Università degli Studi di Parma

Facoltà di Scienze MM.FF.NN
Corso di Laurea in Informatica

JCloset

Software di cloud storage in JAVA

Titolare del corso:
Alfieri Roberto

Autori:
Dughetti Danilo
Monica Gianluca
Allevi Davide

Anno Accademico 2013-2014

Indice

1 Introduzione

1.1 Obiettivi

L'obiettivo del progetto JCloset è quello di realizzare una web Application che offra un servizio SaaS di Cloud Storage stile Dropbox.
Il progetto comprende un applicazione server suddivisa in tre servlet, che gestisce gli utenti e le operazioni sulla porzione di file-system remoto associato ad ognuno di essi. Inoltre, è presente un applicazione desktop stand-alone che funge da interfaccia tra utente e servizio.

1.2 SaaS e Cloud Storage

Introduciamo il progetto enunciando il significato cruciale di SaaS e Cloud Storage.
Innanzitutto SaaS (Software as a service), è un modello di distribuzione del software applicativo dove un produttore di software sviluppa, opera (direttamente o tramite terze parti) e gestisce un'applicazione web che mette a disposizione dei propri clienti via internet.
Il Cloud Storage invece, è un modello di conservazione dati su computer in rete dove i dati stessi sono memorizzati su molteplici server virtuali generalmente ospitati presso strutture di terze parti o su server dedicati.

1.3 Dropbox e pyDio

Riserviamo una menzione speciale ai servizi offerti da Dropbox e pyDio in quanto rappresentano i modelli sui quali abbiamo iniziato a ragionare riguardo il cloud storage.

Dropbox è un software di cloud storage multipiattaforma, che offre un servizio di file hosting e sincronizzazione automatica di file tramite web.
Esso si basa sul protocollo crittografico Secure Sockets Layer (SSL), e i file immagazzinati, accessibili tramite password, vengono cifrati tramite AES con chiave a 256 bit.
Il programma è scaricabile gratuitamente per tutti i sistemi operativi per cui è disponibile. Lo spazio di hosting disponibile varia in funzione del piano tariffario scelto ed è espandibile.
Il servizio può essere usato anche via web, caricando e visualizzando i file tramite il browser, oppure tramite il driver locale che sincronizza automaticamente una cartella locale del file-system con quella condivisa, notificando le sue attività all'utente. L'interfaccia web consente il caricamento di file con dimensione massima pari a 300 Mb ciascuno.

Pydio invece, è un software SaaS open-source che permette a qualsiasi server di diventare una piattaforma di file sharing.
Diversamente da Dropbox non prevede alcun software installabile su desktop ma solo una maschera scritta in Ajax per accedere ai file tramite browser.

1.4 Confronto con piattaforme commerciali

Oggigiorno, le piattaforme di cloud storage maggiormente diffuse ed utilizzate in ambito commerciale sono: Google Drive, Dropbox, iCloud, OneDrive e Ubuntu One.

Nome	Provider	Spazio di archiviazione iniziale	Sincronizzazione file	Crittografia file utente
Dropbox	Dropbox, Inc.	2 GB	Si	Si
Drive	Google	15 GB	Si	Si
iCloud	Apple	2 GB	Si	Si
OneDrive	Microsoft	2 GB	Si	Si
Ubuntu One	Canonical	2 GB	Si	Si

2 Gli Strumenti e tecnologie utilizzate

2.1 Ii linguaggi di programmazione

Per la realizzazione del nostro progetto abbiamo optato per l'utilizzo di JAVA. Con tale linguaggio è stato realizzato il client, le servlet oltre che i package implicati nella gestione del database.
Abbiamo previsto anche una pagina web per potersi iscrivere se si è nuovi utenti e per realizzarla abbiamo usato HTML, CSS e JAVASCRIPT.
Infine, per descrivere i dati trasmessi tra client e server abbiamo optato per l'utilizzo di XML.

2.2 L'IDE

Tutto il progetto è stato svolto grazie all'ausilio di Eclipse Kepler JAVA EE version e per testare il funzionamento del programma, in localhost abbiamo utilizzato TomCat versione 6.0. Mentre per il sistema operativo non vi è alcuna preferenza dato che la portabilità è la caratteristica principale di JAVA.

