange).

L'SMTP iniziò a diffondersi sempre più fin dai primi anni '80. A quel tempo era un protocollo complementare all' UUCP più adatto a gestire il trasferimento di e-mail fra computer la cui connessione era intermittente. In altre parole il protocollo SMTP funziona meglio se i computer sono sempre collegati alla rete.

Sendmail fu uno dei primi, (ma non il primo), agente o programma di trasferimento di posta elettronica ad implementare il protocollo SMTP. Nel 2001 sono almeno 50 i programmi che implementano il protocollo SMTP come client (mittente dei messaggi) o server , (destinatario del messaggio). Vi sono altri programmi, lato server, che usano l'SMTP : Exim, Postfix, Qmail, Microsoft Exchange Server.

L'SMTP era un protocollo basato esclusivamente sul riconoscimento di caratteri di testo ASCII , e quindi ha forti difficoltà a trattare con file binari . Fu quindi sviluppato uno standard MIME per codificare file binari, utile a trasferirli via email. Al giorno d'oggi molti server SMTP supportano l'estensione 8BITMIME . Questa estensione consente un trasferimento più agevole dei file binari, come se fossero file di testo.

L'SMTP è un protocollo "push" che non permette, su richiesta, il "prelievo" di messaggi da server remoto. Per fare questo il mail client deve usare il POP3 , o Post Office Protocol, oppure l' IMAP , o Internet Message Access Protocol. Alcuni server SMTP per fare queste operazioni usano il protocollo ERTN .

Esempio di comunicazione SMTP

Dopo aver stabilito una connessione tra il mittente (il client) e il destinatario (il server), ciò che accade è una sessione SMTP vera e propria. Nella successiva conversazione, qualsiasi cosa inviata dal client è preceduta con "C:", mentre qualsiasi cosa inviata dal server è preceduta da "S:". Su molti computer si può stabilire una connessione mediante il comando telnet :

```
telnet smtp.itis.pr.it 25
```

Questo comando apre un collegamento SMTP verso l'host smtp.itis.pr.it.

S: 220 www.itis.pr.it ESMTP Postfix	<i>risposta del server alla connessione</i>
C: HELO itis.pr.it	<i>si invia un saluto al server con il quale risponde</i>
S: 250 Hello itis.pr.it 	<i>si specifica il mittente</i>
C: MAIL FROM: "Ing. Andrea Modenini" <smodenin@itis.pr.it>	<i>si specifica il destinatario</i>
S: 250 Ok	
C: RCPT TO: "Bill W. Gates III" <billgates@microsoft.com>	
S: 250 Ok	
C: DATA	<i>si indica che d'ora in poi tutto quel che viene inviato è l'email</i>
S: 354 End data with <CR><LF>.<CR><LF>	<i>si specificano gli header del messaggio che si separano dal body da una riga vuota.</i>
C: Subject: mi regali la tua ditta??	<i>Per indicare che si ha finito il messaggio si fa <invio>.<invio></i>
C: From: smodenin@itis.pr.it	
C: To: billgates@microsoft.com	
C:	
C: Mi farebbe molto piacere...	
C: Goodbye.	
C: .	
S: 250 Ok: queued as 12345	
C: quit	<i>si chiude la connessione</i>
S: 221 Bye	

Sebbene opzionale, quasi tutti i client richiedono al server che l'estensione SMTP utilizzi un saluto di tipo "EHLO", piuttosto che di tipo "HELO", mostrato sopra, e il corpo del testo delle email è formattato tipicamente in MIME .

La sicurezza e lo spamming del protocollo SMTP

Una delle limitazioni del protocollo originale SMTP è che non è adeguatamente programmato per l'autenticazione, o verifica delle identità, dei mittenti. Per ovviare a questo problema è stata sviluppata un'estensione chiamata SMTP-AUTH . Nonostante questo, lo spamming rimane ancor oggi problema maggiore. Attualmente non si ritiene conveniente modificare profondamente il protocollo, a causa dell'enorme effetto che avrebbe sull'enorme numero di protocolli SMTP già installati a livello mondiale. Per ovviare a tali inconvenienti è stato proposto comunque un nuovo protocollo chiamato Internet Mail 2000

Per queste ragioni vi sono sempre più proposte per protocolli a modulazione d'ampiezza che assistono le operazioni SMTP. Il gruppo di ricerche Antispam dell' IRTF sta lavorando su un certo numero di proposte per fornire un semplice metodo di autenticazione, flessibile, leggero e scalabile . Tra quelli esaminati, molto probabilmente verrà utilizzato il Sender Policy Framework , formalmente conosciuto come Sender Permitted From. Tuttavia, questo protocollo di sicurezza è messo in discussione a causa di violazioni del copyright nei confronti di brevetti Microsoft .

Gli standard RFC

RFC 821 , pubblicato nel 1982

RFC 1123 pubblicato nel 1989. Correzioni all' RFC 821

RFC 1425 pubblicato nel 1993. Introduce il comando EHLO

RFC 1651 pubblicato nel 1994. Rimpiazza l' RFC 1425

RFC 1869 pubblicato nel 1995. Rimpiazza l' RFC 1651

RFC 1891 pubblicato nel 1996. Corregge l' RFC 1869

RFC 2821 pubblicato nel 2001. Rimpiazza gli RFC 821 , RFC 1123 , RFC 1869

RFC 2822 pubblicato nel 2001.

MIME

Il Multipurpose Internet Mail Extensions , o più brevemente MIME è un protocollo Internet che estende l' SMTP (Simple Mail Transfer Protocol) per permettere ai dati, come dati video, suoni e file binari, di essere trasmessi tramite la posta elettronica senza dover prima essere convertiti in formato ASCII ; questa operazione viene compiuta mediante l'uso di vari tipi di MIME, che descrivono il contenuto di un documento. Un'applicazione compatibile MIME che invia un file, assegna un tipo di MIME al file. L'applicazione ricevente, che deve essere anch'essa compatibile MIME, fa riferimento a un elenco standard di documenti organizzati per tipi e sottotipi di MIME al fine di interpretare il contenuto del file. Per esempio, un tipo di MIME è text che ha un certo numero di sottotipi, tra i quali plain e html. Un tipo di MIME text/html fa riferimento ad un file che contiene un testo scritto in HTML. MIME è parte di HTTP e, sia i browser web che i server HTTP lo utilizzano per interpretare i file di posta elettronica che inviano e ricevono.

Esempio di messaggio MIME:

```
From: John Doe <example@example.com>  
MIME-Version: 1.0  
Content-Type: multipart/mixed;  
boundary="XXXXboundary text"
```

Questo e' un messaggio MIME con piu' parti.

```
--XXXXboundary text  
Content-Type: text/plain
```

```
Questo è il corpo della mail --XXXXboundary text Content-Type: text/plain;  
Content-Disposition: attachment;  
filename="test.txt"
```

Questo invece e' il testo allegato.

```
--XXXXboundary text--
```

In questo messaggio apparirà un file allegato con nome test.txt. La cosa importante da tener presente è che quando si forgia un messaggio la scritta filename="test.txt" deve cominciare esattamente sotto la D di disposition. Lo so perché l'ho provato personalmente con Outlook.

