

Nota Tecnica



**Modulo di gestione dei clienti**,

**modulo di creazione di un nuovo progetto**

**e note riguardo al modulo tema KMS.** 3

# 

Introduzione:

Il presente documento si pone come scopo quello di esporre le scelte implementative e le funzionalità sviluppate relativamente al modulo di gestione dei clienti.

La creazione del modulo è stata suddivisa in varie fasi:

Creazione del modulo

E accenni al funzionamento di Alfresco

Bisogna innanzitutto rispondere ai prerequisiti descritti in <http://docs.alfresco.com/community/concepts/alfresco-sdk-installing-prerequisite-software.html> .

per la creazione dell’archetipo di progetto si rimanda alla documentazione reperibile in <http://docs.alfresco.com/community/concepts/alfresco-sdk-cmd-reference.html>.

e dopo di che importarlo su un IDE Eclipse o IntellijIdea seguendo una delle seguenti guide linkate: Per IntellijIdea:<http://docs.alfresco.com/community/tasks/alfresco-sdk-rad-intellij-import-projects.html>

Per Eclipse: [http://docs.alfresco.com/community/tasks/alfresco-sdk-rad-eclipse-import-projects.html](http://docs.alfresco.com/community/tasks/alfresco-sdk-rad-eclipse-import-projects.htmlb)

Le cartelle su cui andremo a sviluppare saranno la repo-amp e la share-amp. La share-amp verrà utilizzata per sviluppare l’interfaccia grafica del modulo mentre la parte di repo-amp per lo sviluppo del “back-end”, le altre folder create servono per replicare l’ambiente di alfresco in fase di esecuzione del modulo, e quindi non ha senso lavorare su di esse.

è necessario installare il driver jdbc se si intende lavorare nell’SDK, seguire quanto detto in <https://community.alfresco.com/thread/213317-sdk-21-including-third-party-jar-libraries> .

Una volta fatto questo, bisogna aprire nell’IDE o anche a mano all’indirizzo, partendo dalla cartella di installazione (sono riportati gli indirizzi utilizzando i nomi dell’esempio: \alfresco-extension\acme-cms-poc\acme-cms-poc-share-amp\src\main\amp\config\alfresco\web-extension\site-webscripts\com\example\pages

(questo porta a dove è situata la pagina dimostrativa). È sufficiente fermarsi due o tre livelli sopra se l’intenzione è quella di mettere la gerarchia per una pagina custom (come fatto in questo progetto).

Una pagina fatta in alfresco share è composta da tre file, per semplicità prendiamo in considerazione la pagina di esempio già fornita:

un file .xml, che descrive i dati essenziali della pagina:

<webscript>

<shortname>Simple Page</shortname>

<description>Simple page definition</description>

<family>Share</family>

<authentication>user</authentication>

<url>/simple-page</url>

</webscript>

molto importanti sono il tag authentication, che specifica il livello di autenticazione necessario per accedere alla pagina, e il tag url, dal momento che definisce l’indirizzo della pagina, che sarà <ip>:port/share/page/hdp/ws<url> (hdp genera la pagina con header e footer, dp la pagina grezza).

Vi è poi il file .html che si comporta proprio come il body di una pagina html.

Vi è infine un file .js, che serve per applicare delle modifiche via json al model.

Quanto detto è sufficiente per creare una pagina che non necessiti di interazioni con java.

Per quanto riguarda una pagina che invece fa uso delle api di Alfresco il procedimento è più complicato: la pagina che verrà mostrata all’utente è collocata nella cartella alfresco-extension\acme-cms-poc\acme-cms-poc-repo\amp\src\main\amp\config\alfresco\extension\templates\webscripts.

La pagina generata è situata all’indirizzo <ip>:<porta>/alfresco/service<url>.

E’ inoltre necessario creare un bean che fa da collegamento tra il codice java, situato in alfresco-extension\acme-cms-poc\acme-cms-poc-repo-amp\src\main\java\<package>, e la pagina che verrà generata. Per farlo seguire quanto descritto in <http://docs.alfresco.com/community/tasks/bean-config.html>.

Manutenzione moduli

Nel momento in cui si vorrà ad andare a modificare un modulo per evenienze riguardanti aspetti grafici o logici indiretti, bisognerà aprire il modulo con un IDE a piacere, e applicare le modifiche desiderate sui file. Il risultato delle modifiche effettuate è visionabile lanciando il comando ./run.sh una volta posizionti all’interno della folder del modulo su linux, run.bat su windows, in alternativa al comando mvn install -Prun o mvn clean install -Prun, se è necessario anche svuotare la cache; avviato il server basterà

aprire il browser e digitare l’url : <ip>:8080/share/page

Per quanto riguarda l’importazione dei moduli nell’ambiente di alfresco vero e proprio, si dovrà lanciare il comando mvn clean install all’interno sempre del folder del modulo e all’interno della cartella target delle rispettive cartelle repo-amp e share-amp si creeranno un repo-amp file e uno share-amp file. Questi dovranno essere inseriti rispettivamente nelle cartelle amps e amps\_share nell’ambiente dell’alfresco di produzione.

Per installare il modulo è necessario lanciare il file alfresco-community\bin\apply\_amps.bat, oppure lanciandolo tramite il flag -force se si ha già un modulo con lo stesso nome che si vuole sovrascrivere.

