*Università degli Studi di Modena e Reggio Emilia*

---

Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica – *Nuovo Ordinamento*

# Analisi e prototipazione di una interfaccia utente che consenta la realizzazione di semplici pagine web sulla base di template

Relatore:
Prof. Sonia Bergamaschi

Candidato:
Lorenzo Ballasini

Anno Accademico 2003-2004

# *RINGRAZIAMENTI*

*Desidero ringraziare la Prof.ssa Sonia Bergamaschi e l'Ing. Maurizio Vincini per tutto l'aiuto fornito durante la realizzazione di questa tesi.*

*Ringrazio la mia famiglia, per l'incrollabile fiducia concessa durante tutti gli anni di studio e per l'affetto che da sempre mi dimostrano.*
*Un ringraziamento particolare va ad Elisa, che mi è stata vicina nei momenti belli, ma soprattutto in quelli difficili.*

*Ringrazio tutti gli amici, che mi hanno sostenuto e soprattutto sopportato. Sperando di non dimenticare nessuno, in ordine di apparizione, un grazie a: Claudio (il mio ing. di fiducia), i ragazzi dell'ITIS Parma (Simone R., Cristian, Luca, Fabio, Simone P., Marco, Alan), Mariano, Mara, i compagni di corso dell'UniPR (Carlo, Francesco, Andrea, Matteo), Gianalberto, Mattia, i miei grandi compagni di appartamento Davide e Angelo (che hanno resistito alla mia cucina per ben due anni), Riccardo e Daniele (che spero resisteranno per altri due), gli amici di Casalmaggiore (Angelo, Ivano, Laura, Ilaria, Corrado, Alessio, Miky, Giorgia, Martina), Sergio (la mia guida spirituale), Mirco, Serena, Matteo, Alessandro L., Giulio, Fabio, Cristina, Tiziana, Alessandro M. e tutti gli amici dell'UniMO.*

*Un sincero grazie a tutti Voi…*

# Indice

# Indice delle figure

# Introduzione

Negli ultimi anni Internet ha assunto proporzioni inaspettate, entrando nelle case e negli uffici di tutto il mondo, milioni di persone collegate giorno e notte, alla costante ricerca di informazioni di ogni genere. Il World Wide Web rappresenta la nuova frontiera della comunicazione, il mercato globale di cui tutti, in un modo o nell'altro, vorrebbero far parte.
Alla luce di questo trend, un numero sempre crescente di aziende, associazioni, attività commerciali della più varia natura, professionisti e privati manifesta la necessità di appartenere a questa variegata realtà, portando alla proliferazione dei *siti web*.
La realizzazione di un sito Internet richiede competenze che, tipicamente, sono appannaggio di tecnici e programmatori specializzati. Di conseguenza, questo gap tecnologico pone un limite alla diffusione delle informazioni sul Web.
Se da un lato, alcune aziende sono disposte ad investire ingenti somme per espandere il proprio business all'interno della rete, esistono molte realtà che non sono disposte ad impegnare risorse per veder realizzato il proprio sito.
L'applicazione sviluppata vuole andare incontro a tutti coloro che, per passione o necessità, vogliono pubblicare le proprie informazioni all'interno della rete, consentendo di realizzare un sito Internet, in modo veloce e semplice, senza investire risorse per lo sviluppo.
Per interagire con il software realizzato, non sono richieste competenze specifiche o la conoscenza di linguaggi di programmazione orientati al web, ma semplicemente una conoscenza base dell'uso di un calcolatore e della navigazione, requisiti che risultano essere oggigiorno patrimonio della maggior parte degli utenti.

Verrà ora esposta la struttura secondo la quale è organizzata questa tesi:
Nel primo capitolo è riportata una descrizione delle funzionalità dell'applicazione, sulla base dell'interfaccia grafica e delle interazioni con l'utente. Questo capitolo può essere visto come la forma cartacea del manuale utente.
Nel secondo capitolo, dopo una breve introduzione sulla programmazione *web oriented*, si procede alla trattazione dell'ambiente di sviluppo. Tale trattazione riguarda la piattaforma NetBeans IDE 3.5.1 di Sun Microsystem, il container Apache Tomcat e le JavaServer Pages (JSP). In particolare, si espongono i fondamenti del linguaggio, l'interazione con l'utente, la comunicazione tra JSP e database (JDBC) e la pubblicazione di un'applicazione JSP.
Nel terzo capitolo vengono rappresentate in dettaglio, mediante lo standard UML, le classi componenti i package dell'applicazione (*class diagram*) e le principali attività (*activity diagram*).
Il quarto capitolo si occupa di due argomenti cruciali della programmazione web oriented: la gestione delle sessioni e degli errori. Vengono esposte le tecniche offerte da JSP e le politiche di gestione impiegate all'interno dell'applicazione.
Il quinto capitolo si occupa della fase di test ed espone i principali problemi riscontrati in fase di progettazione.
L'ultimo capitolo riporta le conclusioni ed i possibili sviluppi futuri, applicabili al software.
La tesi si conclude con la bibliografia e con un breve glossario, volto a di chiarire il significato dei termini tecnici impiegati durante la trattazione.

# 1 - Panoramica dell'applicazione

L'applicazione è stata sviluppata per consentire agli utenti, privi di competenze relative alla programmazione web oriented, di realizzare, con la massima semplicità, un sito Internet. L'applicazione, sviluppata mediante JSP, è accessibile direttamente dalla rete, quindi immediatamente fruibile da tutti coloro che dispongono di un accesso al web. Per maggiori informazioni sull'accessibilità, consultare il capitolo *Ambiente di sviluppo.*

## *Login*

La fase preliminare è rappresentata dal *login*, procedura mediante la quale gli utenti registrati possono accedere all'applicazione vera e propria, mediante inserimento di *username* e *password*.


**Figura 1 - Login utente**

I dati richiesti sono scelti dall'utente al momento della registrazione (vedere il paragrafo *Registrazione di un nuovo utente*).

In fase di digitazione, lo username viene visualizzato in chiaro, mentre, per motivi di sicurezza, la password è sostituita da caratteri alternativi. In caso di errato login, l'applicazione si limita a visualizzare un messaggio di errore, al fine di non fornire informazioni utili a utenti che cercassero di accedere in modo non autorizzato.

La verifica della correttezza dei dati viene realizzata mediante un accesso a database, nel quale sono memorizzati i dati relativi alle registrazioni e ai siti web degli utenti del sistema.

## *Registrazione di un nuovo utente*

Per gli utenti non registrati, è disponibile un apposito modulo di registrazione, nel quale è richiesto l'inserimento dei seguenti dati:


**Figura 2 - Registrazione di un nuovo utente**

I dati richiesti sono sottoposti ai seguenti vincoli:

- `Nome: una stringa di testo non vuota`
- `Cognome: una stringa di testo non vuota`
- `Indirizzo e-mail: una stringa di testo non vuota`
- `Username: una stringa di testo, priva di spazi, contenente da 8 a 15 caratteri`
- `Password: una stringa di testo, priva di spazi, contenente da 8 a 15 caratteri`

Una volta inseriti e confermati i dati, l'utente viene abilitato ad accedere al servizio

Il controllo dell'input dei dati avviene mediante apposite funzioni JavaScript, richiamate da *onsubmit* del form HTML:

```
<script>
function validate(frm){
	if(!hasData(frm.nome.value)){
		alert("Il campo 'nome' non è corretto!");
		return false;
	}
	…
}
function hasData(s){
	if(s == null)
		return false;
	var n = s.length;
	for(var i=0; i<n;i++){
		var c = s.charAt(i);
		switch(c) {
			case ' ':
			case '\t':
			case '\n':
				continue;
			default:
				return true;
		}
	}
	return false;
}
```

## *Gestione dei Web attivi*

Una volta effettuato il login, l'utente accede alla pagina di gestione dei propri siti:


**Figura 3 - Elenco dei Web attivi**

Ogni utente può disporre di uno o più siti, gestibili individualmente ed in modo indipendente.
Mediante la pagina di amministrazione è possibili creare e cancellare siti web, nonché accedere alla sezione di amministrazione dei singoli siti.
Le operazioni di creazione e cancellazione sono seguite da un apposito messaggio di conferma. Le pagine create mediante l'applicazione saranno disponibili, in real time, all'indirizzo:

`http://indirizzo/nome-web/index.jsp`

dove *`indirizzo`* è l'indirizzo Internet relativo all'applicazione, mentre *`nome-web`* è il nome scelto dall'utente per il proprio sito. In conformità con gli standard per la definizione degli *URL*, tale nome non deve contenere spazi o caratteri di tabulazione.
La creazione di un nuovo sito, porta all'aggiornamento del database utenti e all'interazione con il file system locale:

1. creazione di una cartella con nome uguale a *`nome-web`*
2. creazione di *index.jsp*: file indice della struttura a frame del progetto
3. creazione di *leftframe.jsp*: frame sinistro, destinato ad ospitare il menù di navigazione del sito
4. creazione di *topframe.jsp*: frame alto, predisposto a contenere l'intestazione del sito
5. creazione di *body.jsp*: frame centrale, destinato ai contenuti della pagina iniziale (*homepage*).

La cancellazione di un sito esistente comporta la cancellazione della relativa cartella e del suo contenuto, nonché alla cancellazione della entry relativa nel database utenti.
La modifica di un sito, porta all'apertura della pagina di amministrazione.

## *Amministrazione di un Web*

La pagina di amministrazione consente di accedere a tutte le funzionalità di modifica e manutenzione del singolo sito, da parte dell'utente.


**Figura 4 - Pagina di amministrazione**

La pagina di amministrazione visualizza un insieme di bottoni, che consentono di accedere alle singole funzionalità del sistema. I bottoni sono corredati da un'icona e da un'etichetta, al fine di rendere più intuitiva l'interpretazione dell'interfaccia. Nella parte alta, a destra, è presente inoltre il bottone che consente di accedere alla pagina di *help in linea*.

Nella parte centrale della pagina di amministrazione è presente l'elenco delle pagine componenti il sito, affiancate da un controllo *radio* per la selezione delle pagine e da un'icona rappresentante la collocazione del frame all'interno della pagina.
La selezione delle pagine viene gestita mediante JavaScript, così come la maggior parte delle operazioni di navigazione all'interno delle pagine dell'applicazione:

```
<script>
//funzione di identificazione della pagina selezionata
function select_page(page, width, height)
{
    j=document.all["page"].length;
    for(i=0; i < j; i++){
        if(document.all["page"][i].checked == true){
            s=document.all["page"][i].value;
        }
    }
    window.open(page+'?file='+s, '', 'width='+width+', height='+height+', resizable=no,
    toolbar=no, menubar=no, statusbar=yes');
}
</script>
```

Si passerà ora alla descrizione dettagliata delle singole funzionalità:

## Nuova pagina

Consente di aggiungere una nuova pagina al sito corrente:

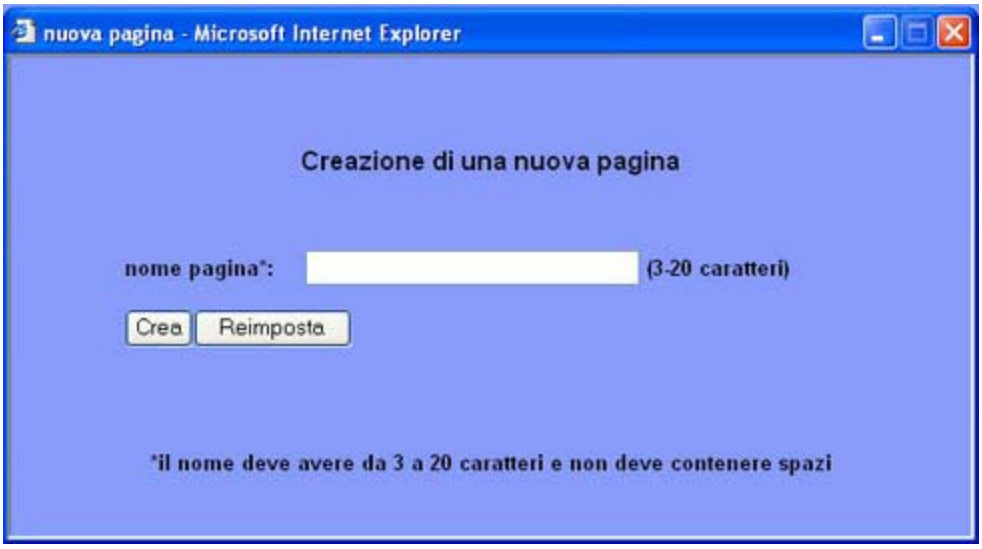


**Figura 5 - Nuova pagina**

La pressione del pulsante “nuova” comporta l’apertura della finestra *popup*, all’interno della quale l’utente è tenuto a specificare il nome della nuova pagina. Tale nome deve contenere da tre a venti caratteri, non intervallati da spazi.
La pressione del bottone “crea” porta all’invocazione del metodo *createNewFile()*, della classe Java *File* e alla successiva invocazione del metodo *writeLine(File f, String s)*, della classe *FileIO* (package *packFile*), preposto alla scrittura di linee di testo all’interno di un file. Mediante writeLine viene scritto il codice HTML relativo ad una pagina vuota. Segue il codice Java relativo al metodo *writeLine(File f, String s)*:

```
/*---------------------------------- writeLine
  scrittura di una linea in un file
*/
    public boolean writeLine(File f, String s){
        byte buf[] = s.getBytes();
        try {
            OutputStream fo=new FileOutputStream(f);
            for(int i=0;i<buf.length;i+=1){
            fo.write(buf[i]);
            }
            fo.close();
        }catch(IOException e){
            return false;
        }
        return true;
    }
```

La scrittura su file viene implementata tramite *write(byte buf[ ])* di FileOutputStream(), mentre la gestione degli errori viene demandata alla struttura *try-catch* di Java. Il valore ritornato, di tipo *boolean*, consente di controllare l’esito dell’operazione effettuata.

## Eliminazione di una pagina

Consente di eliminare una pagina dal web corrente. Solo le pagine aggiunte dall'utente sono eliminabili mentre, per garantire la consistenza del sito, le pagine *leftframe.jsp*, *topframe.jsp* e *body.jsp* non possono essere eliminate.


**Figura 6 - Cancellazione pagina**

L'utente è chiamato a selezionare la pagina da eliminare all'interno di un menù di selezione. Solo le pagine aggiunte dall'utente sono visualizzate in questo menù.
La cancellazione di una pagina per la quale esiste un riferimento all'interno del menù di navigazione (vedere il paragrafo *Menù di navigazione*), comporta al cancellazione del riferimento stesso, al fine di eliminare la presenza di *dead link*. Il metodo preposto a tale verifica è *isLinked(File f, File menu, LinkedList list)*, mentre la cancellazione dei link all'interno del menù è realizzata tramite il metodo *delLink(File f, String lable)*. Segue un estratto di *listLink*:

```
String tag = new String("<!-- menu");
...

while((s = br.readLine()) != null){
   if(s.indexOf(tag) != -1){
      while((s = br.readLine()) != null){
         if(s.indexOf("href") != -1){
            str += s.substring(s.indexOf(">",s.indexOf("href"))+1, s.indexOf("</a>",
             s.indexOf("href")+6));

            al.add(str);
            str = "";
         }
      }
   }
}
```

Analogamente, per i file contenenti un mailForm (vedere il paragrafo *Modifica pagina*), è previsto il metodo *isMailsend(File f, LinkedList list)*, che verifica la presenza di file JSP preposti all'invio di e-mail, collegati alla pagina da eliminare. In caso affermativo, l'applicazione si occupa  di rimuovere anche questi file non più necessari.

## Proprietà di una pagina

Consente di modificare le proprietà del documento corrente, quali colore di sfondo, tipo di carattere e colore del testo.

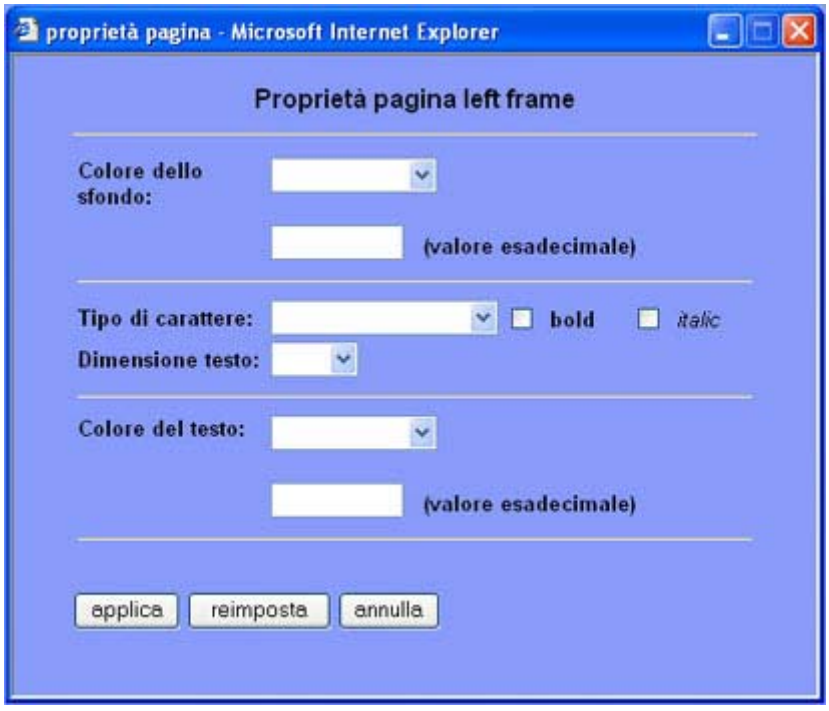


**Figura 7 - Proprietà pagina**

Al fine di consentire la massima configurabilità del sistema, le proprietà *Colore dello sfondo*, *Tipo di carattere*, *Colore del testo* sono contenute all'interno di un file denominato *config.jsp*.
La pressione del tasto *"applica"* comporta l'invocazione del metodo *ChangeAttrib(File f, String attribLabel, String newAttrib)*, preposto alla modifica di un attributo del tag HTML <*body*>. Mediante questo metodo è possibile aggiungere o modificare attributi del tipo *nome=valore*, al fine di applicare al documento le modifiche richieste dall'utente.

Esempio di documento HTML a cui sono state applicate alcune proprietà:

```
<html>

<head>
<title>Body</title>
</head>

<body text="#003366" style="font-family: Arial; font-size: small;
 font-weight: bold" bgcolor="#668CFF">

Hello world!

</body>

</html>
```

## Aggiornamento dell'interfaccia

La pressione del bottone *"aggiorna"* implica un aggiornamento della finestra di amministrazione. Analogamente al *refresh* del browser, consente di allineare il contenuto della finestra ai cambiamenti avvenuti, ad esempio alle modifiche apportate dall'utente alla struttura del sito.

Il codice preposto alla funzione di aggiornamento è il seguente:

```
<input type="image" src="images/aggiorna.jpg" value="aggiorna" name="aggiorna"
   onClick="window.location='admin.jsp'">
```

Per ottenere questa funzionalità, viene sfruttata una combinazione di HTML e JavaScript

## Anteprima

Consente di visualizzare una preview del sito corrente, al fine di consentire all'utente di valutare il lavoro svolto. La pressione del bottone implica l'apertura di una nuova finestra del browser, contenente la *homepage* del sito in costruzione.