2.3 Il controllo della versione

Considerata la vastità del progetto, abbiamo adottato la possibilità di lavorarci attraverso SVN appoggiandoci al servizio hosting di Assembla.
Il progetto è consultabile all'URL: https://www.assembla.com/code/jcloset/subversion/nodes/1

2.4 Gli obiettivi durante lo sviluppo

Data la grande vastità del progetto e le innumerevoli estensioni possibili (che verranno menzionate nel paragrafo degli sviluppi futuri) un obiettivo che ci siamo prefissati durante lo sviluppo è stato quello di rendere il codice mantenibile. Grazie alla natura ad oggetti di JAVA, ogni classe è stata pensata prima come interfaccia e poi implementata in una versione che a noi sembrava la migliore per quel dato problema.
Per fare un esempio, sia il client che il server, per gestire i metadati, utilizzano le strutture dati **DirectoryImpl** e **FileImpl**. Esse però sono implementazioni delle più generali interfacce **Directory** e **File**. Inoltre tutte i metodi che le utilizzano fanno riferimento alle classi base invece che alle implementazioni in questo modo, se in futuro qualcuno volesse cambiare tali strutture dati, sarebbe semplicissimo!

3 Il lavoro nel dettaglio

3.1 La struttura principale

JCloset è diviso in due parti: client e server. Tuttavia il centro di controllo del software è rappresentato dal server. Esso è formato da tre servlet, **ClientManager**, **Confirm** e **Registration**. La prima è interamente dedicata alla gestione delle richieste del client mentre le altre due, sono utilizzate per iscrivere un nuovo utente e creargli la prima prima directory.
Il funzionamento generale del software è il seguente:

- Un ipotetico nuovo utente si connette al sito web di registrazione e compilando il form, riceverà una mail utilizzata per confermare la registrazione.
- Una volta registrato, si inserisce l'utente nel database e viene immediatamente creata fisicamente nel server una directory (all'interno dello spazio dedicato ai file dell'utente cioè **UserSpace**) col nome dell'utente.
- A questo punto il controllo passa al client, dove inizialmente verranno domandate le credenziali per accedere.
- Se le credenziali sono corrette si apre la schermata del client dove da qua è possibile interagire col server per qualsiasi operazione.

3.2 Il Client

Il cient è stato interamente scritto grazie alla libreria grafica di JAVA Swing ed è composto da 9 package per un totale di 29 classi.
Per ciò che concerne la comunicazione, le classi che la implementano sono:

- **User, File, Directory**: Sono le classi da cui derivano le implementazioni utilizzate per concretizzare i metadati scambiati col server.
- **NetworkManager:** E' la classe astratta da cui deriva **NetworkManagerImpl**. Essa svolge il compito di creare e chiudere la connessione col sever, di inviare e ricevere dati via HTTP con metodo POST.
- **XmlUtility**: E' la classe astratta da cui abbiamo ricavato le varie implementazioni utili a creare file XML per ogni operazione richiesta dal client.
- **HistoryStack**: E' una classe che implementa una pila utilizzata per tenere memoria delle cartelle navigate dal client cosi da poter ripercorrerle a ritroso (un po' come funziona sui browser).

3.3 Il Database

Il Database è stato scritto in SQL mentre il DBMS utilizzato è Derby. E formato da quattro entità e grazie alle quali siamo stati in grado di gestire tutti gli aspetti della nostra applicazione.

```
create table ATTIVAZIONI(
      id varchar(50) not null,
      MAIL VARCHAR(50) not null,
      PASSWORD VARCHAR(50) NOT NULL,
      NAME VARCHAR(50) NOT NULL,
      SURNAME VARCHAR(50) NOT NULL
);
```

```
create table USERS(
      MAIL VARCHAR(50) PRIMARY KEY,
      PASSWORD VARCHAR(50) NOT NULL,
      NAME VARCHAR(50) NOT NULL,
      SURNAME VARCHAR(50) NOT NULL
);

CREATE TABLE DIRECTORIES(

      PATH VARCHAR(50) NOT NULL,
      NAME VARCHAR(50) NOT NULL,
      USERS VARCHAR(50) NOT NULL REFERENCES USERS(MAIL),
      SIZE VARCHAR(50) NOT NULL,
      LAST_MODIFY VARCHAR(50) NOT NULL,
      CREATION VARCHAR(50) NOT NULL,
      PRIMARY KEY(PATH,NAME)
);

CREATE TABLE FILES(
      NAME VARCHAR(50) NOT NULL,
      PATH VARCHAR(50) NOT NULL,
      USERS VARCHAR(50) NOT NULL REFERENCES USERS(MAIL),
      SIZE VARCHAR(50) NOT NULL,
      LAST_MODIFY VARCHAR(50) NOT NULL,
      CREATION VARCHAR(50) NOT NULL,
      PRIMARY KEY(PATH,NAME)
);
```

