ALBERTO FRANZIN
543093

UNIVERSITÀ DEGLI STUDI DI PADOVA

LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

REALIZZAZIONE DI UN WIZARD PER LA CREAZIONE DI TEMI GRAFICI PER PORTALI WEB

RELATORE: PROF. SERGIO CONGIU

RELAZIONE DEL TIROCINIO

26 Aprile 2010

Alberto Franzin: *Realizzazione di un wizard per la creazione di temi grafici per portali web,* Relazione del tirocinio, 26 Aprile 2010.

E-MAIL:
alberto.franzin@gmail.com

INTRODUZIONE

Questo documento presenta la relazione del tirocinio da me svolto presso Projectmoon System SRL, con inizio in data 15/12/2009 e termine in data 01/04/2010, e durata prevista di 250 ore.

Obiettivo del tirocinio erano l'utilizzo delle funzionalità avanzate della piattaforma Projectmoon System per la strutturazione e gestione di contenuti, e la creazione di un sistema guidato per la creazione di temi grafici.

ABSTRACT

Durante lo sviluppo di un portale per un cliente, sono state utilizzate le funzionalità avanzate messe a disposizione dalla piattaforma Projectmoon System per la creazione di contenuti strutturati. Per ogni pagina sono stati analizzati dal punto di vista semantico i vari componenti, è stata conseguentemente individuata la tipologia di contenuto più adatta a gestire ogni componente, ed è stato creato un modello in linguaggio Velocity per la visualizzazione dei contenuti.

Vengono poi trattati l'analisi e lo sviluppo di un sistema guidato (wizard) per la creazione di temi grafici per siti e portali sviluppati con la piattaforma Projectmoon System. Vengono svolte in primo luogo un'analisi funzionale, che guiderà la creazione dell'interfaccia grafica e il funzionamento del prodotto finale, e un'analisi tecnica, che spiegherà in maniera dettagliata come procedere durante lo sviluppo dei vari componenti del wizard.

Viene descritto come procedere nella creazione e nella gestione della base di dati usando gli strumenti forniti dalla piattaforma. Viene mostrato quali sono le diverse strutture di appoggio individuate, e come sono state realizzate. Viene proposto il metodo di creazione dei metodi per la gestione dei dati e dell'interfaccia grafica, usando Java e JSP.

RINGRAZIAMENTI

Ringrazio i professori Fulvio Ferroni e Roberto Guernelli dell'ITIS Planck, per avermi messo in contatto con Projectmoon. Ringrazio Projectmoon, Francesco, Pasquale, Elena, Omar, Marco e soprattutto Pier per avermi accolto e seguito passo passo nel corso del mio tirocinio. Ringrazio infine il mio relatore prof. Sergio Congiu, i docenti, i miei colleghi e tutti coloro che hanno contribuito alla mia formazione nel corso di questi anni.

INDICE

1 L'AZIENDA

INDICE

1.1 PRESENTAZIONE DELL'AZIENDA

1.1.1 Introduzione

Projectmoon SRL nasce a Lancenigo di Villorba a febbraio 2008, ed entra da subito a far parte del gruppo SMC. *http://www.projectmoon.it* *http://www.smc.it*

Sviluppa una piattaforma per la creazione di portali e social network chiamata *Projectmoon System*, da cui il motto *on net services* che ne accompagna il logo.

Altro elemento che compone il suo business è la grafica orientata alla comunicazione, quindi flyer, manifesti e tutto ciò che e' rivolto all'advertising e alla comunicazione aziendale.

Attualmente conta un amministratore e 5 dipendenti. Si avvale inoltre della collaborazione di personale SMC, sia per la parte amministrativa, sia per quanto riguarda il reparto tecnico. *Il personale*

1.1.2 Pjoon

Il principale prodotto sviluppato mediante Projectmoon System è il social network Pjoon, strumento che consente agli iscritti di creare profili personali altamente personalizzabili, così come profili per community e aziende. Community e aziende che possono così contare su una base di utenti per i loro gruppi comune, pari agli iscritti a Pjoon. *http://www.pjoon.com*

1.1.3 I partners

I partners sono aziende che hanno stipulato un contratto con Projectmoon SRL per l'uso della tecnologia Projectmoon System per la creazione di siti e portali per i loro clienti.

Sebbene abbiano accesso alla piattaforma come sviluppatori, gli strumenti a loro disposizione sono più limitati rispetto a quelli in uso dal personale interno. Dall'esigenza di migliorare questo aspetto, e potenziare la strumentazione a loro disposizione, nasce il progetto cuore del tirocinio.

Figura 1: Il logo aziendale

2 DESCRIZIONE DEL PROGETTO

Questo capitolo presenta le attività e le problematiche affrontate durante il tirocinio in maniera generale, tematiche la cui analisi e risoluzione verrà trattata nei capitoli successivi.

INDICE

2.1 UTILIZZO DI PROJECTMOON SYSTEM

La prima tematica affrontata consiste nella realizzazione di temi grafici mediante la tecnologia Projectmoon System. Verranno in particolare analizzate in dettaglio l'analisi e la creazione di una struttura centralizzata per la gestione e la suddivisione dei contenuti all'interno di una pagina.

Questo comporterà l'analisi della suddivisione tra elementi grafici e contenuti, per una corretta trattazione semantica dei contenuti.

L'uso degli strumenti che la piattaforma mette a disposizione per risolvere queste problematiche è stata oggetto di studio approfondito rivolto alla creazione di portali per clienti e alla produzione di documentazione, in italiano e inglese, da consegnare ai clienti per permettere loro di fruire in maniera autonoma ma completa del prodotto.

Lo studio delle funzionalità della piattaforma, in particolar modo delle funzionalità trattate in questa relazione, è stato accuratamente documentato ed è diventato manuale d'uso per i clienti.

2.2 WIZARD PER LA CREAZIONE DI TEMI

La seconda tematica riguarda l'analisi e lo sviluppo di uno strumento per consentire ai partners (principalmente) e al personale interno o autorizzato la creazione guidata di temi. Tale strumento sarà quindi realizzato sotto forma di wizard.

2.2.1 Requisiti funzionali informali

Si vuole realizzare uno strumento guidato, che fornisca all'utente la possibilità di modificare le impostazioni del tema base[1] fino a raggiungere il risultato desiderato.

1 Questo passaggio relativo ad aspetti più tecnici, così come i successivi, verrà trattato dettagliatamente nei prossimi capitoli

La prima versione del wizard si limiterà quindi a collezionare le modifiche e produrre come risultato i file con tutte le impostazioni necessarie. Tali file verranno quindi gestiti manualmente dal personale interno.

Una seconda versione del wizard integrerà invece un automatore che provvederà a svolgere le operazioni di validazione e deploy attualmente svolte a mano, rendendo quindi il wizard uno strumento che permetterà ai partners di creare il tema, pubblicarlo ed usarlo, senza l'intervento del personale interno di Projectmoon.

Non ci sono richieste per quanto riguarda le modalità di realizzazione del wizard, che vengono lasciate ad un'analisi più dettagliata. Si vuole però che il wizard sia

1. pronto nel più breve tempo possibile, almeno in una forma base usabile dagli utenti
2. facilmente espandibile e migliorabile, come conseguenza del punto 1.
3. semplice ed intuitivo nell'uso da parte dell'utente finale.

L'analisi e lo sviluppo durante questo tirocinio si focalizzeranno pertanto su una prima versione, anche non ottimizzata, che permetta però

1. ai partners di creare i temi per i propri clienti e inviarli al personale Projectmoon per gli step successivi
2. al personale Projectmoon di proseguire nel suo sviluppo nei mesi successivi, fino ad arrivare ad una versione completa ed ottimizzata.

3 TECNOLOGIE UTILIZZATE

Questo capitolo presenta le tecnologie utilizzate per la risoluzione delle problematiche descritte nel capitolo precedente.

INDICE

3.1 SVILUPPO WEB – LINGUAGGI CLIENT-SIDE

HTML *e* CSS

Lavorando su un prodotto web, le pagine create da Projectmoon System sono pagine HTML. La customizzazione dei temi avviene mediante CSS (fogli di stile), in particolare per quanto riguarda stili del testo, colori, margini, bordi e sfondi, e immagini.

Javascript

Per ottenere particolari effetti grafici si usa Javascript come linguaggio di scripting. Questo permette di ottenere abbellimenti, ma anche, se usato con criterio, una pagina più intuitiva e quindi piu user-friendly.

Per ottenere tale scopo si fa uso della libreria jQuery, che mette a disposizione una vasta serie di funzioni già pronte all'uso.

http://jquery.com

Esiste in rete una vastissima documentazione riguardo HTML, CSS e Javascript, pertanto verranno trattati ulteriormente solo quando interverranno in maniera importante nel resto del documento.

3.2 PROJECTMOON SYSTEM

La piattaforma Projectmoon System è il principale prodotto sviluppato da Projectmoon, su cui sono basati tutti i suoi prodotti. È un CMS avanzato

http://projectmoon.pjoon.com/system

che consente di creare facilmente portali 2.0, multilingua, customizzabili e fortemente orientati al lato *social* di Internet. Mette quindi a disposizione una serie di funzionalità rivolte all'interazione e alla comunicazione tra utenti, come ad esempio chat, blog, forum, RSS, calendari, gallerie immagini e documenti, tagging. Un esempio delle potenzialità della tecnologia è proprio il social network Pjoon.

http://www.liferay.com

Sviluppato in Java basandosi su Liferay, è basato sulle specifiche[1] JSR-168 e JSR-286 sullo sviluppo di portlet, JSR-127 (definisce un framework per componenti lato server di interfaccia utente) e JSR-170 (che regolamenta l'accesso ai contenuti).

MVC

La piattaforma segue il pattern MVC, Model-View-Controller, dove

MODEL fornisce i metodi per accedere ai dati;
VIEW visualizza i risultati del model e interagisce con utenti e agenti;
CONTROLLER riceve i comandi dell'utente (attraverso il view) e li attua, modificando lo stato di view e model.

Tale pattern permette di separare le varie componenti del software che realizzano i vari scopi, in particolar modo *business logic* e interfaccia utente.

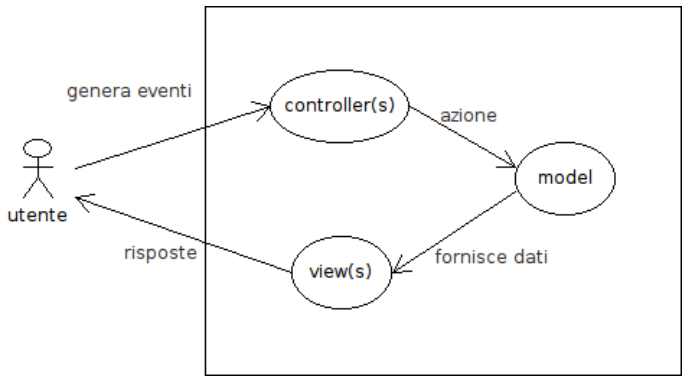


Figura 2: Schema del pattern MVC

In Projectmoon System la parte view è composta dalle pagine JSP, la parte model dall'interfaccia con i database e la parte controller è costituita dai *portlet*.

3.2.1 I portlet

I portlet sono applicazioni che sfruttano le API di un portale. Nel nostro caso, sono applicazioni Java da installare in un *portlet container*. Ciascun portlet è un modulo riusabile all'interno di un portale web, in grado di svolgere una funzione elementare (es: forum, aggregatore RSS, calendario, ...). Un portlet container è invece un insieme di *servlet* che usano le API di Apache Tomcat, più una parte di gestione portlet.

http://tomcat.apache.org

Essendo i portlet moduli indipendenti gli uni dagli altri anche nella stessa pagina, lo sviluppo dei portlet ha portato alla creazione di RUI[2] dalle capacità similari a quelle delle desktop applications. Ciascun portlet si comporta infatti come una finestra all'interno della pagina in cui viene inserito (e che diventa

1 come definite su http://www.jcp.org/en/jsr/overview
2 Rich User Interface

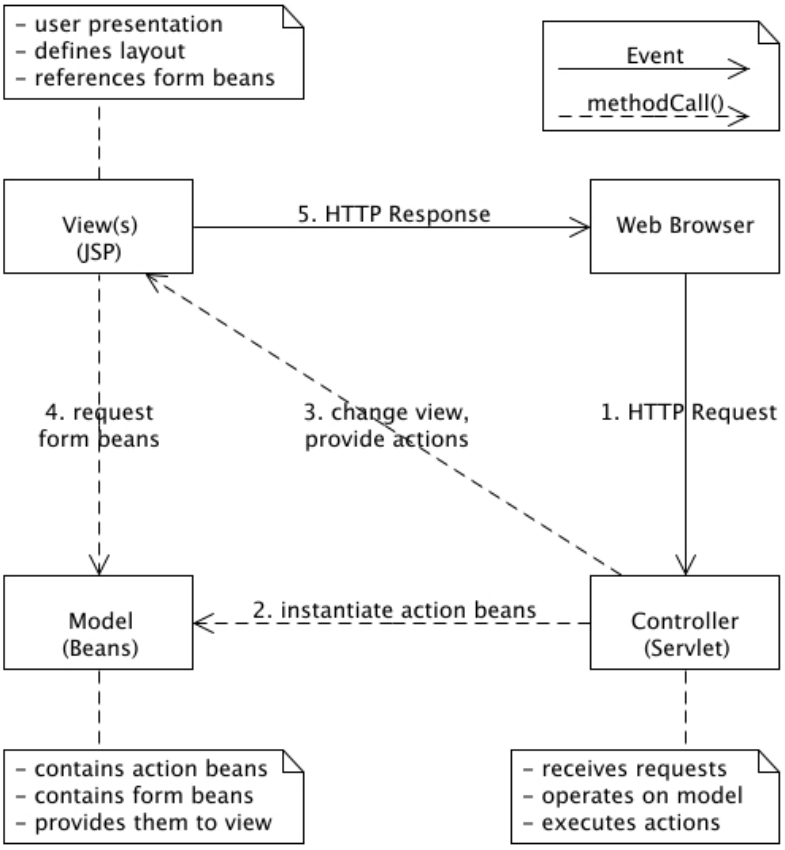


Figura 3: MVC in applicazioni web Java (da http://ifs.tuwien.ac.at/)

un *web container*), e consente quindi agli utenti che hanno privilegi sufficienti di chiuderlo, spostarlo mediante drag&drop o ridurlo. Diventa quindi molto intuitivo e veloce gestire una pagina estremamente personalizzata e al tempo stesso altamente funzionale, lavorando direttamente sull'interfaccia del portale. Non è un caso che HTML5 punti ad avvicinare le webapps alle desktop applications.

Il concetto di portlet è analogo a quelli di *widget* e di *dashboard*, ovvero insieme di unità funzionali non necessariamente in comunicazione tra loro. Si pensi al successo di piattaforme come Wordpress e Joomla, o alle unità funzionali di prodotti come Facebook da un lato, e OpenERP dall'altro.

Linguaggi di sviluppo

La maggior parte dei portlet per Projectmoon System è sviluppata in Java, ma è possibile usare anche linguaggi diversi, tra cui ricordiamo Ruby e PHP. Essendo moduli integrati in pagine web, HTML, CSS e Javascript svolgono un ruolo di primo piano, essendo la maniera con cui un portlet viene visualizzato e permette all'utente di interagire con esso.

3.2.2 La grafica in Projectmoon System

Visto come si realizzano le funzionalità della piattaforma, analizziamo come gestire l'aspetto grafico dei portali. È importante notare come le funzionalità (e quindi i portlet) non definiscano in alcun modo il loro aspetto. Occorre infatti separare[3] la semantica di un elemento funzionale dalla sua presentazione, in quanto sono attività separate, di cui si fanno carico ruoli diversi, e che possono variare nel tempo indipendentemente l'una dall'altra (es: restyling di un sito/applicazione, o upgrade di un portlet). Questo approccio favorisce la portabilità degli elementi - che come abbiamo visto è un elemento delle specifiche alla base di Projectmoon System - la leggerezza, la mantenibilità e il riutilizzo del codice, la customizzazione e la consistenza dello stile degli elementi. Il ruolo di vestire i contenuti spetta quindi al *template*.

Template

Un template è un'applicazione che si occupa di fornire un aspetto grafico alle pagine generate dalla piattaforma. Per ogni elemento definisce lo stile che questo adotterà. Attualmente, il template di default di Projectmoon System, quello cioè su cui ci si basa per costruire i portali, è Pjoon V3, le cui caratteristiche verranno analizzate nel capitolo successivo. Nella costruzione di un portale, il template può venire modificato, e nascono quindi i *temi*.

Temi

Un tema è una customizzazione del template di base, che definisce l'aspetto di ciascun elemento all'interno della pagina, ed è quindi la veste definitiva del portale. È l'elemento del portale su cui si concentrano maggiormente le richieste dei clienti, nonché la problematica alla base di questo tirocinio.

3.3 JOURNAL CONTENT

Il portlet *Journal Content* (in italiano *Crea/Mostra Articolo*) è il principale portlet della tecnologia Projectmoon System. Esso consente l'inserimento di testo, immagini, oggetti in flash (tramite *embed code*) e codice HTML. La sua potenza e flessibilità lo rendono lo strumento ideale per inserire qualsiasi tipo di contenuti all'interno delle pagine.

Strutture e modelli

Potendo quindi inserire articoli molto complessi, nasce la necessità di strutturare le varie tipologie di contenuto. Inoltre, il responsabile dei contenti non è, in linea di principio, lo sviluppatore o il grafico che si occupa dell'aspetto del sito. Il web designer/sviluppatore dovrà quindi poter lavorare sugli oggetti con metodo *black box*, ovvero astraendosi dai contenuti veri e propri. Questo avviene in tre passi:

1. analisi contenutistica: conoscere i contenuti del sito, analizzarli, dividerli in categorie. È l'operazione fondamentale per qualsiasi tipo di attività decisionale sulla presentazione dei contenuti. Il suo risultato è un documento di *content design*;

3 http://en.wikipedia.org/wiki/Separation_of_concerns,
http://en.wikipedia.org/wiki/Separation_of_presentation_and_content

2. definizione degli elementi: ogni contenuto, di qualsiasi tipo sia (dalle ricette alle stampe pubblicitarie alle schede tecniche di un film) si può suddividere in categorie. Questa fase produce la creazione delle *Strutture*;
3. sviluppo: occorre a questo punto implementare la struttura creata, per consentire al content manager di inserire i contenuti veri e propri. Si definiscono quindi i *Modelli*.

Oltre alla correttezza semantica, strutture e modelli presentano un altro vantaggio non indifferente, ovvero la gestione centralizzata. Una volta definiti una struttura ed un modello, si possono riutilizzare più volte all'interno del sito, in articoli diversi. Inoltre, la modifica di un elemento della struttura o del modello si ripercuote su tutti gli articoli che utilizzano quella struttura/modello, consentendo un risparmio di tempo ed energie notevole.

3.3.1 Strutture

Come realizzare una struttura

Vediamo in dettaglio come realizzare una struttura. Il primo passo consiste innanzitutto nel suddividere gli elementi che compongono il contenuto nelle seguenti tipologie:

tipologie di elementi

- testo: riga semplice di testo semplice;
- casella di testo: area composta da più righe di testo semplice;
- area di testo: testo formattato o codice HTML;
- immagine: immagine caricata dal computer dell'utente mediante l'uploader integrato;
- galleria immagini: immagine scelta tra quelle disponibili all'interno della galleria immagini dell'utente;
- gestione documenti: un documento presente nell'archivio dell'utente;
- variabile booleana;
- elenco di selezione: definisce una lista di coppie chiave / valore di elementi (opzioni), lista che verrà poi visualizzata come menu a tendina, tra cui selezionare in seguito una sola tra le opzioni disponibili;
- lista multi-selezione: definisce una lista di coppie chiave / valore di elementi (opzioni), lista da cui sarà in seguito possibile selezionare più opzioni;
- collegamenti ad altre pagine: permette di creare un link ad un'altra pagina del sito (non ad altri siti).

Ogni elemento della struttura diventerà un campo di input nella pagina di inserimento dei contenuti, ed una variabile durante la scrittura del modello. La tipologia scelta per ogni elemento determinerà il formato del campo di input.

organizzazione degli elementi

Gli elementi possono essere organizzati gerarchicamente ad albero, sotto forma di relazione parentale, logica o strutturale. Ad esempio, un titolo e i capitoli di un libro, o una lista e le sue opzioni. Tale caratteristica, oltre ad essere semanticamente corretta, può essere sfruttata per facilitare l'implementazione di un modello (come vedremo in seguito).

A questo punto si passa a definire fisicamente la struttura, tramite un file XML. Tale file può essere creato con un semplice editor di testo, o tramite un editor interattivo adatto agli utenti meno avvezzi alla programmazione, che permette di inserire ogni elemento definendone il nome mediante un campo di input, la tipologia mediante un menu a tendina, e la presenza di eventuali elementi figli.

3.3.2 Modelli

I templates sono gli strumenti usati per separare la grafica dai contenuti e replicare la stessa struttura su molte pagine. Le pagine risultato (dinamiche) sono il prodotto dell'elaborazione di un *template engine*. In Projectmoon System i templates prendono il nome di *modelli*.

Un modello è sostanzialmente l'implementazione di una struttura mediante uno script, scritto in un linguaggio a scelta tra Velocity, XSL[4] e CSS[5]. Il linguaggio usato durante questo tirocinio, è Velocity.

Velocity

http://velocity.apache.org

Velocity è un linguaggio creato da Apache che integra all'interno del codice HTML un limitato set di istruzioni condizionali e di ciclo, identificate ponendo a inizio riga un hash (#), e dei metodi per agire sulle variabili dichiarate nella definizione della struttura. In particolare, le istruzioni:

```
#if ( condizione )
    (...)
#elseif ( altra condizione )
    (...)
#else
    (...)
#end

#foreach( $variabile in $array )
    (...)
#end

## un commento
```

e i metodi

- `$variabile.getName()`
- `$variabile.getType()`
- `$variabile.getData()`
- `$variabile.getChildren()`
- `$variabile.getOptions()` (per liste multi-selezione).

Tramite Velocity è quindi possibile inserire blocchi di HTML se si verificano particolari condizioni, ad esempio la presenza o meno di un particolare contenuto messo a disposizione dalla struttura, o la presenza di elementi figli di un determinato elemento. Vediamo un esempio.

Esempio generico...

Creiamo una struttura con i seguenti elementi:

4 eXtensible Stylesheet Language, per definire fogli di stile in formato XML
5 Cascade Style Sheet

- *Pic1* di tipo *image_gallery*, con i seguenti elementi figli:
  - `PicChildA` di tipo `image_gallery`;
  - `PicChildB` di tipo `image_gallery`.

Creiamo quindi un modello con il seguente codice:

```
<div>
   ## visualizziamo il nome, il tipo di Pic1
   <p>$Pic1.getName()</p>
   <p>$Pic1.getType()</p>

   ## visualizziamo il contenuto di Pic1
   ## essendo un'immagine, all'interno di un tag img
   <img src="$Pic1.getData()" />

   ## scorriamo gli elementi figli di Pic1
   #foreach($child in $e.getChildren())

      ## visualizziamo nome, tipo e contenuto
      ## di ciascun figlio di Pic1
      <blockquote>
         <p>$child.getName()</p>
         <p>$child.getType()</p>
         <img src="$child.getData()" />
      #end
      </blockquote>
   #end
</div>
```

...con liste multi-selezione...