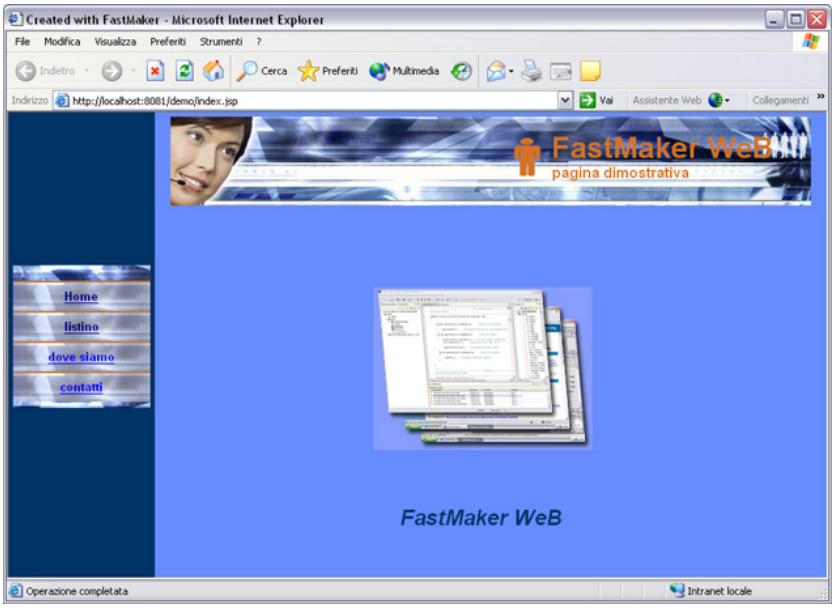


**Figura 8 - Anteprima**

Il codice che si occupa dell'apertura della pagina di anteprima è il seguente:

```
<input type="image" src="images/anteprima.jpg" value="anteprima" name="anteprima"
   onClick="window.open('<%=web%>/index.jsp','preview')">
```

Sono presenti HTML, JavaScript e una espressione JSP, mediante la quale si accede al valore della variabile *web*.

## Menù di navigazione

Il menù di navigazione è uno strumento fornito all'utente al fine di rendere agevole la navigazione all'interno del sito in costruzione. Collocato nel frame sinistro (*leftframe.jsp*), il menù di navigazione è costituito da un'insieme di link di cui solo "home" è predefinito, mentre gli altri sono definibili dall'utente. Questo componente viene gestito mediante un'apposita interfaccia, alla quale si accede mediante la pressione del bottone *menù*, posta alla destra di *leftframe*, nell'elenco delle pagine:


**Figura 9 - Gestione del menù di navigazione**

Le funzioni a disposizione dell'utente sono le seguenti:

- Creazione del menù: Permette di dotare il sito corrente di un menù di navigazione. La creazione del menù si basa su *template*, al fine di consentire l'immissione di elementi grafici anche agli utenti privi di competenze specifiche. Le scelte operabili dall'utente sono l'interfaccia grafica per il menù e il colore del testo dei link.


**Figura 10 - Creazione del menù di navigazione**

La creazione del menù prevede innanzitutto la copia delle immagini relative all'interfaccia scelta dall'utente, mediante il metodo *copyFile(File dest, File surce)*, che effettua la copia di file da sorgente a destinazione.
Successivamente, mediante il metodo *insertMenu(File f, String color)*, l'applicazione provvede ad inserire l'opportuno codice HTML all'interno del file *leftframe.jsp*.
Segue un estratto del codice di *insertMenu*:

```
public boolean insertMenu(File f, String color){
        String s = new String();
        String tag = new String("<body");
        String path = f.getPath();
        File temp = new File(path+".tmp");

        try {
            FileReader fr = new FileReader(f);
            BufferedReader br= new BufferedReader(fr);
            FileWriter fw = new FileWriter(temp);
            while((s = br.readLine()) != null){
                if(s.indexOf(tag) != -1){
                    fw.write(s);
                    fw.write("\n");
                    fw.write("<!-- menu\n");
                      ...    //scrittura su file del codice HTML
                    fw.write("<!-- /menu\n");
                }
                else{
                    if(s.indexOf("menu") != -1){
                      while(((s = br.readLine())!=null)&&(s.indexOf("/menu")==-1)){
                            continue;
                       }
                       s = br.readLine();
                    }
                    fw.write(s);
                    fw.write("\n");
                }
            }
            ...       //rilascio delle risorse (chiusura file)
            f.delete();
            temp.renameTo(f);
        }catch(IOException e){
            return false;
        }
        return true;
    }
```

- Cancellazione del menù: consente di rimuovere il menù di navigazione ed i link in esso contenuti. Il metodo preposto all'eliminazione del menù è *deleteMenu(File f)*, mentre il controllo della presenza del menù all'interno del file è demandato al metodo *existMenu(File f)*.

- Selezione della grafica: permette di modificare l'interfaccia grafica, senza rimuovere i link contenuti nel menù. Viene implementato mediante *copyFile(File dest, File surce)* e *changeMenu(File f, String color)*.

- Aggiunta di un link: consente di aggiungere al menù un riferimento ipertestuale ad una delle pagine del web corrente. L'utente più inoltre scegliere l'etichetta da visualizzare all'interno dell'interfaccia. L'aggiunta di un link è gestita dal metodo *addLink(File f, File link, String lable)*.


**Figura 11 - Aggiunta di un link al menù di navigazione**

- Cancellazione di un link: elimina uno dei link presenti nel menù di navigazione.

## Intestazione grafica

In accordo con le specifiche di progetto, l'utente può dotare il proprio sito di una intestazione grafica. Analogamente al menù di navigazione, viene proposto un insieme personalizzabile di *template*, caricati dinamicamente dall'applicazione, al quale l'utente può accedere.

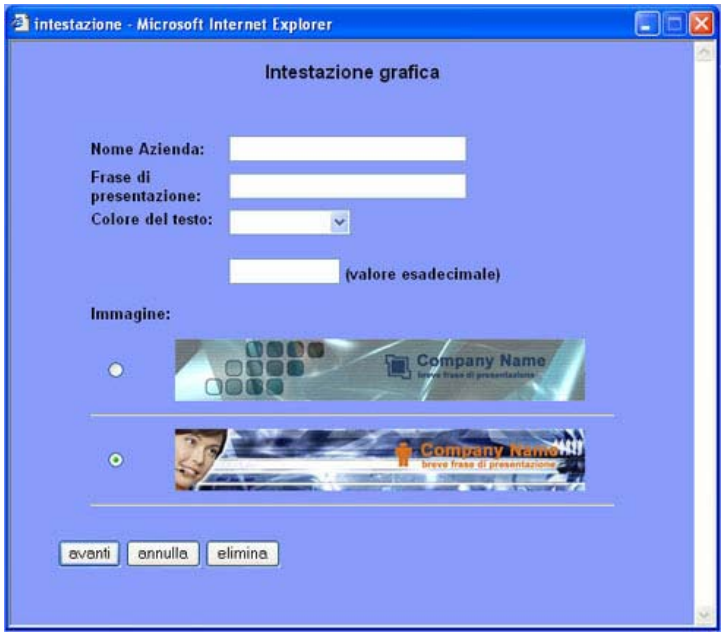


**Figura 12 - Intestazione grafica**

Oltre alla scelta dell'interfaccia, l'utente è chiamato ad immettere il nome dell'azienda, o un nome rappresentativo per il sito, una breve frase di presentazione e il colore del testo. Tali informazioni verranno inserite all'interno dell'intestazione.
Questo stesso modulo può essere utilizzato per modificare o per cancellare un'intestazione preesistente.

I metodi preposti alla gestione dell'intestazione sono i seguenti:

- *copyFile(File dest, File surce)* per la copia del file immagine dell'intestazione
- *insertIntest(File f, String image, String name, String present, String color)* per l'inserimento di una nuova intestazione
- *existIntest(File f)* per verificare la presenza dell'intestazione
- *deleteIntest(File f)* per eliminare un'intestazione

In caso di modifica di un'intestazione preesistente, l'applicazione si occupa di caricare all'interno del modulo i dati da file (titolo e frase di presentazione), al fine di evitare all'utente la reimmissione degli stessi.

## Modifica di una pagina

La modifica di una pagina rappresenta la parte centrale e più complessa dell'applicazione. Il modulo preposto a tale operazione presenta numerose funzionalità, al fine di consentire l'inserimento dei principali elementi di una pagina web. Tali funzioni sono implementate mediante singoli moduli HTML, preposti a raccogliere le preferenze dell'utente, accessibili mediante un menù visualizzato nella parte sinistra dello schermo. L'utente può valutare, real time, le modifiche apportate alla pagina mediante una preview presente nella parte destra dello schermo, che viene aggiornata dopo ogni operazione.
Sono presenti due ulteriori funzioni di cancellazione di un singolo elemento e di re-inizializzazione della pagina.

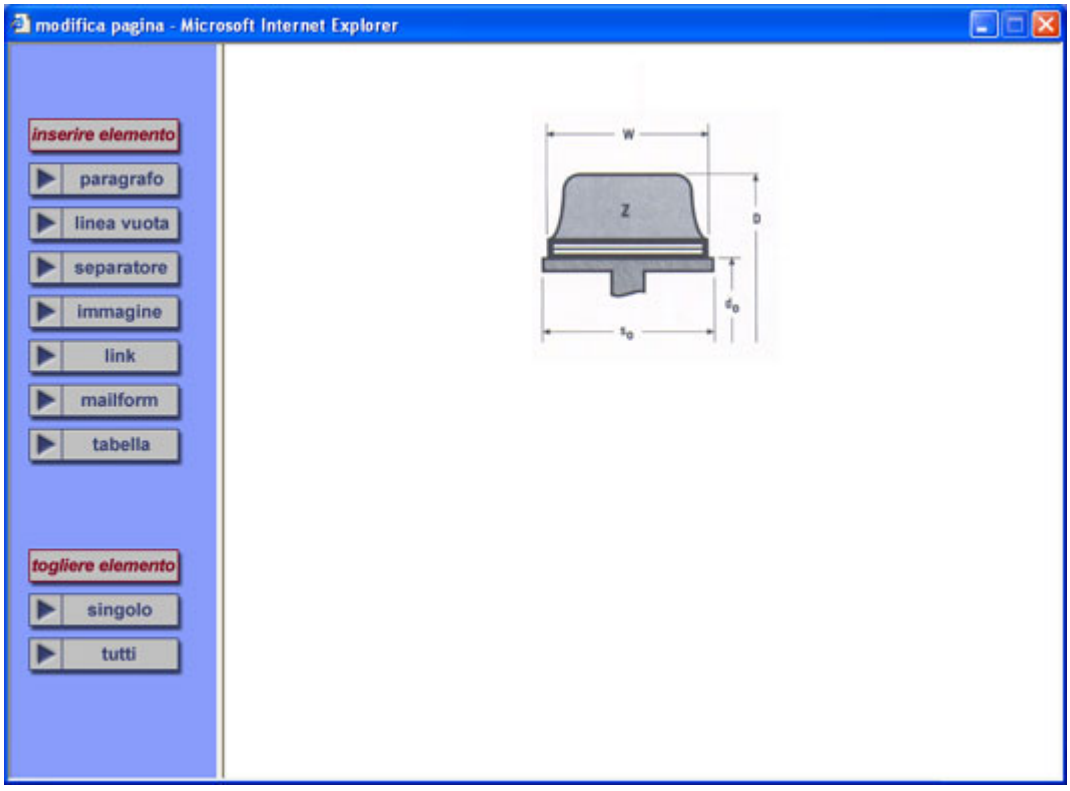


**Figura 13 - Modifica di una pagina**

Si descriveranno ora, in dettaglio, le singole funzionalità di modifica:

***Inserimento di un paragrafo***
Consente di inserire un paragrafo di testo, applicandovi una serie di preferenze:


**Figura 14 - Inserimento di un paragrafo**

Il form HTML raccoglie il testo e le preferenze dell'utente. Le informazioni vengono successivamente elaborate da un documento JSP, che le estrae mediante *request.getParameter(String name)*, metodo che preleva il parametro "*name*" dall'oggetto riferito alla richiesta http, e le passa al metodo *insertParagraph(File f, String text, String textAlign, String font, String bold, String italic, String dimFont, String textColor, String comments)*, incaricato di scrivere su file il codice HTML.
Di particolare rilevanza è il parametro *comments*, che controlla l'inserimento dei commenti HTML preposti a delimitare un paragrafo. Al fine di garantire un corretto funzionamento dell'applicazione, tali commenti non devono essere presenti quando il paragrafo viene collocato all'interno di una tabella.

***Inserimento di una linea vuota***
Permette all'utente di inserire all'interno del documento una linea vuota. La pressione del relativo bottone non comporta l'apertura un form HTML, dato che questo elemento non richiede configurazioni, ma si limita a richiamare il metodo *insertEmpty(File f, String comments)*, che si occupa della scrittura del codice HTML su file. Riguardo al parametro *comments*, valgono le stesse considerazioni viste in precedenza.

***Inserimento di un separatore***
Consente l'inserimento di una linea orizzontale di separazione, che in HTML viene ottenuta mediante il tag **<HR>.**


**Figura 15 - Inserimento di un separatore**

L'utente è tenuto a definire, oltre all'allineamento, la larghezza (in percentuale o in pixel), l'altezza e il colore del separatore. Il metodo preposto all'inserimento del separatore è *insertLine(File f, String allineamento, String larghezza, String altezza, String colore, String continua, String comments)*. Segue il codice di insertLine:

```
public boolean insertLine(File f, String allineamento, String larghezza, String altezza,
                          String colore, String continua, String comments){

        String s = new String();
        String tag = new String("</body");
        String path = f.getPath();
        File temp = new File(path+".tmp");
        int count = countTag(f,"line");

        try {
            FileReader fr = new FileReader(f);
            BufferedReader br= new BufferedReader(fr);
            FileWriter fw = new FileWriter(temp);

            while((s = br.readLine()) != null){
                if(s.indexOf(tag) != -1){
                    if(comments.equals("yes")){
                        fw.write("<!-#- line "+count+">\n");
                    }
                    fw.write("<hr align=\""+allineamento+"\" width=\""+larghezza+"\"
                              size=\""+altezza+"\" color=\""+colore+"\""+continua+">\n");
                    if(comments.equals("yes")){
                        fw.write("<!-#- /line "+count+">\n");
                    }
                }
                fw.write(s+"\n");
            }
            fw.close();
            br.close();
            fr.close();

            f.delete();
            temp.renameTo(f);

        }catch(IOException e){
            return false;
        }
        return true;
    }
```

*insertLine* sfrutta il metodo *countTag(File f, String mytag)* che restituisce il numero delle occorrenze di un determinato tag, contenute all'interno di un file.

### *Inserimento di un'immagine*

E' un modulo per l'inserimento di un'immagine gif o jpg, presente nel file system locale dell'utente, all'interno del documento:

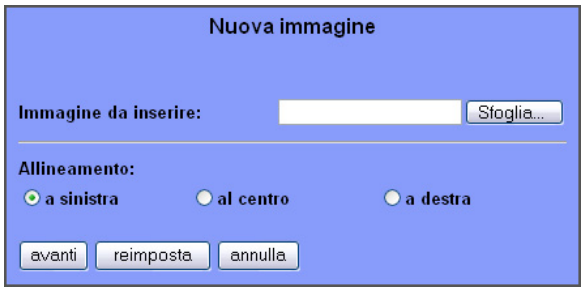


**Figura 16 - Inserimento di un'immagine**

Per consentire il trasferimento di un file immagine, viene utilizzato un controllo di tipo *file* (vedere il capitolo *Ambiente di sviluppo*), rappresentato da un campo di testo e da un pulsante sfoglia. Il nome del file può essere introdotto direttamente, mediante digitazione, o invocando l'apertura di una finestra di dialogo, mediante la pressione del bottone.

Per utilizzare il controllo file, il form HTML deve avere le seguenti caratteristiche: il metodo della richiesta deve essere POST e il tipo di codifica (specificato dall'attributo **enctype** del tag **<FORM>**) deve essere **multipart/form-data.**

Se queste condizioni non sono verificate, il controllo viene comunque visualizzato, ma al server verrà inviato semplicemente il nome del file selezionato dall'utente.

La decodifica del file inviato è demandata ad un apposito codice JSP che si occupa di:

- Estrarre ed interpretare l'header della richiesta HTTP, al fine di estrarre il *boundary* (separatore) per la richiesta corrente
- Verificare che il file inviato sia di tipo consentito (immagine gif o jpg)
- Copiare il file inviato, nella cartella relativa del web corrente, sul file system del server
- Estrarre la preferenza espresse dall'utente riguardo all'allineamento dell'immagine

Nel caso di invio di un file di formato non valido, l'utente viene avvisato da un messaggio di errore.

Di particolare interesse, risulta essere il codice di estrazione dell'header HTTP:

```
//dichiarazione variabili
String boundary= new String();  //stringa di separazione
Enumeration names = request.getHeaderNames();   //header della richiesta

//ricerca della stringa di separazione per il file inviato
while (names.hasMoreElements()) {
    String name = (String) names.nextElement();
    Enumeration values = request.getHeaders(name);  // support multiple values
    if (values != null) {
        while (values.hasMoreElements()) {
            String value = (String) values.nextElement();
            if(value.indexOf("boundary") != -1){
               //salvataggio del separatore in una stringa
               boundary = value.substring(value.indexOf("boundary")+"boundary=".length());
            }
        }
    }
}
```

### *Inserimento di un link*

Permette all'utente di inserire un collegamento ipertestuale, che può essere una risorsa URL o una pagina del sito corrente:

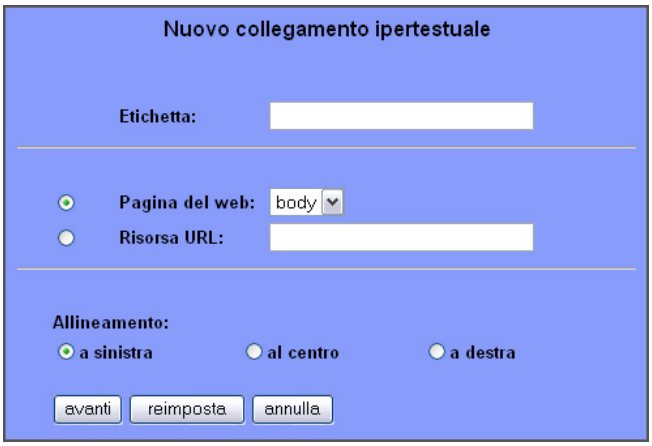


**Figura 17 - Inserimento di un link**

Oltre all'etichetta da associare al link, l'utente può selezionare il tipo di risorsa da collegate (URL o pagina del web). Il metodo preposto all'inserimento del codice HTML relativo è *insertLink(File f, String lable, String URLresource, String target, String align, String comments)*. Il campo *target* specifica il frame in cui visualizzare la risorsa collegata.

### *Inserimento di un mailForm*

Consente all'utente di inserire, all'interno della pagina corrente, un modulo di richiesta informazioni:


**Figura 18 - Inserimento di un mailform**

La sezione *impostazioni* consente l'inserimento dell'indirizzo del destinatario (indirizzo presso il quale saranno inviate le e-mail) e l'oggetto del messaggio. Nella

sezione successiva è possibile selezionare i campi da includere e specificare quali di questi sono obbligatori.
Le modifiche alla sorgente HTML vengono applicate dal metodo *insertMailForm(File f, String mailDest, String object, LinkedHashMap hm, String comments)*.
L'applicazione si occupa inoltre di creare, all'interno della cartella relativa al sito corrente, un file *mailsend.jsp*, finalizzato a contenere il codice di interpretazione e di invio delle e-mail provenienti dal mailform pubblicato sul web.
La formattazione del testo visualizzato all'interno della pagina, sarà determinata dalle preferenze impostate mediante il modulo di modifica delle proprietà.

### *Inserimento di una tabella*

L'inserimento di una tabella risulta essere il modulo più articolato, all'interno della sezione di modifica. Consente l'inserimento di una tabella e la modifica delle singole celle della stessa:

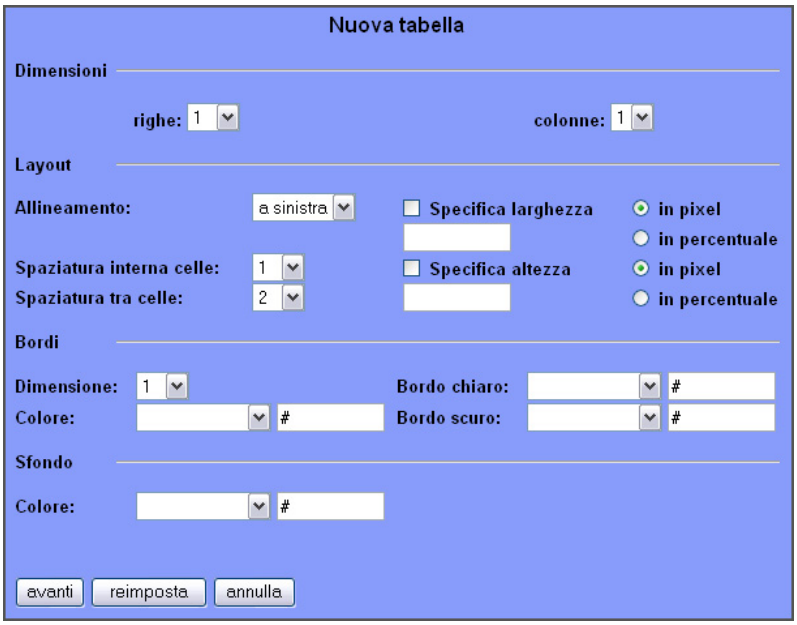


**Figura 19 - Inserimento di una tabella**

L'interfaccia presenta numerosi parametri di configurazione, tra i quali i principali risultano essere il numero di righe (da una a dieci) e di colonne (da una a cinque). L'utente può specificare inoltre un insieme di preferenze relative alla formattazione della tabella: allineamento, dimensioni, spaziatura e colore dello sfondo e del bordo.