I Programmi

JOutlook

Linguaggio: Java 2

Piattaforma di sviluppo: J2SE 5 Update 2

JRE: 1.4.x, 1.5.x

IDE : JBuilder X e 2005



Descrizione

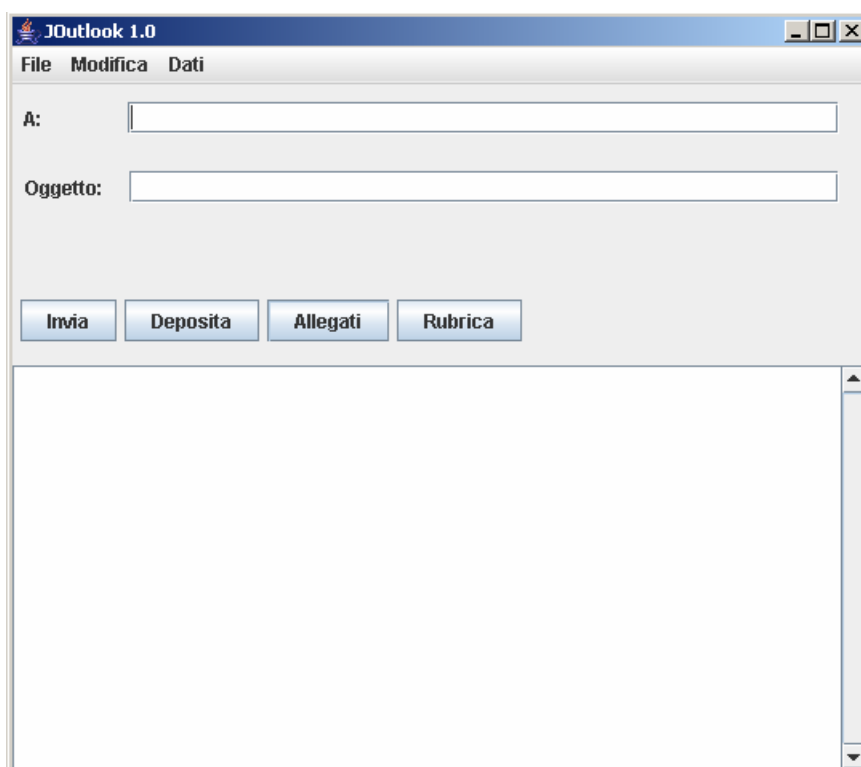
JOutlook è un client di posta elettronica che sfruttando le caratteristiche del POP3 e dell'SMTP permette lo scaricamento e l'invio della posta quando connessi, e di visualizzare e manipolare le email anche se disconnessi.

JOutlook supporta tutti gli account che utilizzano il POP3 ed SMTP. E' in grado di gestire più account memorizzando le impostazioni ed eventualmente anche le password.

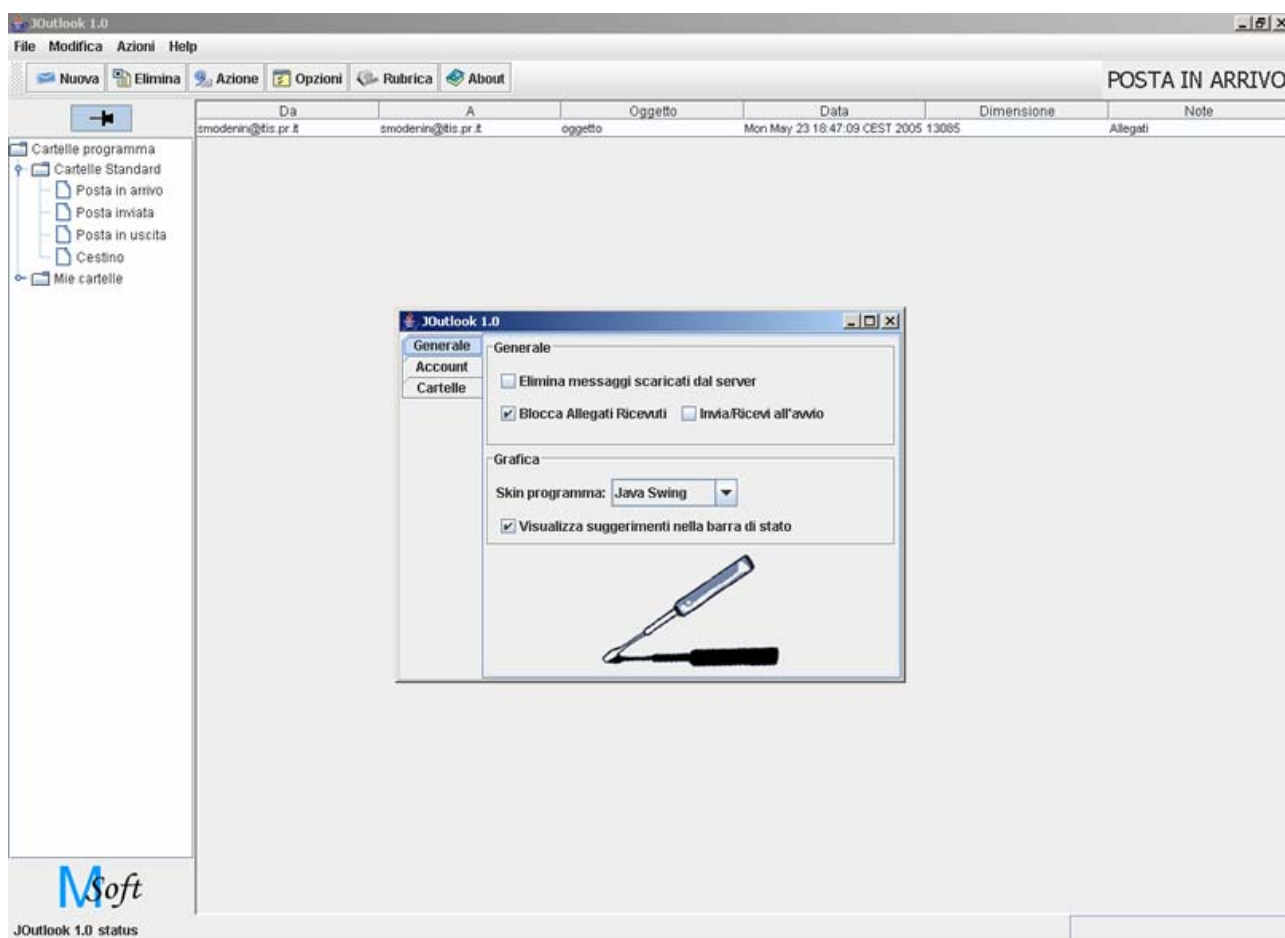
GUI

L'interfaccia grafica di JOutlook è basata sul Windows Design. Presenta la barra dei menù principale (le solite voci: File, Modifica, etc), i bottoni rapidi, una struttura ad albero delle cartelle sulla sinistra, la lista delle email al centro, ed infine una barra di stato contenente informazioni e/o suggerimenti.