3.4 Il server

Il server è composto da 7 package per un totale di 22 classi JAVA, tre pagine HTML e un file CSS.
Per ciò che concerne la comunicazione, le classi che la implementano sono:

- **User, File, Directory**: Sono le classi da cui derivano le implementazioni utilizzate per concretizzare i metadati scambiati col client.
- **DirectoryUtility**: E' la classe che gestisce le operazioni a livello di file-system che sono:
  - Creare, cancellare, rinominare e spostare directory
  - Creare, cancellare, rinominare e spostare file
- **FileSystemManager**: E' la classe che offre i metodi per interagire col database e tra le altre cose offe:
  - Creare un nuovo utente, directory o file
  - Cancellare, spostare, rinominare file e directory
- **XmlUtility**: E' la classe astratta da cui abbiamo ricavato le varie implementazioni utili a creare file XML per ogni operazione richiesta dal client.
- **Registration e Confirm**: Sono le servlet che gestiscono la registrazione dell'utente al database e creano la prima directory in UserSpace.

3.5 Inizializzazione progetto

In questo paragrafo elencheremo i passi necessari per avviare il progetto:

- Assicurarsi che nella directory di JclosetClient siano presenti le directory: “download” e “xml”. La prima sarà lo spazio di destinazione del download dei file da server mentre la seconda conterrà i file XML da inviare.
- Assicurarsi che nella directory di JclosetServer siano presenti le directory: “UserSpace”, “apache-ftpserver-1.0.6”, “database”, e “xml”. Esse contengono lo spazio di archiviazione degli

utenti, il server di Apache per l'FTP, il database del progetto ed infine i file XML.

- Avviare il database con il DBMS Derby entrando nella directory database di JclosetServer a avviando lo script StartNetworkServer. Quindi in un altra shell avviare lo script ij e connettersi attraverso il comando:
  ```
  connect 'jdbc:derby://localhost:1527/JclosetDB;create=true';
  ```
- A questo punto avviare il server ftp andando sempre col terminale all'interno della directory apache-ftpserver-1.0.6 e inserendo il seguente comando:
  ```
  bin/ftpd.sh res/conf/ftpd-typical.xml
  ```
- A questo punto è possibile avviare da Eclipse, il server (con TomCat) e successivamente il Client.

4 La comunicazione e i protocolli utilizzati

4.1 Le scelte progettuali

Tutti i protocolli utilizzati nel progetto sono i seguenti:

Protocollo	Protocollo di trasporto	Porta	Utilizzo comune
FTP	TCP	21	Trasporto file su internet
SMTP	TCP	25	Protocollo per invio di email
HTTP	TCP	80	Usato per la navigazione sul web

Le motivazioni che hanno portato a queste scelte sono le seguenti:

- La scelta dell'HTTP invece è ricaduta grazie alla sua agilità ed efficienza. Inoltre abbiamo constatato che anche altri protocolli come XML-RPC oppure webService come AXIS adottasseri lo stesso protocollo HTTP.
- Infine FTP garantisce una maggiore efficienza durante il trasferimento di file di grandi dimensioni e i limiti della dimensione del file sono dati dalla quantità di spazio sul server. FTP permette anche la ripresa automatica dei download di file o upload senza un gestore di download separato, questo è particolarmente utile per gli utenti con connessioni Internet lente o non affidabili.

Seguono brevi descrizioni di ogni protocollo utilizzato:

4.2 XML

XML(eXtensible Markup Language) è un linguaggio di markup, ovvero un linguaggio marcatore basto su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.
La nostra scelta è ricaduta su esso per la sua grande semplicità nel descrivere dati. Siccome i metadati da inviare nel nostro progetto erano in grande quantità ci è sembrata la scelta migliore. Le classi per la gestione dei file Xml è stata interamente scritta da noi (abbiamo scritto la definizione di **XmlImpl** per gestire il dato Xml stesso) e non è stata utilizzata nessuna libreria esterna.