Oltre all'interfaccia di configurazione, l'applicazione comprende due pagine JSP che consentono, sfruttando una struttura ricorsiva, la modifica delle singole celle della tabella. All'interno di ogni cella, l'utente può scegliere se inserire uno degli elementi analizzati nei paragrafi precedenti o uno spazio vuoto (empty cell). Tale operazione di inserimento viene realizzata richiamando il modulo relativo all'elemento da inserire.

Il FORM visualizzato in figura 19 si occupa di passare le preferenze campionate a *add_tab_edit.jsp*, sorgente preposta alla modifica delle singole celle, mediante un controllo della cella corrente, iterato fino al completamento dell'intera tabella.

Data la particolarità dell'operazione, si riporta a seguito un estratto del codice relativo al controllo della cella corrente e al meccanismo di iterazione mediante pagine JSP:

```
//lettura attributi di sessione
String righe = ((String)session.getAttribute("righe"));
String colonne = ((String)session.getAttribute("colonne"));

//scrittura attributi di sessione
session.setAttribute("targetpage","../add_tab_edit.jsp");   //target  della redirezione
session.setAttribute("comments","no");  //evita la scrittura dei commenti html

//individuazione cella corrente
int currentRow=fa.stringToInt((String)session.getAttribute("currentRow"));
int currentCol=fa.stringToInt((String)session.getAttribute("currentCol"));

//gestione delle righe e delle colonne, con scrittura di html sul file in analisi
if(currentCol!=-1){
    fa.writeTag(f,"   </td>");
}
if(currentCol==cols){
        fa.writeTag(f,"  </tr>");
}
currentCol+=1;
if(currentCol>=cols){
    currentRow+=1;
    currentCol=0;
}
//aggiornamento attributi di sessione
session.setAttribute("currentRow",fa.intToString(currentRow));
session.setAttribute("currentCol",fa.intToString(currentCol));

//se vi sono ancora celle da modificare
if(currentRow<rows)
{
       …      //codice di modifica della singola cella e iterazione
}
//se non vi sono più celle da editare
else{
//chiusura tabella
fa.writeTag(f,"  </tr>");
fa.stopTable(f);

//aggiornamento attributi di sessione
session.setAttribute("targetpage", thispage);
session.setAttribute("comments","yes");
```

Come si può vedere nel codice, la traccia della cella corrente viene mantenuta mediante le variabili di sessione *currentRow* e *currentCol*. La modifica del file HTML, contenente la tabella, viene demandata ai metodi di inserimento visti in precedenza e ai metodi di inizio tabella *startTable(parametri)*, di fine tabella *stopTable(File f)*, e di scrittura di un tag *writeTag(File f, String newtag)*. Ecco l'interfaccia di editing:


**Figura 20 - Editing di una cella**

### *Cancellazione di un singolo elemento*

Consente all'utente di rimuovere un elemento dalla pagina corrente, specificandone il nome:


**Figura 21 - Cancellazione di un singolo elemento**

La cancellazione di un elemento viene realizzata dal metodo *deleteElement(File f, String element)*, che sfrutta i commenti HTML che delimitano i singoli componenti della pagina. Ad esempio, nel seguente codice HTML:

```
<!-# - paragraph 0>
<p align="center"><font face="Arial" size="14 pt" color="#003366">abc</font></p>
<!-# - /paragraph 0>
```

sono ben visibili i commenti di inizio e fine del paragrafo 0, primo paragrafo della pagina.

### *Cancellazione di tutti gli elementi*

Permette di cancellare tutti gli elementi contenuti in una pagina del sito web corrente:

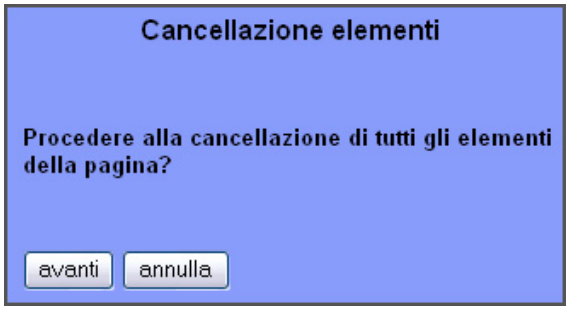


**Figura 22 - Cancellazione di tutti gli elementi**

Il modulo sfrutta *getElements(File f, LinkedList list)* per ottenere la lista degli elementi contenuti nella pagina e li passa al metodo *deleteElement(File f, String element)*:

```
//dichiarazione iteratore per la lista
ListIterator iter=list.listIterator();

//cancellazione elementi della pagina
while(iter.hasNext()){
    selection=(String)iter.next();
    fa.deleteElement(f,selection);
}
```

# 2 - Ambiente di sviluppo

In questo capitolo verranno descritti gli strumenti impiegati durante lo sviluppo dell'applicazione

## *Hardware*

E' stata impiegata una piattaforma basata su processore Intel x86, tale da soddisfare i requisiti previsti da NetBeans IDE e Apache Tomcat.

## *Software*

### Sistema operativo

Microsoft Windows XP Professional, ver. 2002.

### Java IDE

L'applicazione è stata realizzata utilizzando ***NetBeans IDE 3.5.1***, l'ambiente di sviluppo Java Open Source di Sun Microsystem:

*"NetBeans IDE è uno strumento destinato ai programmatori per scrivere, compilare ed eseguire il debug ed il deploy di programmi. E' scritto in Java ma può supportare qualsiasi linguaggio di programmazione. Esiste anche un gran numero di moduli utili per estendere le funzionalità di NetBeans IDE. NetBeans IDE è un prodotto gratuito senza alcuna restrizione riguardante il suo uso".*

### Container JSP

NetBeans include al suo interno ***Tomcat 4.0.6***, per questo motivo è stato scelto come *container* di riferimento.
Tomcat è probabilmente in questo momento il web container più popolare ed utilizzato su internet. Tomcat oltre ad essere la reference implementation per Servlet e JSP API, è distribuito tramite licenza open source e rappresenta anche un ottimo prodotto, sia per stabilità che per performance. Tomcat viene utilizzato nella maggior parte dei casi come container in grado di eseguire applicazioni web basate su servlet e JSP. Anche se al suo interno è presente un connettore HTTP che lo trasforma in un server HTTP a tutti gli effetti, il suo utilizzo più consolidato, specie in scenari di produzione, è quello che lo vede in abbinamento con un server HTTP vero e proprio. Dato che Apache HTTP Server è il server più utilizzato in internet, l'abbinamento di Tomcat con Apache è probabilmente il più diffuso.

# *Evoluzione del World Wide Web*

## La nascita del Web

Il Word Wide Web e il suo protocollo *HTTP (HyperText Transfer Protocol)* sono nati dal lavoro svolto da Tim Berners-Lee presso il CERN nel 1990. Tim Berners-Lee ha sviluppato il protocollo HTTP come un protocollo di rete per la distribuzione di documenti e ha realizzato il primo browser Web. Il sistema fu utilizzato nel 1991-1992 al CERN e in laboratori e università che si occupavano di fisica delle alte energie e in questo modo crebbe la sua popolarità. Nel 1993, l'avvento del browser Mosaic portò all'esplosione dell'utilizzo commerciale del Web. In cinque anni nel mondo nacquero più di 650000 server Web con milioni di utenti.

## Il modello di programmazione Web

L'idea di utilizzare il Web come un ambiente di programmazione si è sviluppata nel corso del tempo e ogni fase tecnologica succedutasi è servita anche per promuovere nuove idee. Il primo modello operativo prevedeva la presenza di un server Web che non faceva altro che servire documenti su richiesta. In quest'ambiente i contenuti non cambiavano mai, se non per un esplicito intervento umano, ovvero quando veniva creata una nuova versione di un documento. HTTP è un semplice protocollo che si basa su richieste e risposte, in cui un browser Web richiede un documento (normalmente utilizzando il comando GET) e il server Web restituisce il documento, sotto forma di dati HTML, preceduto da un'intestazione descrittiva.

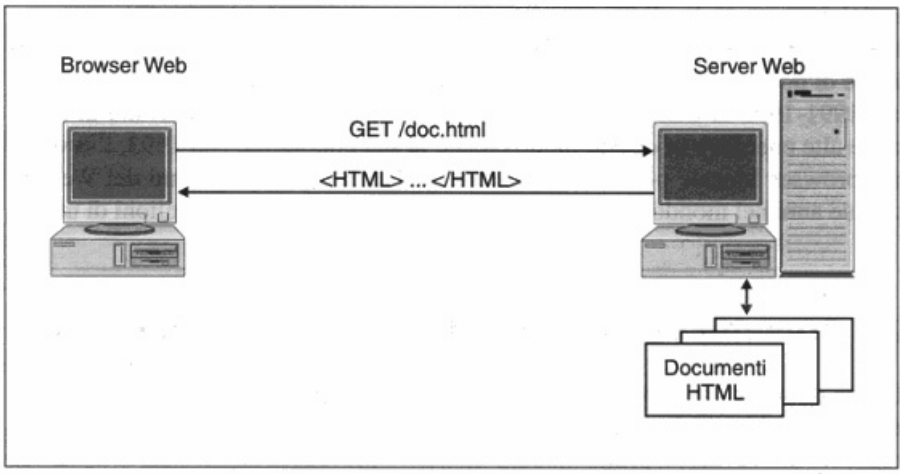


**Figura 23 - Formato a server statico**

Divenne presto chiaro che, se i documenti forniti dal server Web potevano essere scritti da un utente umano, potevano anche essere realizzati automaticamente da un programma, ad esempio da uno script Perl. Il browser Web non si sarebbe accorto della differenza poiché il risultato di una richiesta HTTP è pur sempre costituito da dati HTML. Inoltre in questo modo il browser può inviare qualcosa di più di una semplice richiesta: può inviare al server dei parametri incorporandoli nell'indirizzo URL o tramite la stringa di dati contenenti la richiesta. Questo suggerisce che una richiesta HTTP possa essere interpretata come una query a un database e che i risultati della query possano essere utilizzati per costruire in modo dinamico un documento HTML.

Con lo sviluppo del server Web NCSA (National Center for Supercomputing Applications) HTTPd nacque una nuova specifica chiamata *CGI (Common Gateway Interface).*

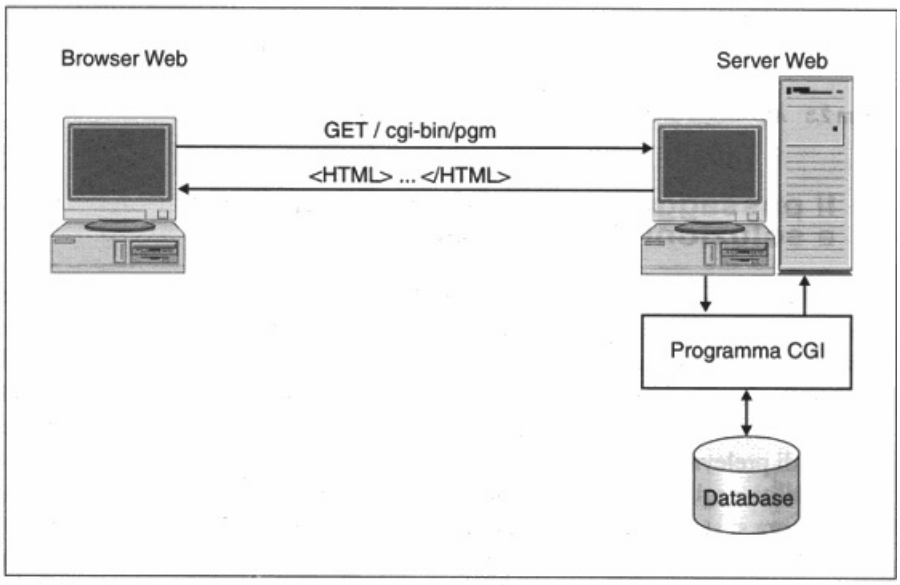


**Figura 24 - Uso di script CGI**

Un programma CGI viene richiamato dal server Web per rispondere a determinati tipi di richieste, normalmente richieste di documenti in una determinata directory o di file con l'estensione .cgi. I parametri della richiesta vengono passati come coppie chiave/valore e le intestazioni della richiesta come variabili d'ambiente. Il programma legge questi parametri e queste intestazioni, svolge le operazioni specificate dall'applicazione (normalmente un accesso a un database) e genera una risposta HTTP. La risposta viene inviata al browser Web che ha effettuato la richiesta come si trattasse di un normale documento statico HTML.
Il sistema CGI è abbastanza comodo ma presenta un grosso difetto: normalmente con il sistema CGI viene creato un nuovo processo per ogni richiesta HTTP (FastCGI però gestisce tutte le richieste come un unico processo). Questo non rappresenta un problema quando il traffico è relativamente ridotto ma crea un elevato sovraccarico a mano a mano che aumenta il livello del traffico. A un certo punto il metodo CGI diventa decisamente inappropriato.
Un significativo miglioramento venne dalla release 1997 dell'API Java Servlet seguita rapidamente dall' API JSP *(JavaServer Pages).* Queste tecnologie per server Web hanno combinato tutte le potenzialità di Java con la connettività a database, l'accesso alle reti, le operazioni multithread e, soprattutto, un nuovo modello di elaborazione. I servlet e le pagine JSP operano da un'unica istanza che rimane in memoria e utilizza più thread per rispondere simultaneamente a più richieste di servizi. I servlet e le pagine JSP possono inoltre utilizzare l'ambiente J2EE *(Java 2 Enterprise Edition)* che consente di realizzare applicazioni sofisticate e solide.

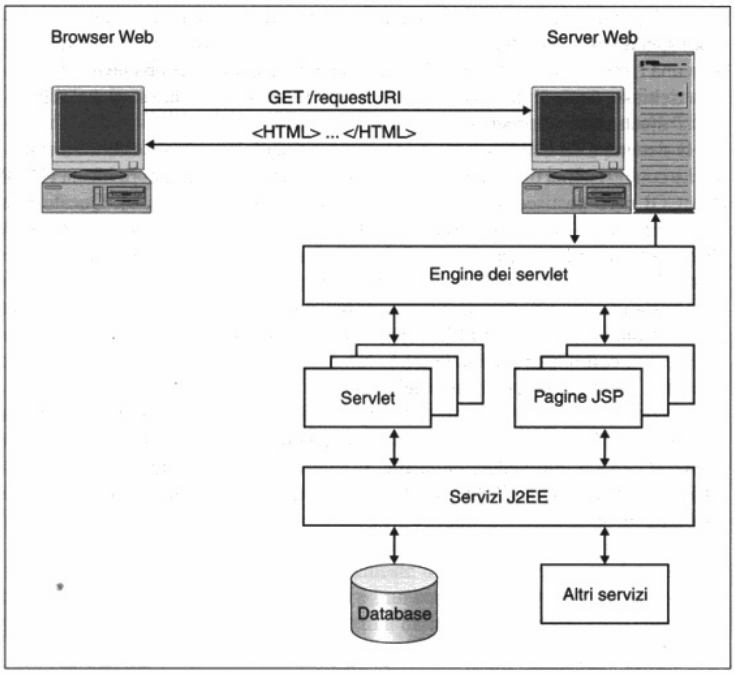


**Figura 25 - J2EE e JSP**

## Passaggio da Client-Side a Server-Side

Il modello delle applicazioni Web si è evoluto a mano a mano che maturava anche il Web e l'esperienza acquisita in ciascuna fase ha portato a nuovi requisiti per l'evoluzione alla fase successiva. L'approccio iniziale che usava moduli Java sul client sotto forma di applet si è molto diffuso ma in realtà ha anche portato ad alcune delusioni. Si sono verificate grandi incompatibilità fra i browser, lunghe sessioni di prelevamento sui modem più lenti e problemi di sicurezza che limitavano l'utilità degli applet. Per questo motivo lo sviluppo degli applet è stato rallentato e ora si sta ampliando sempre più lo sviluppo di codice Java sul server.
Molti osservatori ritengono che l'utilizzo di Java sul client stia per tornare in auge. Il plug-in Java elimina le differenze fra i browser e consente di impiegare componenti Swing. Inoltre l'utilizzo di connessioni Internet sempre più veloci rende le considerazioni sui tempi di prelevamento sempre meno importanti.
I moduli Java sul server non hanno però le restrizioni tipiche dell'ambiente applet. Non si verifica alcuna incompatibilità fra i browser poiché i browser non devono ospitare la macchina virtuale Java. Il browser deve solo interpretare il codice HTML, ma questo è in grado di farlo qualsiasi browser. Inoltre non è necessario eseguire alcuna impostazione sul client, né prelevare grandi file. Analogamente le considerazioni relative alla sicurezza sono sostanzialmente quelle già gestite dal server Web, che in genere è un ambiente chiuso dotato di vari controlli di sicurezza.
JSP si è dimostrata una tecnologia di successo per il server e una base eccellente per lo sviluppo di applicazioni Web.

## *JavaServer Pages (JSP)*

Una pagina JSP può assumere tre diverse forme:

- Il file di codice sorgente .jsp contenente le istruzioni HTML e gli elementi JSP.
- Il codice sorgente Java di un servlet.
- La classe Java compilata.

Innanzitutto lo sviluppatore JSP scrive un file di codice sorgente .jsp e lo memorizza in un documento situato nel file system di un server Web. In questo senso il file di codice sorgente .jsp non differisce da un normale file di codice HTML. L'indirizzo URL con il quale tale file è noto nella rete è lo stesso, tranne per l'estensione .jsp al posto di .html. La prima volta che viene richiamato l'indirizzo URL del file .jsp, il container JSP legge il file, ne analizzata il contenuto e genera un file di codice sorgente contenente il servlet Java equivalente. Poi compila il servlet e crea un file .class. Infine il container JSP carica la classe del servlet e la usa per esaudire la richiesta HTTP. Il passo intermedio (la generazione del codice sorgente del servlet) verrà ripetuta solo se viene modificato il file .jsp.
In questo schema, gli elementi JSP possono influenzare il modo in cui il container JSP opera durante due fasi operative:

- *Tempo di traduzione*: la generazione del codice sorgente del server Java a partire dal file .jsp.
- *Tempo di richiesta*: la chiamata del servlet per esaudire la richiesta HTTP.

## I componenti di una pagina JSP

Un file .jsp può contenere elementi JSP, dati fissi HTML o una combinazione di questi due elementi. Gli elementi JSP sono istruzioni per il container JSP che indicano quale codice generare e come tale codice deve comportarsi. Questi elementi hanno tag di apertura e di chiusura ben precisi che li identificano per il compilatore JSP. Rimane tutto ciò che non viene riconosciuto dal container JSP. In genere si tratta di codice HTML che viene passato senza modifica, in modo che il codice HTML generato contenga i dati esattamente come sono stati codificati nel file .jsp. Esistono tre tipi di elementi JSP:

- direttive;
- elementi script, fra cui espressioni, scriplet e dichiarazioni;
- azioni.

### *Le direttive*

Le *direttive* sono istruzioni per il container JSP che descrivono il codice che deve essere generato. Le direttive hanno la seguente forma generale:

```
<%@ nome-direttiva [attributo"valore" attributo"valore"...] %>
```

Fra il codice di apertura <%@ e il codice di chiusura %> possono trovarsi uno o più spazi. Inoltre dopo il nome della direttiva e fra le coppie attributo/valore possono trovarsi uno o più spazi vuoti. L'unica restrizione è che il tag di apertura <%@ si trovi nello stesso file fisico del tag %> di chiusura.