***Attenzione***

L’ installazione dei moduli comporta un deploy che sovrascriverà il contenuto delle cartelle

tomcat/webapps/share e tomcat/webapps/alfresco. Di conseguenza nel momento in cui si è andati ad effettuare delle estensioni direttamente all’interno di queste cartelle il contenuto andrà perso. Quindi prima di effettuare questa operazione è necessario fare il backup delle cartelle indicate precedentemente.

Per comodità d’ora in poi il nome del progetto sarà indicato come AIOProject.

Per ulteriori dettagli implementativi si rimanda alla visione del codice, data la grande mole di codice.

Preparazione del DB

Per preparare il db bisogna innanzitutto creare un nuovo database.

Nel caso del progetto da me sviluppato si è usato PostgreSQL

Basta quindi spostarsi nella cartella di postgresql tramite dos e digitare il comando

createdb -h localhost -p <porta> -U postgres <nome del db>

Bisogna quindi creare una tabella, che verrà chiamata per comodità da adesso clienti.

La tabella ha la seguente struttura

CREATE TABLE clienti

(

identificativo text NOT NULL,

name text NOT NULL,

piva text NOT NULL,

descr text,

datainizio text NOT NULL,

datafine text,

id bigserial NOT NULL,

CONSTRAINT clienti\_pkey PRIMARY KEY (id)

CONSTRAINT clienti\_identificativo\_name\_piva\_descr\_datainizio\_datafine\_key UNIQUE (identificativo, name, piva, descr, datainizio, datafine)

)

ATTENZIONE! È IMPORTANTE CHE L’OWNER DELLA TABELLA SIA QUELLO SETTATO NELLE PROPERTIES!

L’ultimo è necessario per non avere record che differiscano solo per id.

In particolare si è scelto di tenere le date in formato testuale per rendere il db compatibile a diverse implementazioni sulla rappresentazione della data, e si è scelto di usare come chiave primaria un id chiamato bigserial, in quanto più resistente ad una eventuale estensione dei campi della tabella e più comodo da utilizzare, dal momento che, per le caratteristiche del bigserial, si è certi che l’ultima versione di un record è quella con l’id più alto, quindi per prendere l’ultima versione di un cliente di un derterminato identificativo basta eseguire la query select \* from clienti where id = (select max(id) from clienti where identificativo='param1'). Si è cercato di non utilizzare viste o altri strumenti particolari per evitare di essere troppo dipendenti dalla piattaforma.

Modulo clienti

Creazione della parte share

È quindi stato necessario creare le pagine share e le pagine repo, oltre ai bean di collegamento per poter rendere accessibile il webscript. Si è adottato il metodo POST in tutti quante le invocazioni, in quanto è più sicuro e permette lo scambio di una maggiore quantità di dati.

SI è aggiunta innanzitutto una nuova voce di menu aggiungendo il file **/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-data/extensions/add-create-menuitem-doclib-extension-modules.xml**

<extension>

<modules>

<!-- This module is dependent on the custom content model setup in the repo-amp module -->

<module>

<id>Add a new menu item to Create... menu in DocLib</id>

<version>1.0</version>

<auto-deploy>true</auto-deploy>

<configurations>

<config evaluator="string-compare" condition="DocumentLibrary">

<create-content>

<content id="text-label-clienti" label="Crea nuovo cliente" icon="text" type="pagelink">

<param name="page">hdp/ws/clienti</param>

</content>

</create-content>

</config>

</configurations>

</module>

</modules>

</extension>

Per ulteriore documentazione si faccia riferimento a <http://docs.alfresco.com/5.0/tasks/dev-extensions-share-tutorials-add-menuitem-create-menu.html>

Le pagine includono inoltre nell’HTML la richiesta di JQuery e Bootstrap, mediante le linee

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

<!-- Optional theme -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp" crossorigin="anonymous">

<!-- Latest compiled and minified JavaScript -->

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>

Pagina Clienti

È la pagina di inserimento di un nuovo cliente ed è composta da un semplice form.

È composta dai seguenti file:

* **/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/clienti/pages/clienti.get.desc.xml**, implementato come già detto
* **/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/clienti/pages/clienti.get.html.ftl** che contiene il codice html necessario a generare il form
* **/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/clienti/pages/clienti.get.js** che contiene le istruzioni necessarie a generare l’header. È necessario includere nell’html la linea <@processJsonModel group="share"/>.
* **/AIOProject-share-amp/src/main/amp/web/js/clienti/clienti.js**, incluso con la linea <script src="${url.context}/js/clienti/clienti.js"></script>, che effettua i controlli javascript della pagina. In particolare alla pressione del pulsante di conferma viene fatto il controllo sui dati inseriti, cioè sulla correttezza della partita iva, sul fatto che i campi obbligatori siano definiti, sul fatto che la data di fine sia successiva alla data di inizio. Inoltre si occupa di inoltrare la richiesta AJAX al webscript query\_processor.java che sarà trattato successivamente. Se il webscript riporta uno stato anomalo viene segnalato e così viene anche gestito il caso di inserimento di un identificativo già esistente.