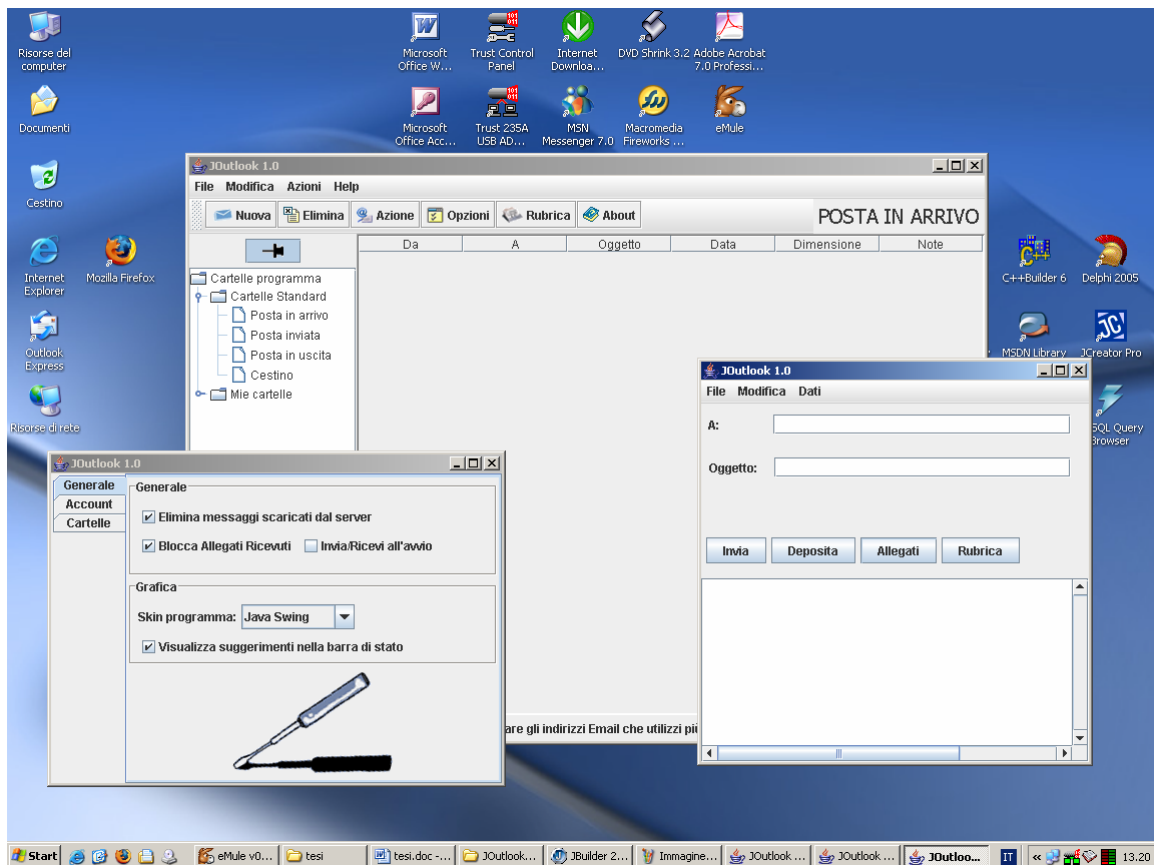
Le finestre sono ridimensionabili grazie all'utilizzo corretto dei layout di Java.



Il frame per l'invio di un messaggio



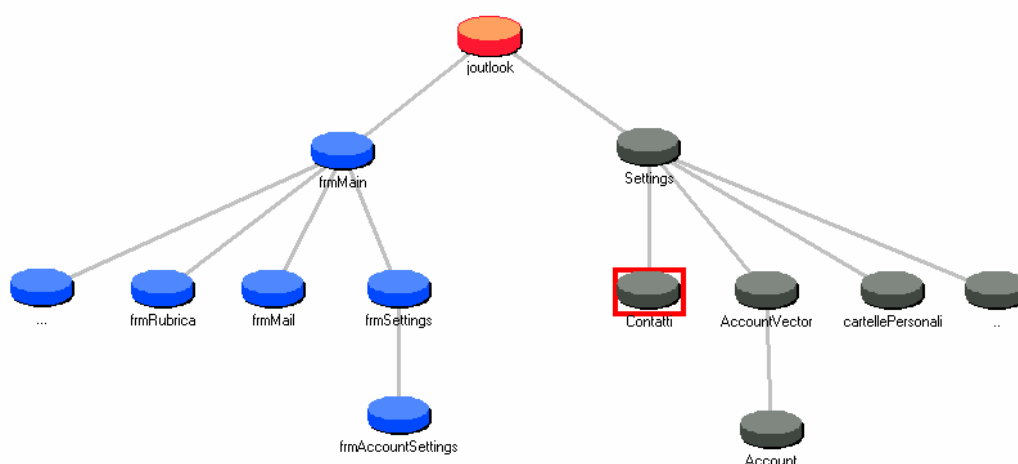
vista generale



Le finestre sono ridimensionabili adattandosi alle necessità dell'utente

Caratteristiche fondamentali del progetto

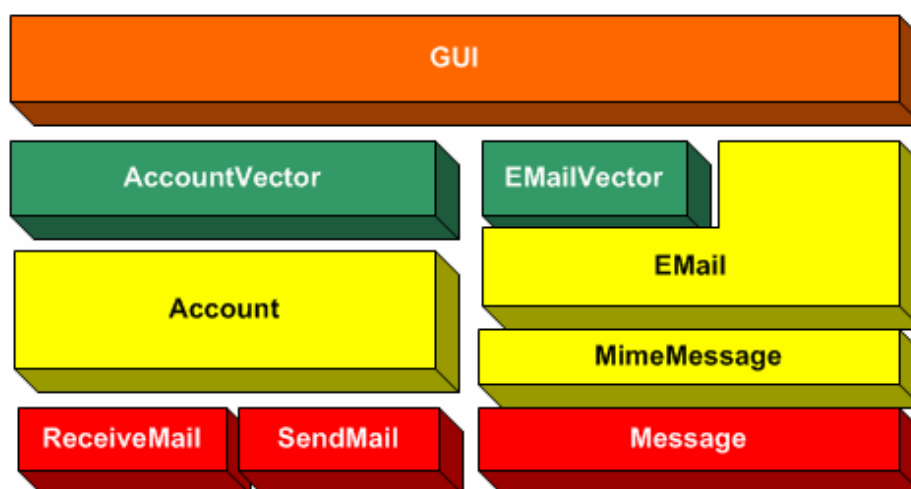
JOutlook è stato sviluppato secondo un progetto ben preciso. Per la visibilità totale tra una classi dell'applicazione si è progettata una **struttura ad albero** (l'ho progettata personalmente senza suggerimenti). La classe principale dell'applicazione, "joutlook", è una classe particolare che viene utilizzata soltanto per l'avvio dell'applicazione. Questa classe contiene un riferimento static al frame principale "frmMain" ed alla classe delle impostazioni "Settings". In questo modo qualunque oggetto del programma è in grado di risalire a tutte gli altri partendo dagli oggetti joutlook.frmMain o joutlook.Settings, percorrendo poi i vari nodi dell'albero.



Esempio:

Alla radice si `joutlook`. Tramite `joutlook` si discende a `frmMain` che ha riferimenti a tutti i sottoframe con i loro componenti. Stessa cosa vale per le impostazioni.

Il progetto ha avuto anche una **suddivisione in strati** formando una pila. Ogni classe è stata fatta il più indipendente possibile per la manutenibilità del programma.



Ogni strato della pila agisce indipendentemente. Nessuna classe fa modifiche o azioni per altre classi, ma semplicemente chiama i loro metodi.

Al cuore del programma, ovvero la base della pila, ci stanno le classi per l'invio e la ricezione e la classe `Message`. La GUI non interagisce in alcun modo direttamente a queste classi. Per farlo deve passare tramite gli account contenuti negli `AccountVector` e le `EMail` (che possono essere nell'`EMailVector`) chiamando opportuni metodi.

Nota: fa in parte eccezione la classe `frmMail` che interagisce direttamente con la `MimeMessage` per la lettura del contenuto di un `EMail` ricevuta (testo, allegati, etc).

Con l'indipendenza dei vari strati, la gerarchia a pila garantisce flessibilità e modularità; in termini pratici se si dovesse modificare una classe (se non un intero strato) del programma, gli altri strati non necessitano di grosse modifiche.

In questo modo la modifica od il rimpiazzo di una classe del programma, se fatta opportunamente, risulta molto semplice e veloce.

Il sistema stratificato ha funzionato. Durante la creazione del programma, inizialmente si era deciso di utilizzare per il POP3 ed SMTP delle classi scritte da me personalmente. Tuttavia in seguito, quando già buona parte del programma era stata scritta, si è deciso di utilizzare la javax.mail.