Le specifiche JSP 1.1 descrivono tre direttive standard disponibili in tutti gli ambienti JSP:

- page
- include
- taglib

### La direttiva page

La direttiva *page* viene utilizzata per specificare gli attributi della pagina JSP. Questa direttiva usa la seguente sintassi:

```
<%@ page [attributo-"valore" attributo-"valore" ...] %>
```

dove gli attributi sono quelli elencati nella seguente tabella:

| ATTRIBUTO | VALORE |
|---|---|
| language | Il linguaggio utilizzato negli scriplet, nelle espressioni e nelle dichiarazioni. In JSP 1.1, l'unico valore valido per questo attributo è *java*. |
| extends | Il nome completo della superclasse di questa pagina JSP. Deve trattarsi di una classe che implementa l'interfaccia HttpJspPage. Le specifiche JSP sconsigliano di utilizzare questo attributo si non si conoscono appieno le sue implicazioni |
| import | Un elenco separato da virgole di uno o più nomi package e/o nomi di classi completamente qualificate.<br>Questo elenco viene utilizzato per creare le corrispondenti istruzioni di importazione nel servlet Java generato. I seguenti package vengono automaticamente inclusi e pertanto non devono essere specificati:<br>`java.lang.*`<br>`java. servlet.*`<br>`java.servlet.jsp.*`<br>`java.servlet.http.*` |
| session | true o false per indicare se la pagina JSP richiede una sessione HTTP. Se il valore è true, allora il servlet generato conterrà il codice che provoca la creazione di una sessione HTTP (o l'accesso alla sessione se esiste già). Il valore predefinito è true. |
| `buffer` | Specifica le dimensioni del buffer di output. Possono essere utilizzate le opzioni nnnkb o none dove nnn è il numero dei KB allocati per il buffer. L'impostazione standard è di 8 KB. |
| autoflush | true se il buffer deve essere esportato automaticamente quando è pieno o false se deve essere lanciata un'eccezione per overflow del buffer. L'impostazione predefinita è true. |
| isThreadSafe | true se la pagina può gestire più richieste da più thread e false in caso contrario. Se è false, il servlet generato dichiara che implementa l'interfaccia SingleThreadModel. |
| info | Una stringa che verrà restituita dal metodo `getServletInfo()` della pagina. |
| isErrorPage | true se questa pagina può essere utilizzata come una pagina d'errore JSP. In questo caso la pagina può essere specificata come il valore dell'attributo errorPage nella direttiva page dell'altra pagina. Specificando true per questo attributo si rende disponibile per questa pagina la variabile implicita exception. L'impostazione predefinita è false. |
| errorPage | Specifica l'indirizzo URL di un'altra pagina JSP che verrà richiamata per gestire eventuali eccezioni non raccolte. L'altra pagina JSP deve specificare isErrorPage="true" nella propria direttiva page. |
| contentType | Specifica il tipo MIME e, opzionalmente, la codifica utilizzata per il servlet generato |

### La direttiva *include*

La direttiva *include* incorpora nello stream di input del codice sorgente il contenuto di un altro file prima della traduzione, comportandosi come la direttiva *#include* del preprocessore C. La sua sintassi è:

```
<%@ include file="nome-file" %>
```

dove *nome-file* è un percorso assoluto o relativo, interpretato secondo il contesto del servlet. Ecco alcuni esempi:

```
<%@ include file="/header.html" %>
<%@ include file="/doc/legal/disclaimer.html" %>
<%@ include file="sortmethod" %>
```

La direttiva *include* è differente dall'opzione *<jsp:include>*, che inserisce nello stream di output della risposta il contenuto di un altro file al momento della richiesta. Questi elementi possono comunque essere utilizzati per includere nelle pagine JSP del codice iniziale o finale standard oppure altri elementi di testo comuni delle pagine.

### La direttiva *taglib*

La direttiva *taglib* consente di eseguire azioni personalizzate nella pagina corrente tramite l'impiego di una libreria di tag. La sintassi della direttiva è:

```
<%@ taglib uri="URI-libreria-tag" prefix="prefisso-tag" %>
```

dove gli attributi possono essere quelli elencati di seguito.

| ATTRIBUTO | VALORE |
|---|---|
| URI-libreria-tag | L'indirizzo URL del descrittore della libreria di tag. |
| prefisso-tag | Prefisso univoco utilizzato per identificare i tag personalizzati che verranno utilizzati nella pagina. |

### Commenti

Le specifiche JSP forniscono due mezzi per includere commenti in una pagina JSP: uno per i commenti nascosti (visibili solo nella pagina JSP) e uno per i commenti inclusi nel codice HTML e XML generato dalla pagina. Il primo tipo usa la seguente sintassi:

```
<%- - Commento nascosto - -%>
```

mentre il secondo presenta il seguente aspetto:

```
<!--Commento incluso nel codice HTML generato - ->
```

Quando il compilatore JSP incontra il tag <%- - di apertura di un commento JSP, ignora tutto ciò che si trova da questo tag al corrispondente tag di chiusura - -%>. Questo significa che i commenti JSP possono essere utilizzati per disattivare delle sezioni della pagina JSP. Questa è una tecnica normalmente utilizzata per attivare o

disattivare temporaneamente delle parti di un programma senza apportare sostanziali modifiche al codice sorgente. Significa anche che i commenti JSP non possono essere nidificati poiché il tag di chiusura del commento più interno verrebbe interpretato come tag di chiusura del commento più esterno.
L'altro tipo di commenti utilizza il normale tag per commenti HTML o XML. I commenti di questo tipo vengono passati inalterati allo stream di output della risposta e vengono inclusi nel codice HTML generato. Essi risulteranno invisibili nella finestra nel browser ma potranno essere osservati richiamando l'opzione *Visualizza codice sorgente* del browser. Se lo scopo del commento è quello di chiarire un frammento di codice alla persona che lo visualizza, il secondo tipo sembra meno utile rispetto al primo per due motivi: viene inserito nel codice HTML generato da un programma e in generale non verrà mai visto da nessuno. Tuttavia, poiché questi commenti HTML sono generati dal computer, incorporano anche numeri di versione, date e altri codici di identificazione che possano essere utili per il personale di supporto tecnico per correggere le applicazioni.

### *Le espressioni*

JSP offre dei metodi semplici per accedere al valore di una variabile Java o di altre espressioni e per inserire tale valore nel codice HTML della pagina. La sintassi utilizzata è:

```
<%= espressione %>
```

dove *espressione è* una qualsiasi espressione Java. L'espressione può contenere qualsiasi valore che possa essere convertito in una stringa. Questa conversione viene normalmente eseguita semplicemente generando un'istruzione *out.print()*
Ad esempio, il codice JSP seguente:

```
Sono le ore <%= new java.util.Date() %>
```

può generare il codice servlet:

```
out.write("Sono le ore ");
out.print( new java.util.Date() );
```

### *Scriptlet*

Uno *scriptlet* è un insieme di una o più istruzioni Java utilizzate per elaborare una richiesta HTTP. Ecco la sintassi di uno script:

```
<% istruzione; [istruzione; ...] %>
```

Il compilatore JSP include semplicemente il contenuto dello scriptlet nel corpo del metodo *jspService().* Una pagina JSP può contenere qualsiasi numero di scriptlet. Se vi sono più scriptlet, essi verranno aggiunti al metodo *jspService()* nell'ordine in cui sono stati specificati. In questo modo uno scriptlet può contenere una parentesi graffa aperta che verrà chiusa in un altro scriptlet.

### *Dichiarazioni*

Come gli scriptlet, anche le *dichiarazioni* contengono istruzioni Java ma come una grande differenza: il codice dello scriptlet entra a far parte del metodo jspService() mentre il codice delle dichiarazioni viene incorporato nel file di codice sorgente generato ma all'esterno del metodo jspService(). La sintassi di una dichiarazione è la seguente:

```
<%! istruzione; [istruzione; ...] %>
```

La sezione delle dichiarazioni può essere utilizzata per dichiarare classi o variabili d'istanza, metodi o classi interne. A differenza degli scriptlet, le dichiarazioni non hanno accesso agli oggetti impliciti descritti nel prossimo paragrafo. Ad esempio, se si usa una sezione di dichiarazione per dichiarare un metodo che deve utilizzare l'oggetto richiesto, occorre passare l'oggetto come parametro del metodo.

### *Oggetti impliciti*

Mentre gli script, le espressioni e i dati HTML vengono incorporati nel metodo jspService, il container JSP scrive lo scheletro di un metodo, inizializzando il contesto della pagina e varie variabili. Queste variabili saranno implicitamente disponibili all'interno degli scriptlet e delle espressioni (ma non delle dichiarazioni). L'accesso può avvenire come per qualsiasi altra variabile ma tali variabili non devono essere dichiarate.

Ad esempio l'oggetto *HttpServletRequest* passato a jspService() è reso disponibile tramite il nome *request*, come illustrato dal seguente scriptlet:

```
<%
String accountNumber = request.getParameter("acct");
if (accountNumber = null){
       // ... gestione della mancanza del numero di conto
}
%>
```

Ecco l’elenco completo delle variabili implicite:

| NOME | VALORE |
|---|---|
| request | La richiesta ServletRequest o HttServletRequest servita. |
| response | La risposta ServletResponse o HttpServletResponse che riceverà l'output HTML generato. |
| pageContext | L'oggetto PageContext di questa pagina. Questo oggetto è un deposito centrale contenente i dati della pagina, della richiesta, della sessione e dell'applicazione. |
| session | Se la pagina JSP usa una HttpSession, questa sarà disponibile con il nome session. |
| application | L'oggetto che rappresenta il contesto del servlet. |
| Out | Lo stream di output utilizzato per generare il codice HTML. |
| Config | L'oggetto ServietConfig per il contesto di questo servlet. |
| Page | Un riferimento alla pagina JSP. |
| Exception | Un'eccezione non raccolta che provoca la chiamata di una pagina d'errore. Questa variabile è disponibile solo per le pagine con isErrorPage="true". |

### *Azioni standard*

Le *azioni* sono elementi JSP di alto livello che creano, modificano o usano altri oggetti. A differenza delle direttive e degli elementi degli script, le azioni vengono codificate utilizzando la normale sintassi XML:

```
<nome-tag [attr="valore" attr="valore" ...] > ... </nome-tag>
```

o, se l'azione non ha corpo, nella seguente forma abbreviata:

```
<nome-tag [attr=" valore" attr=" valore" />
```

La sintassi XML ha i seguenti requisiti:

- ogni tag deve avere un corrispondente tag di chiusura o utilizzare la forma abbreviata /> mostrata in precedenza;
- il valore degli attributi deve essere inserito fra doppi apici;
- i tag devono essere nidificati correttamente: si può usare la forma <A><B> ... </B></A> ma non <A><B> ... </A><B>.

Negli ambienti compatibili JSP 1.1 sono disponibili sette azioni standard. La seguente tabella presenta una sintassi abbreviata delle azioni:

| NOME DEL TAG | DESCRIZIONE |
|---|---|
| <jsp:useBean> | Dichiara un'istanza Java Bean e la associa al nome di una variabile. La sintassi è:<br><jsp:useBean<br>*id="name"*<br>*[ type="type" ]*<br>*[* class="class" ]<br>[ beanName=*"beanName" ]*<br>[scope="page \| request \| *session* \| *application"*]><br>...</jsp:useBean> |
| <jsp: setProperty> | Imposta il valore di una o più proprietà di un bean dichiarato precedentemente con <jsp : useBean>. La sintassi è:<br><jsp:setProperty<br>name="id"<br>*prop-expression/>*<br><br>dove *prop-expression* è una delle seguenti:<br>property="*"<br>property="propName"<br>property="propName" param="parameterName"<br>property="propName" value="valore"<br>property="propName" value=<%= *expression %>* |
| <jsp:getProperty> | Restituisce il valore della proprietà specificata di un bean. La sintassi e:<br><jsp:getPropertyname="id" property="name" /> |
| <jsp: include> | Richiama un'altra risorsa e la unisce allo stream di output costituito dalla pagina JSP. La sintassi è:<br><jsp:include page="*URL"* flush="true" /><br>o, se devono essere passati dei parametri:<br><jsp:include *page="URL"* flush="true"> |

| | |
|---|---|
| | <jsp:param ... /><br><jsp:param<br><jsp:param ... /><br></jsp:include> ... /> |
| <jsp: forward> | Inoltra questa richiesta HTTP a un'altra pagina JSP o a un servlet perché venga elaborata. La sintassi è:<br><jsp:forward *page="URL"* /><br>o, se devono essere passati dei parametri:<br><jsp:forward *page="URL"*><br><jsp:param ... /><br><jsp:param ... /><br><jsp:param ... /><br></jsp:forward> |
| <jsp: param> | Associa un valore a un nome e passa questa associazione a un'altra risorsa richiamata con <jsp:include> o <jsp: forward>. La sintassi è:<br><jsp:param *name="name"* value="valore" /> |
| <jsp: plugin> | Utilizzato per generare il collegamento HTML appropriato per il prelevamento del plugin Java:<br><jsp:plugin<br>*type="bean applet"*<br>*code="objectCode"*<br>*codebase="objectCodebase"*<br>{ align="alignment" }<br>{ archive="archiveList" }<br>{ height="height" }<br>{ hspace="hspace" }<br>{ jreversion="jreversion" }<br>{ *name="componentName"* }<br>{ vspace="vspace" }<br>{ width="width" }<br>{ nspluginurl="url" }<br>{ iepluginurl="url" } ><br>{ <jsp:params><br>{ <jsp:param name="name" value="valore" /> } + </jsp:params><br>}}</jsp:plugin> |

## Inclusione di risorse

HTML non offre metodi diretti per includere nell'output i dati provenienti da altri file. Questo è uno svantaggio poiché in uno stesso sito Web, normalmente in tutte le pagine è presente più o meno lo stesso codice HTML per il logo aziendale, le note di copyright, i collegamenti di navigazione e altre funzionalità. Oltre a queste fonti statiche di testi e immagini, può essere necessario includere anche dei contenuti dinamici. JSP offre due mezzi per incorporare questi dati:

- La direttiva **<%@ include %>**: utilizzata per copiare nel codice sorgente JSP del testo statico prima che venga trasformato in codice sorgente per servlet Java e quindi compilato. In genere questo testo è codice HTML ma può essere qualsiasi cosa che può comparire in una pagina JSP.

- L'azione **<jsp:include>**: fa in modo che la servlet engine richiami un altro indirizzo URL, che unisce il suo output con quello della pagina JSP originaria.

Un punto fondamentale da ricordare è che la direttiva *<%@ include %>* viene eseguita una sola volta, al momento della compilazione, mentre l'azione *<jsp:include>* viene eseguita a ogni richiesta. I prossimi due paragrafi descrivono questi due componenti JSP e il loro funzionamento.

### *La direttiva include*

Quando viene incontrata la direttiva **<%@ include %>**, il container JSP legge il file specificato e aggiunge il suo contenuto al codice sorgente JSP che sta elaborando. Ecco la sintassi:

```
<%@ include file="nome-file" %>
```

Il *nome-file* specificato deve essere un indirizzo URL relativo, ovvero deve contenere solo le informazioni relative al percorso, senza il protocollo e le informazioni sul server. Questa tecnica consente di includere solo le risorse che si trovano nel contesto del servlet.
Se *nome-file* inizia con "/", viene considerato un percorso assoluto rispetto alla cima del contesto del servlet. Altrimenti il *nome file* viene considerato relativo alla pagina JSP corrente. Ad esempio, se un'applicazione Web ha una subdirectory *products* e una pagina *products/search.jsp* contiene la direttiva:

```
<%@ include file="/includes/header.inc" %>
```

allora verrà incluso il file *<percorso>/includes/header.inc* dove *<percorso>* è la directory in cui si trova l'applicazione Web. Se invece la direttiva è:

```
<%@ include file="includes/header.inc" %>
```

allora verrà incluso il file <percorso>/products/includes/header.inc.

### Uso della direttiva include per copiare il codice sorgente

Oltre a consentire di copiare il codice HTML, la direttiva *include* può essere utilizzata anche per includere del codice sorgente Java come una sezione di dichiarazione. Ad esempio si può incorporare, con la direttiva include, una funzione di servizio molto utilizzata che può essere memorizzata in un file.

### *L'azione <jsp:include>*

A differenza della direttiva *include*, l'azione *jsp:include* viene interpretata ogni volta che viene eseguita una richiesta. La sintassi di questa azione è la seguente:

```
<jsp:include page=”nome-risorsa” flush=”true” />
```

dove *nome-risorsa* deve essere un indirizzo URL relativo e deve contenere solo il percorso. Il nome della risorsa viene associato al contesto del servlet così come

accade per il file di una direttiva *include*. Se il nome inizia con "/", fa riferimento ad un percorso che inizia dal contesto del servlet; altrimenti viene interpretato come un percorso relativo che parte dalla directory contenente il codice JSP in cui si trova l'azione. L'attributo *flush* (che è obbligatorio) indica che si deve vuotare l'output *JspWriter* prma di includere la risorsa. L'unico valore utilizzabile in JSP 1.1 è true.

### Il funzionamento dell'azione include

L'azione *<jsp:include>* viene analizzata sintatticamente dal compilatore JSP ma, invece di essere eseguita al momento della compilazione, viene convertita in codice Java che richiama la risorsa nel momento della richiesta. La risorsa può essere costituita da dati statici, ad esempio un file HTML, oppure da dati dinamici, ad esempio una pagina JSP o un servlet.

## JSP & HTML – Interazione con l'utente

La maggior parte delle applicazioni richiede qualche forma di input dell'utente e, nell'ambiente Web, per raccogliere questo input vengono normalmente utilizzati dei moduli HTML. I moduli HTML non differiscono molto dai normali moduli stampati, in quanto sono costituiti da descrizioni e campi disposti secondo una sequenza logica. Quando un utente compila un modulo e fa clic sul pulsante di invio, il nome dei campi e i valori introdotti nei campi vengono trasmessi a un programma associato al server Web che si occuperà della loro elaborazione. Il linguaggio HTML offre vari elementi e controlli di input che rispondono a varie esigenze:

- **Elementi per l'introduzione del testo:** si tratta di caselle rettangolari per l'introduzione di una o più righe di testo.
- **Menu di selezione:** elenchi di opzioni visualizzate a menu. Possono provocare la visualizzazione di un modulo esterno e a ogni elemento è associato un codice.
- **Pulsanti:** caselle rettangolari che simulano il funzionamento di un normale pulsante. In genere vengono utilizzati per avviare un comando, ad esempio per inviare il modulo o per cancellare i campi di input.
- **Caselle di controllo:** piccole caselle quadrate che possono essere selezionate o non selezionate. Vengono utilizzate per specificare le opzioni che possono assumere i valori sì o no.
- **Pulsanti di opzioni:** sono simili alle caselle di controllo e anch'esse indicano valori si/no. La differenza consiste nel fatto che i pulsanti di opzioni normalmente formano gruppi mutuamente esclusivi, ovvero gruppi di pulsanti dei quali uno solo può essere selezionato
- **Elementi per la selezione di file:** controlli che consentono di specificare il nome di un file. In genere questo controllo include un pulsante Sfoglia che consente la visualizzazione di una finestra di dialogo.
- **Elementi nascosti:** elementi non visibili utilizzati per creare parametri con valori costanti.

L'insieme di elementi che possono essere utilizzati in un modulo HTML è standardizzato e documentato in modo formale dalle specifiche HTML (vedere l'indirizzo *http://www.w3.org/TR/html4*. Queste specifiche sono state prodotte dal consorzio *W3C (World Wide Web Consortium)* che le aggiorna periodicamente a mano a mano che emergono nuove funzionalità. Non tutti i browser implementano tutte queste funzionalità.

### *L'elemento FORM*

L'elemento FORM è alla base dei moduli HTML e ha principalmente tre scopi:

- raggruppare sintatticamente gli elementi di input
- identificare il programma sul server che si occuperà di gestire i dati inviati con il modulo
- specificare i valori inviati e la modalità dell'invio.

Un modulo è descritto in HTML dal tag <FORM> che ha la seguente sintassi:

```
<FORM
action="uri"
method="metodo"
enctype="tipo-contenuti"
accept-charset="set-di-caratteri" accept="tipi-di-contenuti"
name="nome-modulo">
</FORM>
```

### *Gli attributi dell'elemento FORM*

Dei sei attributi elencati, l'unico necessario è action. In pratica gli attributi diversi da *action* e *method* vengono utilizzati raramente.