4.3 HTTP

HTTP (protocollo di trasferimento di un ipertesto) è usato come principale sistema per la trasmissione di informazioni sul web.
Un server HTTP generalmente resta in ascolto delle richieste dei client sulla porta 80 usando un protocollo TCP a livello di trasporto.
L'HTTP funziona su un meccanismo richiesta/risposta (client/server): il client esegue una richiesta e il server restituisce la risposta.
Il messaggio di richiesta è composto da tre parti:

- riga di richiesta (request line)
- sezione header (informazioni aggiuntive)
- body (corpo del messaggio)

Il messaggio di risposta è di tipo testuale ed è composto da tre parti:

- riga di stato (status-line)
- sezione header
- body (contenuto della risposta)

Nel nostro progetto HTTP viene utilizzato per quasi tutte le operazioni di comunicazioni col server (in seguito verrà spiegato nel dettaglio).

4.4 SMTP

SMTP (Simple Mail Transfer Protocol) è il protocollo standard per la trasmissione via internet di e-mail.
I protocolli utilizzati per ricevere posta sono invece il protocollo POP e l'IMAP.
E' un protocollo relativamente semplice, testuale, nel quale vengono specificati uno o piu destinatari di un messaggio, verificata la loro esistenza il messaggio viene trasferito. E' abbastanza facile verificare come funziona un server SMTP mediante un client Telnet. Il protocollo SMTP utilizza come protocollo di livello transport TCP. Il client apre una sessione TCP verso il server sulla porta 25.
Poichè SMTP è un protocollo testuale basato sulla codifica ASCII non è permesso trasmettere direttamente testo composto con un diverso set di caratteri e tanto meno file binari.
Tale protocollo viene utilizzato per inviare la mail di conferma all'utente.

4.5 FTP

FTP (File Transfer Protocol) è un protocollo per la trasmissione di dati tra host basato su TCP.
Gli obiettivi principali di FTP descritti nella sua RFC ufficiale furono:

- Promuovere la condivisione di file (programmi o dati)
- Incoraggiare l'uso indiretto o implicito di computer remoti.
- Risolvere in maniera trasparente incompatibilità tra differenti sistemi di stoccaggio file tra host. - - - Trasferire dati in maniera affidabile ed efficiente.

dove:

- **PI** (*protocol interpreter*) è l'interprete del protocollo, utilizzato da client (User-PI) e server (Server-PI) per lo scambio di comandi e risposte. In gergo comune ci si riferisce ad esso come "canale comandi".
- **DTP** (*data transfer process*) è il processo di trasferimento dati, utilizzato da client (User-DTP) e server (Server-DTP) per lo scambio di dati. In gergo comune ci si riferisce ad esso come "canale dati".

FTP, a differenza di altri protocolli come per esempio HTTP, utilizza due connessioni separate per gestire

comandi e dati. Un server FTP generalmente rimane in ascolto sulla porta 21 TCP a cui si connette il client. La connessione da parte del client determina l'inizializzazione del canale comandi attraverso il quale client e server si scambiano comandi e risposte. Lo scambio effettivo di dati (come per esempio un file) richiede l'apertura del canale dati, che può essere di due tipi.