Vediamo ora rapidamente come usare le liste multi-selezione. Da una struttura definita come

- `l` di tipo `multi-selection list`, con elementi figli le coppie
  - `valore1 chiave1`
  - `valore2 chiave2`
  - `valore3 chiave3`

implementiamo il seguente codice in un modello:

```
<p>$l.getName()</p>
<div>
   <ul>
   #foreach ($opt in $l.getOptions())
      <li>$opt</li>
   #end
   </ul>
</div>
```

...e con elenchi di selezione

Ancora più semplice è il caso in cui trattiamo elenchi di selezione, in cui una sola opzione è selezionabile. Quindi, creeremo una struttura del tipo

- `l` di tipo `elenco di selezione`, con elementi figli le coppie

```
        - valore1 chiave1
        - valore2 chiave2
        - valore3 chiave3
```

e un modello con codice

```
<p>$l.getName()</p>
<p>$l.getData()</p>
```

Avendo quindi definito struttura e modello per un particolare contenuto, il content manager dovrà preoccuparsi solo di riempire i campi di input che lo sviluppatore ha definito creando la struttura.

3.4 SVILUPPO IN PROJECTMOON SYSTEM

Come detto nelle pagine precedenti, sviluppare in Projectmoon System, estenderne le funzionalità, vuol dire essenzialmente sviluppare portlet in Java e HTML. La piattaforma mette però a disposizione una serie di strumenti per la gestione dei portlet, che facilitano il compito agli sviluppatori.

3.4.1 Preparazione dell'ambiente

Prima di tutto occorre ovviamente predisporre una versione funzionante della piattaforma. Durante questo tirocinio lo sviluppo è avvenuto su una macchina con sistema operativo GNU/Linux CentOS 5.4. È comunque possibile replicare la procedura d'installazione su una qualsiasi distribuzione linux recente.

Iniziamo creando un utente, nel nostro caso chiamato `pjoon21x`, nel cui spazio installeremo la piattaforma.

Installiamo ora il textscdbms MySQL e il server web Apache Tomcat. Il passo successivo consiste nell'installare la piattaforma vera e propria. Tale operazione può essere svolta solo all'interno di Projectmoon, dal momento che la piattaforma non è rilasciata al pubblico.

hhtp://www.eclipse.org

Per facilitare lo sviluppo si procede ad installare un IDE, nel nostro caso Eclipse, dal momento che offre una serie di tool che aiutano lo sviluppatore nel gestire progetti complessi.

3.4.2 Creazione di un portlet

Essendo la piattaforma basata su portlet, lo sviluppo di questi ultimi è la maniera per ampliarne le potenzialità e le offerte.

È già predisposto uno script che crea l'albero di cartelle e parte dei file componenti il portlet. I file che il programmatore andrà a modificare si trovano in realtà solo in alcune tra queste cartelle.

I file della componente view, vale a dire le pagine JSP sono raggruppati, così come la parte model (la Util per accedere al database) e la parte controller (il codice Java contentente la business logic).

Si può quindi procedere alla compilazione e al deployment, con il comando `ant clean deploy`, e se non si sono commessi errori il portlet è pronto per l'uso.

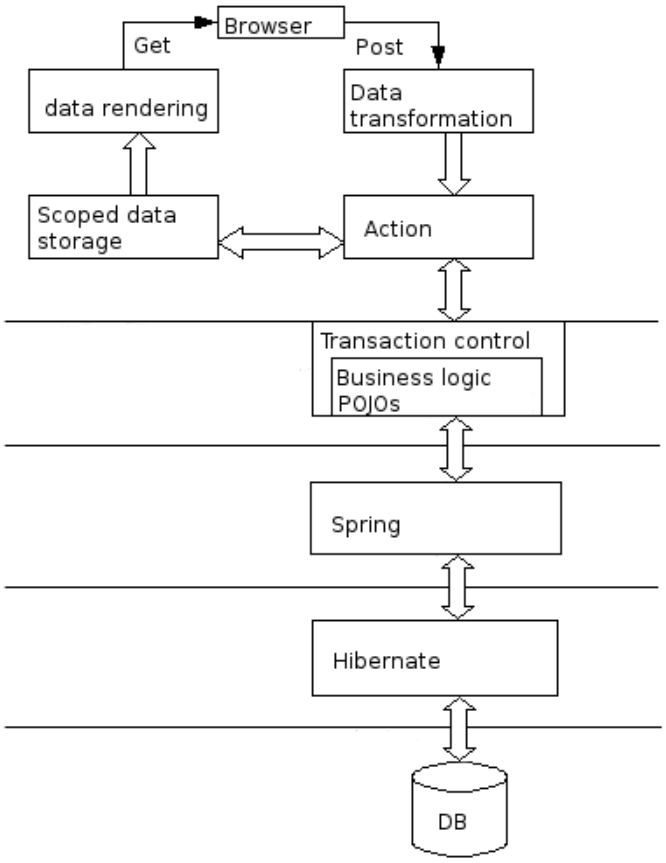


Figura 4: Livelli dell'architettura della piattaforma

3.4.3 Gestione dei database

La piattaforma Projectmoon System si appoggia ai framework Hibernate e Spring per separare la gestione del database dal resto del codice, rendendo indipendente il programma dal DBMS utilizzato.

http://www.hibernate.org
http://www.springframework.org

La piattaforma gestisce i database mediante un tool chiamato *service builder*. Questi si occupa di elaborare l'input dato dallo sviluppatore per creare e gestire la base di dati e i servizi necessari per accedervi.

Il primo passo è creare il database. Non c'è bisogno di scrivere codice SQL. È sufficiente creare un file XML secondo una sintassi predefinita, specificando le tabelle da creare, i campi, le chiavi primarie e i campi su cui si andranno ad effettuare ricerche. Tale file verrà quindi analizzato dal service builder, che creerà le tabelle e i servizi per accedervi direttamente, tra cui i metodi per recuperare le informazioni, per inserire dati ed effettuare ricerche, oltre a tutto il codice SQL necessario — codice che è disponibile ma salvo casi eccezionali non è necessario andare a toccare.

Lo step successivo consiste nell'implementare un livello ulteriore di servizi, che, utilizzando le primitive generate dal service builder, effettuino le operazioni di cui gli sviluppatori avanno bisogno. Si può andare dal semplice inserimento, che non sarà altro che un'interfaccia per uno dei metodi `set`

creati dal service builder, ad una ricerca che rielabora i dati prima di restituirli. Questo è inoltre utile quando si preferisce fare in modo che gli sviluppatori non abbiano accesso diretto alle primitive create dal service builder.

A titolo di esempio, vediamo come si definisce una tabella.

```
<entity name="ExampleEntity" local-service="true" remote-service="
    false">
    <!-- PK fields -->
    <column name="entityId" type="long" primary="true"/>
    <!-- Key fields -->
    <column name="exrternalId" type="long"/>
    <!-- Other fields -->
    <column name="value1" type="int" />
    <column name="value2" type="String" />
    <column name="value3" type="long"/>
    <column name="value4" type="Double"/>
    <!-- Finder methods -->
    <finder name="V1" return-type="Collection">
       <finder-column name="value1" />
    </finder>
    <finder name="V12" return-type="Collection">
       <finder-column name="value1" />
       <finder-column name="value2" />
    </finder>
    <finder name="V123" return-type="entityId" unique="true">
       <finder-column name="value1" />
       <finder-column name="value2" comparator="&gt;" />
       <finder-column name="value3" />
    </finder>
    <!-- References -->
     <reference package-path="com.liferay.counter" entity="Counter"
          />
</entity>
```

I primi campi definiscono gli attributi, specificandone il tipo, e se sono chiavi primarie o chiavi esterne di altre entità. Ci sarà quindi un'altra entità che avrà come chiave primaria `externalId` e che specificherà che all'interno dell'entità `ExampleEntity`vi sarà un campo che riferirà alla sua chiave primaria. Come si può notare, i tipi di dato sono tipi di dato Java, non SQL.

Il service builder predisporrà inoltre la classe Java relativa alla tabella, e una serie di classi per lo sviluppo del secondo strato di servizi.

Eventuali modifiche alla base di dati si apporteranno semplicemente editando il file XML e ricompilando. Verranno quindi ricreati anche i servizi. Lo sviluppatore, avendo precedentemente creato i metodi per sfruttare i servizi, non avrà quindi bisogno di modificare alcuna riga di codice Java per correggere le operazioni.

3.4.4 Taglib

Le *taglib*, contrazione di *tag libraries*, sono, come si evince dal nome, tags JSP definite dall'utente. Permettono di definire un comportamento, una sequenza di azioni, e replicarlo più volte. Una tag riferisce ad una particolare classe, che viene eseguita ogniqualvolta la JVM incontra la taglib.

Il suo uso è analogo a quello dei normali tag HTML, e vanno quindi inserite nel file (`view.jsp`) che si occupa di definire la grafica del portlet.

Come creare una taglib

Per definire una taglib occorrono due componenti:

1. un file TLD (Tag Lib Definition) di tipo XML che specifica i singoli tag, i parametri (quando presenti), il corpo del tag e la classe che si occupa di definirlo;
2. le classi Java che si occupano di gestire effettivamente la taglib.

Utilizzo delle taglib

Le taglib JSP differiscono nell'uso dai normali tag HTML per l'utilizzo del carattere `%`. Una taglib si scriverà dunque `<% taglib` [*parametri*] `%>`

Vediamo un paio di esempi su come utilizzare una taglib in un portlet.

Iniziamo con una taglib che visualizza un messaggio di errore. Questa è una taglib proprietaria di Liferay.

```
<liferay-ui:error key="something-gone-wrong" />
```

Il campo `key` non contiene il messaggio visualizzato, ma una stringa che identifica il messaggio d'errore. Questo permette di tradurre il messaggio e localizzare il prodotto, senza riscrivere la taglib.

Vediamo ora le taglib `c:choose`, `c:when` e `c:otherwise`, che permettono di riprodurre il comportamento di un selettore `switch`. È disponibile anche una taglib `if` che riproduce il comportamento del costrutto `if`, eseguendo il contenuto — e mostrando il codice HTML in essa contenuto — se la condizione specificata è verificata. `c:if` manca però di un branch `else`, quindi per implementare un selettore `if-else` occorre rivolgersi alla taglib `choose-when -otherwise`.

```
<%
int a = 2;
int b = 3;
%>

<c:choose>
   <c:when test="<%= (a < b) %>" >
      Codice HTML visualizzato
   </c:when>
   <c:when test="<% (a == b) %>" >
      Codice non visualizzato
   </c:when>
   <c:otherwise>
      Altro codice non visualizzato
   </c:otherwise>
</c:choose>
```

L'utilizzo delle taglib permette di semplificare di molto la scrittura del codice, pertanto nello sviluppo del wizard verrà creata una taglib apposita, che

sarà un'estensione della taglib `prjmoon-ui`, taglib contenente codice scritto dai programmatori Projectmoon.

3.5 STRUMENTI UTILIZZATI

Ambiente di sviluppo

Per lo sviluppo sulla piattaforma Projectmoon System è necessario un ambiente di prova, consistente in un'installazione locale della piattaforma, dove produrre e testare il codice prima di trasferirlo nella piattaforma reale residente sui server. La piattaforma richiede:

APACHE TOMCAT server web

JVM Java Development Kit

JRE Java Runtime Environment.

Il sistema operativo utilizzato durante lo sviluppo è Linux CentOs 5.4. Una macchina virtuale con Windows Seven e un computer Apple sono stati usati per i test.

L'editor utilizzato è Eclipse, che integra una serie di strumenti e plugins per facilitare lo sviluppo sulla piattaforma, consentendo di gestire progetti di ampie dimensioni.

Per lo sviluppo di strutture e modelli si usano gli strumenti integrati nel portale, pertanto un browser recente (IE 7+, FF2+, Safari, Chrome 4+, Opera 9+) è sufficiente.

Per la stesura della documentazione e dell'analisi sono stati utilizzati OpenOffice e Google Docs per la redazione della documentazione e dell'analisi, Xfig per i mockup grafici e Umbrello per UML. Durante parte del tirocinio è stato usato un Powerbook Apple con Adobe Photoshop e Macromedia Dreamweaver.

Per la stesura di questa tesi si sono usati LaTeX con stile `ArsClassica`, vi e TeXmaker.

4 REALIZZAZIONE DEL PROGETTO

Questo capitolo riassume l'analisi e la risoluzione delle problematiche relative alla strutturazione di contenuti e alla creazione di temi grafici

INDICE

4.1 STRUTTURE E MODELLI

Viene ora presentato un esempio di utilizzo del binomio Strutture e Modelli del portlet Journal Content introdotto nel precedente capitolo.

Il caso in questione segue alle richieste di un cliente, che ha commissionato la realizzazione di un portale per un network di agriturismi nella provincia di Treviso. Mentre venivano prepararati la struttura del sito vera e propria e il tema del sito, a me è stata affidata la predisposizione delle strutture dei contenuti, e la realizzazione di modelli che le integrassero nel portale.

Si tratta in tutto delle seguenti pagine:

- un richiamo in homepage della sezione Eventi;
- la sezione Eventi;
- una pagina per ogni evento;
- la lista degli agriturismi affiliati;
- una pagina per ogni agriturismo, a mo' di carta d'identità.

Per ognuno di questi casi si è provveduto ad usare un solo portlet Journal Content, eventualmente affiancato ad altri portlet (calendario, altre immagini o altri Journal Content) la cui gestione non mi è stata affidata, e quindi non vengono qui trattati.

Il portale è stato realizzato prima in ambiente demo per essere mostrato al committente per una valutazione finale. Una volta ottenuto l'assenso da parte del cliente, è stato portato "in reale" ed è ora raggiungibile all'indirizzo http://terranostra.pjoon.com.

4.1.1 Sezione Eventi in homepage

Richieste

Le richieste provenienti dalla sezione grafica sono state le seguenti:

- creare, mediante un Journal Content, una lista di 5 eventi programmati che abbiano una pagina nel portale, ciascuno in una riga;
- tale portlet deve occupare in larghezza il 70% della larghezza totale della pagina, e sarà posizionato sulla colonna sinistra;
- per ogni evento si desidera mostrare un'immagine presente nella galleria immagini ed una breve descrizione, entrambe linkabili all'evento mostrato;
- le dimensioni dell'immagine sono 120x70 pixel, la sua posizione sarà a sinistra;
- il testo occuperà in larghezza tutto lo spazio alla destra dell'immagine, fino a raggiungere il 70% della pagina;
- al termine, ci sarà un link che porterà alla pagina contenente la lista di tutti gli eventi disponibili.

Analisi

Da richiesta, l'immagine di un evento sarà di tipo `image_gallery`. A livello di Journal Content non può venir effettuato alcun controllo sulla dimensione effettiva dell'immagine caricata, perciò chi inserirà i contenuti si dovrà preoccupare di caricare nella galleria immagini (strumento disponibile all'amministratore del portale) delle immagini della dimensione predefinita (o in scala), mentre a livello di Modello si imporrà comunque la visualizzazione dell'immagine in dimensione 120x70 pixel, per ridimensionare eventuali immagini di dimensioni diverse.

Per la breve descrizione di un evento si sceglie di non usare l'elemento `area di testo`. L'editor visuale infatti è un elemento molto pesante durante il caricamento di una pagina. Durante il periodo di formazione sullo strumento e nelle prove successive, si è visto che inserendo più di due editor visuali si rischia che al termine del caricamento della pagina di inserimento dei contenuti gli editor non vengano correttamente visualizzati, impedendo quindi il loro uso. Non essendoci quindi alcuna necessità di formattare il testo, si sceglie quindi di usare l'elemento `casella di testo`, più semplice e leggero di un'`area di testo`, ma che consente, anche dal punto di vista visivo per l'utente finale, di inserire un testo più corposo rispetta ad un semplice elemento `testo`.

Per gestire i link agli eventi si sceglie di non usare l'elemento `link ad una pagina`, in quanto poco flessibile. Si sceglie invece di usare l'elemento `testo`, in cui verrà inserito il link esatto alla pagina.

Il link alla pagina degli eventi invece può essere scelto come `link ad una pagina`, dato che la posizione della pagina "Eventi" non cambierà nel tempo.

La struttura gerarchica

Come abbiamo visto, sia l'immagine che la descrizione di un evento linkano alla pagina dell'evento stesso. Quindi possiamo considerare il link come un *contenitore* che contiene sia l'immagine che il testo, e predisporre immagine e testo come *elementi figli* del link associato.

Questo avrà un triplice vantaggio:

1. renderà la struttura semanticamente corretta;

2. permetterà la scrittura di un modello più compatto e leggibile, con notevole risparmio di codice. Si potrà infatti iterare sui link e recuperare l'immagine ed il testo in essi contenuti come figli, con i metodi visti nel capitolo precedente;

3. renderà il modello non vincolato al numero di eventi mostrati. Come abbiamo visto, infatti, è possibile scorrere su *tutti* i figli di un elemento padre. In questa maniera, nel caso si dovesse variare il numero di eventi mostrati, sarà sufficiente intervenire sulla struttura, senza dover toccare il modello.

Per permettere ciò, si deve introdurre un'ulteriore variabile che contenga tutti i link agli eventi, che ne diventeranno elementi figli. Si sceglie una variabile di tipo `boolean`, che nella pagina di inserimento contenuti apparirà sotto forma di checkbox.

Come risultato, la struttura sarà la seguente (viene mostrato il file XML generato):

```
<root>
  <dynamic-element name='links' type='boolean'>
    <dynamic-element name='link1' type='text'>
      <dynamic-element name='img1' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo1' type='text_box'></dynamic-element>
    </dynamic-element>
    <dynamic-element name='link2' type='text'>
      <dynamic-element name='img2' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo2' type='text_box'></dynamic-element>
    </dynamic-element>
    <dynamic-element name='link3' type='text'>
      <dynamic-element name='img3' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo3' type='text_box'></dynamic-element>
    </dynamic-element>
    <dynamic-element name='link4' type='text'>
      <dynamic-element name='img4' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo4' type='text_box'></dynamic-element>
    </dynamic-element>
    <dynamic-element name='link5' type='text'>
      <dynamic-element name='img5' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo5' type='text_box'></dynamic-element>
    </dynamic-element>
  </dynamic-element>
  <dynamic-element name='link6' type='link_to_layout'></dynamic-element>
</root>
```

Implementazione del modello

Nel modello andiamo ora a definire le posizioni, le proporzioni e le dimensioni degli elementi. Il codice scritto è il seguente:

```
<div id="contenitoreEventi" style="width:display:block; overflow:hidden;">

## scorriamo tutti gli eventi

#foreach ($link in $links.getChildren())
   <div class="riga" style="clear:both;">

   ## per ogni evento, recuperiamo i figli,
   ## inseriamoli nel tag corretto in base al tipo

   #foreach ($element in $link.getChildren())
      <div style="float:left; margin-bottom:10px;"><a href="$link.getData()">
      #if ($element.getType() == 'image_gallery' )
         <div style="float:left;"><img src="$element.getData()" width="120px"
    height="70px" style="margin-right:10px;"></div>
      #else
         <div style="float:left; max-width:480px">$element.getData()</div>
      #end
      </a></div>
   #end

   </div>
#end

## l'ultimo link, che punta alla pagina Eventi, fuori dal ciclo
<div id="riga6" style="clear:both; float:right">
  <a href="$link6.getUrl()">&gt; vedi tutti</a>
</div>
</div>
```

Come si vede, l'aver strutturato gli elementi secondo una struttura gerarchica, rende possibile scrivere un modello molto breve e molto semplice.

Analizziamo in particolare il frammento di codice che dispone immagine e testo e crea il link:

```
#foreach ($element in $link.getChildren())
   <div style="float:left; margin-bottom:10px;"><a href="$link.getData()">
   #if ($element.getType() == 'image_gallery' )
      <img src="$element.getData()" width="120px" height="70px" style="margin
    -right:10px;">
   #else
      <div style="float:left; max-width:480px">$element.getData()</div>
   #end
   </a></div>
#end
```

Recuperiamo i figli del link ad un evento, ed esaminiamoli uno ad uno.

Possiamo in ogni caso creare una `<div>` contenitore, con un margine inferiore che la separi dalla sottostante, e allineiata a sinistra. Apriamo inoltre il tag `<a>`, e associamo ad `href` il valore della variabile `link` in esame. Questa è un'operazione comune sia all'immagine che al testo.

Ora recuperiamo il tipo dell'elemento figlio del link che dobbiamo esaminare. Se è un'immagine, usiamo il contenuto della variabile come campo `src` del tag `<img>`, settandone la dimensione e il margine che la separerà dal testo.

Altrimenti, sarà per forza di cose il testo descrittivo dell'evento, che verrà mostrato accanto all'immagine.

Chiudiamo quindi il link e la `<div>`.

Con immagini, testo e tema del sito, il risultato è il seguente:

Figura 5: Sezione "Eventi" in homepage

4.1.2 La pagina Eventi

Richieste

Questa pagina sarà molto simile al contenuto del Journal Content trattato nella sezione precedente:

- il suo scopo è contenere tutti gli eventi programmati, mostrando per ciascuno un'immagine ed una breve descrizione linkabili, fino ad un massimo di 10 eventi;
- il portlet occuperà in larghezza il 70% della pagina, il restante spazio conterrà il calendario e dei banner;
- per ogni evento si desidera mostrare un'immagine presente nella galleria immagini ed una breve descrizione, entrambe linkabili all'evento mostrato;
- le dimensioni dell'immagine sono 200x150 pixel, la sua posizione sarà a sinistra;

- il testo occuperà in larghezza tutto lo spazio alla destra dell'immagine, fino a raggiungere il 70% della pagina;

Analisi

Date le richieste molto simili al portlet precedente, le considerazioni sono analoghe. Vi sarà una variabile booleana che farà da contenitore a tutti gli eventi. Ad ogni evento verrà associato un link (elemento `testo`), che a sua volta conterrà un'immagine (di tipo `image_gallery`) ed un testo (elemento `casella di testo`).

Per la struttura verrà quindi creato il seguente codice XML:

```
<root>
  <dynamic-element name='links' type='boolean'>
    <dynamic-element name='link1' type='text'>
      <dynamic-element name='immagine1' type='image_gallery'></dynamic-
    element>
      <dynamic-element name='testo1' type='text_box'></dynamic-element>
    </dynamic-element>
    <dynamic-element name='link2' type='text'>
      <dynamic-element name='immagine2' type='image_gallery'></dynamic-
    element>
      <dynamic-element name='testo2' type='text_box'></dynamic-element>
    </dynamic-element>
    <dynamic-element name='link3' type='text'>
      <dynamic-element name='immagine3' type='image_gallery'></dynamic-
    element>
      <dynamic-element name='testo3' type='text_box'></dynamic-element>
    </dynamic-element>

   [...]

    <dynamic-element name='link9' type='text'>
      <dynamic-element name='immagine9' type='image_gallery'></dynamic-
    element>
      <dynamic-element name='testo9' type='text_box'></dynamic-element>
    </dynamic-element>
    <dynamic-element name='link10' type='text'>
      <dynamic-element name='immagine10' type='image_gallery'></dynamic-
    element>
      <dynamic-element name='testo10' type='text_box'></dynamic-element>
    </dynamic-element>
  </dynamic-element>
</root>
```

Implementazione del modello

Il modello è identico al precedente:

```
<div id="contenitoreEventi" style="display:block; overflow:hidden;">
#foreach ($el in $links.getChildren())
   <div class="riga" style="clear:both;">
   #foreach ($element in $el.getChildren())
      <div style="float:left; margin-bottom:10px;"><a href="$el.getData()" target
    ="_blank">
      #if ($element.getType() == 'image_gallery' )
```

```
        <img src="$element.getData()" width="200px" height="150px" style="
    margin-right:10px;">
      #else
        <div style="max-width:300px;">$element.getData()</div>
      #end
      </a></div>
   #end
   </div>
#end
</div>
```

pertanto valgono le stesse considerazioni.