### L'attributo *action*

Quando un modulo è completo e l'utente fa clic sul pulsante di invio, il browser Web crea una richiesta HTTP contenente tutti i dati del modulo e la invia a un programma in funzione su un server Web; questo programma è specificato nell'attributo *action*.
Il valore dell'attributo action deve essere un identificatore *URI (Uniform Resource Identifier)* HTTP.
In pratica l' identificatore URI avrà la seguente forma:

**[http://<nome-server>][/]<percorso>**

Per inviare il modulo, il browser Web apre una connessione verso il server specificato (che normalmente è il server Web da cui è stata prelevata la pagina HTML) e crea una richiesta HTTP utilizzando il percorso specificato. Normalmente il percorso punta a un servlet, a una pagina JSP o a un programma CGI. Questo programma riceve la richiesta HTTP e i dati del modulo, contenuti nell'identificatore URI o in uno stream di input, a seconda del metodo HTTP utilizzato (vedere la descrizione dell'attributo method).
È anche possibile specificare una stringa di query nell'indirizzo URI. In questo caso i parametri codificati nella stringhe della query vengono uniti a quelli specificati nel

corpo del modulo. Normalmente questo non è necessario perché lo stesso risultato si può ottenere con un campo nascosto.

### L'attributo *method*

Il protocollo HTTP fornisce vari tipi di richieste per il trasferimento, il prelevamento e la cancellazione di file e per operazioni diagnostiche. Fra queste richieste, quelle utilizzabili per i moduli HTTP sono *GET* e *POST*. Per indicare quale richiesta utilizzare viene impiegato l'attributo method.
Una richiesta GET o POST viene normalmente interpretata dal server Web come la richiesta di prelevare il documento specificato nell'identificatore URI. Se il server Web è stato configurato per gestire servlet, programmi CGI o altre forme di script, interpreta una richiesta di queste risorse come la richiesta di eseguire il relativo programma. L'output prodotto da questo programma (normalmente un documento HTML) viene poi rinviato al richiedente, esattamente come se fosse stato richiesto un documento statico.

La differenza fra i metodi GET e POST utilizzati in un modulo HTML è il modo in cui viene fornito l' input al processo sul server:

- GET: i valori del modulo vengono aggiunti all'identificatore URI sotto forma di stringa;
- POST: i valori del modulo vengono forniti nello stream di input.

Anche se è possibile utilizzare entrambi i metodi (e l'API servlet rende la scelta abbastanza trasparente) occorre tenere in considerazione varie caratteristiche. Poiché le richieste GET provocano l'aggiunta dei valori introdotti all'identificatore URI della richiesta, questi risulteranno visibili come coppie *nome/valore* nella riga degli indirizzi del browser e nei file di registrazione del server Web. Questo rende il metodo GET inadatto per inviare dati riservati come ad esempio la password. Inoltre alcuni server e browser possono prevedere restrizioni sulla lunghezza dell'indirizzo URL che può essere inviato. Inoltre, secondo le specifiche http, le richieste GET possono essere sostituite senza effetti collaterali. In alcune circostanze questo significa che un server può dire a un client di utilizzare una copia esistente di una risorsa invece di inviare una nuova copia. Questo normalmente non è ciò che si attende come risposta per l'input rappresentato dal modulo. Per questi motivi in genere si preferisce utilizzare il metodo POST.
L'attributo method è opzionale. Se non viene specificato viene impiegato il metodo GET. Il valore dell'attributo può essere specificato in lettere maiuscole o minuscole.

### L'attributo *enctype*

I valori contenuti nel modulo di input possono essere trasmessi al server in vari modi. Il metodo di codifica dei valori in una stringa di dati è specificato tramite l'attributo *enctype* del metodo POST. Normalmente vengono utilizzati due tipi di codifica:

- application/x-www-form-urlencoded
- multipart/form-data

*multipart/form-data*: è un nuovo approccio utilizzato principalmente per supportare l'invio di file. Secondo questa codifica, ogni campo di input e il relativo valore vengono inviati in un proprio blocco nello stream di input. Vi è una particolare stringa di delimitazione, chiamata boundary, che contrassegna l'inizio e la fine di ciascun blocco: una stringa pseudocasuale scelta dal browser Web e specificata nell'intestazione Content-Type. All'interno di ciascun blocco si trovano una o più intestazioni HTTP seguite da una riga vuota e poi da una riga contenente il valore del campo di input. Il nome del campo viene passato nell'intestazione Content-Disposition.
Lo svantaggio principale della codifica multipart/form-data è il fatto che non è direttamente supportata dall'API servlet corrente. Pertanto non si può usare getParameterNames() per leggere il nome dei campi e getParameterValues() per leggere il loro valore. Per ottenere queste informazioni è necessario leggere e analizzare lo stream di input.

### L'attributo *accept-charset*

In HTTP un set di caratteri è un insieme di regole di conversione di un insieme di byte in un insieme di caratteri. La codifica più utilizzata è ISO-8859-1, un'estensione ASCII che utilizza anche i byte compresi nell'intervallo 127-255. Se viene usato l'attributo accept-charset, questo dovrà contenere un elenco di set di caratteri separati da una virgola o da uno spazio. Lo scopo dell'attributo accept-charset è di indicare il set di caratteri che il programma del server può interpretare ed elaborare. In pratica questo attributo viene utilizzato raramente e sembra essere ignorato da Internet Explorer e Netscape Navigator.

### L'attributo *accept*

Un tag FORM può indicare i tipi di contenuti che devono essere accettati dal programma di gestione situato sul server. Se viene specificato, l'attributo accept deve contenere un elenco di tipi di contenuti separati da una virgola, come *text/html* o *image/jpg.* Questo è solo un suggerimento per il browser Web il quale è però libero di ignorarlo (come avviene normalmente).

### L'attributo *name*

Il modulo può avere un nome cui si fa riferimento nelle sezioni <SCRIPT> in altri punti del documento. Le specifiche HTML sconsigliano di utilizzare l'attributo name, preferendo l'attributo *id* che però non è ancora riconosciuto dal modello a oggetti JavaScript.

### *Gli elementi di input di un modulo*

Nel corpo di un tag <FORM>...</FORM> vengono descritti i vari campi di input. Il codice HTML è costituito da etichette descrittive affiancate da tag HTML che creano i controlli richiesti. In genere i controlli sono inseriti in una tabella HTML per motivi di allineamento.

### *Gli elementi creati con il tag INPUT*

Il tag HTML INPUT può essere utilizzato per vari tipi di elementi. Offre moltissimi attributi, molti dei quali sono specifici di determinati tipi di campi. La seguente sintassi descrive gli attributi comuni alla maggior parte dei tipi.

```
<INPUT
type="text | password | checkbox | radio | submit | reset | file | hidden | image | button"
name="nome"
value="valore"
size="dimensioni">
```

gli attributi hanno il seguente significato:

- **type="tipo":** indica il tipo del campo. Se non viene specificato si presume l'impiego del campo text.
- **name="nome":** utilizzato per assegnare un identificatore al campo in modo che possa essere manipolato da script o fogli di stile. Questo è anche il nome con cui il campo può essere prelevato dal programma operante sul server.
- **value="valore":** può essere utilizzato per assegnare un valore iniziale al campo.
- **size="dimensioni':** indica le dimensioni del campo in pixel o in caratteri (per i campi di testo).

Oltre a questi attributi, il tag INPUT può specificare dei gestori che richiamano apposite azioni nel browser quando si verificano determinati eventi. Il valore dell'attributo di gestione degli eventi è un frammento di codice script, in genere JavaScript. Gli attributi del gestore di eventi hanno un nome che corrisponde all'evento che devono gestire, con il prefisso "*on*":

- onfocus: si verifica quando un utente accede al campo tramite la tastiera o il mouse, ovvero quando la tastiera ha il controllo del campo.
- onblur: si verifica quando un utente lascia il campo con la tastiera o con il mouse, ovvero quando la tastiera non ha più il controllo del campo.
- onselect: si verifica quando vi è del testo selezionato (non è supportato da Netscape Navigator).
- onchange: si verifica quando un utente cambia il valore del controllo e poi accetta la modifica lasciando il campo.

Quando viene inviato un modulo, il browser estrae il nome e il valore di ciascun controllo, converte i dati secondo il tipo di codifica specificata nel tag <FORM> (o implicita) e poi invia il risultato al processo del server.

Di seguito vengono presentati i vari tipi di tag <INPUT>.

### Il controllo *text*

Questa è la forma più semplice e comune di tag <INPUT>, utilizzata per introdurre un'unica riga di testo. La sua sintassi è la seguente:

```
<INPUT
type="text" name="nome" value="valore" size="dimensioni"
maxlength="lunghezza-massima">
```

dove gli attributi hanno il seguente significato:

- **type="text":** indica che questo è un controllo di testo.
- **name="nome":** specifica il nome con cui lo script fa riferimento a questo controllo. Questo è anche il nome con cui il testo può essere letto dal programma sul server.
- **value="valore":** può essere utilizzato per assegnare un valore iniziale al testo. Questo attributo è utile quando un modulo compare nella stessa pagina dell'output del programma che elabora il modulo.
- **size="dimensioni':** la larghezza del campo. Se non viene specificato, il browser sceglie una larghezza standard che potrebbe essere inadatta per il campo. In genere è preferibile specificare esplicitamente le dimensioni.
- **maxl*ength="lunghezza-massima":*** imposta un limite per il numero di caratteri che possono essere digitati nel campo.
-

Normalmente un campo di testo viene visualizzato sotto forma di una casella rettangolare:


**Figura 26 - Controllo casella di testo**

### Il controllo *Password*

È una variante del controllo *text*. L'unica differenza consiste nel fatto che i caratteri introdotti dall'utente non verranno visualizzati ma impiegheranno un carattere di mascheratura, in genere l'asterisco (*).
La sintassi del campo di input password ha il seguente aspetto:

```
<INPUT
type="password"
name="nome"
value="valore"
size="dimensioni"
maxlength="lunghezza-massima">
```

dove gli attributi hanno lo stesso significato descritto in precedenza per il controllo *text*.
Anche un campo per password viene visualizzato sotto forma di casella rettangolare:


**Figura 27 - Controllo password**

## Il controllo *Checkbox*

Una casella di controllo consente di presentare un'opzione che può essere vera o falsa. La sua sintassi è la seguente:

**<INPUT**
**type="checkbox" name=" *nome"***
**value=" valore" checked>**

gli attributi sono definiti nel seguente modo.

- *type="checkbox":* indica che si tratta di una casella di controllo.
- name="nome": specifica il nome con cui gli script possono far riferimento a questa casella di controllo. Questo è anche il nome con cui il programma sul server legge il valore della casella di controllo. Un gruppo di caselle di controllo può condividere lo stesso nome se esse rappresentano valori multipli dello stesso campo che non sono mutuamente esclusivi.
- *value="valore":* può essere utilizzato per specificare il valore restituito quando questa casella è selezionata. Se non viene specificato, il valore è rappresentato dalla stringa di due caratteri "on".
- *checked*: se presente, indica che la casella di controllo inizialmente è selezionata.

La casella di controllo supporta anche un altro attributo.

- *onclick:* si verifica quando l'utente fa clic sulla casella di controllo checkbox. Al contrario il controllo non supporta l'evento onchange.

Normalmente una casella di controllo viene visualizzata come una piccola casella quadrata e lo stato di selezione è rappresentato da un segno di spunta:

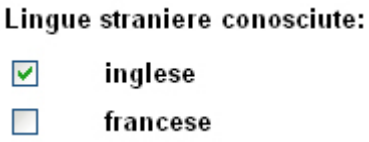


**Figura 28 - Controllo checkbox**

## Il controllo *radio*

Un pulsante di selezione, come una casella di controllo, consente di presentare un'opzione che può essere vera o falsa. La differenza consiste nel fatto che il funzionamento dei pulsanti di selezione è mutuamente esclusivo. Quando si seleziona uno dei pulsanti di un gruppo, tutti gli altri pulsanti del gruppo vengono deselezionati. Per questo si chiamano radio, perché si comportano come i pulsanti di selezione di un apparecchio radio: quando si preme il pulsante di una stazione, tutti gli altri vengono deselezionati. La sintassi del controllo radio è:

```
<INPUT
type="radio" name="nome"
value="valore" checked>
```

dove gli attributi sono definiti nel seguente modo:

- *type="radio":* indica che questo è un pulsante di selezione.
- *name="nome":* specifica il nome con cui gli script possono far riferimento a questo pulsante di selezione. Questo è anche il nome con cui il programma sul server legge il valore del pulsante di selezione. Un gruppo di pulsanti di selezione può condividere lo stesso nome se i pulsanti rappresentano valori mutuamente esclusivi dello stesso campo.
- *value="valore":* specifica il valore restituito dal modulo quando il pulsante è selezionato. Questo è un attributo obbligatorio.
- *checked:* se presente indica che il pulsante è inizialmente selezionato.

Così come la casella di controllo, il pulsante di selezione supporta l'evento *onclick* ma non l'evento *onchange.*
Un pulsante di selezione è sempre rappresentato come un pulsante circolare; quello selezionato contiene al suo interno un piccolo tondo nero:


**Figura 29 - Controllo radio**

### Il controllo *submit*

Per inviare un modulo al server, vi deve essere un modo per indicare che l'utente ha terminato di introdurre i dati. A tale scopo si utilizza il pulsante di input submit. Il pulsante submit è differente dagli altri controlli in quanto normalmente non contribuisce a introdurre informazioni. Ecco la sintassi del pulsante submit:

```
<INPUT
type="submit" name=" nome" value=" valore">
```

dove gli attributi sono definiti nel seguente modo:

- *type="submit":* indica che questo è un controllo submit.
- *name="nome":* specifica il nome con cui gli script o il programma del servlet possono far riferimento a questo pulsante. Normalmente il server non ne ha bisogno poiché è chiaro che l'utente ha fatto clic sul pulsante submit o il modulo non sarebbe stato inviato. Può essere invece utile quando nel modulo sono presenti più pulsanti submit ognuno dei quali ha un significato (e un valore) differente.
- *value="valore":* specifica il valore visualizzato nel pulsante (e restituito insieme al modulo se è presente anche l'attributo name). Se non viene specificato, il valore standard in Internet Explorer 5.0 e Netscape 4.75 è "Submit Query". Altri browser possono produrre valori differenti.

Questo controllo supporta l'evento *onclick* ma non l'evento *onchange*.
Un pulsante submit viene normalmente visualizzato come un pulsante rettangolare contenente il testo specificato dall'attributo value:


**Figura 30 - Controllo submit**

### Il controllo *reset*

Strettamente correlato con il pulsante submit, il pulsante reset consente di riportare tutti i controlli al valore iniziale. Anche reset non contribuisce allo stream di dati inviati al server.
La sua sintassi è:

```
<input type="reset" value="Cancella">
```

dove gli attributi sono definiti nel seguente modo:

- *type="reset":* indica che questo è un controllo reset.
- *value="valore":* specifica il valore visualizzato nel pulsante (e restituito con il modulo se è presente anche l'attributo name). Se non viene specificato, il valore standard è "Reset".

Questo controllo supporta l'evento *onclick* ma non l'evento *onchange.*
Un pulsante reset viene normalmente visualizzato come un pulsante rettangolare contenente il testo specificato nell'attributo value:


**Figura 31 - Controllo reset**

### Il controllo *file*

Alcune applicazioni prevedono l'invio di file al server. Ad esempio le applicazioni di supporto tecnico possono gestire un elenco di risorse inviate dagli utenti; anche i sistemi BBS possono accettare l'invio dei file. Le pagine Web che rappresentano l'interfaccia di applicazioni di questo tipo possono utilizzare il controllo di input file. Segue la sua sintassi:

```
<INPUT
type="file" name="nome" size="dimensioni">
```

gli attributi sono definiti nel seguente modo:

- *type="file":* indica che si tratta di un controllo file.
- *name="nome"* specifica il nome con il quale gli script possono far riferimento a questo controllo.
- *size="dimensioni':* indica la larghezza del campo di input del nome del file.

Normalmente un controllo file produce un campo di testo cui è associato il pulsante Sfoglia. Il nome del file può essere introdotto direttamente nel campo di testo; in alternativa l'utente può fare clic sul pulsante Sfoglia per aprire una finestra di dialogo per la selezione del file:


**Figura 32 - Controllo file**

Per utilizzare il controllo file, un modulo deve avere le seguenti caratteristiche:

- Il metodo della richiesta deve essere POST.
- Il tipo di codifica (specificato dall'attributo enctype del tag <FORM>) deve essere *multipart/form-data.*

Se queste condizioni non sono verificate, il controllo verrà comunque visualizzato ma verrà trattato come un normale campo di testo e al server verrà inviato unicamente il nome del file.

### Il controllo *hidden*

Non tutti i campi di input sono visualizzati all'utente, almeno direttamente. Alcuni moduli possono utilizzare delle costanti specificate all'interno del codice o generate dinamicamente. Un elemento di tipo hidden ha proprio questo scopo. La sua sintassi è la seguente:

```
<INPUT
type="hidden" name=" nome value="valore">
```

Un campo nascosto usa i seguenti attributi.

- *type="hidden":* indica che si tratta di un controllo hidden.
- *name="nome":* specifica il nome con cui gli script possono far riferimento a questo controllo. Questo è anche il nome con cui il testo può essere letto dal programma sul server.
- *value="valore":* deve essere utilizzato per assegnare un valore iniziale al controllo.

Un campo nascosto, non ha alcuna rappresentazione visuale.

### Il controllo *image*

Un campo image può essere utilizzato come campo di input in cui l'utente fa clic con il mouse invece di digitare alla tastiera. In questo caso le informazioni inviate sono rappresentate dal punto dell'immagine in cui è avvenuto il clic. La sintassi del controllo image è:

```
<INPUT
type="image" name="nome" src="url-immagine">
```

Ecco gli attributi di questo campo di input.

- *type="image":* indica che questo è un controllo image.
- *name="nome":* specifica il nome con cui lo script può far riferimento a questo controllo. Questo è anche il nome con cui il programma in esecuzione sul server può determinare il punto in cui l'utente ha fatto clic con il mouse.

- *src="url-immagine":* l'indirizzo URL dell'immagine

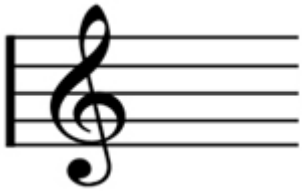

**Figura 33 - Controllo image**

Facendo clic sul controllo image si provoca l'invio del modulo. Quindi non è necessario che l'utente faccia clic sul pulsante di invio.

### Il controllo *button*

Oltre ai pulsanti di Submit e Reset vi è anche un pulsante generico button. La sua sintassi è la seguente:

```
<INPUT
type="button" name="nome" value="valore">
```

dove gli attributi sono definiti nel seguente modo:

- *type="button":* indica che questo è un controllo button.
- *name="nome":* specifica il nome con cui gli script possono far riferimento a questo pulsante.
- *value="valore":* specifica il valore visualizzato in questo pulsante.

Perché questo controllo sia utile, deve definire l'attributo di gestione dell'evento onclick. Una funzione JavaScript potrà quindi far riferimento al nome e al valore del pulsante.

### Controlli creati con *select* e *option*

I tag select e option consentono di creare un elenco di elementi selezionabili in un menu.

Ecco la sintassi del tag select:

```
<select name="nome" size="numero"multiple> opzioni </select>
```

dove gli attributi hanno il seguente significato:

- *name="nome":* assegna al controllo un nome con cui il programma in esecuzione sul server può far riferimento all'elenco.
- *size="numero":* indica il numero di elementi visibili contemporaneamente, ovvero l'altezza del menu. Se il numero è 1, la lista si trasforma in un menu a discesa.
- *multiple:* consente all'utente di selezionare più elementi.

Si noti che il tag <select> ha un corrispondente tag di chiusura </select>.

Il cuore dell'elenco select è rappresentato dai tag option cui è associato un valore e una descrizione. Spesso questi limiti vengono generati dinamicamente da una query a un database. Il tag option ha la seguente sintassi:

```
<option value="valore" selected> testo </option>
```

dove gli attributi che hanno il seguente significato:

- *value="valore":* specifica il valore restituito dal modulo quando è selezionato l'elemento corrispondente. Se questo attributo non viene specificato, viene restituito il corpo del tag option.
- *selected:* se è presente, preseleziona l'elemento.