*Pagina vis\_clienti*

La pagina vis\_clienti è la pagina di gestione dei clienti, ed è suddivisa nei file:

* **/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/visualizza\_clienti/pages/vis\_clienti.get.desc.xml** che funziona come descritto prima.
* **/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/visualizza\_clienti/pages/vis\_clienti.get.html.ftl** che fornisce lo scheletro della pagina, che verrà poi popolata tramite richieste AJAX delle parti dinamiche.
* **/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/visualizza\_clienti/pages/vis\_clienti.get.js**, che funziona come descritto prima.
* **/AIOProject-share-amp/src/main/amp/web/css/vis\_clienti/vis\_clienti.css**, incluso in maniera analoga al javascript come descritto prima e che definisce il CSS della pagina.
* **/AIOProject-share-amp/src/main/amp/web/js/vis\_clienti/vis\_clienti.js**, incluso come descritto prima

In particolare la pagina nella sua parte principale offre la lista degli identificativi dei clienti attivi e non, che è anche il nome con cui sono stati salvati nella cartellatura di Alfresco. Si è scelto questa rappresentazione in quanto è logicamente vicina alla struttura di Alfresco e sarà più facile in futuro implementare funzionalità quali la visualizzazione dei progetti del cliente. I pulsanti dei clienti attivi e inattivi sono generati rispettivamente da clients.java e inactive.java chiamati all’avvio della pagina tramite window.onload che si occupando di definire anche i giusti parametri che verranno poi utilizzati per popolare il form che consente l’aggiornamento dei dati di un particolare cliente. Form già presente nella pagina ma che viene fatto mostrare da una opportuna funzione javascript che si occupa anche di popolare la tabella sottostante tramite una chiamata AJAX al webscript TablePageGenerator.java, che si occupa di fornire la tabella e di iniettare il pulsante cancella e il pulsante per definire la data di fine del rapporto con il corretto id. I pulsanti sono collegati a loro volta a delete.java e update.java, chiamati tramite opportune chiamate AJAX.

I messaggi di errore e di conferma sono gestiti dove possibile in JavaScript e dove non possibile, poiché la risposta dipende dal database o dal repository, tramite la risposta della chiamata AJAX

Per riassumere la pagina fa uso delle seguenti funzioni JavaScript:

* set\_table(identificativo) che si occupa di generare la tabella di un determinato cliente tramite richiesta AJAX
* detail(id,nome,pi,descr,datainizio,datafine) che si occupa di settare

E mostrare il dettaglio di un cliente

* controllaPIVA(pi) che esegue semplicemente il controllo della partita IVA
* sendRequest() Che invia la richiesta AJAX di aggiornamento di un determinato cliente
* set\_data\_fine(ID) che setta la data di fine di un determinato record di un cliente tramite richiesta AJAX
* cancella(key) che fa la richiesta tramite AJAX di cancellazione di un record con un determinato ID
* up() che popola la pagina con i pulsanti dei clienti, tramite richiesta AJAX.
* vai(name) che reindirizza alla cartella con dato nome.

Sono presenti altre funzioni che svolgono compiti banali quale la codifica e decodifica di entità, per garantire il funzionamento della pagina nel caso i record contengano caratteri speciali

Creazione della parte repo

Creazione delle variabili del modulo

Per rendere possibile la modifica di alcune configurazioni anche dopo che il modulo è stato installato in alfresco, si è provveduto a creare un file in **/AIOProject-repo-amp/src/main/amp/config/alfresco/module/AIOProject-repo-amp/config/configModule.properties,** Accessibile da java aggiungendo al codice le linee:

private static Properties properties=new Properties();

…

public static final String getValue(String value) {

return properties.getProperty(value);

}

….

properties.load(getClass().getResourceAsStream("/alfresco/module/AIOProject-repo-amp/config/configModule.properties"));

Questo permette quindi di recuperare le variabili, che si è cercato di documentare direttamente sul codice tramite nomi significativi e una breve descrizione:

#query configuration

queryselection=select id,name, piva, datainizio, datafine, descr from clienti where identificativo='param1' order by id

querylastversion=select \* from clienti where id = (select max(id) from clienti where identificativo='param1')

queryselectactive=select distinct identificativo from clienti where NOT (datafine IS NOT NULL)

queryselectnotactive=select distinct identificativo from clienti where datafine IS NOT NULL and identificativo not in (select distinct identificativo from clienti where NOT (datafine IS NOT NULL))

#do not use values in the form paramX! They may be sobstituted!

queryselectionwithclause=select param1 from param2 where param3

queryinsertion=insert into param1 values ('param2','param3','param4', 'param5', 'param6', 'param7')

queryupdate=UPDATE param1 SET param2 = 'param3' WHERE id = 'param4';

querydelete=DELETE FROM param1 WHERE id = 'param2';

#set param1 in the previous query

querytable=Clienti

#server configuration

serverip=127.0.0.1

serverid=alfresco

serverpassword=admin

serverport=5432

servername=<nome del database creato>

servertype=postgresql

serverconnector=jdbc

#path in lucene, in the form of PATH:\"/{directory}, the "\"" character is needed

#because without it it will be escaped by java, java alone puts the final " so

#you must not put it when you write the query

installationpath=PATH:\"/app:company\_home/st:sites/cm:er/cm:documentLibrary/cm:\_x0030\_2\_x0020\_-\_x0020\_Clients

#down here you can put the names for the folder that are in the client subdirectory

folder.one=01 - Projects

folder.two=02 - General Contracts

folder.three=tre

folder.four=quattro

folder.five=cinque

Come si vede dove necessario si è cercato di parametrizzare le funzioni.