Con immagini e testo di esempio, il risultato è il seguente:

Figura 6: Pagina "Eventi"

4.1.3 La pagina Evento singolo

Richieste

Anche questa pagina è molto semplice:

- il suo scopo è contenere un evento singolo
- il portlet occuperà in larghezza il 100% della pagina;
- per ogni evento si desidera mostrare un'immagine presente nella galleria immagini, larga 750 pixel e alta 300 nella prima riga, e in una seconda riga una descrizione più approfondita a sinistra, e una mappa nella restante porzione destra, per circa un terzo della riga;

Analisi

L'immagine anche in questo caso sarà di tipo `image_gallery`.

La descrizione può essere definita come `area di testo`, in quanto la descrizione necessita di formattazione, e nella pagina di inserimento contenuti sarà l'unico editor presente. Cade quindi la motivazione di carattere prestazionale che ha portato in precedenza ad evitare l'editor avanzato.

Per quanto riguarda la mappa, non si può ovviamente usare il portlet "Mappa di Google" a disposizione con la piattaforma Projectmoon System, in quanto non è possibile importare un portlet dentro un altro. Non è nemmeno consigliabile soluzioni alternative, come ad esempio affiancare i due portlet, o disporli uno sotto l'altro, in quanto il portlet "Mappa di Google" richiede l'API Key di Google e, in generale, un uso molto più difficoltoso di un Journal Content. La soluzione più semplice è quindi utilizzare l'embed code della mappa. Sarà quindi sufficiente una `casella di testo`.

```
<root>
  <dynamic-element name='pic' type='image_gallery'></dynamic-element>
  <dynamic-element name='text' type='text_area'></dynamic-element>
  <dynamic-element name='map' type='text_box'></dynamic-element>
</root>
```

Implementazione in un modello

Anche il questo caso il modello non presenta particolari problemi.

```
<div id="eventContainer" style="overflow:hidden;">
   <div id="immagine" align="center" style="margin:10px 10px 20px 10px;
    overflow:hidden;"><img src="$pic.getData()" alt="foto dell'evento" width="750
    px" height="300px"/></div>
   <div id="textMap">
      <div id="text" style="width:65%;float: left; overflow:hidden; margin-
    left:10px; " >$testo.getData()</div>
      <div id="map" style="width:33%;float: right; overflow:hidden; margin-
    right:10px;" >$mappa.getData()</div>
   </div>
</div>
```

Come si può vedere, `testo` e `mappa` non raggiungono insieme il 100% della larghezza della pagina, in quanto occorre tener conto dei padding interni del portlet.

Figura 7: Pagina per singolo evento

4.1.4 Lista degli agriturismi

Richieste

- il suo scopo è contenere la lista di tutti gli agriturismi. Inizialmente ne verranno inseriti 24;
- il portlet occuperà in larghezza il 100% della pagina. Gli agriturismi saranno disposti su 3 colonne;
- per ogni agriturismo si richiede di mostrare un'immagine 120x70 px ed una breve descrizione, entrambe linkabili, con la pagina di destinazione che si aprirà in una nuova finestra del browser.

Analisi

Non c'è nulla di diverso dai casi precedenti, percio l'immagine sarà di tipo `image_gallery`, la descrizione sarà una `casella di testo`, e il link un elemento di tipo `testo`.

Come detto nelle richieste, a scopo dimostrativo si inseriscono 4 agriturismi. Per aggiungerne altri sarà sufficiente aggiungere elementi nella struttura seguente:

```
<root>
  <dynamic-element name='lista' type='boolean'>
    <dynamic-element name='link1' type='text'>
      <dynamic-element name='img1' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo1' type='text_box'></dynamic-element>
  </dynamic-element>
    <dynamic-element name='link2' type='text'>
      <dynamic-element name='img2' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo2' type='text_box'></dynamic-element>
```

```
  </dynamic-element>
    <dynamic-element name='link3' type='text'>
      <dynamic-element name='img3' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo3' type='text_box'></dynamic-element>
  </dynamic-element>

   [...]

    <dynamic-element name='link24' type='text'>
      <dynamic-element name='img24' type='image_gallery'></dynamic-element>
      <dynamic-element name='testo24' type='text_box'></dynamic-element>
    </dynamic-element>
  </dynamic-element>
</root>
```

Implementazione del modello

Anche in questo caso il modello è simile ai precedenti. Occorre solo prestare attenzione alla disposizione in 3 colonne degli agriturismi.

```
<div id="contenitore" style="overflow:hidden; width:930px; display:block; margin
    -top:20px;">
   #foreach($link in $lista.getChildren())
      <div class="gridElement" style="width:300px; margin-bottom:10px;
    margin-right:0px; display:inline-block; float:left;"><a href="$link.getData()"
    target="_blank">
      #foreach($item in $link.getChildren())
         #if($item.getType() == 'image_gallery')
            <img src="$item.getData()" width="120px" height="70px">
         #else
            <div style=" float:right; max-width:170px; margin-left:10px margin
    -right:10px;">$item.getData()</div>
         #end
      #end
      </a></div>
   #end
</div>
```

4.1.5 Scheda di un agriturismo

Questa pagina fungerà da carta d'identità per un agriturismo.

Richieste

- il suo scopo è quello di presentare un agriturismo, mostrandone foto, caratteristiche, descrizione e mappa;
- la pagina dovrà essere divisa in 6 grandi blocchi, disposti su 2 colonne, secondo lo schema illustrato sotto;
- il primo blocco conterrà 10 immagini, una 400x300 px e altre nove di dimensione 120x90 px disposte in una griglia 3x3;
- il secondo blocco, affiancato al precedente, conterrà la descrizione dell'agriturismo;

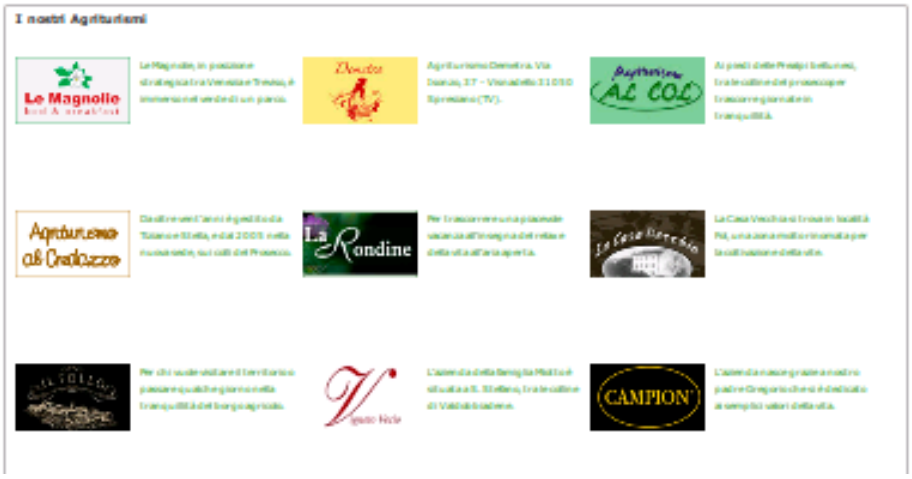


Figura 8: Pagina contenente la lista degli sgriturismi

- il terzo blocco conterrà una serie di icone di altezza fissa e larghezza variabile, scelte tra un gruppo ben definito, che illustrano le varie caratteristiche dell'agriturismo;
- accanto al blocco precedente ci sarà un banner, che linkerà alla pagina principale del network;
- il quinto blocco, in basso a sinistra, conterrà un'immagine 460x320 px;
- il sesto blocco, accanto al precedente, conterrà infine la mappa dove localizzare l'agriturismo.

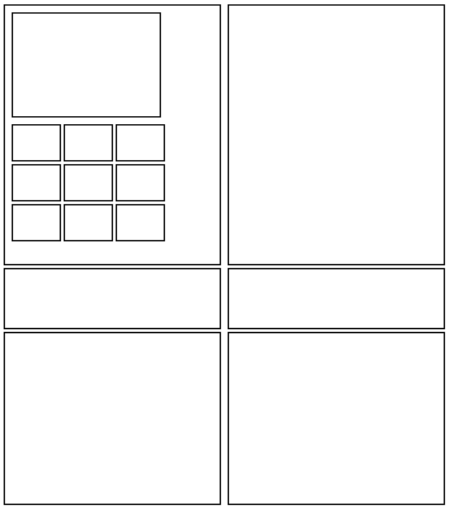


Figura 9: Schema della pagina

Analisi

Anche in questo caso le immagini saranno tutte in formato `image_gallery`. Si contano in totale 12 immagini, suddivise in

- un'immagine 400x300px, nel primo blocco;
- 9 immagini 120x90px, nella griglia del primo blocco;
- un'immagine per il banner nel quarto blocco, la cui altezza viene ricavata analizzando gli elementi del quarto blocco;
- un'immagine 460x320px nel quinto blocco.

Per la descrizione si sceglie di concedere l'uso dell'editor avanzato, in quanto è un solo elemento (quindi non appesantisce il caricamento della pagina di inserimento contenuti) che necessita di formattazione. Sarà quindi di tipo `area di testo`.

Le caratteristiche degli agriturismi sono descritte da 10 icone fisse. Ogni agriturismo dovrà scegliere le proprie. Si userà perciò una `lista multi -selezione`. Ogni elemento di una `lista multi-selezione` è, come visto nei capitoli precedenti, una coppia `(chiave,valore)`. `valore` sarà quindi la descrizione dell'icona (equitazione, passeggiate, ...), ma nasce il problema di come rappresentare la `chiave`, che sarà anche ciò che effettivamente verrà utilizzato nel modello. Il campo della struttura infatti non consente l'inserimento di immagini. Si aggira l'ostacolo in questa maniera:

1. si caricano nella galleria immagini tutti i loghi, recuperandone l'URL;
2. analizziamo la struttura dell'URL: in ambiente di prova è `http://www.projectmoon.eu:8680/image/image_gallery?img_id=XXXXXX` dove XXXXXX è l'ID dell'immagine nella galleria;
3. si può quindi utilizzare come chiave l'ID dell'immagine, e ricostruire l'URL direttamente nel modello.

Questa procedura dovrà essere ripetuta al momento della messa online effettiva del sito, ma il numero limitato delle immagini, il fatto che tale operazione verrà ripetuta una sola volta, e la gestione più intuitiva per l'utente finale rendono consigliabile questo metodo, che è anche il più semplice dal punto di vista implementativo. L'alternativa consiste infatti nel predisporre 10 immagini, in cui l'utente inserirà ogni volta le proprie.

All'immagine per il banner si associa un link, di tipo `testo`.

La mappa si inserirà incollando l'embed code fornito dentro una `casella di testo`.

La struttura in formato XML è:

```
<root>
  <dynamic-element name='immagine1' type='image_gallery'>
    <dynamic-element name='gridImg1' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg2' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg3' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg4' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg5' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg6' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg7' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg8' type='image_gallery'></dynamic-element>
    <dynamic-element name='gridImg9' type='image_gallery'></dynamic-element>
```

```
  </dynamic-element>
  <dynamic-element name='features' type='multi-list'>
    <dynamic-element name='213812' type='bilancia'></dynamic-element>
    <dynamic-element name='213817' type='bottiglie'></dynamic-element>
    <dynamic-element name='213822' type='equitazione'></dynamic-element>
    <dynamic-element name='213827' type='fattorie-dida-coldiretti'></dynamic-
    element>
    <dynamic-element name='213832' type='informazioni'></dynamic-element>
    <dynamic-element name='213837' type='letto'></dynamic-element>
    <dynamic-element name='213842' type='MTB'></dynamic-element>
    <dynamic-element name='213847' type='passeggiate'></dynamic-element>
    <dynamic-element name='213852' type='piatto_posate'></dynamic-element>
    <dynamic-element name='213857' type='spuntino'></dynamic-element>
  </dynamic-element>
  <dynamic-element name='immagine2' type='image_gallery'></dynamic-element>
  <dynamic-element name='testo' type='text_area'></dynamic-element>
  <dynamic-element name='pulsanteTerraNostra' type='image_gallery'></dynamic-
    element>
  <dynamic-element name='linkTerraNostra' type='text'></dynamic-element>
  <dynamic-element name='mappa' type='text_box'></dynamic-element>
</root>
```

Implementazione del modello

Occorre divere il 6 blocchi in 3 macroblocchi, disposti su 3 righe, ciascuno con due blocchi affiancati. Questo modello sintetizza tutti i concetti visti finora, e costituisce un esempio significativo di utilizzo dello strumento Strutture e Modelli del Journal Content.

```
<div id="contenitore" style="width: 960px; text-align:left; overflow:hidden;">
   <div id="block1" style="overflow:hidden">
      <div id="colSx1" style="width:49%; max-width:480px; float:left;">
         <div id="imgTop" style="float:left">
         <img src="$immagine1.getData()" width="400px" height="300px">
         </div>
         <div id="imgGrid1" style="float:left; max-width:480px;">
            #foreach ($child in $immagine1.getChildren())
               <div class="grid1" style="float:left; margin: 10px 10px 10px 0px;"
    >
               <img src="$child.getData()" width="120px" height="90px">
               </div>
            #end
         </div>
      </div>
      <div id="colDx1" style="width:49%; max-width:440px; float:left; margin-
     left:20px;">
         <div id="testo" >$testo.getData()</div>
      </div>
   </div>

   <div id="block2" style="overflow:hidden; margin-top:20px; margin-
     bottom:20px;">
      <div id="colSx2" style="width:49%; max-width:480px; float:left;">
      <div id="gridExt" style="display:block; overflow:hidden;">
      #foreach ($opt in $features.getOptions())
```

```
        <div class="grid2" style="float:left; margin-left:0px; margin-right:10px;
    ">
        <img src="http://www.projectmoon.eu:8680/image/image_gallery?
    img_id=$opt" height="45px" >
        </div>
      #end
     </div>
     <div id="legenda">Legenda dei simboli</div>
      </div>
      <div id="colDx2" style="width:49%; max-width:480px; float:right">
        <div id="pulsante" style="float:left;">
          <a href="$linkTerraNostra.getData()">
          <img src="$pulsanteTerraNostra.getData()" height="70px;" width="
    470px">
          </a>
        </div>
      </div>
   </div>

   <div id="block3" style="overflow:hidden">
      <div id="colSx3" style="width:49%; max-width:480px; float:left;">
        <img src="$immagine2.getData()" width="460px" height="320px">
      </div>
      <div id="colDx3" style="width:49%; max-width:480px; float:right;">
        <div id="mappa" style="width:460px; height:320px;">$mappa.getData()<
     /div>
      </div>
   </div>
</div>
```

Analizziamo in particolare il frammento di codice che recupera le icone delle caratteristiche:

```
<div id="gridExt" style="display:block; overflow:hidden;">
   #foreach ($opt in $features.getOptions())
      <div class="grid2" style="float:left; margin-left:0px; margin-right:10px;">
      <img src="http://www.projectmoon.eu:8680/image/image_gallery?img_id
     =$opt" height="45px" >
      </div>
   #end
</div>
```

È sufficiente scorrere gli elementi della lista e recuperare le chiavi scelte dall'utente, e porle alla fine dell'URL delle immagini. Come già detto sopra, tale URL andrà modificato al momento della messa online del sito.

Figura 10: Pagina per singolo agriturismo

4.2 WIZARD PER LA CREAZIONE DI TEMI

4.2.1 Requisiti

Come già detto nel secondo capitolo, nella definizione dei dettagli del wizard viene lasciata molta libertà. Si provvede quindi a definire ulteriori requisiti, che vanno a completare il quadro da cui partire.

Si vuole creare un wizard che permetta ai partners di creare temi grafici per i siti dei loro clienti.

Requisiti generali

Si vuole rendere disponibile il prima possibile una prima versione del wizard che permetta di definire lo stile scelto e crei la struttura di cartelle `_diffs` contenente le modifiche create. Questa prima parte è quella sviluppata nel corso di questo tirocinio.

Una seconda versione del wizard conterrà un automatore che provvederà a creare la .WAR, deployarla e renderla pronta all'uso nel portale.

Il wizard dovrà essere intuitivo nell'uso e semplice da modificare. La necessità di avere una versione funzionante nel minor tempo possibile suggerisce inoltre di tralasciare tutte le funzionalità non indispensabili.

Elementi di un tema

Un tema completo è costituiti da:

1. CSS;
2. immagini;
3. codice Javascript;
4. templates (codice Velocity o XSL).

Per ovvi motivi di sicurezza si decide di lasciare ai fruitori del wizard la possibilità di gestire solamente CSS e immagini.

4.2.2 Analisi funzionale

Caratteristiche dei temi

Gli elementi che occorre salvare per ogni tema sono:

DATI ANAGRAFICI formati da

- nome del tema;
- descrizione;
- gruppo di riferimento cui è rivolto il tema;
- informazioni di *auditing*, ovvero chi ha creato cosa, quando e come;
- gamma cromatica di base, il punto di partenza per le modifiche cromatiche successive, non è vincolante;
- struttura di base, caratterizzerà la quantità di modifiche apportabili, in particolar modo agli sfondi. È vincolante. Non sarà possibile modificare la struttura, per modificarla sarà necessario eliminare il tema e ricrearlo;

VERSIONAMENTO perché ogni tema avrà più versioni, ed agli utenti sarà consentito tornare a versioni precedenti "migliori", ognuna formata da

- numero di versione *major*;
- numero di versione *minor*;
- modifiche ai CSS
  - portlet

  - navigazione
  - configurazione generale
- immagini
  - header
  - footer
  - sfondi
  - ...

Struttura

La struttura di un tema vincola principalmente gli sfondi, e può essere di 3 tipi:

BOXED semplice customizzazione;
EXTENDED HEADER customizzazione avanzata;
MULTI-BACKGROUND pieno controllo sugli sfondi.

Attributi editabili

Altri elementi su cui si può intervenire:

- personal bar, esclusivamente bianco su nero o viceversa;
- colori e arricchimenti di testo e link;
- elementi della navigazione;
- header e footer;
- aspetto dei portlet;
- margini, bordi e padding;
- maschere di inserimento dati;

Dove possibile le modifiche delle dimensioni sono espresse in *em*, come da default del template Pjoon V3.

Le versioni

Ogni versione conterrà le modifiche incrementali apportate durante una sessione di lavoro sul tema. Il wizard dovrà produrre le modifiche adeguate nella struttura `_diffs`.

La cancellazione di un tema lo rimuove dalla lista dei temi creati dall'utente, ma lo conserva nel database.

Nomenclatura dei temi

Il nome di un tema è normalmente composto da due parti:

- una prima parte che indica la destinazione del tema (club, company, community, ...), d'ora in poi `prefix`;
- il nome vero e proprio, ovvero quello che lo identifica, in genere il nome del cliente/destinatario, separato dal prefix con un underscore (d'ora in poi `nome1`).

Il tema verrà salvato in una cartella il cui nome sarà composto dal nome del tema seguito da -theme, ovvero `prefix_nome1-theme`.

I caratteri ammissibili per il nome di un tema sono:

- lettere minuscole;
- numeri;
- trattini ($-$);
- underscore (_).

Vengono quindi imposti ulteriori vincoli sul nome:

- il `nome1` di un tema non può iniziare con un trattino o un underscore;
- il `nome1` non può contenere parole o espressioni volgari o offensive;
- un utente non può creare due temi con lo stesso nome;
- 20 caratteri per nome1 costituiscono un numero di caratteri sufficiente a garantire ampia libertà di scelta agli utenti.

Essendo ciascun tema creato per uno ed un solo cliente, cade la necessità del `prefix` per distinguere la categoria di destinazione. Può invece esser utile tener traccia anche nel nome del fatto che il tema è stato creato mediante il wizard. Pertanto, all'inizio del nome si sceglie di apporre il prefisso `wz_`. Sommando a ciò il vincolo di unicità sulla coppia `(partner,nome1)` si ottiene il nome completo `wz_IDpartner_nome1`, che risulta quindi facilmente riconoscibile anche ad occhio, grazie alla presenza di `nome1`. Il nome della cartella del tema sarà quindi `wz_IDpartner_nome1-theme`.

I temi eliminati (che vengono solamente "nascosti" al loro creatore, senza venire fisicamente rimossi dal database) continuano ad "occupare" il nome presso il partner che li ha creati.

Struttura del wizard

Maschera iniziale

Ci sarà una maschera iniziale che visualizzerà i temi creati dall'utente, consentendo di

- creare un nuovo tema;
- modificare un tema già esistente;
- pubblicare un tema già esistente;
- eliminare un tema;
- uscire.

Anagrafica

Scegliendo di creare un nuovo tema si arriverà ad una maschera, che chiameremo `anagrafica`, che chiederà di inserire l'anagrafica del tema, la struttura e lo schema base dei colori. Quindi si potrà agire sui due pulsanti disponibili:

SALVA per salvare il tema e proseguire con la modifica degli attributi;
CANCELLA per uscire senza salvare il tema.