Il testo fra i tag di apertura e di chiusura (ovvero il corpo del tag) è ciò che viene visualizzato nel menu. Il tag di chiusura </option> non è necessario e in genere viene omesso.
Quando il modulo viene inviato, all'elemento select viene associato il valore dell'elemento selezionato nel menu.

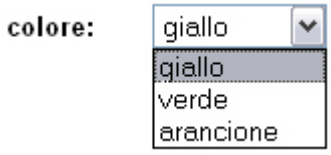


**Figura 34 - Controllo select**

## Il controllo *textarea*

Mentre i campi di testo text e password sono costituiti da un'unica riga, l'elemento textarea può accettare più righe. Questo rende l'elemento textarea utile per introdurre commenti o altro testo libero per il quale normalmente non basta una sola riga. Qui è possibile specificare sia l'altezza che la larghezza e se necessario il browser potrà aggiungere le barre di scorrimento. L'elemento textarea ha la seguente sintassi:

```
<textarea name="nome" rows="numero" cols="numero"> ... testo ...
</textarea>
```

dove gli attributi hanno il seguente significato:

- *name="nome":* assegna il nome con cui questo campo sarà noto agli script e al programma sul server.
- *rows="numero":* specifica il numero di righe dell'area di testo. Questo attributo non limita il numero di righe che possono essere contenute nella casella di testo ma solo il numero di righe visualizzate nella finestra del browser.
- *cols="numero":* specifica la larghezza in caratteri dell'area di testo. Questo non limita il numero delle colonne che possono essere introdotte ma specifica solo la larghezza di visualizzazione.


**Figura 35 - Controllo textarea**

### *Convalida del modulo*

La programmazione dell'interfaccia utente è un argomento complesso. La quantità di codice dedicata alla convalida dei campi di input spesso è maggiore rispetto a quella che svolge le funzioni vere e proprie. Si devono controllare i campi obbligatori per assicurarsi che non siano vuoti e occorre assegnare un valore standard ai campi opzionali. Tutti i campi devono essere convalidati utilizzando un insieme di valori accettabili o degli algoritmi. La validità di alcuni campi può dipendere dal valore specificato in altri campi.
Queste operazioni di convalida che possono essere eseguite dal programma in esecuzione sul server ma il traffico di rete può estendere eccessivamente i tempi di risposta. Per questo motivo è meglio eseguire la convalida sul client, utilizzando un linguaggio per scriptlet come JavaScript. A seguito, si descrivono le operazioni di convalida:

**1. Impostare il trigger**
Innanzitutto occorre prevedere l'esecuzione di un determinato frammento di codice nel momento in cui viene inoltrato il modulo. A tale scopo si imposta l'attributo *onsubmit* nel tag form:

```
<form method="post" action="diag/ShowParms.jsp" onsubmit="return validate(this);">
```

La stringa specificata nell'attributo onsubmit viene valutata prima che il modulo venga inviato. Il modulo viene inviato solo se il valore è true. Il codice di convalida può essere introdotto direttamente nel valore dell'attributo onsubmit ma è più semplice richiamare una funzione e restituire il suo valore. Questo semplifica anche l'aggiunta di nuovo codice quando cambieranno i requisiti di convalida.

**2. Aggiungere un blocco script**
Per incorporare delle istruzioni JavaScript nel codice HTML, basta utilizzare i tag <SCRIPT> ... </SCRIPT>. Per garantire che questi tag vengano caricati e valutati, è opportuno inserirli nella sezione <HEAD> ... </HEAD> del codice HTML.

**3. Scrivere le funzioni di convalida**
Il terzo passo consiste nella realizzazione delle funzioni di convalida.

## JSP & JDBC – Accesso a database

Java usa una serie di classi e interfacce chiamate Java DataBase Connettivity (JDBC) per comunicare con Database. A seguito, si analizzeranno gli aspetti fondamentali dell'uso di JDBC.
Per poter iteragire con un database occorre effettuare una serie di passi generali:

- Creare una istanza del driver JDBC
- Creare una connessione al DB attraverso il driver caricato
- Creare uno o più Statement dall'oggetto Connection
- Eseguire lo Statement per ottenere uno o più ResultSet
- Iteragire con le righe del ResultSet ed estrarre le informazioni volute
- Rilasciare il ResultSet
- Rilasciare lo Statement

- Disconnettersi dal Database

### *Caricamento dei driver*

Per preparare il driver occorre caricare la sua classe ed istanziarlo:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Occorrerà però predisporre un blocco *try-catch* per la gestione di un eventuale errore che si potrebbe generare:

```
try {
	Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(ClassNotFoundException e){
	System.out.println("Impossibile caricare il driver: "+ e);
}
```

### *Connessione al database*

Una volta caricati i driver, è possibile connettersi al database usando i metodi esistenti nella classe java.sql.DriverManager:

- `getConnection(String url, Properties info);`
- `getConnection(String url, String user, String psw);`
- `getConnection(String url);`

Sintassi per ottenere l'oggetto connection:

```
Connection con;
try {
	con=DriverManager.getConnection("Jdbc:Odbc:mioDatabase");
}
catch (SQLException e){
	System.out.println("Impossibile caricare il driver: "+ e);
}
```

### *Le interfacce Statement*

Una volta ottenuto l'oggetto Connection, è possibile usarlo per creare un'interfaccia *Statement* SQL, che ha il compito di inviare i comandi SQL al database:

- comandi di definizione dei dati, come CREATE TABLE o CREATE INDEX
- comandi di manipolazione dei dati come INSERT o UPDATE
- istruzioni SELECT per l'esecuzione di query

Questa interfaccia può essere sia di tipo *Statement* che *PreparedStatement*. Ecco la sintassi di creazione di uno Statement:

```
try {
	Statement st=con.createStatement();
}
catch (SQLException e){
	System.out.println("Errore SQL: "+ e);
}
```

### *ResultSet*

I risultati (ResultSet) sono rappresentati da un elenco ordinato di righe rappresentate in JDBC con l'interfaccia *java.sql.ResultSet*. I risultati vengono prodotti da *executeQuery(),* o da alcune chiamate a metodi. Dopo la creazione, è possibile estrarre i dati nei seguenti modi:

- Richiamando uno dei metodi forniti da JDBC: *absolute()*, *relative()*, *next()*, *previous()*, *first()*, *last()*, *beforeFirst()*, *afterLast()*.
- Prelevando i valori contenuti nella colonna desiderata utilizzando *ResultSet.getxxx(numero-colonna)* o *ResultSet.getxxx(nome-colonna)*, dove *xxx* è il tipo di dati JDBC.

In tabella, si riporta un elenco completo dei metodi getxxx di ResultSet:

| METODO | DESCRIZIONE |
|---|---|
| GetArray | Restituisce un array SQL. |
| getAsciiStream | Restituisce uno stream java.io.InputStream di caratteri ASCII. La traduzione in ASCII (se necessaria) viene gestita dal driver JDBC |
| getBigDecimal | Restituisce un java.math.BigDecimal. |
| getBinaryStream | Restituisce un java.io.InputStream. Non viene eseguita alcuna traduzione. |
| GetBlob | Restituisce un java.sql.Blob *(Binary Large Object).* |
| getBoolean | Restituisce un valore booleano. |
| GetByte | Restituisce un singolo byte. |
| GetBytes | Restituisce un array di byte. |
| getCharacterStream | Restituisce uno stream di caratteri java.io.Reader. |
| GetClob | Restituisce un java.sql.Clob *(Character Large Object).* |
| Gettate | Restituisce una java.sql.Date, una sottoclasse di java.util.Date. |
| getDouble | Restituisce un valore double. |
| GetFloat | Restituisce un valore float. |
| GetInt | Restituisce un valore intero. |
| GetLong | Restituisce un valore intero long. |
| getObject | Restituisce un oggetto java.lang.Object. |
| GetRef | Restituisce un riferimento java.sql.Ref a un tipo strutturato SQL. |
| GetShort | Restituisce un valore intero o short. |
| GetString | Restituisce una stringa. |
| GetTime | Restituisce un valore java.sql.Time. |
| getTimestamp | Restituisce un valore java.sql.Timestamp che include l'ora in nanosecondi. |

## Pubblicazione di un web JSP

Le specifiche API servlet 2.2 e JSP 1.1 considerano un'applicazione Web come una raccolta di risorse cooperanti, associate a un'area comune dello spazio dei nomi del server Web. Questa raccolta può includere servlet, pagine JSP, file HTML, immagini, classi di supporto e dati di configurazione.

### *La struttura delle directory*

Un'applicazione Web ha una determinata struttura di directory che deve essere nota a tutti i servlet. Il livello superiore, la radice dell'applicazione, contiene i documenti HTML, le pagine JSP, le immagini e tutte le altre risorse che costituiscono il contenuto dell'applicazione. Sotto la directory radice si può trovare un numero qualsiasi di subdirectory anch'esse contenenti informazioni per l'applicazione, come le cartelle nell'albero dei documenti di un server Web.
La directory radice contiene anche una speciale directory chiamata *WEB-INF.* Questa directory e le sue subdirectory non sono visibili agli utenti delle applicazioni e contengono i servlet, le classi, i file .jar e i dati di configurazione che costituiscono le parti operative della applicazioni. In WEB-INF vi sono tre elementi degni di nota.

- *classes:* questa directory contiene i servlet e le altre classi. Queste classi vengono trovate e automaticamente dal loader delle classi come se fossero nel percorso delle applicazioni. classes può avere delle subdirectory che corrispondono alla struttura del package, così come ogni altra directory del percorso.
- *lib:* è simile a classes ma contiene i file .jar. Le classi contenute nei file .jar di questa directory vengono rese automaticamente disponibili per il caricatore di classi senza dover essere elencate esplicitamente in qualche percorso.
- *web.xml:* questo è un documento XML che controlla la pubblicazione. È stato definito in modo rigoroso come una struttura indipendente dal produttore e viene utilizzato per configurare i servlet e le altre risorse che compongono l'applicazione Web.

In WEB-INF possono trovarsi anche altri file e subdirectory anche se le specifiche API servlet non dicono nulla a questo proposito. Una subdirectory utilizzata frequentemente è tlds che contiene i descrittori Tag Library Descriptor per i tag personalizzati JSP. Poiché il contenuto di questa subdirectory è visibile per le classi dell'applicazione ma non per gli utenti Web, WEB-INF viene frequentemente utilizzata per scopi specifici. In generale, WEB-INF è disponibile per inserirvi i dati da utilizzare in un'applicazione Web e nel frattempo risulta nascosta agli occhi degli utenti.

### *Il file war (Web Archive)*

Finora è stata descritta la struttura runtime di un'applicazione Web. Quando si pubblica l'applicazione, questa struttura deve essere quella rappresentata da un unico archivio Web costituito da un file *war (Web Archive).* Si tratta semplicemente di un file .jar con un'espressione differente (.war), il cui livello superiore corrisponde alla radice dell'applicazione Web.

Tutte le servlet engine compatibili servlet 2.2 devono accettare direttamente un file .war e costruire l'applicazione Web corrispondente. I mezzi utilizzati per installare il file dipendono dalla servlet engine impiegata.
Tomcat, l'implementazione di riferimento, consente di depositare semplicemente i file .war nella directory *<home_tomcat>/webapps*. Al successivo riavvio di Tomcat, il file .war verrà espanso e convalidato rendendo così disponibile la nuova applicazione. Le servlet engine commerciali in genere offrono uno strumento di amministrazione grafico per svolgere questa operazione.

### *Il descrittore: web.xml*

Il file web.xml della directory WEB-INF è chiamato descrittore; si tratta di un documento XML in un formato ben preciso che specifica la configurazione dell'applicazione Web. Fra le altre cose può essere utilizzato per descrivere:

- sinonimi di servlet, associazioni e parametri di inizializzazione
- i limiti di time-out delle sessioni
- i parametri globali da rendere disponibili nell'applicazione
- la configurazione di sicurezza
- i tipi MIME

Poiché questo file è un documento XML, il suo formato è descritto in una DTD (Document Type Definition). Il file DTD si chiama *web-app_x.y.dtd* dove *x.y* è la versione delle specifiche API servlet, ad esempio 2.2. Viene pubblicato da Sun Microsystems e può essere prelevato ali' indirizzo

```
http://java.sun.com/j2ee/dtds/web-app_2_2.dtd
```

Il file web.xml più semplice ha il seguente aspetto:

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC
      "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
      "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>