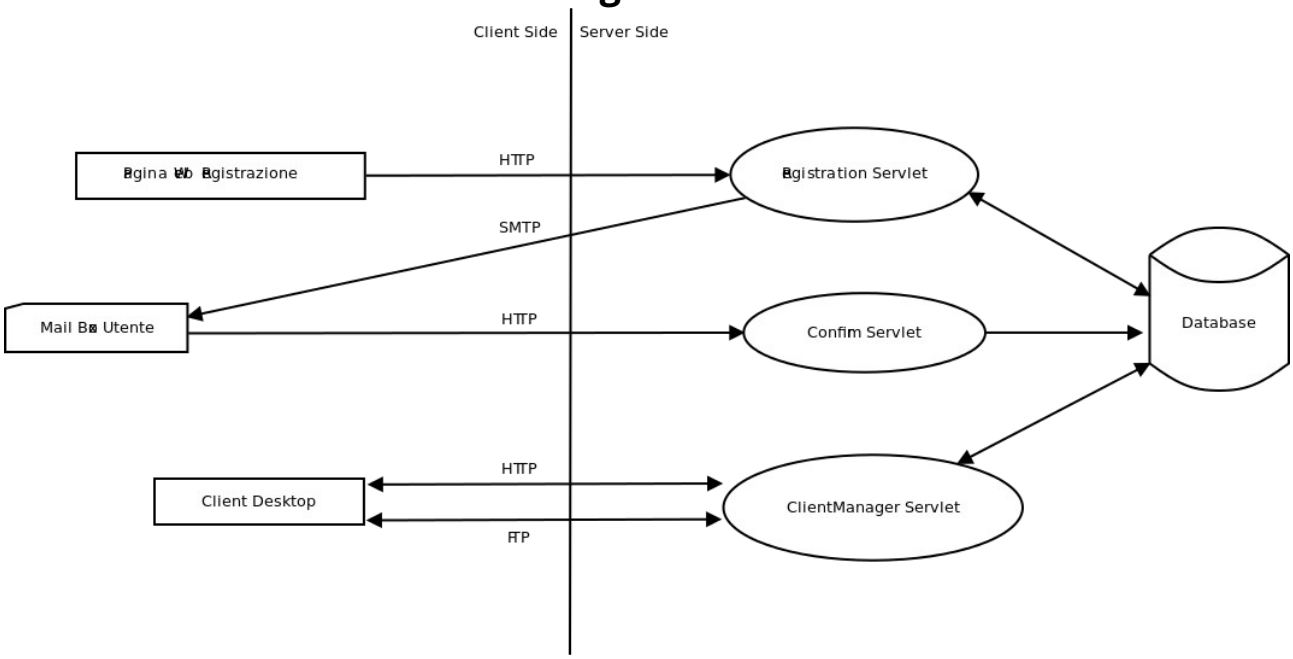
In un canale dati di tipo **attivo** il client apre una porta solitamente casuale, tramite il canale comandi rende noto il numero di tale porta al server e attende che si connetta. Una volta che il server ha attivato la connessione dati al client FTP, quest'ultimo effettua il binding della porta sorgente alla porta 20 del server FTP. A tale scopo possono venire impiegati i comandi **PORT** o **EPRT**, a seconda del protocollo di rete utilizzato.

In un canale dati di tipo **passivo** il server apre una porta solitamente casuale (superiore alla 1023), tramite il canale comandi rende noto il numero di tale porta al client e attende che si connetta. A tale scopo possono venire impiegati i comandi **PASV** o **EPSV**, a seconda del protocollo di rete utilizzato.

Sia il canale comandi, sia il canale dati sono delle connessioni TCP; FTP crea un nuovo canale dati per ogni file trasferito all'interno della sessione utente, mentre il canale comandi rimane aperto per l'intera durata della sessione utente, in altre parole il canale comandi è persistente mentre il canale dati è non persistente.

FTP fornisce inoltre un sistema di autenticazione in chiaro (non criptato) degli accessi. Il client che si connette potrebbe dover fornire delle credenziali a seconda delle quali gli saranno assegnati determinati privilegi per poter operare sul filesystem. L'autenticazione cosiddetta "anonima" prevede che il client non specifichi nessuna password di accesso e che lo stesso abbia privilegi che sono generalmente di "sola lettura".
FTP viene utilizzato nel programma per gestire il download e l'upload di file.

4.6 La comunicazione nel dettaglio



In questo paragrafo verrà descritto precisamente il funzionamento dell'applicazione iniziando dalla comunicazione tra il client desktop e la servlet ClientManager.
La prassi nella comunicazione è la medesima in quasi tutte le parti del programma e per spiegarla prendiamo in esempio l'operazione di **refresh** con la quale il client chiede al server la lista di directory e file presenti nella present work directory ovvero la directory in cui mi trovo sul momento col client:

- Premendo il pulsante “aggiorna” o compiendo una qualsiasi azione si chiama il metodo refresh implementato in ApplicationImpl (l'applicazione vera e propria). A questo punto si istanzia la classe XMLUtility che si occupa di tale operazione e nella fattispecie si istanzia XmlUtilityForVIew.
- Refresh prende in input una stringa pwd che rappresenta la posizione attuale. Dopo che si accede tramite login viene automaticamente impostata alla root directory dell'utente (cioè la directory avente il nome stesso dell'utente).
- Ora si crea il file XML di richiesta tramite XMLUtilityForView e dara come output il seguente file:

  ```
  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <view_request pwd="/dani514@gmail.com/">
  <mail>dani514@gmail.com</mail>
  <password>ciao</password>
  <path>/dani514@gmail.com/</path>
  </view_request>
  ```