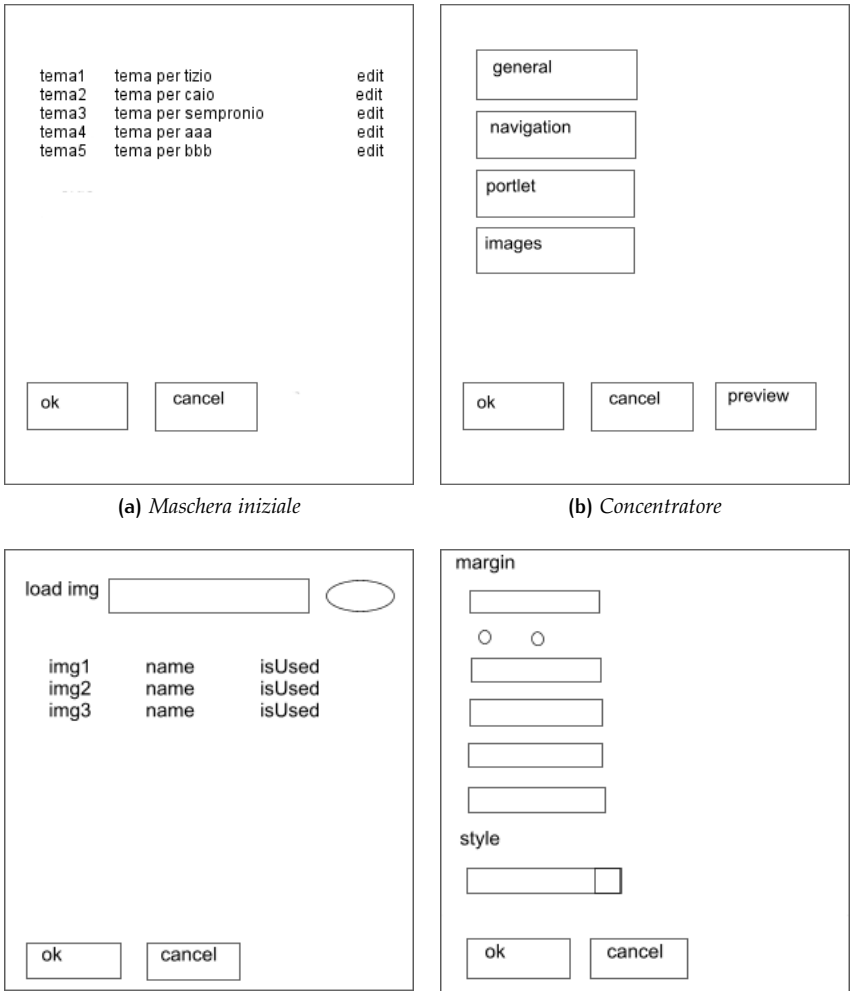
Visto come agire quando si sceglie di pubblicare un tema e quando invece si vuole cancellarne uno, vediamo come il wizard si comporterà quando si modifica un tema esistente, oppure una volta inserita l'anagrafica di un nuovo tema. Una volta inserita l'anagrafica di un tema, o quandi si sceglie di modificare un tema già esistente, si arriverà ad una finestra chiamata `concentratore`, che sarà la base da cui si partirà per compiere tutte le modifiche al tema base. Il concentratore conterrà i pulsanti per agire su:

Concentratore

- immagini;
- aspetto generale;
- navigazione;
- aspetto dei portlet.

Non sarà possibile modificare i dati anagrafici, audit e la struttura. I pulsanti a disposizione saranno:

SALVA per salvare una versione;



(a) *Maschera iniziale*

(b) *Concentratore*

(c) *Caricamento immagini*

(d) *Modifica stili*

Figura 11: Mockup delle diverse maschere del wizard, da usare come linee guida durante lo sviluppo dell'interfaccia

ANTEPRIMA per visualizzare una pagina di anteprima del sito con il nuovo tema;
CHIUDI per terminare le modifiche.

Maschere di inserimento modifiche

Vediamo ora quale sarà l'aspetto delle finestre di inserimento modifiche.

La maschera per il caricamento delle immagini conterrà il campo per il caricamento vero e proprio dal computer dell'utente, e la lista delle immagini caricate.

Le altre maschere conterranno i campi di input per le modifiche, mostrando come default le modifiche apportate nelle precedenti versioni, e consentendo di annullare le modifiche inserite in un campo.

I pulsanti per le azioni saranno:

SALVA per salvare le modifiche apportate;
CANCEL per chiudere la finestra senza salvare le modifiche.

In un secondo step, introdotto in una versione successiva del wizard, le finestre per l'inserimento delle modifiche saranno divise in due parti: modifiche base e modifiche avanzate. Farà ovviamente eccezione la finestra per il caricamento delle immagini.

Selettori ed attributi

I selettori editabili saranno i seguenti:

- body
- #background-wrapper-1
- #background-wrapper-2
- #background-wrapper-3
- #background-wrapper-4
- #header
- #body-page
- #footer
- #navigation
- #navigation li a
- #navigation li a:hover
- #navigation li a:active
- #navigation li a:visited
- #navigation li.selected a
- #navigation li.selected a:hover
- #navigation li.selected a:active
- #navigation li.selected a:visited
- a
- a:link
- a:hover
- a:active
- a:visited
- input[type=submit]
- input[type=button]
- input[type=reset]

Gli attributi modificabili invece saranno:

- margin

- margin-custom
- margin-top
- margin-right
- margin-bottom
- margin-left
- padding
- padding-custom
- padding-top
- padding-right
- padding-bottom
- padding-left
- border-width
- border-width-custom
- border-top-width
- border-right-width
- border-bottom-width
- border-left-width
- border-style
- border-style-custom
- border-top-style
- border-right-style
- border-bottom-style
- border-left-style
- border-color
- border-color-custom
- border-top-color
- border-right-color
- border-bottom-color
- border-left-color
- background-image
- background-color
- background-position-x
- background-position-y
- background-repeat
- font-size
- color
- text-align
- text-transform
- line-height
- letter-spacing
- font-style
- font-weight
- font-variant

dove gli attributi *custom* sono attributi "di servizio" che specificano se l'utente vole usare uno stile unico per i 4 lati del selettore selezionato, o uno stile diverso per ciascun lato.

Chiaramente gli attributi andranno associati ai selettori cui sono riferiti. A loro volta, i selettori andranno associati alla maschera cui appartengono (generale, navigazione, portlet). Il risultato di questa mappatura costituirà l'insieme di campi editabili in una maschera.

Per ogni attributo abbiamo bisogno di almeno due informazioni: valore e tipo. Il valore potrà essere un qualsiasi valore valido inserito dall'utente, mentre il tipo sarà vincolato all'attributo. Ad esempio, `margin-width` potrà essere di tipo `em`, `px` o `%`, non certamente `rigth` o `dotted`, tipi che potranno essere associati rispettivamente a `text-align` e `border-style`.

4.2.3 Analisi tecnica

Temi, versioni e stili

Ogni partner potrà creare temi in numero non limitato. Ciascun tema può variare nel tempo, ma in ogni istante ci sarà al massimo una sola versione usabile (o nessuna, se il tema è stato appena creato e nessuna sua versione è stata pubblicata).

Se ogni tema avrà in un dato istante al massimo una sola versione usabile, vogliamo anche tenere uno storico di tutte le versioni pubblicate. Deve essere inoltre possibile lavorare su un tema in tempi differenti, in maniera da consentire modifiche, miglioramenti, correzioni, ripensamenti.

Si sceglie allora di associare a ciascuna versione una coppia di numeri che la identifichi tra le versioni del tema, e ne rispecchi lo stato (ovvero, se pubblicato o in lavorazione). Il primo numero di versione, detto *major*, viene associato ad una versione pubblicata, mentre il secondo numero, *minor*, ne rispecchia l'avanzamento passo passo. Una versione pubblicata corrisponde all'ultima minor version prodotta.

Versioni major *e* minor

Al momento del salvataggio di una serie di modifiche verrà incrementato il minor della versione, rispetto alla versione da cui si è partiti. Al momento della pubblicazione, verrà invece incrementato il major, mentre il minor verrà azzerato. Ciascuna modifica ulteriore partirà quindi dalla nuova major version. Non sarà possibile modificare temi partendo da versioni più vecchie dell'ultima minor version. Quindi, se ad esempio la versione con coppia major,minor (4,9) viene pubblicata, i suoi numeri di versione diventano (5,0), e le versioni precedenti ad essa diventano non più riutilizzabili (anche se ancora presenti in archivio).

Le minor versions sono quindi solo step intermedi della produzione del tema, contenenti modifiche che verranno consolidate e rese definitive nella major successiva. Al momento della pubblicazione di una versione è quindi possibile eliminare tutte le minor version impiegate, senza perdita di informazioni.

Sia le minor version che le major version conterranno tutti gli stili modificabili, in maniera da rendere più immediata la generazione del tema al momento della pubblicazione.

Versioni di draft

Anche le versioni minor potranno variare nel corso della loro creazione. In particolare, nel passaggio da una finestra di modifiche ad un altra.

La trasmissione di informazioni da una finestra di modifiche al concentratore è un procedimento complicato. La maniera più semplice per gestire le modifiche parziali è quella di salvare queste ultime in versioni temporanee, o versioni di *draft*, nel database, identificandole come tali.

Una versione di draft conterrà solo ed esclusivamente gli attributi modificati nella sessione di lavoro. Ogni tema in ogni istante avrà al massimo una ed una sola versione di draft. Questo permette di identificare la draft assegnandole valori major e minor convenzionali di -1. Nel caso si continuassero ad apportare modifiche ad una draft, queste verrebbero inglobate nella medesi-

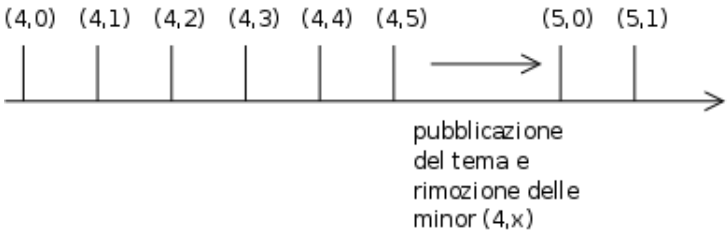


Figura 12: Andamento temporale delle versioni

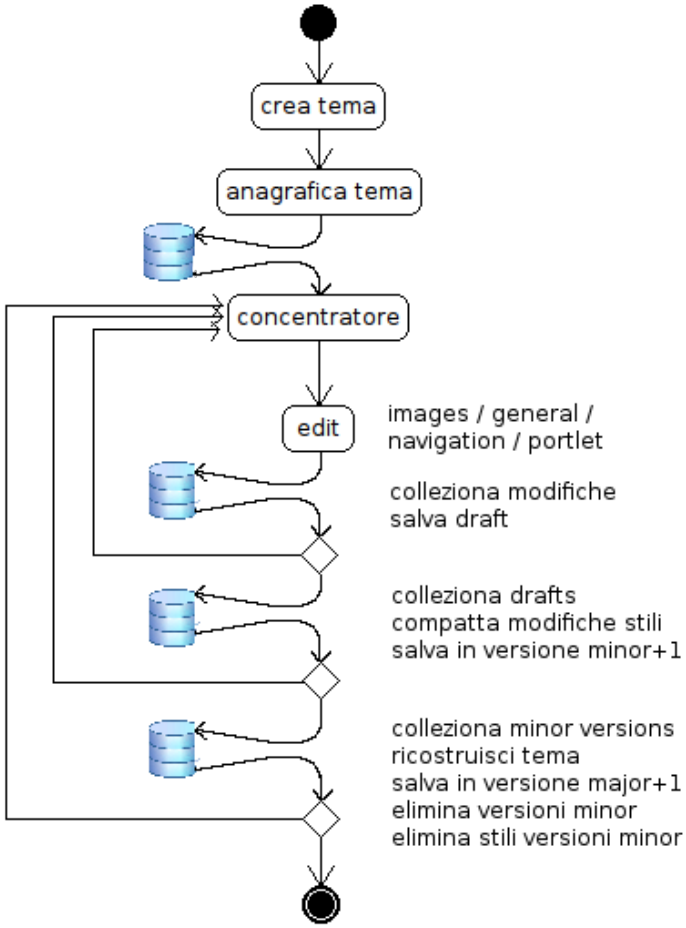


Figura 13: Vita di un tema: dalla creazione al ciclo di lavorazione sulle versioni

ma draft, in quanto le situazioni che si presenterebbero sarebbero le seguenti due:

1. modifica di uno stile non editato in precedenza nella draft, nel qual caso lo stile verrebbe aggiunto alla draft;
2. modifica di uno stile già editato in precedenza nella draft, la quale modifica andrebbe a sovrascrivere la precedente.

Al momento della modifica di una draft, occorre visualizzare un messaggio che avverta che si sta modificando una versione già in corso di editing.

La prima versione del wizard consentirà solo di modificare l'ultima versione. Non saranno quindi possibili versioni "in parallelo", ovvero versioni diverse con la stessa versione di partenza. L'integrazione di questa funzionalità è prevista per le successive verzioni del wizard.

Stili

Uno stile sarà un elemento atomico di una versione. Sarà riferito ad un solo selettore ed un solo attributo. Come già detto, ad ogni stile saranno associati valore e tipo. A queste informazioni affianchiamo un campo che conterrà il valore dello sfondo dell'elemento, da usare nel caso l'elemento sia uno sfondo (ad esempio, in `body`, `background-wrapper` o `border`,). Lo sfondo potrà essere un colore o l'ID di un'immagine tra quelle a disposizione per il tema. Per distinguere tra le due possibilità, si userà il tipo.

Modifiche base e avanzate

Analizziamo quindi in dettaglio come verranno gestiti i passaggi tra modifiche base e modifiche avanzate.

Gli elementi di testo, gli sfondi e gli input avranno lo stesso grado di configurazione. Cambieranno invece le possibilità di customizzazione riguardanti margini, padding e bordi, che da comportamento unico sui 4 lati (base) passeranno ad avere un comportamento per ogni lato. Avremo quindi 6 attributi in totale:

- attributo base;
- custom: modifiche avanzate;
- attributo top;
- attributo left;
- attributo bottom;
- attributo right.

Di volta in volta si analizzeranno gli attributi desiderati, usando l'attributo custom come discriminante.

Gestione delle immagini

La gestione delle immagini in Projectmoon System è centralizzata. Questo vuol dire che nel database un'unica tabella contiene tutte le immagini di tutti i portlet.

Per gestire le immagini del wizard creeremo una tabella in cui verranno salvati gli ID delle immagini caricate durante la lavorazione sui temi, ID a cui associeremo l'ID del tema cui si riferiscono, e il nome dell'immagine.

Vorremo inoltre sapere se l'immagine è usata.

Gestione degli utenti

Anche per la gestione degli utenti ci affidiamo agli strumenti integrati nella piattaforma.

Nel database sono già presenti le tabelle con i dati degli utenti e dei partner, e sono di conseguenza già a disposizione i metodi per accedervi. Per sfruttarli, durante la creazione delle tabelle di cui avremo bisogno andremo ad impostare l'import del package per la gestione degli utenti.

Ricostruzione del tema

Il wizard dovrà compiere in totale due ricostruzioni:

1. da versione di draft a minor version, alla pressione del tasto "save" del concentratore;
2. da major version a tema completo, in fase di pubblicazione.

Da versione di draft a minor version

La versione draft contiene solamente gli stili modificati durante l'ultima sessione di lavoro. È comunque possibile risalire alla versione di partenza, e sfrutteremo questa informazione per ricostruire la minor version, che a differenza della draft contiene tutti i selettori e gli attributi.

Scorriamo tutti i selettori, e, per ogni selettore, gli attributi che lo caratterizzano. Controlliamo, per ogni stile così definito, se esso è presente nella draft. In caso affermativo, lo inseriamo nella nuova minor, altrimenti andiamo a inserire lo stile corrispondente presente nella minor di partenza. Fatto ciò salviamo la minor creata, ed eliminiamo la draft con i suoi stili.

Da major version a tema completo

In fase di pubblicazione invece dobbiamo generare il tema completo, ovvero, nel nostro caso, CSS e immagini.

Procediamo come nel caso precedente, scorrendo selettori e attributi. In questo caso, però, non potremo importare tutti gli stili così come si trovano, ma dovremo prestare attenzione a:

- selezionare solo gli stili effettivamente modificati;
- distinguere tra stili foreground e background;
- distinguere tra i vari tipi assegnati agli stili, e gestire il relativo output di conseguenza.

Quindi, per ogni stile:

1. controlliamo se è stato modificato;
2. se è un attributo custom, controlliamo il suo valore. Se non selezionato, inseriamo lo stile "completo", altrimenti andremo ad inserire i 4 stili "custom";
3. se l'attributo prevede uno sfondo, controlliamo se lo sfondo è settato, e se è un colore o un'immagine;
4. a questo punto inseriamo valore, background e tipo, a seconda di cosa richiede lo stile, e di cosa è stato settato dall'utente.

Formattiamo il risultato e inseriamolo in un file chiamato `custom.css`, che inseriremo in una struttura di cartelle `_diffs`.

Per la creazione dell'archivio compresso da inviare a Projectmoon viene usata la classe ZipWriter di Liferay, al cui interno inseriremo la cartella `diffs` contenente il file `custom.css` creato e le immagini del tema.

L'anteprima

Vediamo ora come procedere per l'anteprima del tema. Al click sul relativo pulsante si apre una nuova finestra che visualizza una pagina statica ma rappresentativa, vale a dire contenente il maggior numero possibile di elementi modificabili.

La finestra inietterà all'interno della pagina statica lo stile ricostruito al momento, recuperando l'ultima minor creata. Per questa ragione la pagina statica non conterrà `head`, ma solo `body`, tag escluso.

Base di dati

Analizziamo ora quale sarà la base di dati che il wizard andrà ad utilizzare. È fondamentale ricordare che la piattaforma utilizza un proprio service builder e framework come Hibernate e Spring per la gestione del proprio database, cosa di cui l'analisi dovrà tener conto.

Entità

Essendo la gestione degli utenti e dei partner già presente nel database, le entità che dovremo creare sono le seguenti:

- Temi
- Versioni_tema
- Stili
- Immagini

Esse derivano direttamente dai requisiti e dalle analisi funzionale e tecnica svolte.

Oltre agli attributi definiti precedentemente, definiremo un ID chiave primaria per ogni entità.

Definiremo anche i campi su cui andremo ad effettuare ricerche, l'ordine delle tuple e le risorse che sfrutteremo (gestione dei contatori, gestione degli utenti, ...). Ciò si rende necessario dal momento che è su queste informazioni che il service builder costruisce i metodi per inserimento, recupero e manipolazione dei dati.

TWTheme

La creazione del database consiste nella compilazione del seguente file `service.xml`, che andiamo ad analizzare.

```
<?xml version="1.0"?>
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder
    5.1.0//EN" "http://www.liferay.com/dtd/liferay-service-
    builder_5_1_0.dtd">

  <service-builder package-path="it.prjmoon.portlet.themewizard">
  <namespace>TW</namespace>

  <!-- Themes -->

  <entity name="TWTheme" local-service="true" remote-service="
      false">

    <!-- PK fields -->

    <column name="themeId" type="long" primary="true" />

    <!-- Audit fields -->
```

```
<column name="companyId" type="long" />
<column name="userId" type="long" />
<column name="userName" type="String" />
<column name="createDate" type="Date" />
<column name="modifiedDate" type="Date" />

<!-- Other fields -->

<column name="name" type="String"/>
<column name="description" type="String" />
<column name="limitedGroupId" type="long" />
<column name="structureType" type="int"/>
<column name="scopeType" type="int"/>
<column name="baseScheme" type="int"/>
<column name="creatorPartnerId" type="long"/>
<column name="creatorUserId" type="long" />
<column name="publishedVersionId" type="long" />

<!-- Relationships -->

<column name="versions" type="Collection" entity="TWVersion"
    mapping-key="themeId" />

<!-- Order -->

<order by="desc">
   <order-column name="createDate" />
</order>

<!-- Finder methods -->

<finder name="CP_CU" return-type="Collection">
   <finder-column name="creatorPartnerId" />
   <finder-column name="creatorUserId" />
</finder>

<finder name="CreatorPartnerID" return-type="Collection">
   <finder-column name="creatorPartnerId" />
</finder>

<finder name="LimitedGroupId" return-type="Collection">
   <finder-column name="limitedGroupId" />
</finder>

<!-- References -->

<reference package-path="com.liferay.counter" entity="Counter
    " />
<reference package-path="com.liferay.portal" entity="Image" /
    >
<reference package-path="com.liferay.portal" entity="Resource
    " />
```

```
    <reference package-path="com.liferay.portal" entity="User" />

</entity>
```

Abbiamo creato l'entità `TWTheme`, con attributi

- themeId `long` chiave primaria;

i campi di audit

- companyId `long` ID del partner;
- userId `long` ID dell'utente che ha creato il tema;
- userName `String` nome dell'utente che ha creato il tema;
- createDate `Date` data di creazione;
- modifiedDate `Date` data di modifica;

quindi gli attributi che abbiamo ricavato durante l'analisi

- name `String` il nome del tema;
- description `String` breve descrizione del tema;
- limitedGroupId `long` gruppo di riferimento del tema;
- structureType `int` struttura del tema;
- scopeType `int` destinazione del tema;
- baseScheme `int` schema colori base;
- creatorPartnerId `long` ID del partner;
- creatorUserId `long` ID dell'utente che ha creato il tema;
- publishedVersionId `long` ID della versione del tema attualmente pubblicata.

Ricordiamo che il tipo di dato non è tipo di dato SQL ma Java. Sarà compito del service builder effettuare la conversione.

Segue la definizione della chiave esterna nella tabella delle versioni, specificando che l'ID del tema sarà chiave esterna nella tabella delle versioni.

Quindi specifichiamo in che ordine vogliamo siano inserite le tuple. Nel nostro caso, ordine discendente per data di creazione.

Nella sezione `Finder` definiamo i campi su cui andremo ad effettuare ricerche. Cercheremo i temi in 3 maniere:

1. per utente o partner creatore;
2. per partner creatore;
3. per gruppo di riferimento.

Vedremo in seguito come sarà possibile raffinare le ricerche, ma la base saranno i metodi generati dal service builder su questi e `finder`.

Si specificano infine quali sono i package da importare: `Counter`, `User`, `Image` e `Resource`, che consentiranno di gestire i contatori, gli utenti e le immagini.

Immagini

```
<entity name="TWThemeImage" local-service="true" remote-service=
    "false">

   <!-- PK Fields  -->

   <column name="themeImageId" type="long" primary="true" />
```

```
    <!-- Key fields -->

    <column name="themeId" type="long" />
    <column name="imageId" type="long" />
    <column name="name" type="String" />
    <column name="isUsed" type="boolean" />

    <!-- Order -->

    <order by="desc">
       <order-column name="themeId" />
       <order-column name="isUsed" />
       <order-column name="imageId" />
    </order>

    <!-- Finder methods -->

    <finder name="ThemeId" return-type="Collection">
       <finder-column name="themeId" />
    </finder>

</entity>
```

L'entità `TWThemeImage` lega le immagini al tema per cui sono state caricate. Vediamone gli attributi.

- themeImageId `long` chiave primaria;
- themeId `long` ID del tema;
- imageId `long` ID dell'immagine (nella tabella immagini comune a tutto il database);
- name `String` nome dell'immagine;
- isUSed `boolean` indica se l'immagine è usata;

Le immagini vengono ordinate per

1. ID del tema;
2. immagine usata o no;
3. ID dell'immagine.

Si effettueranno ricerche per tema.