</web-app>
```

Nel corpo di <web-app> si trovano gli altri elementi che descrivono la configurazione dell'applicazione.
E' importante ricordare che gli elementi contenuti nel corpo <web-app> devono essere riportati esattamente nell'ordine specificato. Se possono essere utilizzati più elementi dello stesso tipo, questi devono comunque trovarsi insieme e non inframmezzati ad altri. Ad esempio tutti gli elementi <servlet> devono trovarsi prima di tutti gli elementi <servlet-mapping>.

# 3 - Descrizione dell'applicazione

In questo capitolo verranno descritte, mediante la sintassi *UML*, le classi sulle quali si basa l'applicazione.

## *Package diagram*

Le classi sono suddivise in due package:

- *packDB*: destinato alla gestione del database utenti
- *packFile:* preposto alle operazioni di interazione con i file

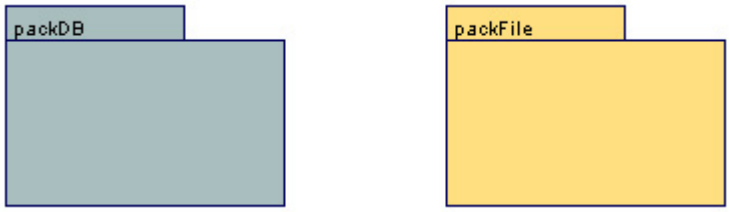


**Figura 36 - Packages**

## *Class diagram*

Il class diagram fornisce una descrizione delle classi e dei metodi che stanno alla base dell'applicazione, da un punto di vista Object Oriented:

### Package packDB

#### *Classe dbIO*

E' la classe che si occupa dell'interazione con il database utenti:

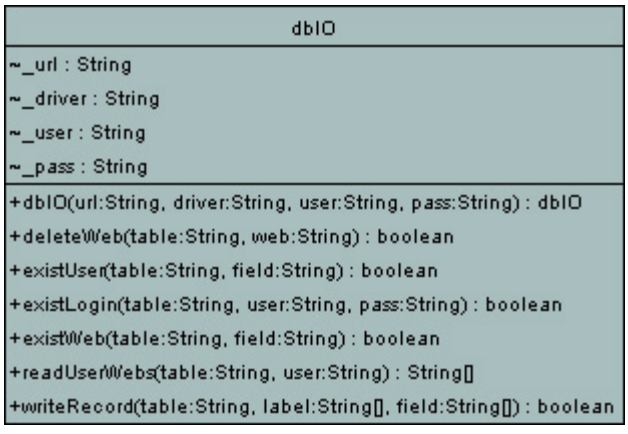


**Figura 37 - Classe dbIO**

Metodi:

- dbIO: costruttore
- deleteWeb: cancella una entry dalla tabella *Web*
- existUser: verifica la presenza di un nome utente
- existLogin: verifica la correttezza dei dati immessi in fase di login
- existWeb: verifica la presenza di un sito web

- readUserWebs: restituisce l'elenco dei siti gestiti da un particolare utente
- writeRecord: inserisce un nuovo record in tabella

## Package packFile

### *Classe OnlyExt*

Classe utilizzata per implementare un filtro sull'estensione dei file:

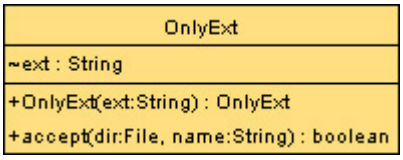


**Figura 38 - Classe OnlyExt**

Metodi:

- OnlyExt: costruttore
- accept: ritorna i nomi dei file con estensione corretta

### *Classe fileAdd*

Classe che si occupa dell'inserimento del codice HTML, relativo ai nuovi elementi, all'interno dei file:


**Figura 39 - Classe fileAdd**

Metodi:

- fileAdd: costruttore
- countTag: ritorna il numero di occorrenze di un determinato tag all'interno di un file.
- insertParagraph: inserisce il codice HTML relativo a un nuovo paragrafo

- insertImage: inserisce il codice HTML relativo a una nuova immagine
- insertMailForm: inserisce il codice HTML relativo a un nuovo modulo di richiesta informazioni
- insertLink: inserisce il codice HTML relativo a un nuovo collegamento ipertestuale
- insertEmpty: inserisce una cella vuota in tabella
- insertLine: inserisce il codice HTML relativo a un nuovo separatore
- insertTable: inserisce il codice HTML relativo a una nuova tabella
- startTable: inserisce il codice HTML di apertura della tabella
- stopTable: inserisce il codice HTML di chiusura della tabella
- writeTag: scrive di un tag su file
- getElements: ritorna la lista degli elementi presenti nella pagina
- deleteElement: cancella un elemento dal file
- stringToInt: converte da stringa a intero (funziona nell'intervallo [-1,10])
- intToString: converte da intero a stringa (funziona nell'intervallo [-1,10])

***Classe fileGram***

Si occupa della gestione della grammatica HTML di un file:

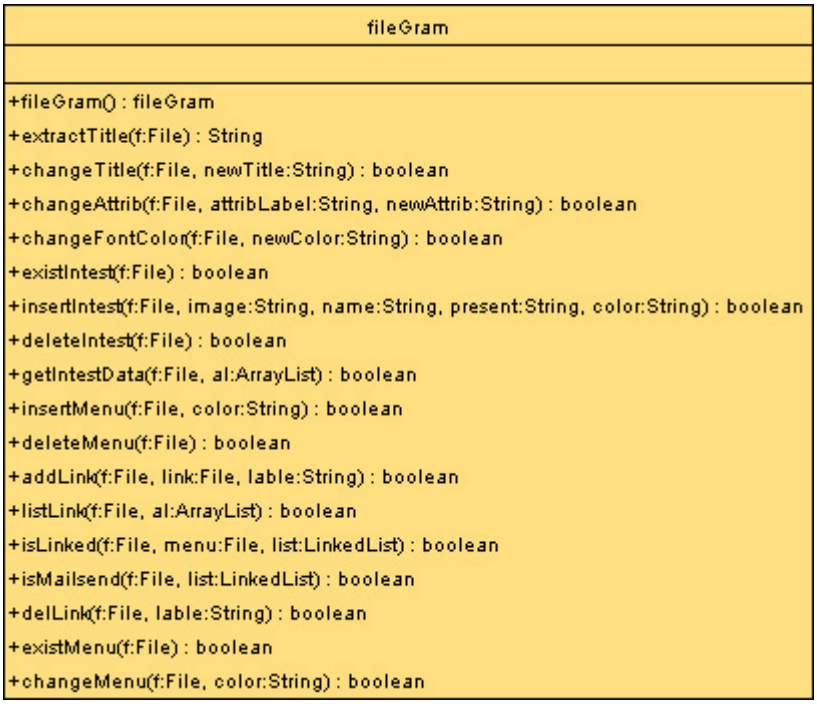


**Figura 40 - Classe fileGram**

Metodi:

- fileGram: costruttore
- extractTitle: ritorna il titolo di una pagina HTML
- changeTitle: modifica il titolo di una pagina HTML
- changeAttrib: modifica il valore di un attributo
- changeFontColor: modifica il colore del testo
- existIntest: verifica la presenza dell'intestazione grafica
- insertIntest: inserisce l'intestazione grafica

- deleteIntest: cancella l'intestazione grafica
- getIntestData: ritorna i dati (titolo e frase di presentazione) dell'intestazione grafica
- insertMenu: inserisce il menù di navigazione
- deleteMenu: cancella il menù di navigazione
- addLink: aggiunge un link al menù
- listLink: ritorna la lista dei link presenti nel menù
- isLinked: verifica la presenza di un link a file nel menù
- isMailSend: verifica la presenza di un mailSend, collegato ad un file
- delLink: candella un link dal menù
- existMenu: verifica la presenza del menù di navigazione
- changeMenu: modifica l'interfaccia grafica del menù di navigazione

***Classe fileIO***

Si occupa dell'input/output su file e dell'interazione con il filesystem:

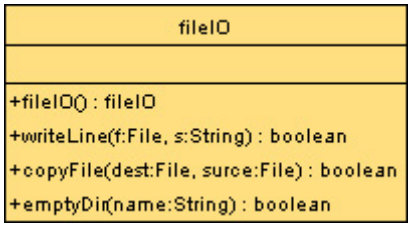


**Figura 41 - Classe fileIO**

Metodi:

- fileIO: costruttore
- writeLine: scrive una stringa su file
- copyFile: effettua la copia di un file, da sorgente a destinazione
- emptyDir: elimina una directory ed il suo contenuto

## *Activity Diagram*

In questo paragrafo si descriveranno le azioni di maggior interesse tramite diagrammi delle attività. Si porrà particolare attenzione ai legami logici presenti.

### Login utente

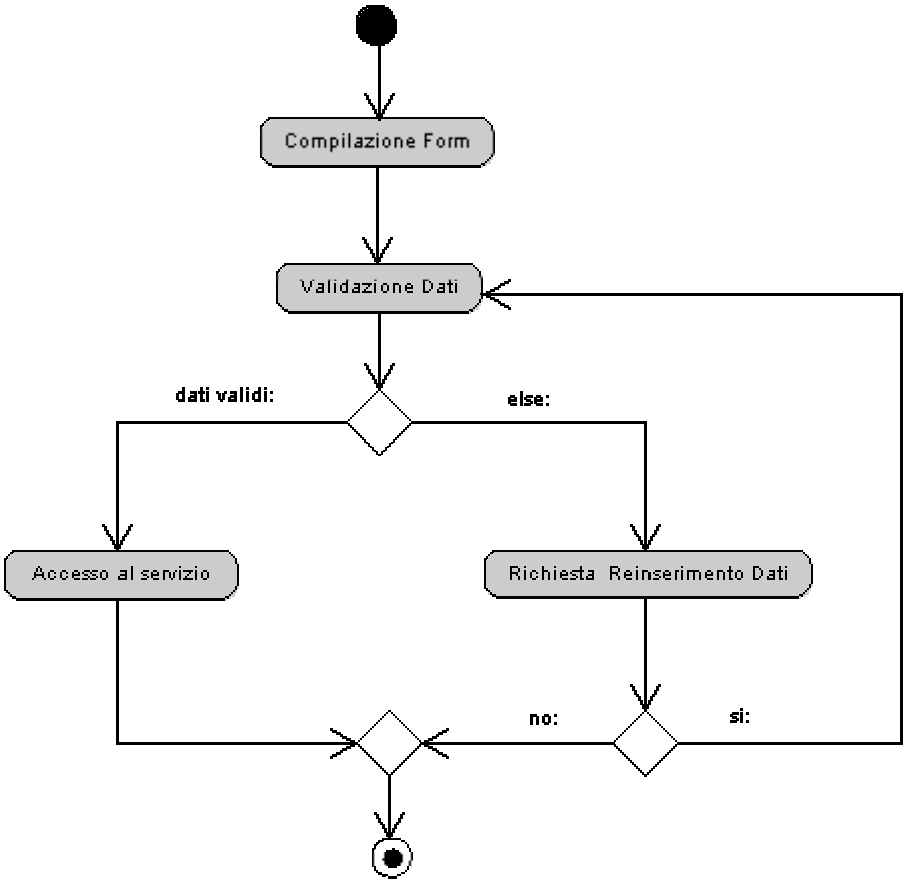


**Figura 42 - Login utente**

La procedura di login prevede la compilazione di un form e la successiva validazione dei dati inseriti. In caso di corretta autenticazione, si permette l'accesso al servizio, in caso contrario si consente la reintroduzione dei dati.

## Registrazione nuovo utente

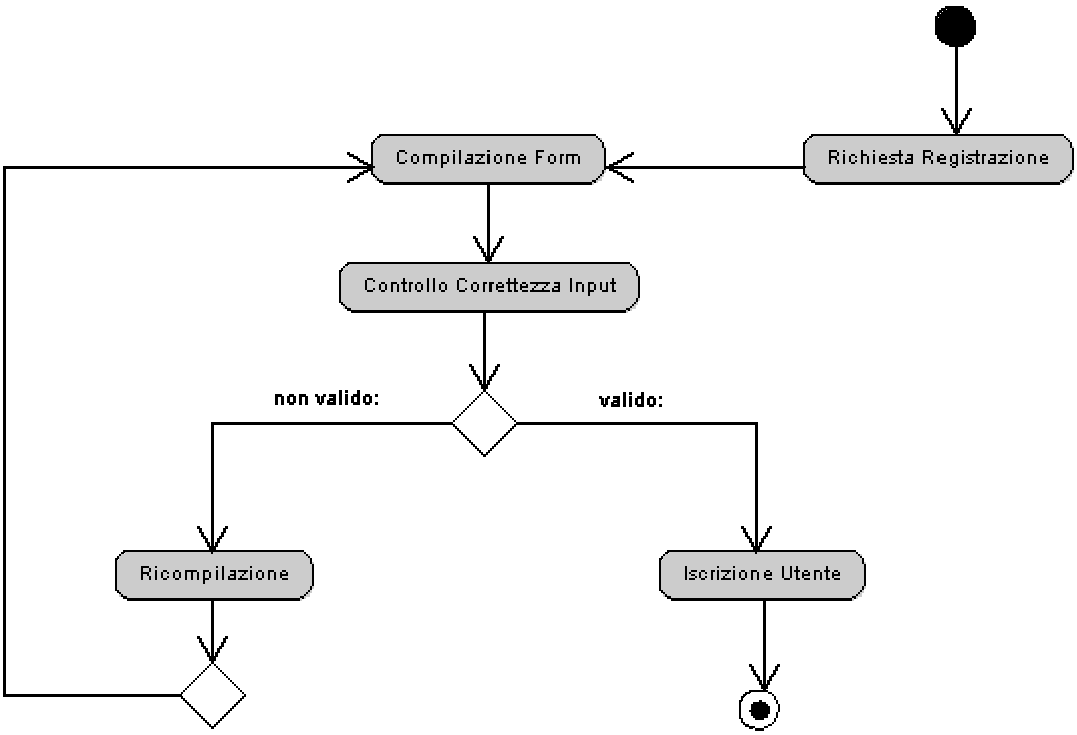


**Figura 43 - Registrazione nuovo utente**

Ogni nuovo utente è tenuto a seguire una apposita procedura di registrazione, prima di poter accedere ai servizi offerti. La registrazione prevede la compilazione (completa) di un form di raccolta delle informazioni. In caso di errata o incompleta compilazione, l'utente è avvisato da un messaggio di errore.

## Creazione di un nuovo sito

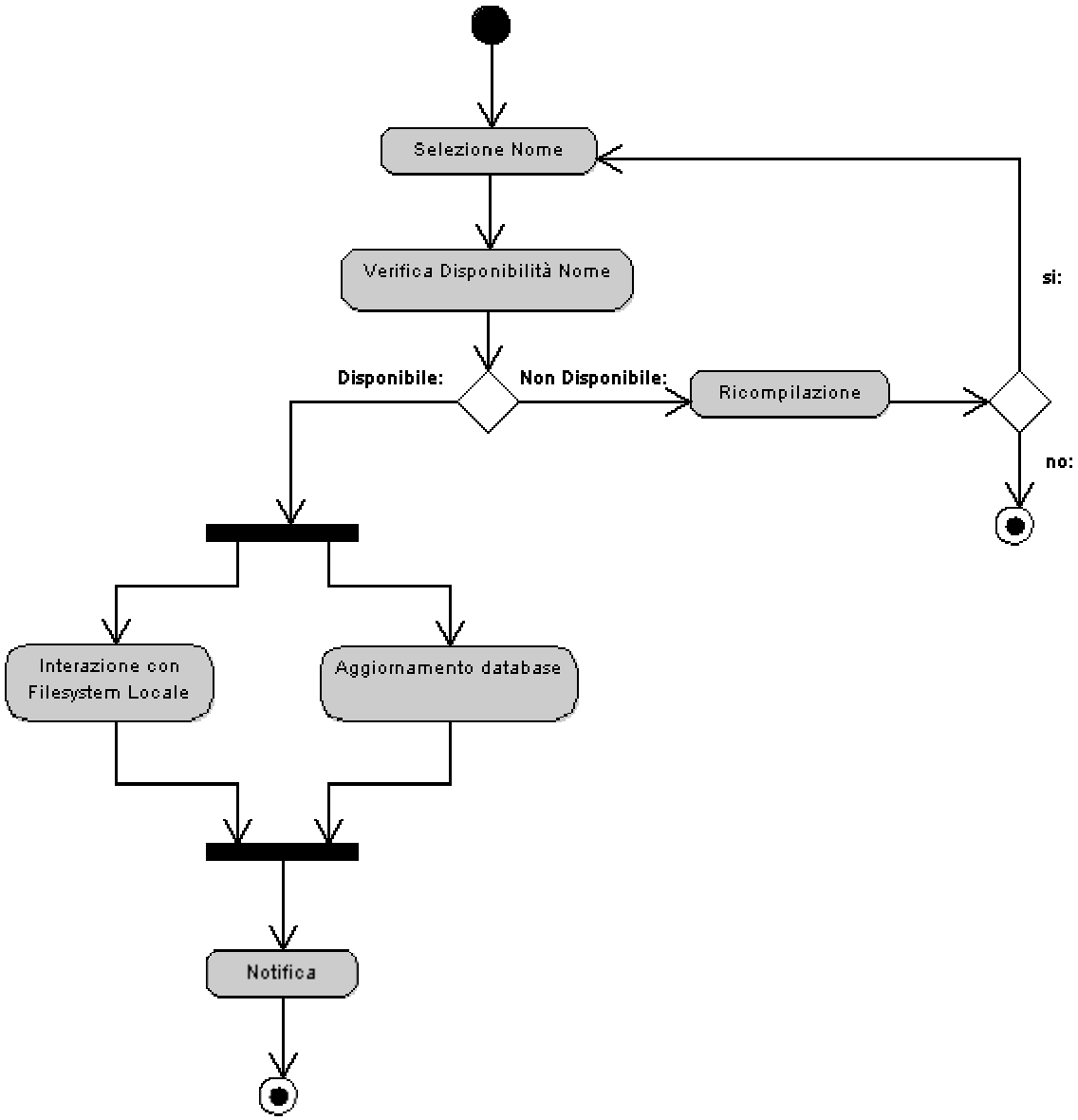


**Figura 44 - Creazione di un nuovo sito**

Una volta che l'utente ha selezionato il nome per il sito, l'applicazione ne verifica la disponibilità. In caso negativo, richiede la ricompilazione del modulo, mentre, in caso affermativo, procede alla creazione dei sito su filesystem locale e all'aggiornamento del database utenti. L'operazione viene notificata mediante un apposito messaggio di conferma.

## Inserimento di un elemento

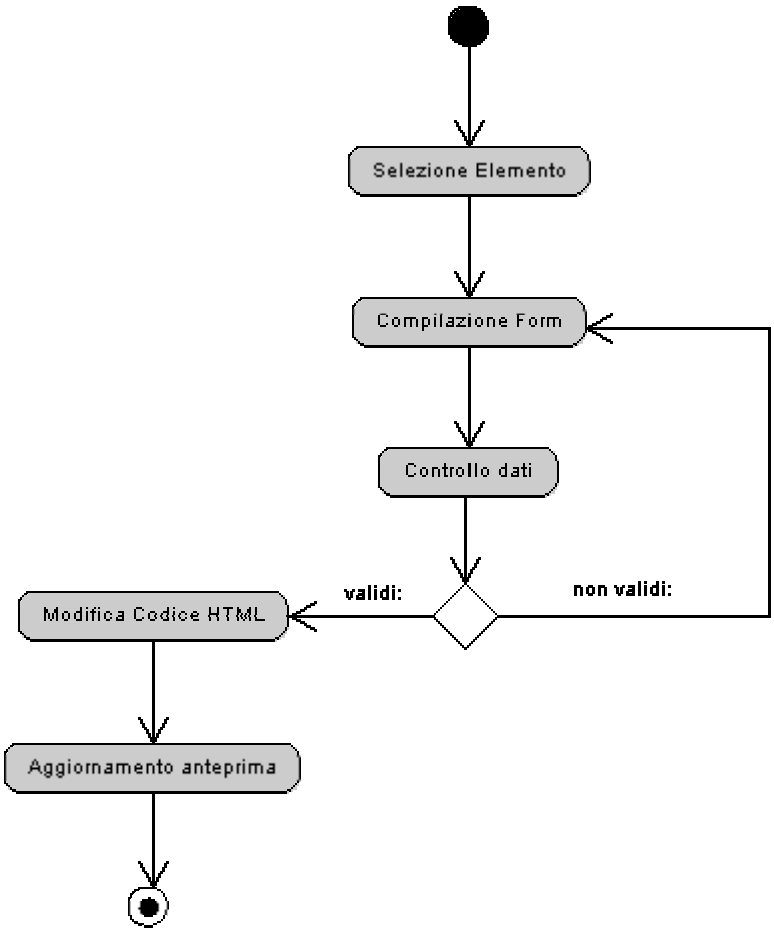


**Figura 45 - Inserimento di un elemento**

Il diagramma illustra la procedura di inserimento di un generico elemento, durante la fase di modifica di una pagina del sito corrente.
Dopo la compilazione del modulo, destinato a raccogliere le preferenze dell'utente, l'applicazione verifica la correttezza dei dati immessi. In caso di errore di compilazione, viene richiesta la correzione del dato non valido.
Terminata la compilazione del form, il software si occupa di apportare le modifiche richieste al codice HTML della pagina corrente e di aggiornare l'anteprima.

# 4 - Sessioni e gestione degli errori

## *Gestione delle sessioni*

Il protocollo *HTTP (HyperText Transfer Protocol)* è stato originariamente sviluppato per distribuire documenti e immagini tramite il World Wide Web. Per questo motivo utilizza un modello di comunicazioni piuttosto semplice. Un client richiede un documento, il server risponde con il documento o un codice d'errore e la trasmissione ha termine. Il server non conserva alcuna registrazione della richiesta. La prossima volta che il client eseguirà una richiesta, il server non avrà alcun modo per distinguerlo da un altro client. Per questo motivo il protocollo HTTP è definito un protocollo "*senza stati*".
Solo poche applicazioni sono adatte a questo modello a richiesta e risposta. Nella maggior parte dei casi, per ottenere un risultato di qualche rilievo sono necessarie più richieste. Una difficoltà ulteriore è data dal fatto che alcuni processi eseguiti sul server richiedono un certo tempo, più di quanto un server Web possa attendere per garantire prestazioni ragionevoli.
Non si tratta di problemi nuovi. I programmi *CGI (Common Gateway Interface)* e i sistemi di elaborazione delle transazioni hanno dovuto preoccuparsi di questi stessi problemi per molti anni. Le tecniche applicate in questi ambienti sono utilizzabili anche nell'ambiente servlet / JSP, ma *l'API servlet Java* offre vari meccanismi interni che garantiscono una soluzione più elegante e di facile uso: **le sessioni http.**

## Controllo delle sessioni

Poiché il server Web non è in grado di ricordare da una richiesta alla successiva le operazioni svolte da un client, per gestire una sessione occorre fare i modo che siano i client a memorizzarne lo stato. Questo può essere ottenuto in due modi:

- Facendo in modo che il client conservi tutti i dati della sessione e li invii al server.
- Facendo in modo che sia il server a conservare tutti i dati, gli assegni un identificatore e fornisca al client solo tale identificatore.

Il primo approccio è facile da implementare e non richiede l'introduzione di particolari funzionalità sul server. Questo approccio, però, può generare lo scambio di grandi quantità di dati con un impatto negativo sulle prestazioni. Un altro problema riguarda le risorse del server, che devono essere reinizializzate a ogni richiesta. Per questi motivi questo approccio è più adatto a sessioni a lunga persistenza che impiegano piccole quantità di dati, come ad esempio le preferenze o i numeri di conto.
Il secondo approccio offre funzionalità più ampie. Una volta che un server attiva una sessione e il client la accetta, il server può costruire oggetti attivi complessi e gestire grandi quantità di dati, distinguendo le sessioni attive tramite una semplice chiave.
Per far si che il client memorizzi i dati e li restituisca al server Web, vengono utilizzate principalmente quattro tecniche:

- i campi nascosti
- la riscrittura dell'indirizzo URL
- i cookie
- l'API per sessione http

L'applicazione sviluppata impiega quest'ultima tecnica, in quanto le altre metodologie di gestione presentano alcuni svantaggi.

I campi nascosti non forniscono la necessaria flessibilità richiesta da un'applicazione composta da numerosi moduli interagenti tra loro. La gestione dei parametri passati mediante campi *hidden* risulta essere, il più delle volte, molto laboriosa.
La riscrittura dell'indirizzo URL, d'altra parte, presenta evidenti problemi di sicurezza (i parametri passati risultano visibili in chiaro) e risente delle impostazioni del browser.
Infine, si è preferito non impiegare cookie (sebbene siano una tecnica oramai molto diffusa) in quanto vengono visti da molti utenti, come una sorta di intrusione. La disabilitazione dei cookie a livello di browser porterebbe a gravi malfunzionamenti dell'applicazione.

## L'API per sessioni

Il package javax.servlet.http, che costituisce l'API per sessioni, contiene tre classi:

- *HttpSession:* un'interfaccia che si comporta come una mappa o una tabella hash e che è in grado di memorizzare e prelevare gli oggetti per nome. Viene creata una sessione tramite una chiamata a *HttpServletRequest.getSession();* questa sessione persiste fino a una determinata scadenza o finché non viene chiusa da un servlet della sessione. Le richieste HTTP in arrivo che trasportano l'identificatore di sessione vengono associate automaticamente alla sessione.
- *HttpSessionBindingListener:* un'interfaccia che consente a un oggetto di conoscere quando è stato memorizzato o rimosso da una sessione. L'interfaccia contiene i metodi callback *valueBound()* e *valueUnbound()*, che l'oggetto deve implementare per poter ricevere le notifiche di associazione.
- *HttpSessionBindingEvent:* un oggetto passato ai metodi *valueBound()* e *valueUnbound()* di *HttpSessionBindingListener*. L'evento offre dei metodi per restituire la sessione e il nome con il quale l'oggetto è stato connesso alla sessione.

## Creazione di sessioni

Un servlet indica che vuole utilizzare una sessione richiamando i metodi *getSession()* o *getSession(boolean create)* di HttpServletRequest nel seguente modo:

```
HttpSession session = request.getSession(true);
```

Il metodo getSession() senza parametri richiama semplicemente getSession(true). Il parametro create indica se la servlet engine deve creare una nuova sessione se non esiste già. Se il parametro è false, il servlet può operare solo in una sessione preesistente. In ogni caso, la richiesta viene esaminata per vedere se contiene un codice di sessione valido. In caso affermativo il container del servlet restituisce un riferimento all'oggetto della sessione che può essere utilizzato per memorizzare e prelevare gli attributi della sessione.
In una pagina JSP, la creazione di una sessione è automatica, tranne quando viene soppressa nella direttiva page. All'inizio del metodo jspService() del servlet generato, viene creato e inizializzato l'oggetto *PageContext*. Nell'ambito dell'inizializzazione, il metodo JspFactory.getPageContext() richiama request.getSession(true). La sessione creata o richiamata viene restituita al

servlet generato quando richiama pageContext.getSession(). La sessione diventa accessibile alla parte rimanente della pagina JSP e all'interno della variabile implicita session.
Se una pagina JSP non deve utilizzare una sessione, dovrà sopprimere la creazione automatica nella direttiva page:

```
<%@ page session="false" %>
```

Questa evita alla servlet engine di dover creare e gestire una sessione non necessaria. Il risparmio di memoria può essere significativo.

Quando la sessione viene creata, il client (il browser Web) non la conosce ancora. Quando il codice di sessione viene inviato al client e il client lo rinvia nella richiesta successiva, si dice che il client si *connette alla sessione*. Un servlet o una pagina JSP può rilevare se il client si è connesso, utilizzando il metodo *isNew():*