Creazione dei webscript

Come già accennato, le pagine in share per il loro completo funzionamento necessitano di webscript di supporto, per interrogare il database e generare le tabelle contenenti i dati. Tutti quanti estendono la classe AbstractWebScript, sono contattabili tramite POST e fanno uso del JDBC per contattare il server remoto dove è presente la tabella. Il file .properties descritto prima permette di cambiare secondo le necessità la configurazione della connessione, definita nella riga

connection=DriverManager.getConnection((connector+":"+type+"://"+ip+":"+port+"/"+server),id,password);

Che sono i parametri definiti nel paragrafo query configuration del .properties

I WebScript sono poi registrati nel file **/AIOProject-repo-amp/src/main/amp/config/alfresco/module/AIOProject-repo-amp/context/webscript-context.xml** secondo le modalità già descritte.

Per comodità nella chiamata l’indirizzo è stato settato con lo stesso nome della funzione java ad esso corrispondente. Inoltre alcuni accettano parametri in JSon definiti in coppie {key:value} in un JSonObject

Passiamo ora in rassegna i vari webscript implementati, in ordine alfabetico.

# clients.java

Questo webscript, situato in **/AIOProject-repo-amp/src/main/java/clients.java** e dalla pagina definita in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/clients.post.desc.xml** e in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/clients.post.html.ftl** si occupa di generare i pulsanti necessari a visualizzare poi la pagina di dettaglio per un utente attivo. Si occupa quindi di interrogare il database e di definire di conseguenza i parametri dei pulsanti e il loro nome, che chiamano uno la funzione detail(id,nome,pi,descr,datainizio,datafine),e l’altro la funzione vai(name). Si appoggia sulla query definita in queryselectactive per selezionare gli utenti attivi, percorrendo poi il result set per di volta in volta assegnare i parametri.

Viene chiamato senza parametri.

La classe si compone delle seguenti funzioni:

* getValue(String value), che recupera una property con una data value
* inizialize\_values (), che inizializza le variabili utilizzate, recuperandole dalle properties.
* public void execute(WebScriptRequest req, WebScriptResponse res) metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* execute(Connection con, String query, WebScriptResponse res), metodo che esegue la query.
* writedata(ResultSet rs, String result, Connection con), metodo che si occupa di ritornare il codice html necessario a generare i pulsanti dei clienti attivi

# delete.java

Questo webscript, situato in **/AIOProject-repo-amp/src/main/java/delete.java** e dalla pagina definita in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/delete.post.desc.xml** e in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/delete.post.html.ftl** si occupa semplicemente di cancellare un elemento che abbia un determinato id, fornito come parametro della chiamat. Si appoggia sulla query definita alla voce querydelete.

La classe si compone delle seguenti funzioni:

* getValue(String value), che recupera una property con una data value
* inizialize\_values (), che inizializza le variabili utilizzate, recuperandole dalle properties.
* public void execute(WebScriptRequest req, WebScriptResponse res) metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* execute(Connection con, String query), metodo che esegue la query di delete di un dato record del database.

# inactive.java

Questo webscript, situato in **/AIOProject-repo-amp/src/main/java/inactive.java** e dalla pagina definita in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/inactive.post.desc.xml** e in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/inactive.post.html.ftl** si occupa di generare i pulsanti necessari a visualizzare poi la pagina di dettaglio per un utente non più attivo. Si occupa quindi di interrogare il database e di definire di conseguenza i parametri del pulsante e il suo nome, che chiama la funzione detail(id,nome,pi,descr,datainizio,datafine). Si appoggia sulla query definita in queryselectnotactive per selezionare gli utenti non attivi. Percorrendo poi il result set per di volta in volta assegnare i parametri.

Viene chiamato senza parametri.

La classe si compone delle seguenti funzioni:

* getValue(String value), che recupera una property con una data value
* inizialize\_values (), che inizializza le variabili utilizzate, recuperandole dalle properties.
* public void execute(WebScriptRequest req, WebScriptResponse res) metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* execute(Connection con, String query, WebScriptResponse res), metodo che esegue la query.
* writedata(ResultSet rs, String result, Connection con), metodo che si occupa di ritornare il codice html necessario a generare i pulsanti dei clienti non attivi.

Si è deciso di renderla autonoma rispetto a clients per consentire una migliore estensione ad ulteriori eventuali funzionalità.