Versioni

```
<!-- Themes versions -->

<entity name="TWVersion" local-service="true" remote-service="
    false">

    <!-- PK fields -->

    <column name="versionId" type="long" primary="true" />

    <!-- Audit fields -->
```

```
        <column name="userId" type="long" />
        <column name="userName" type="String" />
        <column name="createDate" type="Date" />

        <!-- Key fields -->

        <column name="themeId" type="long" />
        <column name="parentVersionId" type="long" />

        <!-- Other fields -->

        <column name="versionMajor" type="int" />
        <column name="versionMinor" type="int"/>

        <!-- Relationships -->

        <column name="styles" type="Collection" entity="TWStyle"
            mapping-key="versionId" />

        <!-- Order -->

        <order by="desc">
           <order-column name="themeId" />
           <order-column name="createDate" />
        </order>

        <!-- Finder methods -->

        <finder name="ThemeId" return-type="Collection">
           <finder-column name="themeId" />
        </finder>
        <finder name="ParentVersionId" return-type="Collection">
           <finder-column name="parentVersionId" />
        </finder>
        <finder name="T_M" return-type="Collection">
           <finder-column name="themeId" />
           <finder-column name="versionMajor" />
        </finder>
        <!-- easy way to get genealogy -->
        <finder name="T_M_mM" return-type="Collection">
           <finder-column name="themeId" />
           <finder-column name="versionMajor" />
           <finder-column name="versionMinor" comparator="&lt;=" />
        </finder>

        <!-- References -->

        <reference package-path="com.liferay.counter" entity="Counter
            " />
        <reference package-path="com.liferay.portal" entity="Resource
            " />
        <reference package-path="com.liferay.portal" entity="User" />
```

```
</entity>
```

L'entità `TWversion` contiene le versioni. I suoi attributi sono:

- versionId `long` chiave primaria;

i campi di audit

- userId `long` ID dell'utente che ha creato il tema;
- userName `String` nome dell'utente che ha creato il tema;
- createDate `Date` data di creazione;

quindi gli altri attributi che abbiamo ricavato durante l'analisi

- themeId `long` il tema cui riferiscono;
- parentVersionId `long` la versione da cui derivano;
- versionMajor `int` numero di versione major;
- versionMinor `int` numero di versione minor;

Viene inoltre specificato che nella tabella degli stili vi sarà un campo che riferisce alla tabella delle versioni.
Le versioni verranno ordinate per tema e data di creazione.
Si effettueranno ricerche per

1. tema;
2. versione di partenza;
3. versionMajor di un tema;
4. versioni di un tema appartenenti alla stessa major e precedenti ad una data minor di partenza.

Quest'ultima ricerca servirà a ricostruire la genealogia quando saranno consentite modifiche in parallelo alle versioni.
Infine, importiamo i package che consentiranno di gestire utenti e contatori.

Stili

```
<!-- incremental edits -->

<entity name="TWStyle" local-service="true" remote-service="
    false">

   <!-- PK fields -->

   <column name="styleId" type="long" primary="true"/>

   <!-- Key fields -->

   <column name="versionId" type="long"/>
   <column name="parentVersionId" type="long"/>

   <!-- Other fields -->

   <column name="selector" type="int" />
   <column name="attribute" type="int" />
   <column name="type" type="int"/>
```

```
        <column name="value" type="Double"/>
        <column name="background" type="long"/> <!-- imageId XOR 0x
            to dec -->

        <!-- Finder methods -->

        <finder name="VersionId" return-type="Collection">
           <finder-column name="versionId" />
        </finder>

        <finder name="V_S" return-type="Collection">
           <finder-column name="versionId" />
           <finder-column name="selector" />
        </finder>

        <finder name="V_S_A" return-type="Collection">
           <finder-column name="versionId" />
           <finder-column name="selector" />
           <finder-column name="attribute" />
        </finder>

        <finder name="V_PV_S_A" return-type="TWStyle" unique="true">
           <finder-column name="versionId" />
           <finder-column name="parentVersionId" comparator="&gt;" />
           <finder-column name="selector" />
           <finder-column name="attribute" />
        </finder>

        <!-- References -->

        <reference package-path="com.liferay.counter" entity="Counter
            " />

    </entity>
</service-builder>
```

L'entità `TWStyle` contiene gli stili. I suoi attributi sono:

- styleId `long` chiave primaria;
- versionId `long` la versione cui appartengono;
- parentVersionId `long` la versione da cui derivano;
- selector `int` il selettore;
- attribute `int` l'attributo;
- type `int` il tipo del valore/background;
- value `long` il valore dell'attributo;
- background `long` il colore di sfondo, o l'ID dell'immagine di sfondo.

Si effettueranno ricerche per

1. versione;
2. versione e selettore;
3. versione, selettore e attributo;

4. versione, versioni di riferimento più recenti di una data parentVersion, selettore e attributo.

Sarà così possibile risalire a tutti gli stili possibili, per costruire la versione completa.
Infine, importiamo il package per la gestione dei contatori.

Ora ci occupiamo della mappatura tra finestre, selettori, attributi e tipi.
Questa parte non entra nel database ma viene gestita mediante array di costanti, in quanto la mappatura presumibilmente non varierà nel corso del tempo. Dividiamo selettori, attributi e tipi in gruppi da massimo 10 elementi, eventualmente saltando qualche elemento dell'array. Questo facilita il riconoscimento degli abbinamenti.
Per i selettori definiremo il seguente schema:

1..10 body, header, footer, sfondi;
11..20 navigation bar;
21..30 aspetto dei portlet;
31..40 campi di input e altri elementi;

mentre per quanto riguarda gli attributi ci rifaremo alla seguente regola:

11..20 ampiezza dei margini;
21..30 padding;
31..40 larghezza dei bordi;
41..50 colore dei bordi;
51..60 stile dei bordi;
61..70 sfondi;
71..80 proprietà del testo.

Ora non resta che associare gli attributi ai selettori cui riferiscono: *Costanti selettori-attributi*

```
public static final int[][] SELECTORS_ATTRIBUTES = {
   {},
   {61,62,63,64,65,71,72,73,74,75,76,77,78,79}, //body
   {61,62,63,64,65}, //#background-wrapper-1
   {61,62,63,64,65}, //#background-wrapper-2
   {61,62,63,64,65}, //#background-wrapper-3
   {61,62,63,64,65}, //#background-wrapper-4
   {61,62,63,64,65}, //#header
   {61,62,63,64,65}, //#body-page
   {61,62,63,64,65}, //#footer
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,35,36,41,42,
      43,44,45,46,51,52,53,54,55,56,61,62,63,64,65},// #navigation
   {},

   /*11..20*/
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,72,73,74,75,76,77,78,79},//#navigation li a
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,72,73,74,75,76,77,78,79},//#navigation li a:
          hover
```

```
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,72,73,74,75,76,77,78,79},// #navigation li a:
          active
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,72,73,74,75,76,77,78,79},//#navigation li a:
          visited
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,72,73,74,75,76,77,78,79},//#navigation li.
          selected a
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,
      72,73,74,75,76,77,78,79},// #navigation li.selected a:hover
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,
      72,73,74,75,76,77,78,79}, // #navigation li.selected a:active
   {11,12,13,14,15,16,21,22,23,24,25,26,31,32,33,34,
      35,36,41,42,43,44,45,46,51,52,53,54,55,56,61,
      62,63,64,65,71,72,73,74,75,76,77,78,79},// #navigation li.
          selected a:visited
   {},{},

   /*21..30*/
   {71, 72, 73, 74, 75, 76, 77, 78, 79 }, //a
   {71, 72, 73, 74, 75, 76, 77, 78, 79 }, //a:link
   {71, 72, 73, 74, 75, 76, 77, 78, 79 }, //a:hover
   {71, 72, 73, 74, 75, 76, 77, 78, 79 }, //a:active
   {71, 72, 73, 74, 75, 76, 77, 78, 79 }, //a:visited
   {},{},{},{},{},

   /*31..33*/
   {51,62,72}, //input[type=submit]
   {51,62,72}, //input[type=button]
   {51,62,72}, //input[type=reset]
};
```

Costanti dei tipi

Vediamo ora i possibili tipi che gli attributi possono assumere:

```
public static final String[] TYPES = {
   "",
   "px","em","%",
   "url",
   "transparent",
   "no-repeat",
   "repeat-x",
   "repeat-y",
   "repeat",
   "solid",
```

```
    "dotted",
    "dashed",
    "left",
    "right",
    "center",
    "uppercase",
    "lowercase",
    "small–caps",
    "normal",
    "bold",
    "italic",
    "boolean"
};
```

e associamo i tipi agli attributi:

Costanti attributi-tipi

```
public static final int[][] ATTRIBUTE_TYPES = {
    {},{},{},{},{},{},{},{},{},{},{},

    /*11..20*/
    {1,2,3}, // margin
    {22}, // margin-custom
    {1,2,3}, // margin-top
    {1,2,3}, // margin-right
    {1,2,3}, // margin-bottom
    {1,2,3}, // margin-left
    {},{},{},{},

    /*21..30*/
    {1,2,3}, // padding
    {22}, // padding-custom
    {1,2,3}, // padding-top
    {1,2,3}, // padding-right
    {1,2,3}, // padding-bottom
    {1,2,3}, // padding-left
    {},{},{},{},

    /*31..40*/
    {1,2,3}, // border-width
    {22}, // border-width-custom
    {1,2,3}, // border-top-width
    {1,2,3}, // border-right-width
    {1,2,3}, // border-bottom-width
    {1,2,3}, // border-left-width
    {},{},{},{},

    /*41..50*/
    {10,11,12}, // border-style
    {22}, // border-style-custom
    {10,11,12}, // border-top-style
    {10,11,12}, // border-right-style
    {10,11,12}, // border-bottom-style
    {10,11,12}, // border-left-style
```

```
   {},{},{},{},

   /*51..60*/
   {0}, // border-color
   {22}, // border-color-custom
   {0}, // border-top-color
   {0}, // border-right-color
   {0}, // border-bottom-color
   {0}, // border-left-color
   {},{},{},{},

   /*61..70*/
   {4}, // background-image
   {0,5}, // background-color
   {1,2,3}, // background-position-x
   {1,2,3}, // background-position-y
   {6,7,8,9}, // background-repeat
   {},{},{},{},{},

   /*71..80*/
   {1,2,3}, // font-size
   {0}, // color
   {13,14,15}, // text-align
   {16,17}, // text-transform
   {1,2,3}, // line-height
   {1,2,3}, //letter-spacing
   {19,20}, //font-style
   {19,21}, //font-weight
   {18,19}, //font-variant
   {},
};
```

Costanti maschera-selettori

Infine, associamo i selettori alle maschere di inserimento dati:

```
public static final int[][] EDITABLE_PROPERTIES = {
   {1,2,3,4,5,6,7,8}, //general
   {9,11,12,13,14,15,16,17,18}, //navigation
   {21,22,23,24,25,31,32,33}, //portlet
};
```

Per comodità definiamo anche altri due array di costanti, che indicano se l'attributo ha sfondi, e se dipende da un booleano (attributi custom).

Essendo solo array "di appoggio" per semplificare la scrittura del codice non vengono qui riportati per esteso.

Costanti delle versioni

Riportiamo invece le costanti di cui abbiamo bisogno nella gestione delle versioni.

```
public static final int UNPUBLISHED_THEME_VERSION_ID = -1;
public static final int INITIAL_VERSION_MAJOR_NUMBER = 0;
public static final int INITIAL_VERSION_MINOR_NUMBER = 0;
public static final long NO_PARENT_VERSION_ID = 0;
```

```
public static final int NO_VERSION_MAJOR_NUMBER = -1;
public static final int VERSION_MINOR_INCREMENT = 1;
public static final int VERSION_MAJOR_INCREMENT = 1;
public static final int DRAFT_MAJOR_NUMBER = -1;
public static final int DRAFT_MINOR_NUMBER = -1;
public static final int PREVIEW_MAJOR_NUMBER = -2;
public static final int PREVIEW_MINOR_NUMBER = -2;
```

4.2.4 Implementazione

Creazione del portlet

Il primo passo per la creazione del portlet è posizionarsi nella cartella root dell'ambiente di lavoro `/home/liferaay/pjoon21x/workspace/plugins/portlets/` e lanciare lo script `bash create.sh nome-del-portlet descrizione-del-portlet`, che provvederà a creare la struttura di cartelle del portlet, popolandola con i file base per il suo sviluppo (`service.xml`, `view.jsp`, ...).

Andiamo quindi a inserire nel file `portlet.xml` i parametri necessari all'identificazione del portlet da parte della piattaforma e all'impostazione dei permessi d'uso. Al momento impostiamo dei permessi generici, al momento della messa online del portlet essi verranno corretti.

```
<?xml version="1.0"?>

<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
    app_2_0.xsd" version="2.0" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml
    /ns/portlet/portlet-app_2_0.xsd http://java.sun.com/xml/ns/
    portlet/portlet-app_2_0.xsd">
   <portlet>
      <portlet-name>theme_wizard</portlet-name>
      <display-name>Theme Wizard</display-name>
      <portlet-class>it.prjmoon.portlet.themewizard.portlet.
          ThemeWizardPortlet</portlet-class>
      <init-param>
         <name>view-jsp</name>
         <value>/html/theme_wizard/view.jsp</value>
      </init-param>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
      </supports>
      <portlet-info>
         <title>Theme Wizard</title>
         <short-title>Theme Wizard</short-title>
         <keywords>Theme Wizard</keywords>
      </portlet-info>
      <security-role-ref>
         <role-name>administrator</role-name>
      </security-role-ref>
      <security-role-ref>
         <role-name>guest</role-name>
```

```
      </security-role-ref>
      <security-role-ref>
         <role-name>power-user</role-name>
      </security-role-ref>
      <security-role-ref>
         <role-name>user</role-name>
      </security-role-ref>
   </portlet>
</portlet-app>
```

Nel file `liferay-display.xml` invece impostiamo la categoria del portlet, cioè il gruppo in cui sarà contenuto nella finestra `aggiungi applicazioni` del portale. Nel nostro caso, la categoria è `admin`.

```
<?xml version="1.0"?>
<!DOCTYPE display PUBLIC "-//Liferay//DTD Display 5.1.0//EN" "http:
    //www.liferay.com/dtd/liferay-display_5_1_0.dtd">

<display>
   <category name="category.admin">
      <portlet id="1" />
   </category>
</display>
```

Generazione del database

Una volta redatto il file `service.xml` si potrà lanciare il comando `ant build -service` che provvederà a generare le tabelle, i metodi `get` e `set` per manipolare gli attributi e i metodi per la ricerca sui campi `finder` che abbiamo specificato. Verranno inoltre create le classi in cui definiremo a nostra volta i metodi per interfacciarci con le primitive create dalla piattaforma, metodi che, in linea di principio, verranno messi a disposizione di sviluppatori che non hanno creato la base di dati, e a cui non devono poter accedere direttamente.

Questo permetterà di lavorare sulla base di dati senza aver, sostanzialmente, scritto una sola riga di codice SQL. Sarà tuttavia possibile, per chi desiderasse, analizzare o modificare il codice generato dal service builder.

Nel caso ci si accorgesse di aver commesso errori, o si rendesse necessario apportare modifiche, sarà sufficiente modificare il file `service.xml` e ricompilarlo — ed ovviamente modificare le funzioni che abbiamo creato, dove necessario.

I servizi

Come anticipato nelle sezioni precedenti, vediamo come creare i servizi per accedere al database.

Dividiamo i servizi in base alla tabella, e quindi all'entità, cui si riferiscono. Avremo quindi 4 file, creati dal service builder, in cui andremo ad inserire i metodi che creeremo.

`TWTHEMELOCALSERVICEIMPL.JAVA` È il file che gestisce gli accessi alla tabella `TWTheme`. Vediamone i metodi principali.

`addTheme` Il metodo `addTheme` è il metodo che inserisce un tema nel database, creando

il record, inserendo gli attributi passati come parametri e aggiornando la persistenza del database.

```
public TWTheme addTheme(long userId, String name, String
    description,
      long limitedGroupId, int structureType, int scopeType,
      int baseScheme, long creatorPartnerId, long creatorUserId)
throws SystemException, PortalException {

   Date now = new Date();

   User user = userPersistence.findByPrimaryKey(userId);
   long themeId = counterLocalService.increment();

   TWTheme theme = twThemePersistence.create(themeId);

   theme.setUserId(user.getUserId());
   theme.setUserName(user.getFullName());
   theme.setCompanyId(user.getCompanyId());
   theme.setCreateDate(now);
   theme.setModifiedDate(now);

   theme.setName(name);
   theme.setDescription(description);
   theme.setLimitedGroupId(limitedGroupId);
   theme.setStructureType(structureType);
   theme.setScopeType(scopeType);
   theme.setBaseScheme(baseScheme);
   theme.setCreatorPartnerId(creatorPartnerId);
   theme.setCreatorUserId(creatorUserId);
   theme.setPublishedVersionId(0);

   twThemePersistence.update(theme, false);

   return theme;
}
```

`unpublishTheme`

Il metodo `unpublishTheme` rende non più disponibile un tema, impostando la sua versione pubblicata ad un valore di default.

```
public void unpublishTheme(long themeId)
throws SystemException, PortalException {
   TWThemeLocalServiceUtil.getTWTheme(themeId).
       setPublishedVersionId(
      TWConstants.UNPUBLISHED_THEME_VERSION_ID);
   }
```

`setBaseScheme`

Il metodo `setBaseScheme` aggiorna lo schema colori base di un tema.

```
public void setBaseScheme(long themeId, int baseScheme)
throws SystemException, PortalException {

   TWThemeLocalServiceUtil.getTWTheme(themeId).setBaseScheme(
       baseScheme);
```

```
}
```

updateTheme

Il metodo updateTheme aggiorna il nome di un tema.

```
public TWTheme updateTheme(long themeId, String name, String
    description)
throws SystemException, PortalException {

   Date now = new Date();

   TWTheme theme = twThemePersistence.findByPrimaryKey(themeId);

   theme.setModifiedDate(now);
   theme.setName(name);
   theme.setDescription(description);
   theme.setPublishedVersionId(0);

   twThemePersistence.update(theme, false);

   return theme;
}
```

getThemes

Il metodo getThemes ritorna l'elenco dei temi creati da un partner o un utente.

```
public List<TWTheme> getThemes(long creatorPartnerId, int start,
    int end)
throws PortalException, SystemException{

   return getThemes(creatorPartnerId, 0, start, end);
}

public List<TWTheme> getThemes(long creatorPartnerId,
      long creatorUserId,  int start, int end)
throws PortalException, SystemException {

   if (creatorUserId <= 0) {
      /* themes created by partners
       */
      return twThemePersistence.findByCreatorPartnerID(
            creatorPartnerId, start, end);
   } else {
      /* themes created by users which are not in a partner company
           have a
       * a creatorPartnerId of 0
       */
   return twThemePersistence.findByCP_CU(
            creatorPartnerId, creatorUserId, start, end);
   }
}
```

getTheme

Il metodo getTheme semplicemente ritorna un tema, dato l'ID del tema cercato.

```
public TWTheme getTheme(long themeId)
throws PortalException, SystemException {
   return twThemePersistence.findByPrimaryKey(themeId);
}
```

Il metodo `addImage` inserisce un'immagine nel database, associandola al tema per il quale è stata caricata. Il metodo riceve come parametri l'ID del tema e l'immagine in formato `File`. L'immagine viene quindi scomposta in un array di `byte` e scalata se supera le dimensioni massime. Infine, viene salvata nella tabella delle immagini, e il suo ID, il suo nome (ricavato come campo dall'immagine in formato `File`) e l'ID del tema vengono salvati nella tabella che associa le immagini ai temi.

`addImage`

```
public void addImage(long themeId, File file)
   throws PortalException, SystemException {

   byte[] bytes = null;
   byte[] scaled = null;

   try {
      bytes = FileUtil.getBytes(file);
      scaled = ImageUtilExt.scaleImage(
         bytes,
         PortalPropsExtUtil.usersImageMaxHeigth(),
         PortalPropsExtUtil.usersImageMaxWidth());
   } catch (Exception e) { }

   long themeImageId = counterLocalService.increment();

   long imageId = counterLocalService.increment();

   imageLocalService.updateImage(imageId, scaled);

   TWThemeImage twThemeImage = twThemeImagePersistence.create(
       themeImageId);

   twThemeImage.setThemeId(themeId);
   twThemeImage.setImageId(imageId);
   twThemeImage.setName(file.getName());

   twThemeImagePersistence.update(twThemeImage, false);

}
```

Il metodo `getImages` ritorna una lista contenente tutte le immagini associate al tema richiesto.

`getImages`

```
public List<TWThemeImage> getImages(long twThemeId)
   throws PortalException, SystemException {

   return twThemeImagePersistence.findByThemeId(twThemeId);
}
```

removeMinorVersions

Il metodo `removeMinorVersions` è usato in fase di pubblicazione di un tema. Dato il tema pubblicato e la version major di partenza, il metodo recupera tutte le minor appartenenti alla major data. Quindi, per ognuna ne rimuove tutti gli stili, eliminando alla fine anche la versione minor stessa.

```
public void removeMinorVersions(long themeId, int versionMajor)
   throws PortalException, SystemException {

   List<TWVersion> versions = TWVersionLocalServiceUtil.getVersions
       (themeId, versionMajor);
   List<TWStyle> styles = new ArrayList<TWStyle>();
   TWVersion version = null;
   for(int i = 0; i < versions.size() -1; i++)  {
      version = versions.get(i);
      styles = TWStyleLocalServiceUtil.getStyles(version.
          getVersionId());
      for( TWStyle style : styles ) {
         TWStyleLocalServiceUtil.removeStyle(style.getStyleId());
      }
      TWVersionLocalServiceUtil.removeVersion(version.getVersionId
          ());
   }
}
```

updatePublishedVersion

Ultimo, `updatePublishedVersion` è il metodo che aggiorna lo stato delle versioni pubblicate di un tema. Tutto ciò che fa è recuperare il tema aggiornato, aggiornare l'attributo `publishedVersionId` e aggiornare il database.

```
public void updatePublishedVersion(long themeId, long
    publishedVersionId)
   throws PortalException, SystemException {

   TWTheme theme = twThemePersistence.findByPrimaryKey(themeId);
   theme.setPublishedVersionId(publishedVersionId);
   twThemePersistence.update(theme, false);
}
```

TWVERSIONLOCALSERVICEIMPL.JAVA È il file che gestisce l'accesso alla tabella `TWVersion`. Si occupa inoltre di ricostruire le versioni, quando richiesto.

addVersion

Il seguente metodo crea una nuova versione, specificando che tale versione non è una draft. Tutto ciò che fa è richiamare il metodo successivo.

```
public TWVersion addVersion(long userId, long themeId,
    long parentVersionId, boolean publishing)
throws PortalException, SystemException {

  return addVersion(userId, themeId, parentVersionId, publishing,
      false);
}
```

Il medoto `addVersion` crea una nuova versione, sia essa minor, major o draft. Vediamo in dettaglio il suo comportamento.