```
HttpSession session = request.getSession();
if (session.isNew()) {
        ... // Codice di inizializzazione
}
```

*session.isNew()* è *true* se la sessione è appena stata creata e se il client non è ancora stato informato o se il client è stato informato ma sceglie di non parteciparvi.

## Memorizzazione e prelevamento degli oggetti dalle sessioni

Gli oggetti vengono associati a una sessione tramite il metodo *setAttribute():*

```
session.setAttribute("user", userID);
```

Il nome con il quale un oggetto viene associato può essere una qualunque stringa univoca. In una sessione è possibile memorizzare ogni tipo di oggetto, ma poiché le sessioni possono essere serializzate, è opportuno che gli oggetti della sessione implementino java.io.Serializable. Possono essere memorizzati solo gli oggetti e non i tipi primitivi come *int*, *char* o *double*. Per memorizzare i tipi primitivi occorre utilizzare i rispettivi wrapper: *Integer*, *Character* o *Double*.

Gli oggetti possono essere estratti da una sessione con il metodo *getAttribute():*

```
String userlD = (String) session.getAttribute( "user");
```

Come una mappa o una tabella hash, una sessione memorizza solo oggetti e quindi i dati prelevati devono essere convertiti nel tipo appropriato. I tipi primitivi contenuti nelle classi wrapper devono essere estratti dai metodi forniti nella classe wrapper:

```
Integer countObject = (Integer) getAttribute("count");
int count = countObject.intValue();
```

Normalmente, se si memorizza un attributo in una sessione, si conosce il suo nome e il suo tipo ed è possibile richiederlo direttamente in questo modo. E anche possibile ottenere un elenco dei nomi di attributi utilizzando il metodo *getAttributeName():*

```
Enumeration enames = session.getAttributeNames();
while (enames.hasMoreElements()) {
```

```
        String name = (String) enames.nextElement();
        Object value = session.getAttribute(name);
        out.println(name + " = " + value);
}
```

Quando un oggetto non è più necessario, può essere eliminato dalla sessione con *removeAttribute():*

```
session.removeAttribute("user");
```

L'operazione viene eseguita automaticamente quando viene chiusa la sessione ma vi sono casi in cui un attributo deve essere eliminato prima della chiusura della sessione.

## Distruzione delle sessioni

Dopo la creazione, una sessione normalmente persiste fino alla sua scadenza. La scadenza (*timeout*) fa riferimento alla durata massima in cui la sessione rimarrà valida. Questa è una considerazione importante poiché il server non ha alcun modo per sapere se un client ha terminato di lavorare con una sessione se non gli viene comunicato esplicitamente.
L'intervallo di timeout standard può essere impostato nel descrittore *web.xml*:

```
<web-app>
   <session-config>
      <session-timeout> 30 </session-timeout>
   </session-config>
   ...
</web-app>
```

L'intervallo è specificato in minuti, normalmente 30. Il valore specificato riguarda tutte le sessioni dell'applicazione, a meno che le sessioni non modifichino singolarmente questo valore.
Alcune applicazioni che utilizzano risorse disponibili in numero limitato, possono scegliere una scadenza più ravvicinata, usando il metodo *setMaxInactiveInterval()* per impostare un intervallo più breve:

```
session.setMaxInactiveInterval(180);
```

L'argomento fornito a *setMaxInactiveInterval()* è il numero di secondi. Il valore corrente può essere ottenuto con *getMaxInactiveInterval().* Se si specifica un valore negativo, la sessione non scade mai.
In alcuni casi, può essere specificata una conclusione ben precisa per la sessione. In questi casi, può essere utilizzato il metodo *invalidate():*

```
session.invalidate();
```

Questo metodo contrassegna la sessione come inattiva ed elimina tutte le associazioni con gli oggetti allocati.

## *Gestione degli errori*

### Gli attributi errorPage e isErrorPage

Se si verifica un'eccezione mentre viene valutata una pagina JSP, la servlet engine in genere invia al browser uno *stack trace*. Ma questo comportamento può essere utile solo per un programmatore in fase di sviluppo ma è indesiderabile in un'applicazione Web commerciale. JSP offre una soluzione semplice e comoda che richiede l'uso coordinato di due attributi: *errorPage* e *isErrorPage*.
Una pagina JSP può indicare che quando lancia un'eccezione non raccolta venga richiamata una determinata pagina d'errore:

```
<%@ page errorPage="url-errore" %>
```

dove *url-errore* è l'indirizzo URL di un'altra pagina JSP nello stesso contesto. Questa pagina JSP deve utilizzare il seguente attributo nella propria direttiva page:

```
<%@ page isErrorPage="true" %>
```

Una pagina d'errore ha accesso all'eccezione tramite la variabile implicita exception (questa è l'unica circostanza in cui una pagina JSP ha accesso a questa variabile). Da qui può estrarre il messaggio d'errore con *getMessage()* per visualizzarlo o registrarlo se necessario. Inoltre può generare uno stack trace con *exception. printStackTrace().*
Poiché una pagina d'errore è anch'essa una pagina JSP, ha accesso al contesto, alla sessione (se esiste), alla richiesta e agli altri oggetti del servlet.

### Controllo della validità della sessione

La persistenza dei dati gestiti dall'applicazione può essere compromessa da un periodo di inattività sufficientemente lungo da causare il *timeout* della sessione.
Di conseguenza è stato previsto, all'interno della *errorPage*, un controllo di validità della sessione corrente.
Nel caso l'errore riscontrato sia dovuto ad un timeout di sessione, *errorPage* fornisce all'utente gli strumenti per ripristinare il proprio lavoro.

```
<%
//controllo della validità della sessione
boolean b = request.isRequestedSessionIdValid();

//se la sessione è scaduta
if(!b){
	...	//codice HTML: gestione errore di sessione
}

//se l'errore non è di sessione
else{
	...	//codice HTML: avviso di errore imprevisto
}
%>
```

# 5 - Test e problemi riscontrati

## *Test dell'applicazione*

Il test dell'applicazione è stato condotto mediante la realizzazione di tre siti dimostrativi, al fine di valutare le effettive potenzialità del progetto sviluppato. In particolare, si è cercato di riprodurre il sito:

http://dbgroup.unimo.it/bdatia


**Figura 46 - Sito dimostrativo**

La riproduzione ottenuta è risultata soddisfacente, sia dal punto di vista grafico che dal punto di vista del mantenimento delle funzionalità presenti. L'unica modifica apportata al layout ha riguardato l'impostazione dell'intestazione di pagina, caratterizzato da una tabella con struttura non riproducibile.
E' risultato inoltre impossibile includere direttamente il "*NewsViewer*", componente dinamico finalizzato alla visualizzazione delle notizie. Tale componente può comunque essere collegato mediante un riferimento ipertestuale.

I siti realizzati sono stati valutati sia mediante NetBeans IDE 3.5.1, che include Tomcat 4.1, sia tramite una versione *stand alone* di Tomcat 5.0 (in particolare, la release 5.0.19), che risulta essere attualmente il container di riferimento per quanto riguarda le applicazioni JSP.

## *Problemi riscontrati*

**Installazione di Apache Tomcat 5.0.19**
Inizialmente, è stato impossibile disporre di Apache Tomcat a causa di un conflitto, con Window XP Professional, sull'uso della porta 8005, richiesta dal webserver come *Server port*.
Il problema è stato risolto modificando il file di configurazione *server.xml*, presente nella sottocartella *conf*, impostando come S*erver port* la porta 8085.

**Errore di compilazione durante la fase di test**
Apache Tomcat non comprende un compilatore Java. Per questo motivo, l'apertura di un sito JSP non compilato produce l'errore:

```
org.apache.jasper.JasperException: Unable to compile class for JSP
```

Il problema può essere risolto copiando manualmente *$JAVA_HOME/lib/tools.jar* da JDK alla cartella *common/lib* di Tomcat.

**Instabilità di Tomcat 4.0.6**
In alcune circostanze, Tomcat 4.0.6, che equipaggia NetBeans IDE 3.5.1, è risultato instabile. In particolar modo, il webserver ha impedito, senza un motivo tangibile, l'accesso al file system da parte dell'applicazione.
A causa della non ripetitibilità e della manifestazione casuale dell'errore, si è preferito procedere alla reinstallazione dell'intero sistema (sistema operativo, Java IDE, webserver), al fine di ripristinare il corretto funzionamento della piattaforma.

**Instabilità di NetBeans IDE 3.5.1**
Durante lo sviluppo dell'applicazione è stato necessario procedere alla disinstallazione completa di NetBeans e di JDK, al fine di ripristinare il corretto funzionamento dell'IDE Java, la cui interfaccia utente risultava, inspiegabilmente e sensibilmente compromessa.
Dopo la reinstallazione è stato necessario impostare un nuovo file di progetto, al fine di ripristinare la piena funzionalità della piattaforma.

**Controllo delle sessioni**
Il metodo *request.isRequestedSessionIdValid()* si è dimostrato, in alcune circostanze, non pienamente affidabile. Di conseguenza errorPage potrebbe non gestire correttamente gli errori dovuti allo scadere della sessione corrente.
Questo inconveniente può essere risolto mediante un controllo del valore (*boolean*) ritornato dai metodi che sfruttano variabili di sessione.

**Impostazione del layout**
L'applicazione sviluppata permette di creare siti con varie tipologie di interfaccia, garantendo all'utente una buona flessibilità. Tuttavia, per comporre layout complessi, è necessaria una buona conoscenza dell'applicazione e delle sue potenzialità. Molto spesso è necessario combinare vari elementi (tabelle, paragrafi, link, immagini e separatori) al fine di raggiungere il risultato desiderato.
Potrebbe essere utile fornire agli utenti alcuni suggerimenti sulle tecniche di composizione dei principali elementi di una pagina web, anche se, solo l'esperienza li porterà ad ottenere le soluzioni cercate.

# 6 - Conclusioni e sviluppi futuri

Obiettivo del progetto è la realizzazione di un'applicazione che consenta di generare siti web, con l'ausilio di template.
Sulla base del concetto di flessibilità, si è cercato di fornire uno strumento versatile, semplice da utilizzare e facilmente aggiornabile.
L'utente può disporre di elementi grafici precostruiti, qualora non disponesse del bagaglio tecnico necessario per crearne di nuovi, ma può anche decidere di gestire ogni elemento del suo sito in piena libertà. Può infine scegliere di far coesistere elementi grafici preformattati ed elementi creati autonomamente.
In questo modo, si è cercato di soddisfare la più ampia gamma di utilizzatori possibile.

Su Internet, è disponibile una vasta gamma di editor HTML e di strumenti *freeware* dedicati a facilitare la realizzazione del proprio sito, come *AceHTML*, *EasyHTML*, *e PSPad Editor*. Molti fornitori di servizi di *hosting* (come Digilander di libero.it e Tripod di lycos.it) dispongono di form che consentono di automatizzare le fasi di pubblicazione del lavoro creato. Sono sempre più diffusi i *blog*, le gallerie fotografiche precostruite, i *file sharer (strumenti dedicati alla condivisione di file)*.
*Nessuno di questi strumenti consente però all'utente di creare in piena libertà, in modo semplice, integrato ed indipendentemente dal codice HTML, un sito web completo*. La struttura del progetto JSP inoltre, non prevede la fase di pubblicazione, dato che i file vengono creati direttamente sul server web, risparmiando così all'utente una fase dispersiva e ripetitiva.

Per questi motivi, l'applicazione sviluppata nel corso di questa tesi, può rappresentare un valido strumento per tutti coloro che, pur non disponendo di risorse ingenti o conoscenze tecniche specifiche, desiderano realizzare e vedere pubblicato il proprio sito personale.

Personalmente, ritengo questa esperienza positiva ed altamente formativa. Lo sviluppo di questo progetto mi ha permesso di migliorare sensibilmente la conoscenza della programmazione *web oriented* e di acquisire un nuovo ed importante strumento, le JavaServer Pages (JSP).
Ho potuto valutare le potenzialità di NetBeans IDE, uno dei più importanti ambienti di sviluppo Java oggi disponibili e ho familiarizzato con l'uso e la configurazione di Apache Tomcat, il container JSP di riferimento. La stessa redazione di questo documento mi ha consentito di valutare e di impiegare alcune delle competenze acquisite durante il mio percorso di studi.

## *Possibili sviluppi futuri*

Data la varietà degli elementi integrabili in una pagina web, è stato necessario effettuare delle scelte, nel tentativo di fornire agli utenti gli strumenti effettivamente indispensabili per la realizzazione di un buon sito.

Internet è una realtà in continua evoluzione, di conseguenza, le tecnologie e le soluzioni messe a disposizione dalla rete sono in costante crescita.

Per questo motivo, sarà necessario aggiornare l'applicazione sviluppata, al fine di integrare le nuove funzionalità offerte. La struttura modulare della programmazione ad oggetti Java e l'organizzazione dei package impiegati nel progetto, consentiranno un'agevole upgrade del software.

Tra le varie possibilità di sviluppo, le seguenti risultano essere, attualmente, le più utili:

*Potenziamento della gestione delle tabelle:*

- modifica della gestione delle singole celle (dimensione delle singole celle ed inserimento di più elementi in una cella)
- possibilità di inserire una nuova tabella all'interno di una cella
- modifica del contenuto delle singole celle di tabelle esistenti

*Inserimento di link dinamici:*
Possibilità, da parte dell'utente, di creare e gestire collegamenti a risorse dinamiche, quali database, script, webmail.

*Creazione di nuovi elementi template:*
Integrazione di nuovi strumenti di autocomposizione, quali gallerie fotografiche, moduli di gestione di cataloghi e listini, file server.

*Pubblicazione file:*
Estensione del tipo e della gestione dei file pubblicabili, mediante modifica del *parse* dei file *multipart*, inviati da form HTML.

# Bibliografia

### Documentazione online

1. Sito ufficiale Java, di Sun Microsystem:
   http://java.sun.com

2. Sito ufficiale di NetBeans:
   http://www.netbeans.org

3. Sito ufficiale del progetto Jakarta:
   http://jakarta.apache.org

4. World Wide Web Consortium:
   http://www.w3.org

### Testi di riferimento

1. Titolo: Progetto di Basi di Dati Relazionali
   Autori: Domenico Beneventano, Sonia Bergamaschi, Maurizio Vincini
   Casa editrice: Pitagora

2. Titolo: JSP, La guida completa
   Autore: Phil Hanna
   Casa editrice: McGraw Hill

3. Titolo: Java 2, La guida completa, quinta ed.
   Autore: Herbert Schildt
   Casa editrice: McGraw Hill

4. Titolo: HTML 4, Tutto & Oltre
   Autore: Rick Darnell
   Casa editrice: Apogeo

5. Titolo: JavaScript, La guida
   Autore: D. Goodman
   Casa editrice: McGraw Hill

6. Titolo: UML Distilled, prima ed. italiana
   Autori: Martin Fowler e Kendall Scott
   Casa editrice: Addison – Wesley

# Glossario

Al fine di semplificare la lettura di questo documento, seguirà una breve spiegazione dei termini tecnici menzionati durante la trattazione:

***Absolute path:*** indirizzo completo di un file, a partire dalla root directory

***Account:*** iscrizione registrata su un server che, tramite inserimento di username e password, consente l'accesso ai servizi

***ANSI:*** serie di 256 caratteri. Standard definito da ANSI, che consente di scambiare dati tra computer con diverso sistema operativo o tra diversi programmi.

***ASCII (American Standard Code for Information Interchange):*** L'insieme dei codici binari che vengono attribuiti a ciascun carattere, compresi alcuni codici speciali, affinché i dati possano essere scambiati tra computer con diverso sistema operativo o tra diversi programmi.

***Autenticazione:*** Procedura che, tramite la verifica di username e password, consente l'accesso al servizio solamente al possessore dei dati di un account attivo.

***Beans:*** componenti riutilizzabili per comporre applicazioni Java, applet e servlet

***Blog (weB LOG):*** pagina personale su un sito pubblico, che elenca cronologicamente una serie di interventi pubblicati dall'autore

***Browser:*** software che consente la visualizzazione delle pagine in Internet. Può essere utilizzato anche per la visualizzazione di documenti locali

***Cookie:*** file che viene creato durante una sessione di collegamento in Internet e che viene memorizzato sull'hard disk dell'utente. Tipicamente, il cookie è un file di dati che contiene informazioni per il collegamento ad un determinato sito.

***Dead link:*** link che porta ad una pagina inesistente

***Directory:*** una zona del disco rigido, o di un'altra unità di memoria di massa, contenente file e subdirectory

***DNS Server:*** server Internet che gestisce le richieste di URL da parte degli utenti. Effettua la trasformazione dell'URL alfanumerico mnemonico nel corrispondente indirizzo IP.

***Field:*** in un database, indica una serie omogenea di dati con valore differente per ogni record.

***File server:*** server dedicato a contenere i file personali appartenenti agli utenti.

***File system:*** la parte del sistema operativo che si occupa della gestione dei file.

***Font:*** disegno grafico di lettere, numeri e simboli tipografici. Ogni font può assumere diversi stili (normale, corsivo, grassetto e corsivo grassetto).

***Form:*** modulo HTML, contenente un insieme di campi compilabili dall'utente. Viene solitamente usato per la trasmissione di informazioni.

***Frame:*** ciascuna delle parti di una pagina Internet di tipo frameset, che ospita un documento o un file.

***Frameset:*** pagina Internet composta da diversi documenti.

***Framework:*** la struttura software nell'ambito della quale un'applicazione viene eseguita.

***GIF:*** Standard per la grafica, messo a punto da CompuServe, per archiviare immagini in modo compresso.

***Header:*** intestazione contenente le proprietà di un file.

***Home page:*** pagina principale di un sito

***Hosting:*** disponibilità di spazio (gratuito o a pagamento) su un server Internet, per la pubblicazione di file personali o per la realizzazione di un sito.

***HTML (HiperText Markup Language):*** linguaggio utilizzato per la composizione di pagine Internet.

***HTTP:*** Protocollo per il trasferimento dati in una rete TCP/IP

***Java:*** linguaggio di programmazione ad oggetti, sviluppato da Sun Microsystem

***JavaScript:*** linguaggio di programmazione sviluppato da Netscape, che consente l'esecuzione di procedure interattive all'interno di documenti HTML

***JPEG:*** standard per la compressione di immagini, a fattore di qualità variabile.

***JSP:*** pagine contenenti script Java di tipo server-side

***Libreria:*** insieme di componenti software, utilizzabili all'interno di un'applicazione

***Login:*** connessione al server, mediante digitazione di username e password

***MIME (Multipurpose Internet Mail Extension):*** estensione del normale protocollo di posta elettronica (SMTP). Consente l'invio di messaggi contenenti HTML, audio, video, elementi multimediali e garantisce la compatibilità con i set di caratteri orientali.

***Package:*** insieme di classi Java, raccolte in una libreria.

***Path:*** indirizzo che identifica la posizione di un file all'interno della memoria di massa.

***Plug-in:*** software accessorio che aggiunge determinate funzionalità ai programmi installati (come programmi grafici o browser).

***Protocollo:*** insieme di regole condivise da un insieme eterogeneo di entità che comunicano per raggiungere un fine comune

***Query:*** interrogazione per la ricerca, selezione ed estrazione di dati specifici da un database

***Record:*** in un database, è l'insieme dei dati riferiti allo stesso oggetto

***Server:*** computer che fornisce un insieme di servizi ad altri computer (client) ad esso collegati.

***Servlet:*** programmi Java che vengono eseguiti sui server.

***SQL (Structured Query Language):*** linguaggio standard di interrogazione dei database, che permette la consultazione (e la modifica) dei dati, indipendentemente dall'ambiente di creazione.

***Stand alone:*** Software che può essere eseguito in modo indipendente da qualsiasi altro.

***Tag:*** comandi di formattazione, delimitati da caratteri convenzionali che consentono la loro identificazione all'interno del testo.

***Timeout:*** intervallo di tempo oltre il quale il tentativo di connessione, o di lettura dati, viene annullato.

***UML:*** insieme di metodologie di analisi e progettazione orientate agli oggetti, apparse tra la fine degli anni '80 e i primi anni '90.

***Upload:*** trasferimento di un file locale al file system di un server (o di un host generico)

***URL (Uniform Resource Locator):*** indirizzo (completo) di una risorsa su Internet, in formato alfabetico mnemonico.

***Username:*** parola che identifica un determinato utente all'interno di una rete.