Grazie al sistema stratificato utilizzato non ho dovuto buttare via l'intero programma, ma semplicemente modificare le classi riguardanti l'ultimo strato (tempo impiegato: 15 min).

Documentazione Classi più complesse

Class joutlook

```
public final class joutlook
```

L'Applicazione. Tutti gli attributi sono static. Istanziarla più di una volta non ha senso.

Constructor Summary

```
public joutlook()  
istanzia gli elementi dell'applicazione, li imposta, e visualizza il frame  
principale
```

Main Field Summary

<code>public static frmMain</code>	frame il frame principale dell'applicazione
<code>public static Settings</code>	settings le impostazioni utente del programma
<code>static class</code>	JEMLFilter filtro per cercare i file estensione ".jempl"

Main Method Summary

<code>public static void</code>	centra(Window w) centra nello schermo la finestra passata
<code>private static void</code>	leggiImpostazioni() legge le impostazioni salvate su disco e le mette in Settings.
<code>private static void</code>	inizializzaImpostazioni()

	carica le impostazioni Settings sulle classi interessate
public static	getDefaultSession() restituisce la Session di default
public static void	notifica(String msg) notifica il messaggio nella finestra DOS
public void	salva() salva le impostazioni su disco.
public static void	main() inizio del programma.

Class frmMain

```
public class frmMain
extends JFrame
```

è il Frame principale del programma.

Constructor Summary

```
frmMain()
costruttore del Frame
```

Main Field Summary

class	EventoMenuPersonali classe che gestisce gli eventi dei JMenuItem riguardanti le cartelle personali
class	InviaMessaggi extends Thread classe chiamata da inviaMessaggi() per inviare i messaggi senza bloccare il codice
class	RiceviMessaggi extends Thread classe chiamata da riceviMessaggi() per ricevere i messaggi senza bloccare il codice
class	NascondiPannelloCartelle extends Thread classe utilizzata per far passare 2 secondi prima di nascondere le cartelle
byte	CURRENT_FOLDER indica la cartella corrente del programma
public static final byte	POSTA_IN_ARRIVO identifica la cartella posta in arrivo per la CURRENT_FOLDER
public static final byte	POSTA_IN_USCITA identifica la cartella posta in uscita per la CURRENT_FOLDER
public static final byte	POSTA_INVIATA

	identifica la cartella posta inviata per la CURRENT_FOLDER
public static final byte	CESTINO identifica la cartella cestino per la CURRENT_FOLDER
public boolean	onCartelle true se il puntatore del mouse è sopra il JTree. False altrimenti
String[]	z_MailListColoumnName sono i nomi delle colonne del JTree
DefaultMutableTreeNode[]	trnCartelleStd array contenente i nodi del JTree delle cartelle standard
DefaultMutableTreeNode	trnRoot nodo radice del JTree
DefaultMutableTreeNode	trnPIA nodo posta in arrivo del JTree
DefaultMutableTreeNode	trnPIU nodo posta in uscita del JTree
DefaultMutableTreeNode	trnCES nodo cestino del JTree
DefaultMutableTreeNode	trnPIN nodo posta inviata del JTree

Main Method Summary

private void	jbinit() è il metodo d'inizializzazione grafica
public void	refreshTabella() aggiorna la tabella contenente la lista delle e-mail
public void	cambiaCartella() cambia la cartella delle e-mail visualizzata
public void	caricaCartellePersonalì() metodo utilizzato per rendere effettive tutte le impostazioni riguardanti le cartelle personali nel programma
public void	controllaPostaPer(Account c) controlla la posta per un determinato account
public synchronized void	inviaMessaggi() invia tutti i messaggi della cartella posta in uscita
public synchronized void	riceviMessaggi() riceve tutti i messaggi di tutti gli account e li mette in posta in arrivo
public void	caricaEmailVectorInTabella(EMailVector em,DefaultTableModel mdl) carica l'EmailVector Specificato nel TableModel dato
public void	showMessaggioInStatus(String msg) imposta il messaggio da visualizzare nella barra dei suggerimenti
public void	setStatusInfo(String msg) visualizza il messaggio nella barra di stato

public void	visualizzaPannelloCartelle() mostra il JTree espandibile
public void	nascondiPannelloCartelle() nasconde il JTree espandibile

Class EMail

```
public class EMail
extends MimeMessage
```

classe che rappresenta la singola EMail. Estensione della javax.mail.MimeMessage che ereditando tutti i suoi attributi e metodi, semplifica l'utilizzo di tale classe. Per l'insieme completo degli attributi e metodi ereditati fare riferimento alla documentazione Sun della MimeMessage

Constructor Summary

```
public EMail(Session s)
public EMail(MimeMessage m)
public EMail(Session s,InputStream in)
```

Main Field Summary

public final static String	HIGH_PRIORITY identifica la stringa dell'alta priorità negli Header
public final static String	NORMAL_PRIORITY identifica la stringa della priorità normale negli Header

Main Method Summary

public String	getMittente() restituisce il mittente principale dell'e-mail
public String	getOggetto() restituisce l'oggetto dell'e-mail
public String	getDestinatario() restituisce il destinatario principale dell'e-mail
public String	getDate() restituisce la Data d'invio dell'e-mail
public int	getSize() restituisce in byte la dimensione dell'e-mail
public	getMessaggio()

String	restituisce il messaggio originale con Header e Body
public static EMail	<pre>genereEMail(String mittente,String destinatario,String oggetto,String messaggio) genera un EMail con il messaggio in text/plain. Può essere utilizzata successivamente per modifiche, come l'aggiunta di allegati.</pre>
public void	<pre>salvaSuDisco(String nomeFile) salva l'email nella cartella delle email salvate del programma</pre>
public void	<pre>aggiungiAllegato(File f) aggiunge l'allegato all'email.</pre>
public boolean	<pre>haAllegati() true se l'EMail ha allegati, false altrimenti</pre>
public boolean	<pre>haAltaPr() true se ha alta priorità, false altrimenti</pre>
public void	<pre>setPriorità(String type) setta la priorità del messaggio. E' consigliato utilizzare HIGH_PRIORITY e NORMAL_PRIORITY</pre>

Class EMailVector

```
public class EMailVector
extends Vector
implements Serializable
```

Classe utilizzata per contenere più EMail. Fa da interfaccia al Vector in modo che si semplifichino le operazioni di Casting.

Constructor Summary

```
public EMailVector()
```

Main Field Summary

private String	<pre>nome è il nome dell'EMailVector. Utilizzato per i nomi delle cartelle</pre>
----------------	--

Main Method Summary

public void	<pre>addEmail(EMail e) aggiunge un Email all'EmailVector</pre>
public	<pre>getEmail(int index)</pre>

EEmail	restituisce il riferimento all'EEmail d'indice index
public EEmail[]	toEEmailArray() restituisce l'array delle EEmail all'interno del EEmailVector
public void	setName(String n) imposta il nome di questo EEmailVector
public String	getName() restituisce il nome di questo EEmailVector

Class Account

```
public class Account
implements Serializable
```

Classe utilizzata per la gestione di un singolo Account di posta elettronica.