  In tale file troviamo, come in tutte le operazioni, il ripresentarsi della mail e password e ogni volta a livello server viene rifatta l'autenticazione per poter svolgere l'operazione.
- Tale XML viene convertito in stringa e impacchettato in una richiesta post come la seguente:

  ```
  action=view&content=<?xml version="1.0" encoding="UTF-8" standalone="no"?
  ><view_request
  pwd="/dani514@gmail.com/"><mail>dani514@gmail.com</mail><password>ciao</pas
  sword><path>/dani514@gmail.com/</path></view_request>
  ```

- Tale stringa viene inviata grazie al metodo di NetworkManagerImpl, sendByPost al server.

- Il server a questo punto riceverà il messaggio grazie a doPost e facendo il parsing del tag action capirà il tipo di azione da intraprendere:
  - Istanzierà a sua volta XMLUtilityForView e lancerà il metodo per leggere il file XML e fare il parsing. Una volta estratte dal file le variabili mail, password e path contatterà il database prima per controllare le credenziali dell'utente e poi per trovare i file e directory nella data pwd. I risultati verranno salvati su un file XML di risposta e inviati al client.
- Il client ora leggerà la risposta grazie al metodo readXml di XMLUtilityForView che in tal caso darà in output una lista di **DirectoryImpl** e una lista di **FileImpl** da passare alla parte grafica e visualizzarla su nella tabella principale.
- Infine viene chiusa la connessione con closeConnection.

Tale procedimento viene eseguito per tutte le altre operazioni del programma cambiando di volta in volta le istruzioni svolte all'interno di readXml e createXml.
Se la risposta del server è un booleano invece, vengono utilizzati i metodi createSuccessTest e verifySuccessTest di XmlUtility.
Tuttavia, questa è un operazione di visualizzazione per cui il fallimento è accettabile. Mentre per operazioni come creare una directory o spostare un file abbiamo previsto un passaggio intermedio dove viene prima controllata a livello server la fattibilità della operazione, interrogando il database, e dopodichè viene effettuata l'operazione fisica sul file-system del server.
Ad esempio, nella porzione di codice qui proposta viene descritto come viene controllata se in caso di creazione di una directory esista o meno una directory nella pwd con lo stesso nome. Viene creata solo in caso negativo.

```
try {
	Boolean existDuplicateDir = true;
	try {
		existDuplicateDir = this.fileSystemManager.isThereSameDirectory(pwd,
name);
	} catch (SQLException e) {
		e.printStackTrace();
	}
	if(!existDuplicateDir){
		DirectoryManager.createDirectory(pwd, name);
		Directory directory = null;
		try {
			directory = DirectoryManager.getDirectoryInfo(pwd, name,mail);
		} catch (ParseException e) {
		e.printStackTrace();
		}
		try {
			this.fileSystemManager.createDirectory(pwd, mail, name,
directory.getSize(), directory.getLastModify(), directory.getCreation());
		} catch (SQLException e) {
			e.printStackTrace();
		}
		}
		//Controllo la presenza di directory duplicate
		String result =
XmlUtility.DocumentToString(XmlUtility.createSuccessTest(!existDuplicateDir));
		response.setContentType("text/xml");
		PrintWriter out = response.getWriter();
		out.println(result);
		out.flush();
```

Per quanto riguarda invece la registrazione:
Quando lanciamo JclosetServer ci viene posta la pagina di registrazione dove metteremo le nostre credenziali e tramite http con metodo POST vengono inviate alla servlet Registration.

Chi si registra viene inserito nella tabella ATTIVAZIONI e nel frattempo tramite protocollo SMTP viene mandata una e mail all'indirizzo specificato nel form. L'utente dovrà cliccare su un link contenente una variabile generata casualmente e salvata sul database che dovrà poi combaciare.
Una volta confermata la nostra registrazione all'interno del database veniamo tolti nella tabella ATTIVAZIONI e veniamo inseriti nella tabella USERS e ci viene creata una directory iniziale.
Infine per quanto riguarda l'invio e la ricezione dei file abbiamo utilizzato il protocollo FTP. Prima di inviare o scaricare ogni file avviene un passaggio di comunicazione tramite HTTP e XML per determinare la fattibilità della operazione. In caso affermativo, si procede all'invio del file tramite il metodo sendByFTP o receiveByFTP e se l'operazione va a buon fine si registrano le modifiche sul database.
Per quanto riguarda la parte server abbiamo fatto affidamento all' Apache FTP Server.