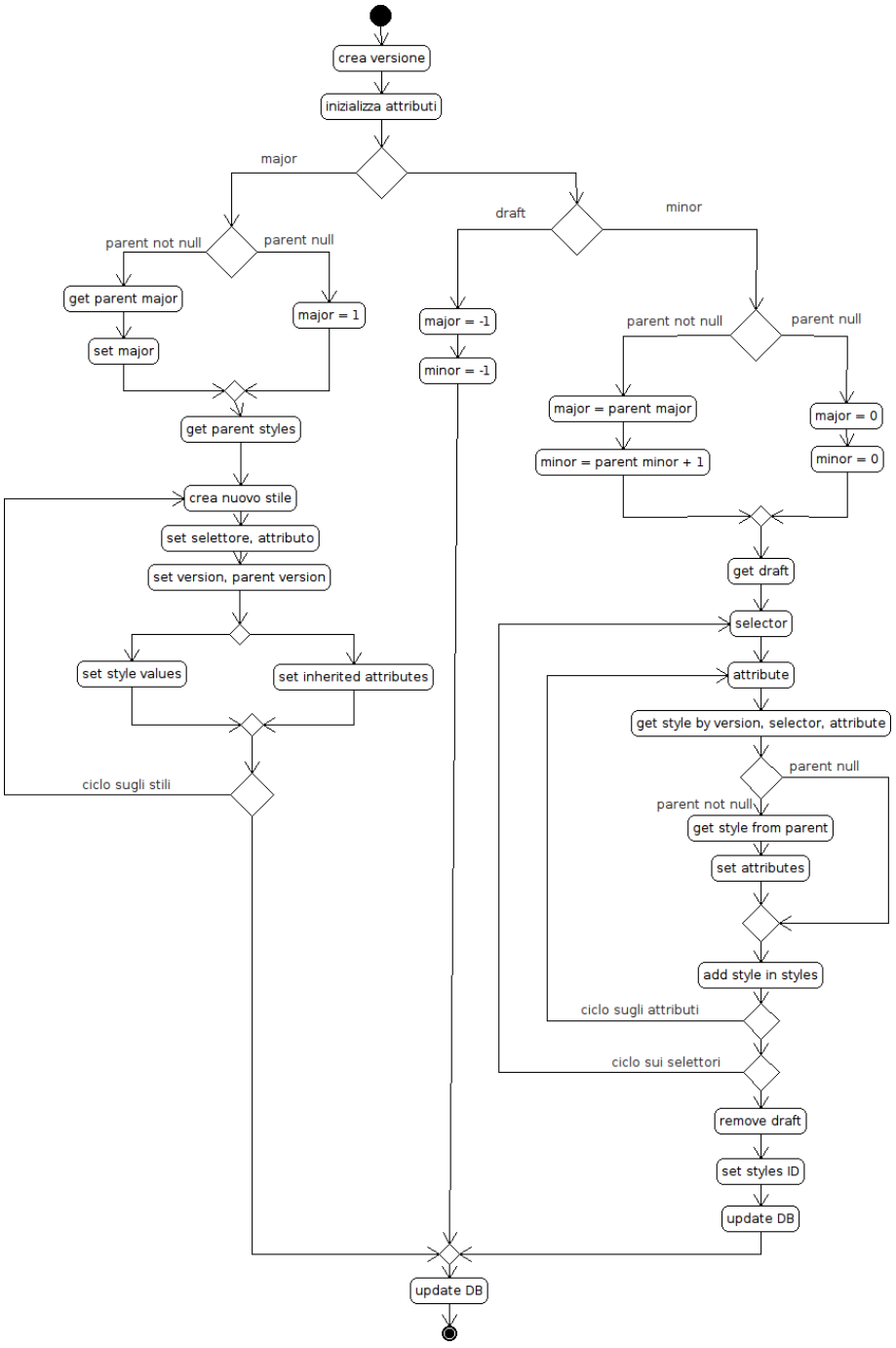


Figura 14: Funzione `addVersion`

Per prima cosa, creare la versione, e settare audit, tema e versione genitrice, se esistente, oppure un valore di default.

Vediamo il caso in cui si stia procedendo alla pubblicazione di una versione major. Se esiste una versione genitrice, recuperarne il valore di major e incrementarlo, per ottenere il major della versione corrente, altrimenti impostare il major a un valore di default (`1`). Il minor in ogni caso sarà `0`. Recuperare quindi gli stili della minor in fase di pubblicazione, e, per ciascuno, creare un nuovo stile che ne erediti gli attributi. In caso lo stile presenti una versione genitrice non nulla (e quindi lo stile è ereditato da una versione precedente) recuperarlo, e iniettarne gli attributi nel nuovo stile. Quindi aggiornare la tabella degli stili.

Nel caso si stia salvando una versione di draft, impostare minor e major number a `-1`.

Il terzo caso contemplato è il salvataggio di una minor version. Analogamente a quanto fatto nel caso della pubblicazione della major version, settare i numeri di versione (in questo caso major e minor) in base alla versione genitrice, se presente. Recuperare quindi la draft da pubblicare, e scorrere i selettori e gli attributi. Per ogni stile così determinato, provare a recuperare gli attributi dalla draft. Se non è presente, recuperarlo dalla versione genitrice. Come ultima operazione, eliminare la draft e aggiornare la tabella degli stili.

Come ultima operazione per tutte le tre possibilità, aggiornare la tabella delle versioni.

```
public TWVersion addVersion(long userId, long themeId,
    long parentVersionId, boolean publishing, boolean draft)
throws PortalException, SystemException {

  Date now = new Date();

  long versionId = counterLocalService.increment();

  TWVersion version = twVersionPersistence.create(versionId);

  String userName = userLocalService.getUser(userId).getFirstName()
      ;

  version.setUserId(userId);
  version.setUserName(userName);
  version.setCreateDate(now);
  version.setThemeId(themeId);
  version.setParentVersionId(parentVersionId);
  TWVersion parentVersion = null;

  if (parentVersionId != TWConstants.NO_PARENT_VERSION_ID) {
    parentVersion = getVersion(parentVersionId);
  }

  if (publishing) {
    // PUBLISING
    // prev major + increment
    if(Validator.isNotNull(parentVersion)) {
```

```
    version.setVersionMajor(parentVersion.getVersionMajor()
      + TWConstants.VERSION_MAJOR_INCREMENT);
  } else {
    version.setVersionMajor(TWConstants.
        INITIAL_VERSION_MAJOR_NUMBER);
  }
  // initial minor (~x.0)
  version.setVersionMinor(TWConstants.
      INITIAL_VERSION_MINOR_NUMBER);

  List<TWStyle> styles = TWStyleLocalServiceUtil
    .getStyles(parentVersionId);
  TWStyle newStyle = null;

  for (TWStyle style : styles) {
    newStyle = twStylePersistence.create(counterLocalService
        .increment());
    newStyle.setVersionId(versionId);
    newStyle.setParentVersionId(TWConstants.NO_PARENT_VERSION_ID)
        ;
    newStyle.setSelector(style.getSelector());
    newStyle.setAttribute(style.getAttribute());
    if (style.getParentVersionId() == TWConstants.
        NO_PARENT_VERSION_ID) {
      newStyle.setType(style.getType());
      newStyle.setValue(style.getValue());
      newStyle.setBackground(style.getBackground());
    } else {
      TWStyle inherited = TWStyleLocalServiceUtil.getStyle(
          style.getParentVersionId(), style.getSelector(),
          style.getAttribute());
      newStyle.setType(inherited.getType());
      newStyle.setValue(inherited.getValue());
      newStyle.setBackground(inherited.getBackground());
    }
    twStylePersistence.update(newStyle, false);
  }
} else if (draft) {
  // DRAFT VERSION (-1.-1)
  version.setVersionMajor(TWConstants.DRAFT_MAJOR_NUMBER);
  version.setVersionMinor(TWConstants.DRAFT_MINOR_NUMBER);
} else {
  // SAVING DRAFT TO VERSION
  // prev major
  if (Validator.isNotNull(parentVersion)) {
    // prev minor + increment (x.3 -> x.4, x.43 -> x.44)
    version.setVersionMajor(parentVersion.getVersionMajor());
    version.setVersionMinor(parentVersion.getVersionMinor()
        + TWConstants.VERSION_MINOR_INCREMENT);
  } else {
    version.setVersionMajor(TWConstants.
        INITIAL_VERSION_MAJOR_NUMBER);
```

```
      version.setVersionMinor(TWConstants.
          INITIAL_VERSION_MINOR_NUMBER);
    }
    TWVersion draftVersion = getDraftVersion(themeId, userId);
    TWStyle style = null;
    List<TWStyle> newStyles = new ArrayList<TWStyle>();
    for (int iterS = 0; iterS < TWConstants.SELECTORS.length; iterS
        ++) {
      if (Validator.isNotNull(TWConstants.SELECTORS[iterS])) {
        for (int iterA = 0; iterA < TWConstants.
            SELECTORS_ATTRIBUTES[iterS].length; iterA++) {
          int iterSA = TWConstants.SELECTORS_ATTRIBUTES[iterS][
              iterA];

          style = null;
          try {
            style = TWStyleLocalServiceUtil.getStyle(versionId,
                iterS, iterSA);
          } catch (Exception e) {
          }

          if (Validator.isNull(style)) {
            long parentStyleVersionId = 0;
            try {
              parentStyleVersionId = TWStyleLocalServiceUtil
                  .getStyle(parentVersionId, iterS,
                      iterSA).getParentVersionId();
            } catch (Exception e) {
            }
            style = twStylePersistence.create(0);
            style.setVersionId(draftVersion.getVersionId());
            style.setSelector(iterS);
            style.setAttribute(iterSA);
            style.setParentVersionId(parentStyleVersionId);
          }
          newStyles.add(style);
        }
      }
    }
    twStylePersistence.removeByVersionId(draftVersion.getVersionId
        ());
    for (TWStyle tmpStyle : newStyles) {
      long tmpStyleId = counterLocalService.increment();
      tmpStyle.setStyleId(tmpStyleId);
      twStylePersistence.update(tmpStyle, false);
    }
  }

  twVersionPersistence.update(version, false);
  return version;
}
```

Il metodo `getDraftVersion` restituisce la versione draft al momento in uso per un determinato tema. In caso non vi sia alcuna draft attiva, ne viene creata una.

`getDraftVersion`

```
public TWVersion getDraftVersion(long themeId, long userId)
throws PortalException, SystemException {

  System.out.println("working on theme no. " + themeId);

  if (twVersionPersistence.countByT_M(themeId,
      TWConstants.DRAFT_MAJOR_NUMBER) > 0) {
    return (TWVersionLocalServiceUtil.getVersions(themeId,
        TWConstants.DRAFT_MAJOR_NUMBER).get(0));
  } else {
    if (getVersions(themeId).size() > 0) {
      TWVersion pv = getVersions(themeId).get(0);
      return (TWVersionLocalServiceUtil.addVersion(userId, themeId,
          pv.getVersionId(), false, true));
    } else {
      return (TWVersionLocalServiceUtil.addVersion(userId, themeId,
          TWConstants.NO_PARENT_VERSION_ID, false, true));
    }
  }
}
```

Il seguente metodo, `testDraftVersion` controlla se esiste una draft version per un determinato tema.

`testDraftVersion`

```
public boolean testDraftVersion(long themeId)
throws PortalException, SystemException {

  return (twVersionPersistence.countByT_M(themeId,
      TWConstants.DRAFT_MAJOR_NUMBER) > 0);
}
```

Il metodo `getVersion` restituisce la versione corrispondente all'ID specificato.

`getVersion`

```
public TWVersion getVersion(long versionId)
throws PortalException, SystemException {

  return twVersionPersistence.findByPrimaryKey(versionId);
}
```

I metodi `getVersions` ritornano una lista di versioni contenente tutte le versioni di un tema. Se specificato, tutte e sole le versioni di un tema dato un major number.

`getVersions`

```
public List<TWVersion> getVersions(long themeId)
throws PortalException, SystemException {

  return getVersions(themeId, TWConstants.NO_VERSION_MAJOR_NUMBER);
}
```

```
public List<TWVersion> getVersions(long themeId, int versionMajor)
throws PortalException, SystemException {

  if (versionMajor == TWConstants.NO_VERSION_MAJOR_NUMBER) {
    return twVersionPersistence.findByThemeId(themeId);
  } else {
    return twVersionPersistence.findByT_M(themeId, versionMajor);
  }
}
```

getVersionGenealogy

I metodi `getVersionGenealogy` ritornano le versioni di un tema, rispettivamente tutt quelle di una major e tutte quelle di una major fino ad una data minor. Nel caso di versioni non "in parallelo" il risultato è identico, ma può variare nel caso di versioni non "in parallelo", funzionalità che si vuole implementare nelle versioni sucessive del wizard.

```
public List<TWVersion> getVersionGenealogy(long themeId, int
    versionMajor)
throws PortalException, SystemException {

  return twVersionPersistence.findByT_M(themeId, versionMajor);
}

public List<TWVersion> getVersionGenealogy(long themeId, long
    versionId)
throws PortalException, SystemException {

  return twVersionPersistence.findByT_M_mM(themeId, getVersion(
      versionId)
      .getVersionMajor(), getVersion(versionId).getVersionMinor());
}
```

getSmartVersionGenealogy

Come nel caso precedente, il metodo `getSmartVersionGenealogy` è un metodo che verrà utilizzato in caso di versioni parallele. Data una versione minor di un tema, procede a ritroso in base alla versione genitrice, fino ad raggiungere una major. Le versioni incontrate nel cammino vengono inserite in una lista.

```
public List<TWVersion> getSmartVersionGenealogy(long themeId, long
    versionId)
throws PortalException, SystemException {

  List<TWVersion> genealogy = new ArrayList<TWVersion>();

  TWVersion version = getVersion(versionId);

  int major = version.getVersionMajor();

  while (version.getVersionMinor() >= 0
       && version.getVersionMajor() == major) {
    genealogy.add(version);
    version = getVersion(version.getParentVersionId());
  }
```

```
  return genealogy;
}
```

Il seguente metodo `saveDraftToVersion` trasforma una draft in minor version. Il suo comportamento è simile al branch `minor` del metodo `addVersion`.

`saveDraftToVersion`

```
public void saveDraftToVersion(long draftVersionId, long userId)
throws Exception {

  if(TWStyleLocalServiceUtil.countStyles(draftVersionId) <= 0) {
    twVersionPersistence.remove(draftVersionId);
    return;
  }

  TWVersion version = getVersion(draftVersionId);
  long themeId = version.getThemeId();

  TWVersion parentVersion = null;
  long parentVersionId = TWConstants.NO_PARENT_VERSION_ID;

  try {
    parentVersion = getVersions(themeId).get(1);
    parentVersionId = parentVersion.getVersionId();
  } catch (Exception e) {
  }

  // SAVING DRAFT TO VERSION
  // prev major
  if (Validator.isNotNull(parentVersion)) {
    version.setVersionMajor(parentVersion.getVersionMajor());
    version.setVersionMinor(parentVersion.getVersionMinor()
        + TWConstants.VERSION_MINOR_INCREMENT);
    version.setParentVersionId(parentVersionId);
  } else {
    // prev minor + increment (x.3 -> x.4, x.43 -> x.44)
    version.setVersionMajor(TWConstants.
        INITIAL_VERSION_MAJOR_NUMBER);
    version.setVersionMinor(TWConstants.
        INITIAL_VERSION_MINOR_NUMBER);
    version.setParentVersionId(TWConstants.NO_PARENT_VERSION_ID);
  }

  List<TWStyle> newStyles = new ArrayList<TWStyle>();
  TWStyle style = null;

  for (int iterS = 0; iterS < TWConstants.SELECTORS.length; iterS
      ++) {
    if (Validator.isNotNull(TWConstants.SELECTORS[iterS])) {
      for (int iterA = 0; iterA < TWConstants.SELECTORS_ATTRIBUTES[
          iterS].length; iterA++) {
        int iterSA = TWConstants.SELECTORS_ATTRIBUTES[iterS][iterA
            ];
```

```
        style = null;
        try {
          style = TWStyleLocalServiceUtil.getStyle(
              draftVersionId, iterS, iterSA);
        } catch (Exception e) {
        }

        if (Validator.isNull(style)) {
          long parentStyleVersionId = 0;
          try {
            parentStyleVersionId = TWStyleLocalServiceUtil
                .getStyle(parentVersionId, iterS, iterSA)
                .getParentVersionId();

            if (parentStyleVersionId == 0) {
              parentStyleVersionId = parentVersionId;
            }

          } catch (Exception e) {
          }
          style = twStylePersistence.create(0);
          style.setVersionId(draftVersionId);
          style.setSelector(iterS);
          style.setAttribute(iterSA);
          style.setParentVersionId(parentStyleVersionId);
        }
        newStyles.add(style);
      }
    }
  }
  twStylePersistence.removeByVersionId(draftVersionId);
  for (TWStyle tmpStyle : newStyles) {
    long tmpStyleId = counterLocalService.increment();
    tmpStyle.setStyleId(tmpStyleId);
    twStylePersistence.update(tmpStyle, false);
  }
  twVersionPersistence.update(version, false);
}
```

`updateVersion` Il metodo `updateVersion` aggiorna una versione, dato il suo ID e una lista di stili aggiornati.

```
public void updateVersion(long versionId, List<TWStyle> styles)
throws Exception {

  long parentVersionId = getVersion(versionId).getParentVersionId()
      ;

  for (TWStyle style : styles) {

    twStylePersistence.removeByV_S_A(versionId, style.getSelector()
        ,
```

```
        style.getAttribute());
    if (style.getType() >= 0) {
      long styleId = counterLocalService.increment(TWStyle.class
          .toString());

      style.setStyleId(styleId);
      style.setVersionId(versionId);

      twStylePersistence.update(style, false);
    }
  }
}
```

Il metodo `removeVersion` elimina una versione specifica, dato il suo ID. `removeVersion`

```
public void removeVersion(long versionId)
throws PortalException, SystemException {

  twVersionPersistence.remove(versionId);
}
```

Il metodo `removeDraftVersion` elimina la draft di un tema. `removeDraftVersion`

```
public void removeDraftVersion(long themeId)
throws PortalException, SystemException {

  twVersionPersistence.removeByT_M(themeId, TWConstants.
      DRAFT_MAJOR_NUMBER);
}
```

Il metodo `buildThemePreview` ricostruisce il tema che verrà visualizzato nella finestra di `preview`. Viene invocato ogni volta che si richiede di visualizzare una preview, e ciò può accadere alla creazione di una draft, di una versione minor o di una major. Non potendo sapere in quale momento il metodo verrà invocato, dobbiamo gestire il caso generale, e ricostruire il tema. `buildThemePreview`

In una prima fase di inizializzazione recuperiamo l'ultima versione del tema inserita nel database, che chiameremo `baseVersion`, e creiamo la nostra versione di preview, che verrà identificata dagli appositi minor e major numbers.

A questo punto scorriamo tutti i selettori, e, per ciascuno di essi, tutti gli attributi ad esso associati. Creiamo così per ogni coppia selettore/attributo un nuovo stile. Proviamo a recuperare lo stile da `baseVersion`. Se lo stile era presente in essa (quindi, versione non di draft) copiamone i parametri nella versione di preview, altrimenti andiamo a recuperarli dallo stile corrispondente nella versione genitrice di `baseVersion`, la quale a questo punto può essere solo una draft. Salviamo lo stile nel database.

Salviamo anche la preview version nel database, generiamo il tema mediante l'apposito metodo (usato anche in fase di pubblicazione) ed eliminiamo la preview version dal database, che a questo punto non ci serve più.

```
public String buildThemePreview(long themeId)
throws PortalException, SystemException {

  String result = null;
```

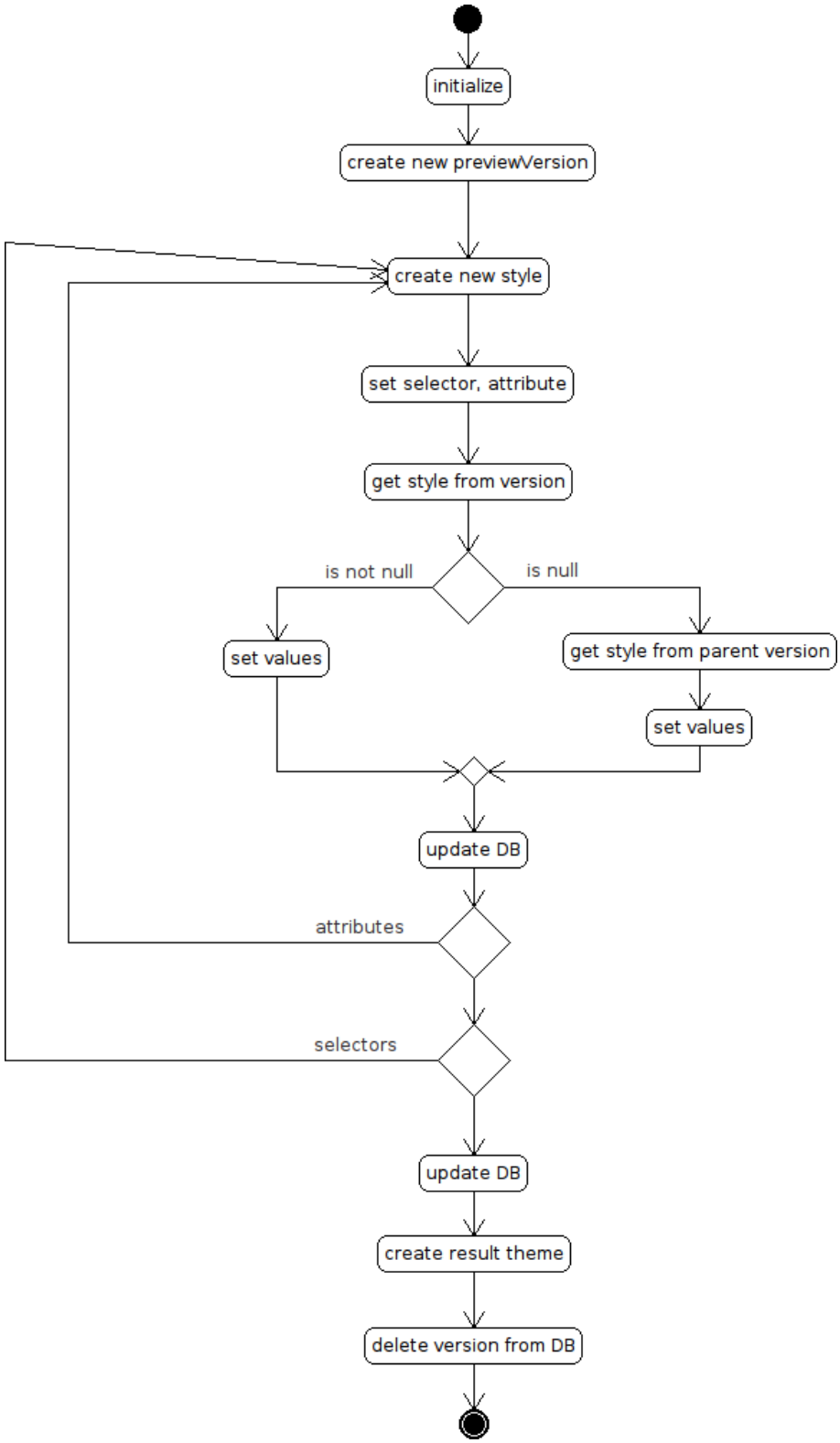


Figura 15: Funzione `buildThemePreview`

```
  long versionId = TWVersionLocalServiceUtil.getVersions(themeId).
      get(0).getVersionId();
  TWVersion baseVersion = getVersion(versionId);
  TWStyle newStyle = null;
  long newVersionId = counterLocalService.increment();
  TWVersion newVersion = twVersionPersistence.create(newVersionId);
  newVersion.setThemeId(themeId);
  newVersion.setVersionMajor(TWConstants.PREVIEW_MAJOR_NUMBER);
  newVersion.setVersionMinor(TWConstants.PREVIEW_MINOR_NUMBER);

  for (int iterS=0; iterS<TWConstants.SELECTORS.length; iterS++){
    if(Validator.isNotNull(TWConstants.SELECTORS[iterS])) {

      for (int iterSA=0; iterSA < TWConstants.SELECTORS_ATTRIBUTES[
          iterS].length; iterSA++) {
        if(Validator.isNotNull(TWConstants.SELECTORS_ATTRIBUTES[
            iterS][iterSA])) {

          int iterA = TWConstants.SELECTORS_ATTRIBUTES[iterS][
              iterSA];
          TWStyle style = TWStyleLocalServiceUtil.getStyle(
              versionId, iterS, iterA) ;
          newStyle = twStylePersistence.create(counterLocalService.
              increment());
          newStyle.setVersionId(newVersionId);
          newStyle.setParentVersionId(baseVersion.getVersionId());
          newStyle.setSelector(iterS);
          newStyle.setAttribute(iterSA);

          if (style != null) {
            newStyle.setType(style.getType());
            newStyle.setValue(style.getValue());
            newStyle.setBackground(style.getBackground());
          } else {
            TWStyle inherited = TWStyleLocalServiceUtil.getStyle(
                baseVersion.getParentVersionId(), iterS, iterA);
            newStyle.setType(inherited.getType());
            newStyle.setValue(inherited.getValue());
            newStyle.setBackground(inherited.getBackground());
          }
          twStylePersistence.update(newStyle, false);
        }
      }
    }
  }
  twVersionPersistence.update(newVersion, false);
  result = ThemeWizardUtil.createResultTheme(themeId, true, false);
  twVersionPersistence.removeByT_M(themeId, TWConstants.
      PREVIEW_MAJOR_NUMBER);
  return result;
}
```

testPublishingVersion

L'ultimo metodo, `testPublishingVersion` controlla l'esistenza o meno di versioni di preview.

```
public boolean testPublishingVersion(long themeId)
throws PortalException, SystemException {

  return (twVersionPersistence.countByT_M(themeId,
    TWConstants.PREVIEW_MAJOR_NUMBER) > 0);
}
```