# query\_processor.java

Questo webscript, situato in **/AIOProject-repo-amp/src/main/java/query\_processor.java** e dalla pagina definita in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/query\_processor.post.desc.xml** e in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/query\_processor.post.html.ftl .** Si occupa di inserire un oggetto nel database, utilizzando la query definita in queryinsertion e deve essere chiamato con i seguenti parametri:

* ID:l’identificativo del cliente,
* Nome:il nome esteso del cliente,
* Piva:la partita iva del cliente,
* Descr:una descrizione generica,
* Datainizio:la data di inizio del rapporto,
* Datafine:la data di fine del rapporto(se non presente, verrà settata a null),
* Mode:1=inserimento con la creazione delle cartelle, 2=inserimento senza creazione di cartelle ,
* Box1:creazione della cartella di nome definito in folder.one ,
* box2:creazione della cartella di nome definito in folder.two ,
* box3:creazione della cartella di nome definito in folder.three ,
* box4:creazione della cartella di nome definito in folder.four ,
* box5:creazione della cartella di nome definito in folder.five ,

Lo script si appoggia, oltre al JDBC per l’inserimento nel database, sulla API di Alfresco

FileFolderService per creare la gerarchia di cartelle corrispondente, di permissionservice per settare i permessi di accesso alle cartelle (per il momento codificati nel Java e non configurabili dopo l’installazione se non mediante l’editor dei permessi di Alfresco)

E di nodeservice per settare la descrizione della cartella principale.

Per includere il serviceregisty, necessario per accedere alla API di Alfresco, è necessario, oltre a specificare nel bean, il cui funzionamento è stato già descritto, tramite le righe

<property name="serviceRegistry">

<ref bean="ServiceRegistry" />

</property>

E poi nel codice java con l’aggiunta di

import org.alfresco.service.ServiceRegistry;

….

private ServiceRegistry serviceRegistry;

public void setServiceRegistry(ServiceRegistry serviceRegistry) {

this.serviceRegistry = serviceRegistry;

}

Ritorna una stringa di risultato dove è specificato l’esito dell’operazione.

La classe si compone delle seguenti funzioni:

* setServiceRegistry(ServiceRegistry serviceRegistry), che inizializza il parametro per il service registry di Alfresco. È richiesto obbligatoriamente da Alfresco se si vogliono utilizzare le sue funzionalità
* getValue(String value), che recupera una property con una data value
* inizialize\_values (), che inizializza le variabili utilizzate, recuperandole dalle properties.
* public void execute(WebScriptRequest req, WebScriptResponse res) metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* execute(Connection con, String query), metodo che esegue la query.
* create\_folder\_tree(String ID, String descr,Boolean box1, Boolean box2, Boolean box3, Boolean box4, Boolean box5), metodo che si occupa di creare la cartellatura di un nuovo cliente e di settare i permessi alle cartelle e la descrizione della cartella padre.

# TablePageGenerator.java

Questo webscript, situato in **/AIOProject-repo-amp/src/main/java/TablePageGenerator.java** e dalla pagina definita in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/TablePageGenerator.post.desc.xml** e in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/TablePageGenerator.post.html.ftl** si occupa di generare la tabella dello storico di un determinato codice cliente, fornito come parametro della chiamata AJAX. Si appoggia sulla query definita alla voce queryselection.

Il webscript si occupa di generare il codice della tabella e di iniettare i giusti parametri nel pulsante per cancellare, e, nel caso la data di fine non sia stata settata, anche di generare il pulsante per la modifica della data di fine rapporto.

La classe si compone delle seguenti funzioni:

* getValue(String value), che recupera una property con una data value
* inizialize\_values (), che inizializza le variabili utilizzate, recuperandole dalle properties.
* public void execute(WebScriptRequest req, WebScriptResponse res) metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* execute(Connection con, String query, WebScriptResponse res), metodo che esegue la query.
* dumpData(ResultSet rs, String result), metodo che si occupa di ritornare il codice html necessario a generare la tabella dello storico di un cliente.

# update.java

Questo webscript, situato in **/AIOProject-repo-amp/src/main/java/update.java** e dalla pagina definita in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/update.post.desc.xml** e in **/AIOProject-repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts/update.post.html.ftl** si occupa semplicemente di cancellare un elemento che abbia un determinato id, un campo da aggiornare e un valore con cui aggiornarlo, forniti come parametro della chiamata. Si appoggia sulla query definita alla voce queryupdate.

La classe si compone delle seguenti funzioni:

* getValue(String value), che recupera una property con una data value
* inizialize\_values (), che inizializza le variabili utilizzate, recuperandole dalle properties.
* public void execute(WebScriptRequest req, WebScriptResponse res) metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* executeupdate(Connection con, String query), metodo che esegue la query di update della data di fine di un dato record del database.

Modulo di inserimento progetto

Questo modulo si basa fortemente sulle tecniche e le funzioni già implementate e descritte durante la descrizione del precedente modulo, tuttavia si è cercato di eseguire la maggior parte del lavoro di composizione delle pagine tramite javascript, per testare la velocità della soluzione sia in termine di prestazioni che di tempi di sviluppo. Inoltre è stata inserita una integrazione con LDAP.

Il progetto punta a sfruttare il modulo di inserimento di clienti per aggiungere nuove funzionalità.

A tale scopo, anche il modulo riguardante i clienti andrebbe aggiornato, per sfruttare tutte le funzionalità introdotte. Tuttavia il modulo clienti è completamente funzionante *stand alone*, mentre il modulo di un nuovo progetto è pensato per funzionare in abbinamento con il modulo della gestione dei clienti e necessita di esso.