Constructor Summary

```
public Account()
crea un account con tutti i campi come stringhe vuote
```

```
public Account(String Id, String pop3,String smtp, String user, String pass)
crea un account con i campi specificati
```

Main Field Summary

private String	ID nome dell'account
private String	POP_3 l'indirizzo POP3 dell'account
private String	SMTP l'indirizzo SMTP dell'account
private String	ADDRESS l'indirizzo di posta elettronica dell'account
private String	USER l'username dell'account
private String	PASS la password dell'account

Main Method Summary

public String	getPop3() restituisce l'indirizzo POP3
------------------	---

public String	getSmtp() restituisce l'indirizzo SMTP
public String	getUser() restituisce l'username
public String	getPass() restituisce la password. null se la password non è memorizzata
public String	getID() restituisce l'ID
public String	getAddress() restituisce l'indirizzo di posta elettronica
public void	setPop3(String inf) imposta l'indirizzo POP3
public void	setSmtp(String inf) imposta l'indirizzo SMTP
public void	setUser(String inf) imposta l'username
public void	setPass(String inf) imposta la password. Può essere null.
public void	setID(String inf) imposta l'ID
public void	setAddress(String inf) imposta l'indirizzo di posta elettronica.
public String	toString() restituisce l'ID

Class AccountVector

```
public class AccountVector
implements Serializable
```

Usata per il contenimento di Account. Fa da interfaccia al Vector in modo da semplificare le operazioni di Casting.

Constructor Summary

```
public AccountVector()
crea un AccountVector vuoto
```

Main Field Summary

None

Main Method Summary	
public void	addAccount(Account e) aggiunge un Account all'AccountVector
public Account	getAccount(int index) restituisce il riferimento all'Account d'indice index
public Account[]	toAccountArray() restituisce l'array delle Account all'interno del AccountVector

Class Settings

```
public class Settings
implements Serializable
```

Classe contenente tutte le impostazioni utente. In tutto il programma attualmente ce n'è solo una all'interno della classe joutlook. E' stata utilizzata per semplificare future aggiunte al programma come la multiutenza.

Constructor Summary	
public Settings()	non fa niente

Main Field Summary	
public AccountVector[]	accounts l'array di tutti gli account dell'utente
public boolean	tglButton
public boolean	deleteFromPop3 se true alla ricezione vengono cancellati i messaggi dal server
public boolean	inviaRiceviAvvio se true all'avvio del programma viene effettuato un Invia/Ricevi tutti
public boolean	bloccaAllegati se true gli allegati ricevuti vengono bloccati
public byte	skin identifica lo skin 0- Swing 1- Swing Special 2- Winzoz
public String[]	nomiCartelleStd array di stringhe con I nomi delle cartelle standard
Vector	cartellePersonali Vector contenente le cartelle personali

Vector	contatti Vector contenente i contatti della rubrica
--------	--

Main Method Summary

None

Class ReceiveMail

```
public class ReceiveMail
```

Classe utilizzata per la ricezione dei messaggi di posta elettronica. E' il cuore del programma.

Constructor Summary

None

Main Field Summary

None

Main Method Summary

public static void	receiveMail(String host, String user, String pass, EMailVector ricevute, boolean delete) throws MessagingException riceve l'email per il determinato host, user e pass e le aggiunge nel EMailVector. Se delete è true le cancella dal server.
--------------------------	---

Class SendMail

```
public class SendMail
```

Classe utilizzata per l'invio dei messaggi di posta elettronica. E' il cuore del programma.

Constructor Summary

None

Main Field Summary

None

Main Method Summary

public	inviaEmail(Message m) throws MessagingException
static	
void	invia il messaggio m

Class Contatto

```
public class Contatto
implements Serializable
```

Rappresenta un contatto della rubrica

Constructor Summary

```
public Contatto()
Contatto con le proprietà ""
```

```
public Contatto(String nome, String indirizzo)
crea un contatto con le specifiche date
```

Main Field Summary

public String	NOME Il nome visualizzato per il contatto in Rubrica
---------------	---

public String	INDIRIZZO L'indirizzo associato
---------------	------------------------------------

Main Method Summary

None

Class frmMail

```
public class frmMail
implements JFrame
```

E' il frame che permette l'editazione e lettura delle EMail all'utente.

Constructor Summary

```
public frmMail()
```

Main Field Summary

private boolean	MODAL specifica la modalità di lettura/scrittura dell'email. Fare riferimento alle costanti READ_MODAL e WRITE_MODAL
public static final boolean	READ_MODAL attribuito a MODAL indica che l'email è in lettura
public static final boolean	WRITE_MODAL attribuito a MODAL indica che l'email è in scrittura
Vector	allegati contiene gli allegati ricevuti o che si stanno per inviare
private EMail	CURRENT_EMAIL è l'email che si sta visualizzando/editando
class	Allegato rappresenta un allegato dell'EMail. Sono tutti contenuti nel Vector apposito.
public class extends JDialog	RimozioneAllegato Dialog che si apre per scegliere quale allegato rimuovere dal Vector.

Main Method Summary

private void	setEmail(EMail m) imposta il frame per l'EMail data
public void	setLetturaModal(EMail m) imposta il frame in modalità lettura per l'EMail data

public void	setScritturaModal(EMail m) imposta il frame in modalità scrittura per l'EMail data
public boolean	getModal() restituisce la proprietà MODAL
public void	showLetturaModal(EMail m) chiama setLetturaModal(m) e mostra il frame
public void	showEditazioneModal(EMail m) chiama setScritturaModal(m) e mostra il frame
jbInit()	inizializzazione grafica

Class JOperationBar

```
public class JComponent
implements ActionListener
```

Componente grafico barra che scorre. Utilizzato per indicare quando c'è un operazione in corso.

Constructor Summary

```
public JOperationBar()
istanzia il componente
```

Main Field Summary

private Timer	time timer utilizzato per l'effetto grafico di scorrimento
private int	posizione indice utilizzato per l'effetto grafico di scorrimento

Main Method Summary

public void	start() fa scorrere la barra
public void	stop() blocca lo scorrere della barra

Frazioni di codice importanti

Ricezione di un EMail

```
import javax.mail.*;
import javax.mail.internet.*;

public class ReceiveMail
{
    public static final int POP3_PORT = 110;
    public static final char POP3_ERROR_CODE1 = '4';
    public static final char POP3_ERROR_CODE2 = '5';

    public static void riceviEMail(String host,String user,String pass,EMailVector
ricevute,boolean delete)throws MessagingException{
        Session session = joutlook.getDefaultSession();

        Store store = session.getStore("pop3");
        store.connect(host, user, pass);

        Folder folder = store.getFolder("INBOX");
        if (folder != null) {
            folder.open(Folder.READ_WRITE);
            Message[] elencoMessaggi = folder.getMessages();

            for (int i = 0; i < elencoMessaggi.length; i++) {
                EMail m=new EMail((MimeMessage) elencoMessaggi[i] );
                m.saveChanges();
                elencoMessaggi[i].setFlag(Flags.Flag.DELETED,delete);
                ricevute.addEMail(m);
                //((MimeMessage)elencoMessaggi[i]).writeTo(new FileOutputStream(i+".txt"));
            }
            folder.close(delete);
        }
        store.close();
    }
}
```

Invio di un EMail

```
public class SendMail
{
    public static final int SMTP_PORT = 25;
    public static final char SMTP_ERROR_CODE1 = '4';
    public static final char SMTP_ERROR_CODE2 = '5';

    public static void inviaEMail(Message m) throws MessagingException{
        Transport.send(m);
    }
}
```