4.6 Precaricamento vs. caricamento real-time

Un dubbio progettuale che ci ha colpito all'inizio della stesura del codice è stata la seguente:
Creare una struttura dati interna al client su cui caricare i dati all'accesso e lavorare su essa o chiedere di volta in volta lo stato al server?
Inizialmente avevamo optato per la prima soluzione creando la struttura dati DirTree ovvero un' albero formato da DirNode, nodi, contenti come figli due liste una di FileImpl (ovvero i file contenuti) e una di DirNode(Ovvero la directory) e come valore del DirNode stesso una DirectoryImpl.
Questo ci avrebbe permesso di offrire all'utente operazioni locali come la ricerca rapida di file e directory e interrogazioni complesse.
Tuttavia andando avanti nello sviluppo ci accorgemmo come tale soluzione fosse inadatta per un'applicazione online in quanto, in previsione di una remota possibilità di ampliare l'applicazione con altre interfacce (come potrebbe essere un sito AJAX come PyDio), tale soluzione non sarebbe potuta più essere sostenibile poichè fonte di errori di sincronizzazione.

4.7 I controlli sulle operazioni e la sicurezza

Come già accennato in precedenza i controlli vengono effettuati solamente a livello di server e sul database.
Una volta che si riceve la richiesta per una data operazione, prima di eseguirla fisicamente attraverso i metodi della classe **DirectoryManager** si controlla la fattibilità interrogando il database.
I controlli effettuati sono:

- Controllo se l'utente è già registrato
- Controllo se esiste già un file o una directory con lo stesso nome nella data pwd
- Controllo l'esistenza del dato path
- Controllo se la directory è vuota

Questi controlli vengono effettuati per mantenere corretto il file-system e per mantenere vigenti alcune restrizione da noi adottate per semplificare la sua gestione:

- Non è possibile eliminare directory non vuote
- Non è possibile spostare directory non vuote
- Non è possibile spostare directory o file in directory contenenti directory o file con lo stesso nome

Per quanto riguarda la sicurezza, ogni richiesta del client contenente le credenziali dell'utente e le operazioni prevedono un nuovo login livello server per ognuna di esse. Questo evita che un utente possa compiere azioni su file o cartelle di altri utenti.
Tuttavia, le password ed email viaggiano in modo continuo (per ogni azione) in chiaro. Questo implica una grande facilità di intercettazioni di tali dati attraverso sniffing.
Il sistema è vulnerabile anche ad attacchi di tipo SQLInjection.
Infine sia i file che le directory salvate fisicamente sul server non sono cifrate il che rappresenta un pericolo per la privacy dell'utente.

5 Sviluppi futuri e conclusioni

5.1 Sviluppi futuri

Considerata la grandissima vastità del progetto la mole di sviluppi futuri è secondo noi consistente ma allo stesso tempo affascinante in quanto, grazie all'impostazione robusta conferita da noi al progetto sono facilmente implementabili.
I principali sono:

- Gestione dello spazio disponibile ad ogni utente così come avviene nelle piattaforme commerciali.
- Dislocazione del file-system in più server.
- Implementazione della sicurezza. Essa rappresenta il punto debole del progetto in quanto viene poco considerata rispetto ad altri aspetti. Punti su cui partire sono sicuramente utilizzare HTTPS anziché HTTP, cifrare la password, cifrare i file dell'utente magari implementando sistemi di sicurezza avanzati.
- Aggiungere al client un driver per la sincronizzazione automatica di una cartella scelta dall'utente.
- Permettere la condivisione di file con altri utenti del servizio o con servizi esterni come i social network
- Porre controlli sulle dimensioni massime dei file.

5.1 Conclusioni

L'esperienza maturata grazie a questo progetto è stata senz'altro positiva in quanto ci ha portato a capire il grandissimo numero di aspetti da considerare quando si affrontano servizi di tale complessità e il grande numero di possibilità che sono offerte per implementarne ogni parte.