Nota:in seguito si useranno per i nomi le stesse convenzioni già presentate

# Aggiunte al DB

Per il corretto funzionamento del modulo sono necessarie le seguenti tabelle aggiuntive:

CREATE TABLE progetti

(

nome text NOT NULL,

descr text,

CONSTRAINT progetti\_pkey PRIMARY KEY (nome)

)

Che è una tabella che semplicemente deve contenere la lista dei progetti e la loro descrizione (si assume che il nome di un progetto sia univoco, altrimenti basta aggiungere un id in auto\_increment alla tabella). La seguente tabella è stata pensata per garantire la possibilità di aggiunta di attributi ad un progetto, senza aggiungere complessità alla tabella di coloro che sono impiegati in un progetto.

È necessaria per poter gestire il caso di un progetto senza alcun assegnatario.

Più degna di nota è la tabella delle persone impegnate su un progetto:

CREATE TABLE lavoratori

(

mail text NOT NULL,

project text NOT NULL,

client serial NOT NULL,

descr text,

CONSTRAINT lavoratori\_pkey PRIMARY KEY (mail, project, client),

CONSTRAINT lavoratori\_client\_fkey FOREIGN KEY (client)

REFERENCES clienti (id) MATCH SIMPLE

ON UPDATE NO ACTION ON DELETE NO ACTION

)

Come si vede vi è qui diretto collegamento tra il progetto e il cliente che lo ha commissionato (si assume nell’assegnazione del progetto che la versione del cliente a cui viene assegnato un progetto sia quella dell’ultima modifica nota eseguita tramite il modulo clienti. Ciò è garantito del fatto di selezionare l’id maggiore, in quanto il tipo serial dell’id cliente non decresce alla cancellazione di un record; si ha quindi la certezza che ad un dato nome di un cliente, la sua ultima versione è quella con id maggiore.

# Definizione delle properties

Il sistema fa uso delle seguenti properties, riportate a questo punto per far capire meglio le query utilizzate:

#query configuration

queryselectallprojects=select project,mail,lavoratori.descr,identificativo from lavoratori join clienti on Client=ID order by project

queryselectallclients=select distinct identificativo from clienti

#do not use values in the form paramX! They may be sobstituted!

insertworker=insert into lavoratori values ('value1','value2',(select max(ID) from clienti where identificativo='value3'),'value4');

insertproject= insert into progetti values ('value2','value4')

#server configuration

serverip=localhost

serverid=alfresco

serverpassword=admin

serverport=5432

servername=testdb

servertype=postgresql

serverconnector=jdbc

ldap.authentication.java.naming.provider.url=ldap://192.168.55.73:389

#path in lucene, in the form of PATH:\"/{directory}, the "\"" character is needed

#because without it it will be escaped by java, java alone puts the final " so

#you must not put it when you write the query

clientinstallationpath=PATH:\"/app:company\_home/st:sites/cm:er/cm:documentLibrary/cm:\_x0030\_2\_x0020\_-\_x0020\_Clients/cm:put\_name\_here/cm:\_x0030\_1\_x0020\_-\_x0020\_Projects

#down here you can put the names for the folder that are in the client subdirectory

folder.father.project.one=01 - Technical Documentation

folder.father.project.two=02 - Management Documentation

folder.child.one.one=01 - Analysis

folder.child.one.two=02 - Design

folder.child.one.three=03 - Communication and Marketing

folder.child.one.four=04 - Manuals and Tutorials

folder.child.one.five=05 - Reports, Records, Meeting Minutes

folder.child.two.one=01 - Project Management

folder.child.two.two=02 - Financial

folder.child.two.two.one=01 - Proposals

folder.child.two.two.two=02 - Contracts

folder.child.two.two.three=03 - Accounting

folder.child.two.two.four=04 - Invoices

Come si nota sono configurabili, oltre ai parametri delle varie connessioni, anche i nomi delle cartelle e le query.

Parte share

La parte share si basa su una struttura simile a quella del modulo clienti. Per ragioni di tempo, non si è potuto sviluppare un modify per i progetti e la pagina di visualizzazione dei progetti è quindi un semplice dump dei dati. La struttura è stata tuttavia predisposta all’aggiunta di funzionalità, soprattutto quella relativa alla creazione di una mailing list relativa al progetto, non implementata per ragioni di tempo.

Pagina di visualizzazione dei progetti.

Vista la struttura della tabella dei lavoratori, la pagina si occupa di mostrare una riorganizzazione dei dati di quella tabella per dare una migliore leggibilità.

È composta prevalentemente attraverso l’iniezione di codice HTML da parte di funzioni JavaScript che manipolano i risultati del webservice che viene chiamato tramite AJAX.

La parte statica è definita nei file

**/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/dipendenti/pages/dipendenti.get.desc.xml**

**/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/dipendenti/pages/dipendenti.get.html.ftl**

**/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/dipendenti/pages/dipendenti.get.js**

Il CSS è invece situato nel file **/AIOProject-share-amp/src/main/amp/web/css/workers/workers.css** e il JavaScript nel file **/AIOProject-share-amp/src/main/amp/web/js/workers/workers.js**. In particolare sono state implementate le seguenti funzioni, oltre a quelle di escape/encoding/decoding di stringhe:

* **setup()**, chiamata al window.onload, che si occupa di fare la chiamata AJAX per ottenere il contenuto della tabella progetti, che è una stringa nella quale il contenuto delle celle è separato dai caratteri “###,” mentre le righe sono separate tramite i caratteri “###;” la stringa verrà poi trasformata in codice HTML tramite la funzione
* **parse(response)** che si occupa di ritrasformare in un array di array la stringa passata dalla funzione di setup, e di tradurre i dati contenuti in codice HTML più leggibile e chiaro.