TWSTYLELOCALSERVICEIMPL.JAVA

Gestisce l'accesso alla tabella `TWStyle`.

addStyle

Il metodo `addStyle` crea un nuovo stile, con i valori passati come parametri come attributi.

```
public TWStyle addStyle(long versionId, long parentVersionId, int
    selector,
   int attribute, int type, Double value, long background)
throws SystemException, PortalException {

   long styleId = counterLocalService.increment();
   TWStyle style = twStylePersistence.create(styleId);

   style.setVersionId(versionId);
   style.setParentVersionId(parentVersionId);
   style.setSelector(selector);
   style.setAttribute(attribute);
   style.setType(type);
   style.setValue(value);
   style.setBackground(background);

   twStylePersistence.update(style, false);
   return style;
}
```

updateStyle

Il metodo `updateStyle` aggiorna gli attributi di uno stile con i nuovi valori passati come parametri.

```
public TWStyle updateStyle(long styleId, long versionId, long
    parentVersionId,
   int selector, int attribute, int type, Double value, long
       background)
throws PortalException, SystemException{

   TWStyle style = twStylePersistence.findByPrimaryKey(styleId);

   style.setVersionId(versionId);
   style.setParentVersionId(parentVersionId);
   style.setSelector(selector);
   style.setAttribute(attribute);
   style.setType(type);
   style.setValue(value);
   style.setBackground(background);
```

```
   twStylePersistence.update(style, false);
   return style;
}
```

Il metodo `getStyle` utilizza il metodo predisposto dal `service builder` per ritornare lo stile richiesto. `getStyle`

```
public TWStyle getStyle(long styleId)
throws PortalException, SystemException{

   return twStylePersistence.findByPrimaryKey(styleId);
}
```

Il metodo `getStyles` ritorna una lista contenente tutti gli stili associati ad una versione, utilizzando il metodo fornito dal `service builder`. `getStyles`

```
public List<TWStyle> getStyles (long versionId)
throws PortalException, SystemException {

   return twStylePersistence.findByVersionId(versionId);
}
```

Il metodo `countStyles` conta il numero di stili associati ad una versione, utilizzando il metodo appositamente creato dal `service builder`. `countStyles`

```
public int countStyles (long versionId)
throws PortalException, SystemException {

   return twStylePersistence.countByVersionId(versionId);
}
```

Il metodo `getStyle` restituisce lo stile scelto, dati versione, selettore e attributo, gestendo opportunamente il caso in cui lo stile non sia presente. `getStyle`

```
public TWStyle getStyle (long versionId, int selector, int
    attribute)
throws PortalException, SystemException {

   if(countStyles(versionId) == 0 || versionId == 0) {
      return null;
   }

   TWStyle style = null;
   try {
      style = twStylePersistence.findByV_S_A(versionId, selector,
          attribute).get(0);
   } catch(Exception e) {
      style = null;
   }
   return style;
}
```

L'ultimo metodo, `removeStyle`, elimina lo stile selezionato dalla tabella. `removeStyle`

```
public void removeStyle (long styleId)
```

```
throws PortalException, SystemException {

   twStylePersistence.remove(styleId);
}
```

TWTHEMEIMAGELOCALSERVICEIMPL.JAVA Gestisce l'accesso alla tabella `TWThemeImage`.

`setUsedImages` Il metodo `setUsedImages` imposta il flag `isUsed` a `true` se l'immagine è usata nel tema.

```
public void setUsedImages(long themeId)
throws PortalException, SystemException{

   long versionId = TWThemeLocalServiceUtil.getTheme(themeId).
       getPublishedVersionId();
   for (int iterS = 0 ; iterS < TWConstants.SELECTORS.length ;
       iterS++) {
      try {
         TWStyle style = TWStyleLocalServiceUtil.getStyle(versionId
             , iterS, 61);
         if(style.getBackground() > 0 && style.getType() == 4) {
            TWThemeImage img = TWThemeImageLocalServiceUtil.
                getTWThemeImage(style.getBackground());
            img.setIsUsed(true);
         }
      } catch(Exception e) {}
   }
}
```

Le azioni

Vediamo ora la componente più propriamente *controller*, secondo il pattern textscmvc, del portlet, vale a dire la classe che si occupa di gestire le azioni eseguite dall'utente, ogniqualvolta un pulsante viene cliccato.

`processAction` Il metodo `processAction` è il metodo che viene invocato al momento del click su un pulsante del portlet. Come dice il nome, riconosce l'azione (il cui nome è impostato in uno dei parametri dell'`ActionRequest`) e invoca a sua volta il metodo adibito alla gestione dell'evento, scegliendolo tra quelli scritti di seguito..

```
private String _uploadRequestRedirect = StringPool.BLANK;

public void processAction(ActionRequest actionRequest,
    ActionResponse actionResponse)
throws PortletException {

  try {
    String actionName = ParamUtil.getString(actionRequest,
        ActionRequest.ACTION_NAME);
    boolean myAction = false;

    if(actionName.equalsIgnoreCase("updateTheme")) {
```

```
      myAction = true;
      updateTheme(actionRequest);
    } else if (actionName.equalsIgnoreCase("addImage")) {
      myAction = true;
      addImage(actionRequest);
    } else if (actionName.equalsIgnoreCase("saveVersion")) {
      myAction = true;
      saveVersion(actionRequest);
    } else if(actionName.equalsIgnoreCase("saveDraft")) {
      myAction = true;
      saveDraft(actionRequest);
    } else if(actionName.equalsIgnoreCase("publishVersion")) {
      myAction = true;
      publishVersion(actionRequest, actionResponse);
    }

    if (myAction) {
      if (SessionErrors.isEmpty(actionRequest)) {
        SessionMessages.add(actionRequest, "request_processed");
      }
    }

    String redirect = ParamUtil.getString(actionRequest, "redirect"
        );

    if( Validator.isNotNull(_uploadRequestRedirect)) {
      redirect = _uploadRequestRedirect;
    }

    if (Validator.isNotNull(redirect)) {
      actionResponse.sendRedirect(redirect);
    }
  } catch (Exception e) {
    throw new PortletException(e);
  }
}
```

Il metodo newTheme crea un tema vuoto. newTheme

```
protected TWTheme newTheme(ActionRequest actionRequest)
throws Exception {
  String name = ParamUtil.getString(actionRequest, "name");
  String description = ParamUtil.getString(actionRequest, "
      description");

  TWTheme theme = TWThemeLocalServiceUtil.addTheme(0, name,
      description, 0, 0, 0, 0, 0, 0);

  return theme;
}
```

Il metodo updateTheme invece imposta gli attributi di un tema, dati i parametri passati. Se il tema richiesto non esiste lo si crea. updateTheme

```
protected void updateTheme(ActionRequest actionRequest)
throws Exception {

  ThemeDisplay themeDisplay =
    (ThemeDisplay)actionRequest.getAttribute(WebKeys.THEME_DISPLAY)
        ;

  if (!themeDisplay.isSignedIn()) {
    return;
  }

  long userId = themeDisplay.getUserId();
  long groupId = themeDisplay.getScopeGroupId();
  long twThemeId = ParamUtil.getLong(actionRequest, "twThemeId");
  String name = ParamUtil.getString(actionRequest, "name");
  String description = ParamUtil.getString(actionRequest, "
      description");

  if(twThemeId > 0) {
    // CREATE
    TWThemeLocalServiceUtil.updateTheme(twThemeId, name,
        description);
  } else {
    // UPDATE
    TWThemeLocalServiceUtil.addTheme(userId, name, description,
        groupId, 0, 0, 0, 0, userId);
  }
}
```

`addImage` Il metodo `addImage` inserisce un'immagine nel database, sotto formato di `file`.

```
protected void addImage(ActionRequest actionRequest)
throws Exception, UploadException {

  ThemeDisplay themeDisplay =
    (ThemeDisplay)actionRequest.getAttribute(WebKeys.THEME_DISPLAY)
        ;

  if (!themeDisplay.isSignedIn()) {
    return;
  }

  UploadPortletRequest uploadRequest = PortalUtil.
      getUploadPortletRequest(
      actionRequest);
  _uploadRequestRedirect = ParamUtil.getString(uploadRequest, "
      redirect");
  long twThemeId = ParamUtil.getLong(uploadRequest, "twThemeId");
  File file = uploadRequest.getFile("img");
  TWThemeLocalServiceUtil.addImage(twThemeId, file);
}
```

Il metodo `saveVersion` viene invocato al momento del salvataggio di una draft in minor version, e a sua volta invoca il metodo `saveDraftToVersion`.

`saveVersion`

```
protected void saveVersion(ActionRequest actionRequest)
throws Exception {

  ThemeDisplay themeDisplay =
    (ThemeDisplay)actionRequest.getAttribute(WebKeys.THEME_DISPLAY)
        ;
  long userId = themeDisplay.getUserId();
  long themeId = ParamUtil.getLong(actionRequest, "twThemeId");
  TWVersion draft = TWVersionLocalServiceUtil.getDraftVersion(
      themeId, userId);
  TWVersionLocalServiceUtil.saveDraftToVersion(draft.getVersionId()
      , userId);
}
```

Il metodo `publishVersion` invece viene invocato al momento della pubblicazione di un tema. Recupera l'ultima versione minor, la pubblica, rimuove le minor e le draft relative, aggiorna il tema e il flag `isUsed` delle immagini pubblicate, invocando i metodi predisposti. Infine, invoca il metodo che crea l'archivio compresso contenente la struttura `_diffs`.

`publishVersion`

```
protected void publishVersion(
  ActionRequest actionRequest, ActionResponse actionResponse)
throws Exception{

  ThemeDisplay themeDisplay =
    (ThemeDisplay)actionRequest.getAttribute(WebKeys.THEME_DISPLAY)
        ;
  long userId = themeDisplay.getUserId();
  long themeId = ParamUtil.getLong(actionRequest, "twThemeId");
  TWVersion lastVersion = TWVersionLocalServiceUtil.getVersions(
      themeId).get(0);

  if ( lastVersion.getVersionMajor() == TWConstants.
      DRAFT_MAJOR_NUMBER &&
      TWThemeLocalServiceUtil.getTheme(themeId).
          getPublishedVersionId() != 0) {
    lastVersion = TWVersionLocalServiceUtil.getVersions(themeId).
        get(1);
  } else if (TWVersionLocalServiceUtil.getVersions(themeId).size()
      == 0) {
    lastVersion = null;
  }

  if(lastVersion != null) {
    long lastVersionId = lastVersion.getVersionId();
    boolean publishing = true;
    TWVersion version = TWVersionLocalServiceUtil.addVersion(userId
        , themeId, lastVersionId, publishing);
    TWThemeLocalServiceUtil.removeMinorVersions(themeId, version.
        getVersionMajor()-1);
```

```
    TWVersionLocalServiceUtil.removeDraftVersion(themeId);
    TWThemeLocalServiceUtil.updatePublishedVersion(themeId, version
        .getVersionId());
    TWThemeImageLocalServiceUtil.setUsedImages(themeId);
    ThemeWizardUtil.writeZip(themeId);
  } else {
    System.out.println("No versions available to be published!!!");
  }
}
```

saveDraft

L'ultimo metodo, `saveDraft`, salva una draft nel database. È il metodo più complesso, in quanto è quello che si occupa di analizzare e trattare opportunamente i dati inseriti dall'utente. Recupera la draft del tema, e, se non presente, la crea. Dopodiché scorre tutti i selettori e i relativi attributi di un tema e recupera lo stile così ottenuto dai parametri. Quindi ne estrae gli attributi. Se lo stile è stato modificato, viene inserito nella draft. Infine vengono aggiornate la tabella degli stili e quella delle versioni, salvando la draft e gli stili modificati ad essa collegati.

```
protected void saveDraft(ActionRequest actionRequest)
throws Exception {

  ThemeDisplay themeDisplay =
    (ThemeDisplay)actionRequest.getAttribute(WebKeys.THEME_DISPLAY)
        ;
  long userId = themeDisplay.getUserId();
  long twThemeId = ParamUtil.getLong(actionRequest, "twThemeId");
  TWVersion draft = TWVersionLocalServiceUtil.getDraftVersion(
      twThemeId, userId);
  long versionId = draft.getVersionId();
  List<TWStyle> styles = new ArrayList<TWStyle>();

  String modified = "";
  double value = 0;
  int type = 0;
  long background = 0;

  for(int iterS = 0; iterS < TWConstants.SELECTORS.length ; iterS
      ++) {
    if (Validator.isNotNull(TWConstants.SELECTORS[iterS])) {
      for(int iterA = 0 ; iterA < TWConstants.SELECTORS_ATTRIBUTES[
          iterS].length ; iterA++) {
        int iterSA = TWConstants.SELECTORS_ATTRIBUTES[iterS][iterA
            ];

        TWStyle style = TWStyleUtil.create(0);
        style.setSelector(iterS);
        style.setAttribute(iterSA);

        modified = ParamUtil.getString(
            actionRequest,
            iterS+"_"+iterSA+ "_M");
```

```
type = ParamUtil.getInteger(
    actionRequest,
    iterS+"_"+iterSA+ "_T",-1);

value = ParamUtil.getDouble(
    actionRequest,
    iterS+"_"+iterSA+ "_V",0);

if (type == 4) {
  background = ParamUtil.getLong(
        actionRequest,
        iterS+"_"+iterSA+ "_B",0);
} else {
  String tmpString = ParamUtil.getString(actionRequest,
        iterS+"_"+iterSA+"_B");
  if (modified.equalsIgnoreCase("changed") &&
      Validator.isNotNull(tmpString) &&
      tmpString.length() > 6) {
    tmpString = tmpString.substring(tmpString.length()-6);
    background = Long.parseLong(tmpString,16);
    if(background == 0) {
      /* workaround to make the #000000 color considered
          not null
       * needs sibling hack in ThemeWizardUtil.
           createResultTheme().
       * It sets #0 to #1000000, only the last 6 chars
       * of the resulting string will be considered
       * */
      background = 16777216;
    }
  } else {
    background = 0;
  }
}

style.setVersionId(versionId);
if (Validator.isNotNull(modified)) {
  if (modified.equalsIgnoreCase("changed")){
    style.setSelector(iterS);
    style.setAttribute(iterSA);
    style.setValue(value);
    style.setType(type);
    style.setBackground(background);
    style.setParentVersionId(TWConstants.
        NO_PARENT_VERSION_ID);
    styles.add(style);
  } else if (modified.equalsIgnoreCase("revert")) {
    style.setSelector(iterS);
    style.setAttribute(iterSA);
    style.setValue(0.0);
    style.setType(-1);
    style.setBackground(0);
```

```
            style.setParentVersionId(TWConstants.
                NO_PARENT_VERSION_ID);
            styles.add(style);
          }
        }
      }
    }
  }

  TWVersionLocalServiceUtil.updateVersion(draft.getVersionId(),
      styles);
}
```

Le utilities

Le utilities sono dei metodi che affiancano la componente *controller* del portlet, che abbiamo visto nella sezione precedente. Non hanno accesso diretto al database, ma si appoggiano alla componente *model*, vale a dire i servizi visti nelle pagine precedenti.

Nel nostro caso le utilities servono a ricostruire un tema e a creare la struttura `_diffs` nell'archivio compresso.

`createResultTheme`

Il metodo `createResultTheme` viene invocato quando si deve produrre il file CSS contenente le modifiche apportate al tema, sia nel caso in cui si debba pubblicare il tema, e quindi creare il file da inserire nella struttura `_diffs`, sia nel caso in cui si debba creare lo stile da iniettare dinamicamente nella pagina di preview. Tale metodo genererà perciò la stringa da restituire al metodo chiamante, che si occuperà di gestirla come sua competenza. La stringa è il risultato dell'append delle sottostringhe in uno `StringBuilder`, classe molto più efficiente della semplice concatenazione di stringhe.

Gli stili sono stati divisi in due categorie:

1. custom, vale a dire booleani o dipendenti da un booleano;
2. non dipendenti da un booleano.

La prima categoria include bordi, margini e padding, che possono essere trattati in maniera unica, o separatamente in superiore, inferiore, destro e sinistro. Dato che nel corso della lavorazione un utente può aver impostato tutti i campi, vogliamo inserire nel file `custom.css` solo una delle due opzioni, ovvero quella indicata spuntando i `radio button` corrispondenti nella maschera di editing. Se la scelta è caduta sulla prima opzione, verrà recuperato e formattato lo stile relativo, altrimenti verranno processati i quattro stili custom, ricordando che nel database, e negli array di costanti di appoggio, lo schema è il seguente:

1. stile unico;
2. custom option;
3. stile superiore;
4. stile destro;
5. stile inferiore;
6. stile sinistro.

Nel caso invece in cui si abbia a che fare con un attributo non dipendente da un attributo `custom`, esso verrà trattato direttamente, in base alle proprie caratteristiche.

Se uno stile è stato modificato, il relativo selettore viene quindi inserito nello `StringBuilder` contenente il risultato.

Vediamo l'algoritmo della funzione, aiutandoci con lo schema in figura 16 nella pagina seguente che ci aiuta a comprendere meglio il percorso.

Per la corretta generazione dello stile, occorre valutare diversi casi. Detto degli attributi custom, che servono solamente per valutare quale delle corrispondenti opzioni vanno trattate, vediamo in quale maniera le restanti tipologie di attributi vengono gestite. Innanzitutto, ricordiamo che per far ciò ci appoggiamo agli array di costanti predisposti nel file `TWConstants.java`.

Le dimensioni si ottengono concatenando i `value` e `type` ottenuti dagli array in base agli indici indicati, che non sono altro che i valori salvati nel database.

I colori invece sono gli unici stili che non hanno tipo. Vengono stampati solamente in formato esadecimale RRGGBB, quindi non in forma verbale (black, white, ...), apponendo prima del colore il carattere speciale `#`. Occorre prestare attenzione al caso in cui si vuole visualizzare il colore nero (`#000000`). Poiché il colore non ha tipo, il valore `0` è il valore che per `value` e `background` indica che l'attributo non è stato settato. Si provvede quindi a salvare, in luogo dello `0`, il valore esadecimale `0x1000000`. Al momento dell'inserimento del colore nella stringa, solo gli ultimi sei caratteri verranno considerati. Tale hack viene effettuato al momento del salvataggio del colore nel database. Tutti gli altri colori vengono invece gestiti come semplice conversione dal formato decimale, formato in cui sono salvati nel database, al formato esadecimale.

Per quanto riguarda le immagini, i loro ID sono contenuti anch'essi nel campo `background`, come i colori. Essendo tuttavia ID di tuple in una tabella, rimangono in formato decimale. È possibile distinguerli dai colori in quanto alle immagini è associato un tipo diverso da `0`. Per inserire le immagini quindi si ricava il nome dell'immagine e si ricostruisce il percorso per arrivarvi.