Impostazione dell'Email nel frmMail

```
private void setEmail(EMail m){
    CurrentEmail=m;
    allegati.removeAllElements();
    mnuAllegati.removeAll();
    if(m==null) return;
    /**Visualizzazione Messaggio**
    try{
        Object obj = m.getContent();
        //se è testo viene stampato normalmente
        if(obj instanceof String)
            txtMail.setText((String)obj);
    }
```

```
else if(obj instanceof Multipart){
    Multipart mp=(Multipart) obj;
    String s="";

    //gestione d'ogni parte del messaggio
    int npart=mp.getCount();
    for(int i=0;i<npart;i++){
        Part p=mp.getBodyPart(i);

        String disposition=p.getDisposition();
        String contentType=p.getContentType();
        //SE E' UN ALLEGATO
        if (disposition != null && (disposition.equals(Part.ATTACHMENT) ||
disposition.equals(Part.INLINE))){
            String filename=p.getFileName();
            if(filename==null) filename="att"+i;

            Allegato l=new Allegato();
            l.filename=filename;
            l.stream=p.getInputStream();

            allegati.add(l);
            //imposto il menu
            JMenuItem mni=new JMenuItem(filename);
            //EVENTO MENU
            mni.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    if(joutlook.settings.bloccaAllegati){
                        frmInputError.showError(null,tblAllegati,"Gli allegati sono stati bloccati");
                        return;
                    }
                    JMenuItem i=(JMenuItem) e.getSource();
                    int index=mnuAllegati.getComponentIndex(i);
                    if(index!=-1){
                        Allegato c=(Allegato)allegati.get(index);
                        JFileChooser fc=new JFileChooser();
                        fc.setSelectedFile(new File(c.filename));
                        if(fc.showSaveDialog(joutlook.frame)==JFileChooser.APPROVE_OPTION){
                            File destinazione=fc.getSelectedFile();
                            try{
                                FileOutputStream fout=new FileOutputStream(destinazione);
                                byte[] buffer=new byte[4096];
                                while(true){
                                    int readed=c.stream.read(buffer);
                                    if(readed===-1)
                                        break;
                                    fout.write(buffer);
                                }
                                fout.flush();
                                fout.close();

                            }catch(IOException ex){
                                JOptionPane.showMessageDialog(joutlook.frame,"Impossibile salvare il
file",joutlook.title,JOptionPane.ERROR_MESSAGE);
                                joutlook.notifica(ex.toString());
                            }
                        }
                    }
                }
            }); //FINE EVENTO
            mnuAllegati.add(mni);
        }
        else{ //non è un'allegato
            //aggiungi contenuto in formato testo

            BufferedReader br=new BufferedReader(new InputStreamReader(p.getInputStream()));
            String app;
            while((app=br.readLine())!=null)
                s+=app+"\n";
        }
    }
    txtMail.setContentType();
    txtMail.setText(s);
    lblInfo.setVisible(m.haAltaPr());
    mniDatiAltaPr.setState(m.haAltaPr());
    tblAllegati.setEnabled(mnuAllegati.getComponentCount(>0);
    lblAllegati.setVisible(mnuAllegati.getComponentCount(>0);
}
}
```

```
    }
    catch(Exception e){
        JOptionPane.showMessageDialog(this,"Si è verificato un errore nella codifica: l'Email
potrebbe non apparire correttamente",joutlook.title,JOptionPane.ERROR_MESSAGE);
        joutlook.notifica(e.toString());
    }

    if(this.getModal()==frmMail.READ_MODAL)
        txtFromTo.setText(m.getMittente());
    else
        txtFromTo.setText(m.getDestinatario());
    txtObject.setText(m.getOggetto());
}
```

l'EMail

```
import java.util.Properties;
import java.util.Date;
import java.util.Enumeration;
import javax.activation.*;

/**
 * <p>Title: JOutlook</p>
 * <p>Description: programma di posta elettronica</p>
 * <p>Copyright: Copyright (c) 2004 Andrew Mods</p>
 * <p>Company: MSoft</p>
 * @author Andrew Mods & Andrea Modenini
 * @version 1.0
 */

class EMail extends MimeMessage{
    public final static String HIGH_PRIORITY="High";
    public final static String NORMAL_PRIORITY="Normal";

    public EMail(Session s){
        super(s);
    }

    public EMail(MimeMessage m)throws MessagingException{
        super(m);
    }

    public EMail(Session s,InputStream in) throws MessagingException{
        super(s,in);
    }

    public String getMittente(){
        String mittenti="";
        try{
            Address[] a=this.getFrom();
            if(a==null || a.length==0) return "";

            for(int i=0;i<a.length-1;i++)
                mittenti+=a[i].toString()+",";

            mittenti+=a[a.length-1].toString();
            return mittenti;
        }
        catch(MessagingException e){
            return "";
        }
    }

    public String getOggetto(){
        try{
            return this.getSubject();
        }
        catch(MessagingException e){
            return "";
        }
    }

    public String getDestinatario(){
        String destinatari="";
        try {
```

```
Address[] a = this.getAllRecipients();
if(a==null || a.length==0) return "";

for (int i = 0; i < a.length-1; i++)
    destinatari += a[i].toString() + ",";

    destinatari+=a[a.length-1].toString();
return destinatari;
}
catch (MessagingException e) {
    return "";
}
}

public String getDate(){
    try{
        return this.getSentDate().toString();
    }
    catch(MessagingException e){
        return "";
    }
}

public int getSize(){
    try{
        return super.getSize();
    }
    catch(MessagingException e){
        return 0;
    }
}

public String getMessaggio() throws IOException,MessagingException{
    String messaggio="";
    Enumeration en=this.getAllHeaders();
    while(en.hasMoreElements()){
        Header h=(Header)en.nextElement();
        messaggio += h.getName()+":"+h.getValue()+"\n";
    }
    messaggio+="\n\n";
    /*BufferedReader bf = new BufferedReader(new InputStreamReader(this.getInputStream()));
    String app;
    while((app=bf.readLine())!=null)
        messaggio+=app+"\n";*/
    return messaggio;
}

public static EMail generaEMail(Account mittente,String destinatario,String oggetto,String
messaggio) throws MessagingException,IOException{
    Properties props = System.getProperties();
    props.put( "mail.smtp.host", mittente.getSmtp() );

    Session session = Session.getDefaultInstance( props );
    EMail message = new EMail( session );

    InetAddress from = new InetAddress( mittente.getAddress() );
    InetAddress to[] = InetAddress.parse( destinatario );

    message.setFrom(from);
    message.setRecipients( Message.RecipientType.TO, to );
    message.setSubject(oggetto);
    message.setSentDate( new Date() );

    MimeMultipart mm=new MimeMultipart();
    MimeBodyPart bodypart=new MimeBodyPart();
    bodypart.setText(messaggio);

    mm.addBodyPart(bodypart);
    message.setContent(mm);

    return message;
}

public void salvaSuDisco(String nomeFile) throws MessagingException,IOException{
    //se cartella non esiste creala
    File folder=new File("saved_data");
    if(!folder.exists() || folder.isFile()){
        if(!folder.mkdir())
    
```

```
        throw new IOException("impossibile creare cartella");
    }

    //salvataggio su file
    nomeFile = "saved_data/" + nomeFile;
    FileOutputStream f = new FileOutputStream(nomeFile);

    this.writeTo(f);

    f.flush();
    f.close();
}

public void aggiungiAllegato(File f) throws IOException, MessagingException { //solo se il messaggio
è Multipart
    Multipart m=(Multipart)this.getContent();
    MimeBodyPart bp=new MimeBodyPart();

    DataSource source = new FileDataSource(f);

    bp.setFileName(f.getName());
    bp.setDataHandler(new DataHandler(source));

    m.addBodyPart(bp);
}

public boolean haAllegati(){
    try{
        if (this.getContent() instanceof Multipart) {
            MimeMultipart m = (MimeMultipart)this.getContent();

            for (int i = 0; i < m.getCount(); i++) {
                Part p=m.getBodyPart(i);
                String disp = p.getDisposition();
                if (disp!=null && (disp.equals(Part.ATTACHMENT) || disp.equals(Part.INLINE)))
                    return true;
            }
        }
    }
    catch (Exception ex) {}
    return false;
}

public boolean haAltaPr(){
    try{
        return this.getHeader("X-MSMail-Priority")[0].compareToIgnoreCase("high") ==
            0;
    }catch(Exception e){
        return false;
    }
}

public void setPriorità(String type) throws Exception{
    this.setHeader("X-MSMail-Priority", type);
}
}
```

MailNET

Linguaggio: ASP.NET C#

Piattaforma di sviluppo: .NET SDK 1.1

Framework: 1.1

IDE : Visual Studio 2003, DreamWeaver MX



Descrizione

MailNET è un'applicazione web che consente all'utente di gestire il proprio account di posta elettronica via Web (http). MailNET svolge le operazioni di invio e ricezione messaggi. Questa operazione può essere molto utile per utenti con una banda limitata, che in questo modo prima di scaricare l'intero messaggio possono leggere solo l'intestazione e decidere in seguito se scaricare anche il testo del messaggio ed eventuali allegati.