Pagina di inserimento di un progetto

La pagina si occupa di inserire i dati relativi al progetto, a coloro ai quali è assegnato e a creare la cartellatura desiderata assegnando i permessi di visione alla sola cartella tecnica a coloro ai quali è stata assegnato il progetto.

La pagina che si occupa di inserire un progetto e dove è possibile assegnare utenti (viene fornita una lista di quelli presenti nell’LDAP) ad un determinato progetto è definita nella sua parte statica dai file

**/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/progetti/pages/progetti.get.desc.xml**

**/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/progetti/pages/progetti.get.html.ftl**

**/AIOProject-share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/progetti/pages/progetti.get.js**

Il CSS è invece situato nel file **/AIOProject-share-amp/src/main/amp/web/css/progetti/progetti.css** e il JavaScript nel file **/AIOProject-share-amp/src/main/amp/web/js/progetti/progetti.js**.

Il JavaScript utilizza le variabili globali buttons\_unassigned buttons\_assigned assigned

Per tenere traccia di quali bottoni sono stati premuti e quindi anche di coloro che sono stati assegnati al progetto. Implementa, oltre alle funzioni base di manipolazione delle stringhe, anche le seguenti funzioni:

* **sendRequest()** che fa il submit del form al webscript project\_creator che si occupa di eseguire il lato backend dell’inserimento.
* **setup()**, che si occupa di fare le richieste necessarie ad ottenere le informazione necessarie quali la lista dei clienti e la lista degli utenti registrati nell’LDAP.
* **process\_response(res)**, che si occupa di processare la lista degli utenti LDAP fornita dalla parte repo e di trasformarla in button html e di settarne l’action onclick.
* **refresh()**, che banalmente fa il refresh dei pulsanti mostrati nell’area degli utenti disponibili e in quella degli assegnati.
* **assign(mail)**, che assegna un utente alla lista degli utenti a cui è stato assegnato il progetto, si occupa di ricreare il codice html dei bottoni e di invocare la funzione refresh(). Implementa anche un sistema per controllare che l’utente che si tenta di assegnare non sia già stato assegnato, poichè testando l’applicativo è capitato di notare che era possibile riuscire ad essere abbastanza rapidi da premere il pulsante di un utente 2 volte prima che la funzione refresh() lo togliesse.
* **unassign(mail)**, che si occupa di togliere un utente dalla lista di coloro ai quali è stato assegnato un progetto e di rigenerare il codice html dei pulsanti e di rigenerarli invocando la funzione refresh()
* **ControllerTD()**,**ControllerMD()**, **ControllerF()**, che banalmente si occupano del check e uncheck e disabilitazione dei figli al cambiamento di uno dei padri nelle checkbox.
* **aggiungi()**, che si occupa di gestire il popup per il pussante di aggiunta di un nuovo utente assegnatario del progetto, invocando poi la funzione assign(mail) per generarne il codice HTML corrispondente al relativo pulsante

Parte Repo

In questo modulo, rispetto a quello precedente descritto e come già accennato, si è cercato di ridurre il carico nella parte repo per spostare le computazioni relative alla generazione del codice nel JavaScript.

Tuttavia è stata necessaria l’implementazione di quattro classi Java:

**NB**:ricordarsi, se si utilizza la response per ritornare codice HTML o stringhe qualsiasi, di includere il seguente codice:

res.setContentType("text/html; charset=UTF-8");

res.setContentEncoding("UTF-8");

Altrimenti la codifica dei caratteri non riesce in maniera corretta.

### Clienti.java

Questa pagina è chiamata senza parametri tramite richiesta POST al webscript “clienti” dalla funzione “setup()” della pagina “progetti”, e si occupa di ritornare il codice HTML necessario a generare un select con i clienti, recuperati dal database, tra le varie option.

Si compone delle seguenti funzioni:

* **getValue(String value)**, che recupera una property con una data value
* **inizialize\_values ()**, che inizializza le variabili utilizzate, recuperandole dalle properties.
* **public void execute(WebScriptRequest req, WebScriptResponse res)** metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* **execute(Connection con, String query, WebScriptResponse res)**, metodo che esegue la query.
* **writedata(ResultSet rs, String result, Connection con)**, metodo che si occupa di ritornare il codice html necessario per generare il select e le option iterando il result set risultato dell’interrogazione del database.

### GetUsers.java

Questa pagina è chiamata senza parametri tramite richiesta POST al webscript “GetUsers” dalla funzione “setup()” della pagina “progetti”, e si occupa di interrogare l’LDAP al fine di ottenere una lista di tutti gli utenti che sono registrati nell’LDAP. La lista è ritornata tramite una stringa con tutti i record separati con una serie di caratteri “sentinella” che fanno da separatori e che poi il Javascript si occupa di rimuovere e di riorganizzare. Esso è composto dalle seguenti funzioni:

* **public static DirContext ldapContext ()** che fa da costruttore senza parametri di un ldap context. Viene seguito da
* **public static DirContext ldapContext (Hashtable <String,String>env)** che invece è il suo costruttore con parametri.
* **public static String getUsers (),** che è il metodo principale della classe e che si occupa di creare la richiesta degli utenti all’LDAP e di iterare la risposta ottenuta al fine di formare una stringa che rispetti le condizioni già citate.
* **public static String getcontextFactory()** e **public static DirContext getLdapcontext()** che semplicemente ritornano il valore degli omonimi parametri.
* **public void execute(WebScriptRequest req, WebScriptResponse res),** metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.

### GetWorkers.java

Questa pagina è chiamata senza parametri tramite richiesta POST al webscript “GetWorkers” dalla funzione “setup()” della pagina “dipendenti”. Essa si occupa di ritornare il risultato dell’interrogazione della tabella lavoratori, con il valore delle varie celle separati da un opportuno separatore e le righe separate da un diverso separatore. Ciò permette in seguito al codice JavaScript di ricomporla in un array di array di stringhe che è possibile iterare.

Esso si compone delle seguenti funzioni:

* **getValue(String value)**, che recupera una property con una data value
* **inizialize\_values ()**, che inizializza le variabili utilizzate, recuperandole dalle properties.
* **public void execute(WebScriptRequest req, WebScriptResponse res)** metodo obbligatorio che si può intendere come il metodo che viene invocato alla chiamata Ajax del webscript e che contiene le istruzioni per gestirla.
* **execute(Connection con, String query, WebScriptResponse res)**, metodo che esegue la query.
* **writedata(ResultSet rs, String result, Connection con)**, metodo che si occupa di formare la stringa di risposta che corrisponde ai criteri descritti prima.

### Project\_creator.java

Questa pagina è chiamata con svariati parametri tramite richiesta POST al webscript “project\_creator” dalla funzione “sendRequest()” della pagina “progetti”. È la classe più corposa implementata in tutto il progetto. In quanto si occupa di gestire l’inserimento di un nuovo progetto nel DB e di crearne la cartellatura corrispondente, assegnando nel mentre i permessi di accesso e modifica alla sola cartella di documentazione tecnica (e suoi figli) a coloro ai quali è stato assegnato il progetto, in addizione ai gruppi ai quali i permessi vengono invece dati di default.

Essa si compone delle seguenti funzioni:

* **private void inizialize\_values ()** che inizializza i valori dei vari parametri ottenuti dalle properties.
* **public static final String getValue(String value)** che ritorna il valore della property passata come argomento alla funzione.
* **public void give\_permissions(String[]users, String permission,NodeRef nodeRef, PermissionService permissionService)** che è usata per dare i permessi all’array di stringhe contenente la lista degli utenti a cui è stato assegnato il progetto.

**Nota:** dato che è possibile inserire utenti non LDAP e non di Alfresco, può succedere che l’utente a cui si danno i permessi non sia registrato da nessuna parte. Ciò non rompe nulla in quanto Alfresco lo rappresenta nella lista dei permessi con uno spazio bianco fino al momento in cui non viene definito un utente con quello username. In quel momento comparirà il nome. Ciò è stato verificato provandolo effettivamente nell’Alfresco di test. Quindi il seguente caso è tollerato, anche se non è consigliabile ricorrere a questa pratica.

* **public void execute(WebScriptRequest req, WebScriptResponse res),** che viene invocato alla chiamata AJAX e si occupa di invocare e gestire le varie operazioni compiute.
* **private void execute(Connection con, String query)**, che esegue l’inserimento del record nelle tabelle relative agli impiegati e ai progetti.
* **private void create\_folder\_tree(String ID, String descr, String[] workers),** che crea iterativamente l’albero della cartellatura assegnando i permessi di default e invocando give\_permissions alla creazione della cartella relativa alla documentazione tecnica.

# Note riguardo al modulo tema.

Per quanto riguarda il modulo tema, valgono le considerazioni fatte per quanto riguarda il tema Ennova non da me prodotto, in quanto la strategia adottata per crearlo lo ricalca in maniera quasi totale nella struttura. Nel modulo da me prodotto i due temi sono distribuiti assieme in un’unica AMP ed è possibile alternarli a piacimento dal menu di amministrazione. Per ottenere ciò si è dovuti però fare un refactoring di alcune parti del tema ennova e del modulo della password dimenticata, con l’unico scopo di poter spostare il css nella parte specifica del tema piuttosto che in quella relativa alla pagina stessa, non sovrascrivibile.

Ho infatti sfruttato il fatto, prima non sfruttato, che i file di css del tema vengono automaticamente caricati assieme alla pagina share, quindi definendo lo stile con classi e ID custom nei file del tema e non in file specifici per la pagina, è possibile poter cambiare il css e le immagini importate e mostrate.

ATTENZIONE:SE SI CAMBIA TEMA, GLI UTENTI CHE NON SONO AMMINISTRATORE NON VEDRANNO NELLE LORO DASHBOARD IL NUOVO TEMA FINCHÈ NON VERRÀ RIAVVIATO IL SERVER O SVUOTATA LA CACHE.

D I V I N G I N T O T H E F U T U R E

MILAN - VENICE

12, Via della Liberta’ | 30175 Venezia | Italy

VAT # 0377 3530 278 | Tel: +39 (0)41 0992180 |Fax: +39 (0)41 0992181 