Utile anche se l'utente non sta utilizzando il computer sul quale opera normalmente, in quanto il client che vi si connette, non dovendo gestire nulla, è indifferente.

GUI

L'interfaccia grafica di MailNET è molto semplice. A sinistra si ha un menù laterale per la navigazione alle varie pagine dell'applicazione web. Nella zona centrale si ha il contenuto della pagina. Per fare le pagine con la stessa struttura grafica si sono utilizzati i modelli di DreamWeaver MX.

Home

Leggi

Invia

Server POP3: pop3.ittis.pr.it

Server SMTP: smtp.ittis.pr.it

Username: smodenin

Password:

Indirizzo e-mail: smodenin@ittis.pr.it

Ricordami i dati

Entra

©2005 MailNET 1.0 by Modenini Andrea

Caratteristiche principali del progetto

MailNET, sfruttando le caratteristiche di Visual Studio 2003, fa una netta distinzione tra il design della pagina e la programmazione utilizzando i CodeBehind, ovvero dei riferimenti della pagina Web a file C# (*.cs). Questo ha permesso di dividere il lavoro di programmatore da quello di grafico. Infatti mentre la programmazione di MailNET è avvenuta con Visual Studio 2003, la grafica è stata fatta con Macromedia DreamWeaver.

Il linguaggio C# presenta per ASP .NET una classe apposita per l'invio di e-mail tramite Web. Questo ha semplificato molto il codice per la trasmissione di un messaggio. Al contrario C# non presenta classi per la ricezione. Per farla ho dovuto scrivere personalmente una classe per la gestione del POP3, e sono ricorso ad una DLL di terze parti per la gestione dei MIME dell'e-mail ricevuta.

Il passaggio dei dati principali tra le pagine, ovvero gli host e le informazioni utente, avviene tramite le variabili definite con l'oggetto Session. Le variabili definite con l'oggetto Session sono variabili visibili da tutte le pagine che, come fa intuire il nome stesso, valgono per la sessione di navigazione attiva dell'utente, e quindi terminano quando il client termina definitivamente la connessione col sito (ovvero la chiusura del browser o la disconnessione dalla rete Internet). Con le Session è quindi possibile memorizzare informazioni del utente (come l'username con il quale ha effettuato il login) con un livello di sicurezza abbastanza elevato.

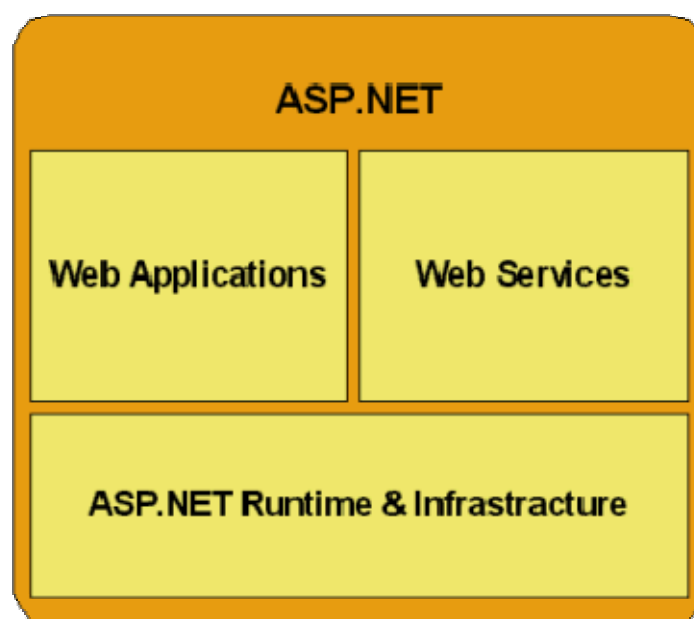
L'unica pagina che utilizza dei parametri è la pagina di visualizzazione dell'email. Tale pagina prende in ingresso il parametro "index" (che identifica il numero dell'e-mail) passatogli in QueryString (*pagina?parametro=valore;...*).

Il Linguaggio

Più che dire il linguaggio (che sarebbe il C# per questa applicazione) c'è da parlare d'ambiente. ASP .NET è l'evoluzione di ASP 3.0 ed è una nuova tecnologia di programmazione rivoluzionaria che permette lo sviluppo di applicazione basate sul web in modo semplice e veloce.

Oltre alle "vecchie" capacità, ha alcune nuove funzionalità molto importanti:

- Per la prima volta la programmazione in ASP utilizza un linguaggio vero e proprio a tutti gli effetti, eliminando gli obsoleti linguaggi di scripting. Il linguaggio non è più interpretato, ma è compilato (od almeno in parte), e sono stati introdotti i tipi di dato.
- Si è sviluppato maggiormente il concetto di Common Language Runtime. In ASP .NET il programmatore può scegliere il linguaggio che preferisce come sintassi tra VB .NET, C#, C++ .NET, J# senza problemi di compatibilità e comunicazione tra oggetti. In questo modo ASP .NET , a differenza del nonno ASP che doveva ricorrere per esempio a COM+, viene utilizzato direttamente per lo sviluppo di tutti gli ambiti di un Sito web.



- Infine l'ultima fondamentale caratteristica è che non viene eseguito direttamente da IIS ma si appoggia alla piattaforma .NET Framework aumentando così le potenzialità ed in teoria la portabilità (in teoria siccome Microsoft ancora non si è decisa di produrre un Framework per OS diversi da Windows).

Documentazione delle Classi più complesse

Class POP3Connection

```
public class POP3Connection
```

Componente grafico barra che scorre. Utilizzato per indicare quando c'è un operazione in corso.

Constructor Summary

`public POP3Connection(String host,String user,String pass)`
istanzia la classe per il determinato host, user, pass

Main Field Summary

<code>private String</code>	<code>POP3</code> l'host del POP3
<code>private String</code>	<code>USER</code> l'username con cui fare il login
<code>private String</code>	<code>PASS</code> la password con cui fare il login
<code>private const int</code>	<code>PORT</code> identifica la porta
<code>private TcpClient</code>	<code>tcpc</code> la connessione TCP/IP
<code>private NetworkStream</code>	<code>nstream</code> lo stream del flusso dati
<code>private StreamReader</code>	<code>stream</code> sempre lo stream, istanziato per la lettura

Main Method Summary

<code>public bool</code>	<code>connect()</code> effettua la connessione al server POP3
<code>public void</code>	<code>close()</code> chiude la connessione
<code>public int[]</code>	<code>getStat()</code> restituisce le statistiche della mailbox. [0]-numero dei messaggi nuovi [1]-dimensione totale
<code>public String</code>	<code>readMessage(int index)</code> legge il messaggio index
<code>public String</code>	<code>readTopMessage(int index)</code> legge il top del messaggio index
<code>public bool</code>	<code>delMessage(int index)</code> elimina il messaggio index
<code>private bool</code>	<code>connettiUtente()</code> connette l'utente una volta aperta la connessione. Chiamato dal metodo connect()
<code>private void</code>	<code>sendCommand(String cmd)</code> invia il comando stringa sul flusso dati
<code>private String</code>	<code>readSingleLineResponse()</code>

	legge un responso di una riga
private String	readMultilineResponse() legge un responso di più righe terminante per <crLf>.<crLf> (ovvero il body del message)

Frazioni di codice importanti

Gestione POP3

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Web.Mail;
using System.IO;

namespace MailNET
{
    public class POP3Connection
    {
        #region Proprietà

        private String POP3;
        private const int PORTA=110;
        private String USER;
        private String PASS;
        private TcpClient tcpc;
        private NetworkStream nstream;
        private StreamReader stream;
        //costruttore
        #endregion

        #region Costruttore

        public POP3Connection(String host,String user,String pass)
        {
            POP3=host; USER=user; PASS=pass;
        }
        //effettua la connessione al server pop3=true se riuscita

        #endregion

        #region Metodi
        public bool connect()
        {
            String resp;

            try
            {
                tcpc=new TcpClient(POP3,PORTA);
            }
            catch(Exception ex)
            {
                return false;
            }
            nstream=tcpc.GetStream();
            stream=new StreamReader(nstream);

            resp=readSingleLineResponse();

            if(resp.StartsWith("+OK"))
                return connettiUtente();
            return false;
        }

        public void close()
        {
            sendCommand("QUIT\r\n");
        }
    }
}
```

```
        if(stream!=null)
            stream.Close();
        if(nstream!=null)
            nstream.Close();
        if(tcpc!=null)
            tcpc.Close();
    }
    //esegue il comando STAT e restituisce il num. msg nuovi e la dimensione totale
    public int[] getStat()
    {
        String resp;
        int[] dati=new int[2];
        sendCommand("STAT\r\n");
        resp=readSingleLineResponse();

        if(resp==null ||!resp.StartsWith("+OK"))
            return null;

        String[] parts=resp.Split(' ');
        dati[0]=Int32.Parse(parts[1]);
        dati[1]=Int32.Parse(parts[2]);

        return dati;
    }

    public String readMessage(int index)
    {
        String resp;
        sendCommand("RETR "+index.ToString()+"\r\n");
        resp=readSingleLineResponse();
        if(!resp.StartsWith("+OK")) return null;
        return readMultiLineResponse();
    }

    public String readTopMessage(int index)
    {
        String resp;
        sendCommand("TOP "+index.ToString()+" 1\r\n");
        resp=readSingleLineResponse();
        if(!resp.StartsWith("+OK")) return null;
        return readMultiLineResponse();
    }

    public bool delMessage(int index)
    {
        String resp;
        sendCommand("DELE "+index.ToString()+"\r\n");
        resp=readSingleLineResponse();
        return resp.StartsWith("+OK");
    }

    private bool connettiUtente()
    {
        String resp;
        //invio username
        sendCommand("user "+USER+"\r\n");
        resp=readSingleLineResponse();
        if(!resp.StartsWith("+OK")) return false;
        //invio password
        sendCommand("pass "+PASS+"\r\n");
        resp=readSingleLineResponse();
        if(!resp.StartsWith("+OK")) return false;

        return true;
    }

    #endregion

    #region MetodiBase
    /***FUNZIONI BASE***/
    private void sendCommand(String cmd)
    {
        byte[] send=Encoding.ASCII.GetBytes(cmd);
        nstream.Write(send,0,send.Length);
    }

    private String readSingleLineResponse()
    {

```

```
        return stream.ReadLine();
    }

    private String readMultiLineResponse()
    {
        String msg="";

        do
        {
            msg+=stream.ReadLine()+"\r\n";
        }while(!msg.EndsWith("\r\n.\r\n"));

        return msg.Substring(0,msg.Length-4);
    }
    #endregion
}
}
```

Invio di un EMail

```
private void btnInvia_Click(object sender, System.EventArgs e)
{
    TimeSpan t=(lastInvio - DateTime.Now);
    if(t.Seconds<30)
    {
        lblInfo.Text="Non puoi inviare subito un messaggio dietro l'altro";
        return;
    }
    try
    {
        MailMessage m=new MailMessage();
        m.From=FROM;
        m.To=txtTo.Text;
        m.Subject=txtSubject.Text;
        m.Body=txtMail.Text;
        SmtMail.SmtpServer=SMTP;

        //allegati
        for(int i=0;i<lstAllegati.Items.Count;i++)
            m.Attachments.Add(new MailAttachment(lstAllegati.Items[i].Value));

        //*****

        SmtMail.Send(m);
        lblInfo.Text="Messaggio inviato con successo";
        lastInvio=DateTime.Now;
    }
    catch(Exception ex)
    {
        lblInfo.Text=ex.ToString();
    }
}
```

Salvataggio di un Cookie

```
HttpCookie ck=new HttpCookie(COOKIE_NAME);
ck.Values.Add("pop3",txtPOP3.Text);
ck.Values.Add("smtp",txtSMTP.Text);
ck.Values.Add("user",txtUsername.Text);
ck.Values.Add("pwd",cript(txtPassword.Text));
ck.Values.Add("email",txtEmail.Text);
ck.Expires=DateTime.Now.AddDays(10); //scade tra 10 giorni
ck.Secure=false;
Response.Cookies.Add(ck);
```

Upload di un Allegato

```
private void btnAggiungi_Click(object sender, System.EventArgs e)
```

```
{  
    string filename=System.IO.Path.GetFileName(MyFile.PostedFile.FileName);  
    string path=Server.MapPath("upload/files")+@"\"+CID.ToString()+filename;  
    MyFile.PostedFile.SaveAs(path);  
    lstAllegati.Items.Add(new ListItem(filename,path));  
}
```

Note Finali

Lo sviluppo d'applicazione per la gestione della posta elettronica, nonostante l'uso in alcuni casi di codice già scritto (per evitare la programmazione di basso livello per quanto riguarda POP3, SMTP, MIME), può risultare molto complessa. Le caratteristiche definite dai vari RFC hanno reso alto il potenziale contenuto di un e-mail ma anche molto complessa la gestione, in particolare per quel che riguarda i MIME. L'esempio sono questi due programmi che si limitano alla gestione di account unicamente POP3 ed SMTP (gli altri protocolli che possono essere usati oggi sono IMAP, HTTP, UUCL). Inoltre ci si è limitati a semplice EMail senza nessun meccanismo di encrypt, firma, o certificati e a connessioni standard senza utilizzare i sistemi di sicurezza, quali SSL, APOP, MD5.

Bibliografia

- <http://www.ietf.org/rfc.html>
Il sito ufficiale dei vari RFC's
- <http://www.mokabyte.com>
Sito dedicato alla programmazione Java
- Io programma “Edizione Master” – Anno VIII n. 1;
Io programma “Edizione Master” – Anno VIII n. 2;
Rivista dedicata alla programmazione. Utilizzate per ottenere informazioni sulla javax.mail
- <http://www.sun.com/java>
Il sito ufficiale di Java. Utilizzato per notizie, informazion, documentazioni
- <http://msdn.microsoft.com>
Il sito ufficiale dell'MSDN. Utilizzato per notizie, informazion, documentazioni riguardo ASP.NET
- <http://www.wikipedia.com>
Enciclopedia libera on-line. Utilizzata per trovare spiegazioni non troppo tecniche