Nel caso si abbia invece a che fare con stili che richiedono solo il tipo e non alcun valore, come ad esempio lo stile del testo, o l'allineamento del testo, si ricava il tipo e lo si concatena nella stringa, senza dover considerare il contenuto di `value` o `background`.

```
public static String createResultTheme(long themeId)
throws SystemException, PortalException {

  return createResultTheme(themeId, true, true);
}

public static String createResultTheme(long themeId, boolean
    verbose, boolean publishing)
throws SystemException, PortalException {

  StringBuilder sb = new StringBuilder();
  long renderingVersionId =0 ;
  if (publishing ||
      !(TWVersionLocalServiceUtil.testPublishingVersion(themeId) ||
        TWVersionLocalServiceUtil.testDraftVersion(themeId) )
    ) {
    renderingVersionId = TWThemeLocalServiceUtil.getTheme(themeId).
        getPublishedVersionId();
```

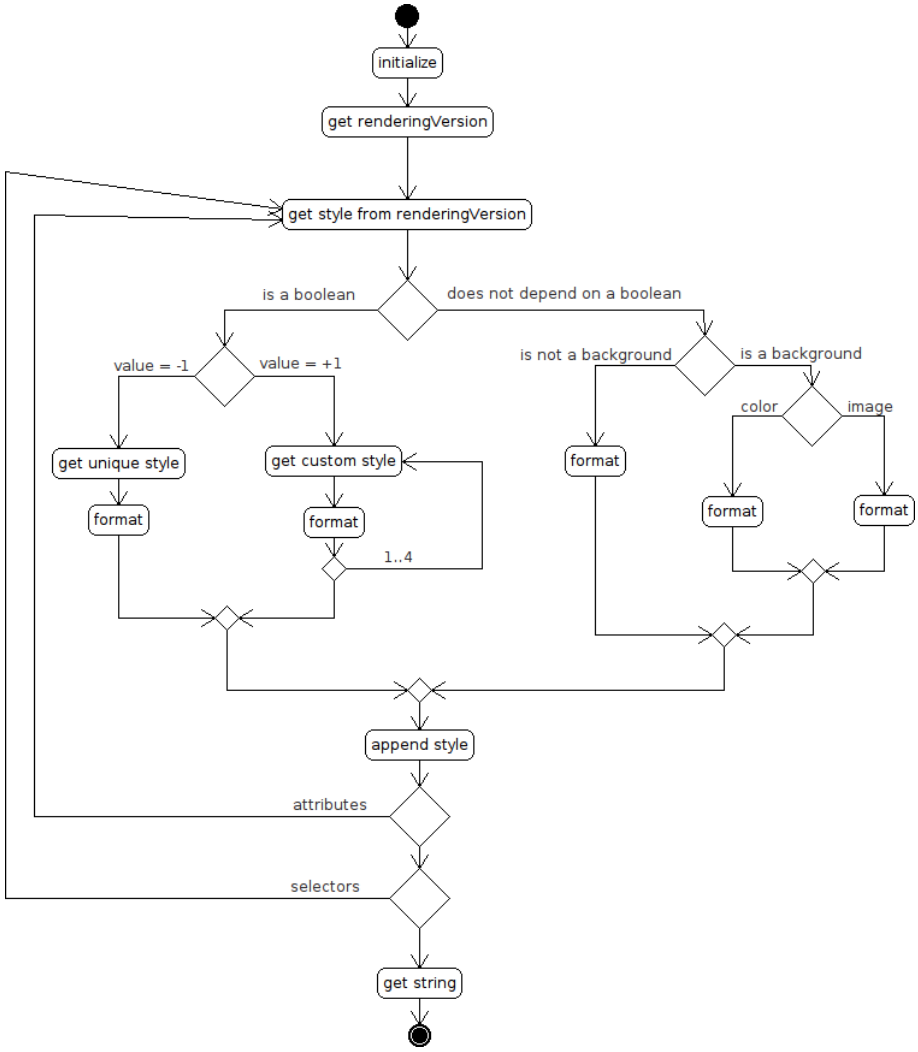


Figura 16: Funzione `buildThemePreview`

```
} else if(TWVersionLocalServiceUtil.testPublishingVersion(themeId
    )){
  renderingVersionId = TWVersionLocalServiceUtil.getVersions(
      themeId,
    TWConstants.PREVIEW_MAJOR_NUMBER).get(0).getVersionId();
} else {
  renderingVersionId = TWVersionLocalServiceUtil.getVersions(
      themeId).get(0).getVersionId();
}
TWStyle style = null;

for (int iterS = 0 ; iterS < TWConstants.SELECTORS.length ; iterS
    ++) {

  if(Validator.isNotNull(TWConstants.SELECTORS[iterS])) {

    StringBuilder tmpSb = new StringBuilder();
    boolean selectorHasBeenEdited = false;

    for(int iterSA = 0 ; iterSA < TWConstants.
        SELECTORS_ATTRIBUTES[iterS].length ; iterSA++) {

      if(Validator.isNotNull(TWConstants.SELECTORS_ATTRIBUTES[
          iterS][iterSA])) {

        int iterA = TWConstants.SELECTORS_ATTRIBUTES[iterS][
            iterSA];

        style = TWStyleLocalServiceUtil.getStyle(
            renderingVersionId,
            iterS,
            iterA);

        if(TWConstants.ATTRIBUTE_TYPES[iterA][0] == 22) {

          if(style.getValue() == -1) {

            TWStyle style2 = TWStyleLocalServiceUtil.getStyle(
                renderingVersionId, iterS, iterA-1);
            String hexer = Long.toHexString(style2.getBackground
                ());
            int elType = style2.getType();
            double elValue = style2.getValue();

            if (elType == 0 && Validator.isNotNull(hexer)){

              tmpSb.append("\t");
              tmpSb.append(TWConstants.ATTRIBUTES[iterA-1]);
              tmpSb.append(": ");

              tmpSb.append("#");
```

```
                if(hexer.equalsIgnoreCase("1000000")) {
                  hexer = "000000";
                }else if(hexer.length() < 6) {
                  int i = hexer.length();
                  while(i < 6) {
                    tmpSb.append("0");
                    i++;
                  }
                }

                tmpSb.append(hexer.toUpperCase());
                tmpSb.append(" ");
                tmpSb.append(TWConstants.TYPES[elType]);
                tmpSb.append(";");
                if (verbose) tmpSb.append("\n");
                selectorHasBeenEdited = true;

              } else {
                if(style.getType() > 0) {
                  if(verbose) tmpSb.append("\t");
                  tmpSb.append(TWConstants.ATTRIBUTES[iterA-1]);
                  tmpSb.append(": ");
                  if(TWConstants.HAS_INPUT[iterA-1] == 1) {
                    tmpSb.append(elValue);
                  }
                  tmpSb.append(TWConstants.TYPES[elType]);
                  tmpSb.append(";");
                  if (verbose) tmpSb.append("\n");
                  selectorHasBeenEdited = true;
                }
              }
            } else {
              for(int inc = 1 ; inc <=4 ; inc++) {

                TWStyle style2 = TWStyleLocalServiceUtil.getStyle(
                    renderingVersionId, iterS, iterA+inc);

                String hexer = Long.toHexString(style2.
                    getBackground());
                int elType = style2.getType();
                double elValue = style2.getValue();

                if (elType == 0 && Validator.isNotNull(hexer) && !
                    hexer.equals("0")){
                  tmpSb.append("\t");
                  tmpSb.append(TWConstants.ATTRIBUTES[iterA+inc]);
                  tmpSb.append(": ");

                  tmpSb.append("#");

                  if(hexer.equalsIgnoreCase("1000000")) {
                    hexer = "000000";
```

```
          }else if(hexer.length() < 6) {
            int i = hexer.length();
            while(i < 6) {
              tmpSb.append("0");
              i++;
            }
          }

          tmpSb.append(hexer.toUpperCase());
          tmpSb.append(" ");
          tmpSb.append(TWConstants.TYPES[elType]);
          tmpSb.append(";");
          if (verbose) tmpSb.append("\n");
          selectorHasBeenEdited = true;

        } else {
          if(style.getType() > 0) {
            if(verbose) tmpSb.append("\t");
            tmpSb.append(TWConstants.ATTRIBUTES[iterA+inc])
                ;
            tmpSb.append(": ");
            if (TWConstants.HAS_INPUT[iterA+inc] == 1) {
              tmpSb.append(elValue);
            }
            tmpSb.append(TWConstants.TYPES[elType]);
            tmpSb.append(";");
            if (verbose) tmpSb.append("\n");
            selectorHasBeenEdited = true;
          }
        }
      }
    }
    selectorHasBeenEdited = true;
  }

  if(TWConstants.DEPENDS_ON_BOOLEAN[iterA] == 0) {
    TWStyle style2 = TWStyleLocalServiceUtil.getStyle(
        renderingVersionId, iterS, iterA);
    String hexer = Long.toHexString(style2.getBackground())
        ;
    long bgId = style2.getBackground();
    int elType = style2.getType();
    double elValue = style2.getValue();

    if (TWConstants.ATTRIBUTE_BG[iterA] > 0 && Validator.
        isNotNull(style.getBackground())){

      if(style.getType() == 4) {

        String bgNm = TWThemeImageLocalServiceUtil.
            getTWThemeImage(bgId).getName();
        if(verbose) tmpSb.append("\t");
```

```
            tmpSb.append(TWConstants.ATTRIBUTES[iterA]);
            if(publishing) {
              tmpSb.append(": url(../images/common/"+bgNm+")");
            } else {
              tmpSb.append(": url(../images/common/"+bgNm+")");
            }
            tmpSb.append(";");
            if (verbose) tmpSb.append("\n");
            selectorHasBeenEdited = true;
          } else {
            tmpSb.append("\t");
            tmpSb.append(TWConstants.ATTRIBUTES[iterA]);
            tmpSb.append(": ");
            tmpSb.append("#");

            if(hexer.equalsIgnoreCase("1000000")) {
              hexer = "000000";
            }else if(hexer.length() < 6) {
              int i = hexer.length();
              while(i < 6) {
                tmpSb.append("0");
                i++;
              }
            }
            tmpSb.append(hexer.toUpperCase());
            tmpSb.append(" ");
            tmpSb.append(TWConstants.TYPES[elType]);
            tmpSb.append(";");
            if (verbose) tmpSb.append("\n");
            selectorHasBeenEdited = true;
          }
        } else if(style.getType() > 0) {
          if(verbose) tmpSb.append("\t");
          tmpSb.append(TWConstants.ATTRIBUTES[iterA]);
          tmpSb.append(": ");
          if (TWConstants.HAS_INPUT[iterA] == 1) {
            tmpSb.append(elValue);
          }
          tmpSb.append(TWConstants.TYPES[elType]);
          tmpSb.append(";");
          if (verbose) tmpSb.append("\n");
          selectorHasBeenEdited = true;
        }
      }
    }
  }

  if(selectorHasBeenEdited) {
    sb.append(TWConstants.SELECTORS[iterS] + (verbose ? " {\n"
        : "{" )  );
    sb.append(tmpSb);
    sb.append(verbose ? "}\n\n" : "}");
```

```
      }
    }
  }
  System.out.println(sb.toString());
  return sb.toString();
}
```

Il metodo `writeZip` crea l'archivio compresso contenente la struttura `_diffs` , in cui ha precedentemente inserito il file `custom.css` e le immagini usate nel tema. Per creare l'archivio ci si appoggia alla classe `ZipWriter` fornita da Liferay.

`writeZip`

```
public static byte[] writeZip(long themeId)
throws SystemException, PortalException {

  try {
    String prova = createResultTheme(themeId);
    ZipWriter zipWriter = new ZipWriter();

    zipWriter.addEntry("/_diffs/css/custom.css", prova);
    List<TWThemeImage> images = TWThemeLocalServiceUtil.getImages(
        themeId);
    for (TWThemeImage image : images) {
      if(image.getIsUsed()) {
        byte[] bytes = ImageLocalServiceUtil.getImage(image.
            getImageId()).getTextObj();
        zipWriter.addEntry("/_diffs/images/wizard/"+image.getName()
            , bytes);
      }
    }
    return zipWriter.finish();
  } catch (Exception e) {
    System.out.println(e);
  }
  return new byte[] {};
}
```

Il metodo `createPreviewTheme` si occupa di generare lo stile da iniettare dinamicamente nella pagina di preview. In maniera diretta, invocando il metodo `createResultTheme`, se si sta trattando una versione completa (vale a dire una versione pubblicata o una minor), oppure incaricando il metodo `buildThemePreview` di ricostruire una versione completa nel caso in cui si stia chiedendo di visualizzare la preview di una draft. Questa distinzione è necessaria in quanto il metodo `createResultTheme` tratta solo versioni complete.

`createPreviewTheme`

```
public static String createPreviewTheme(long themeId)
throws PortalException, SystemException {

  if(TWVersionLocalServiceUtil.testDraftVersion(themeId) ||
      TWVersionLocalServiceUtil.testPublishingVersion(themeId)) {
    return TWVersionLocalServiceUtil.buildThemePreview(themeId);
  } else {
```

```
    return createResultTheme(themeId);
  }
}
```

La taglib

Per semplificare la creazione delle finestre di inserimento modifiche creiamo una taglib che, dati i parametri riguardanti le modifiche che interessa proporre, esponga gli input di conseguenza. La taglib gestirà volta per volta un singolo stile del tema. Se uno stile è stato modificato nella sessione di lavoro, e quindi è presente nella draft del tema, il valore viene visualizzato come default del campo di input.

La taglib apparterrà al pacchetto taglib `prjmoon-ui`, quindi la chiameremo con il nome `prjmoon-ui:css-settings`.

Definizione della taglib

Il primo passo è definire i parametri che la taglib avrà, specificandone il nome e l'obbligatorietà. Tale operazione si compie modificando il file textscxml `prjmoon-ui.tld` (TagLib Definition).

Gli attributi che ci interessano sono:

- selettore
- attributo
- tipo
- value
- background

che sono gli attributi degli stili. Ad essi aggiungiamo

- images

vale a dire la lista delle immagini disponibili per un tema.

Vediamo di seguito la porzione di codice che ci interessa:

```
<tag>
   <name>css-settings</name>
   <tag-class>com.ext.taglib.ui.CssSettingsTag</tag-class>
   <body-content>JSP</body-content>
   <attribute>
      <name>selector</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
      <name>attribute</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
      <name>type</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
```

```
      <name>value</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
      <name>background</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
      <name>images</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
</tag>
```

Al codice sopra listato occorre affiancare la classe `CssSettingsTag`, definita nel file `CssSettingsTab.java` `CssSettingsTag`

```
package com.ext.taglib.ui;

import javax.servlet.http.HttpServletRequest;

import com.liferay.taglib.util.IncludeTag;

public class CssSettingsTag extends IncludeTag {

   public int doStartTag() {
      HttpServletRequest request =
         (HttpServletRequest)pageContext.getRequest();

      request.setAttribute("prjmoon-ui:css-settings:type", _type);
      request.setAttribute("prjmoon-ui:css-settings:value", _value)
          ;
      request.setAttribute("prjmoon-ui:css-settings:background",
          _background);
      request.setAttribute("prjmoon-ui:css-settings:selector",
          _selector);
      request.setAttribute("prjmoon-ui:css-settings:attribute",
          _attribute);
      request.setAttribute("prjmoon-ui:css-settings:images",
          _images);

      return EVAL_BODY_BUFFERED;
   }

   public int getType() {
      return _type;
   }

   public void setType(int type) {
      _type = type;
   }
```

```
public double getValue() {
   return _value;
}

public void setValue(double value) {
   _value = value;
}

public long getBackground() {
   return _background;
}

public void setBackground(long background) {
   _background = background;
}

public int getSelector() {
   return _selector;
}

public void setSelector(int selector) {
   _selector = selector;
}

public void setAttribute(int attribute) {
   _attribute = attribute;
}

public int getAttribute() {
   return _attribute;
}

public void setImages(String images) {
   _images = images;
}

public String getImages() {
   return _images;
}

protected String getDefaultPage() {
   return _PAGE;
}

private static final String _PAGE =
   "/html/taglib/ext/ui/css_settings/page.jsp";

private int _type;
private double _value;
private long _background;
private int _selector;
```

```
    private int _attribute;
    private String _images;
}
```

Con questi due file abbiamo definito gli attributi della taglib e i metodi `getter` e `setter` per ciascuno di essi.

Vediamo quindi il codice JSP che definisce il comportamento della taglib, e, conseguentemente, l'aspetto dei campi di input. *Codice della taglib*

```
<%@ include file="/html/taglib/init.jsp" %>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<%@ page import="com.liferay.portal.kernel.util.Validator" %>
<%@ page import="com.ext.portal.util.TWConstants" %>
<%@ page import="com.liferay.portal.kernel.util.GetterUtil"%>
<%@ page import="java.util.ArrayList"%>
<%@ page import="com.liferay.portal.kernel.util.StringUtil"%>
<%
int selectorIndex = GetterUtil.getInteger(request.getAttribute("
    prjmoon-ui:css-settings:selector").toString());
int attributeIndex = GetterUtil.getInteger(request.getAttribute("
    prjmoon-ui:css-settings:attribute").toString());
int baseType = GetterUtil.getInteger(request.getAttribute("prjmoon-
    ui:css-settings:type").toString());
double baseValue = GetterUtil.getDouble(request.getAttribute("
    prjmoon-ui:css-settings:value").toString());
long baseBackground = GetterUtil.getLong(request.getAttribute("
    prjmoon-ui:css-settings:background").toString());
String images = request.getAttribute("prjmoon-ui:css-
    settings:images").toString();

String[] imgIdsStr = images.split(",");
String[] imgIds = StringUtil.split(images,",");

String selector = TWConstants.SELECTORS[selectorIndex];
String attribute = TWConstants.ATTRIBUTES[attributeIndex];
String type = TWConstants.TYPES[baseType];
%>

<script type="text/javascript">
function <portlet:namespace />isChanged(name,revert) {
  document.<portlet:namespace />fm[name].value = "changed";
  jQuery('#'+revert).show();
}

function <portlet:namespace />revert(name) {
  var answer = confirm("Revert values for this attribute?");
  if(answer) {
    nameM = name + '_M';
    nameT = name + '_T';
    nameV = name + '_V';
    nameB = name + '_B';
    nameR = name + '_REVERT';
```

```
    document.<portlet:namespace />fm[nameM].value = "unchanged";
    document.<portlet:namespace />fm[nameT].value = "<%= baseType
        %>";
    document.<portlet:namespace />fm[nameT].selected = "<%= false
        %>";
    document.<portlet:namespace />fm[nameV].value = "<%= baseValue
        %>";
    document.<portlet:namespace />fm[nameB].value = "<%=
        baseBackground %>";
    jQuery('#'+nameR).hide();
  }
}
</script>

<dt class="dt-attributes"><%= attribute %></dt>
<dd class="dd-attributes">
<input name="<portlet:namespace /><%= selectorIndex %>_<%=
    attributeIndex %>_M" type="hidden" value="unchanged" />

<c:choose>

  <c:when test="<%= TWConstants.ATTRIBUTE_BG[attributeIndex] == 0
      %>" >

  <%if(TWConstants.ATTRIBUTE_TYPES[attributeIndex][0] == 22) {%>

    choose one:

    <input type="radio"
      name="<portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_V"
      value="-1";
      onchange="<portlet:namespace />isChanged('<portlet:namespace
          /><%= selectorIndex %>_<%= attributeIndex %>_M',
        '<portlet:namespace /><%= selectorIndex %>_<%=
            attributeIndex %>_REVERT');"
      <%= ( baseValue < 0 ? "checked='checked'" : "") %> >
    don't use <%= attribute %> - <%= baseValue %>
    </input>

    <input type="radio"
      name="<portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_V"
      value="1";
      onchange="<portlet:namespace />isChanged('<portlet:namespace
          /><%= selectorIndex %>_<%= attributeIndex %>_M',
        '<portlet:namespace /><%= selectorIndex %>_<%=
            attributeIndex %>_REVERT');"
      <%= ( baseValue > 0 ? "checked='checked'" : "") %> >
    use <%= attribute %> - <%= baseValue %>
    </input>
```

```
  <input type="hidden" name="<portlet:namespace /><%=
      selectorIndex %>_<%= attributeIndex %>_T"
    value="22" />

<%} else { %>

  <%if (TWConstants.HAS_INPUT[attributeIndex] == 1) {%>
    <input name="<portlet:namespace /><%= selectorIndex %>_<%=
        attributeIndex %>_V" type="text" value="<%= baseValue %>"
      onchange="<portlet:namespace />isChanged('<
          portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_M',
        '<portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_REVERT');" />
  <%} %>

  <%if(TWConstants.ATTRIBUTE_TYPES[attributeIndex][0] > 0) {%>
    <select name="<portlet:namespace /><%= selectorIndex %>_<%=
        attributeIndex %>_T" type="text" value=""
      onchange="<portlet:namespace />isChanged('<
          portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_M',
        '<portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_REVERT');" >
        <option value=""> </option>
      <%for(int iterAttType = 0 ; iterAttType < TWConstants.
          ATTRIBUTE_TYPES[attributeIndex].length ; iterAttType++)
           { %>
        <option value="<%=TWConstants.ATTRIBUTE_TYPES[
            attributeIndex][iterAttType] %>"
          <%= (TWConstants.ATTRIBUTE_TYPES[attributeIndex][
              iterAttType] == baseType ? "selected=\"selected\""
              : "") %>
        >
          <%=TWConstants.TYPES[TWConstants.ATTRIBUTE_TYPES[
              attributeIndex][iterAttType]] %>
        </option>
      <%} %>
    </select>
  <%} %>

  <input type="button" value="revert" id="<portlet:namespace
      /><%= selectorIndex %>_<%= attributeIndex %>_REVERT" style
      ="display:none"
    onclick="<portlet:namespace />revert('<portlet:namespace /><
      %= selectorIndex %>_<%= attributeIndex %>')" />

<%} %>

</c:when>

<c:otherwise>
```

```
<select name="<portlet:namespace /><%= selectorIndex %>_<%=
    attributeIndex %>_T" type="text" value=""
  onchange="<portlet:namespace />isChanged('<portlet:namespace
      /><%= selectorIndex %>_<%= attributeIndex %>_M',
   '<portlet:namespace /><%= selectorIndex %>_<%= attributeIndex
       %>_REVERT');" >

  <%for(int iterAttType = 0 ; iterAttType < TWConstants.
      ATTRIBUTE_TYPES[attributeIndex].length ; iterAttType++) {
       %>
    <option value="<%=TWConstants.ATTRIBUTE_TYPES[
        attributeIndex][iterAttType] %>"
      "<%= (TWConstants.ATTRIBUTE_TYPES[attributeIndex][
          iterAttType] == baseType ? "selected=\"selected\"" :
          "") %>"
    >
      <%=TWConstants.TYPES[TWConstants.ATTRIBUTE_TYPES[
          attributeIndex][iterAttType]] %>
    </option>
  <%} %>
</select>

<%if (attributeIndex == 61) {   %>
  <select name="<portlet:namespace /><%= selectorIndex %>_<%=
      attributeIndex %>_B" type="text" value=""
      onchange="<portlet:namespace />isChanged('<
          portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_M',
      '<portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_REVERT');" >
    <option value=""> </option>
    <%for(int iterImgs = 0 ; iterImgs < imgIds.length ;
        iterImgs++) { %>
      <option value="<%= imgIds[iterImgs] %>"
        <%= (imgIds[iterImgs].equalsIgnoreCase(""+
            baseBackground) ?
                "selected=\"selected\"" : ""
        ) %>
      >
        <%= imgIds[iterImgs] %>
      </option>
    <%} %>
  </select>

<%} else { %>

  <%
  String tmpDisplayedValue = Long.toHexString(baseBackground).
      toUpperCase();
  String displayedValue = "#";
  if(tmpDisplayedValue.length() < 6) {
```

```
        int i = tmpDisplayedValue.length();
        String dv = "";
        while(i < 6) {
          dv += "o";
          i++;
        }
        displayedValue += dv;
      }
      displayedValue += tmpDisplayedValue;
      %>

      <input name="<portlet:namespace /><%= selectorIndex %>_<%=
          attributeIndex %>_B"
           id="<portlet:namespace /><%= selectorIndex %>_<%=
               attributeIndex %>_B"
           type="text"
           value="<%= displayedValue %>" />
      <script type="text/javascript">
        jQuery( function() {
          var <portlet:namespace /><%= selectorIndex %>_<%=
              attributeIndex %>_B = new Liferay.ColorPicker( {
            item: "#<portlet:namespace /><%= selectorIndex %>_<%=
                attributeIndex %>_B",
            onChange: function () { <portlet:namespace />isChanged(
              "<portlet:namespace /><%= selectorIndex %>_<%=
                  attributeIndex %>_M",
              "<portlet:namespace /><%= selectorIndex %>_<%=
                  attributeIndex %>_REVERT"
            ) },
            onClose: function () {<portlet:namespace />isChanged(
              "<portlet:namespace /><%= selectorIndex %>_<%=
                  attributeIndex %>_M",
              "<portlet:namespace /><%= selectorIndex %>_<%=
                  attributeIndex %>_REVERT"
            ) },
          } );
        } );
      </script>
    <% } %>

    <input type="button" value="revert" name="<portlet:namespace
        /><%= selectorIndex %>_<%= attributeIndex %>_REVERT" style
        ="display:none"
      onclick="<portlet:namespace />revert('<portlet:namespace /><
          %= selectorIndex %>_<%= attributeIndex %>_REVERT')" />

  </c:otherwise>

</c:choose>
```

Analizziamone il codice.

Innanzitutto recuperiamo i parametri che abbiamo passato al momento della chiamata della taglib, ovvero i parametri definiti in `CssSettingsTab.java`. Dividiamo la stringa contenente gli ID delle immagini negli ID corrispondenti. Inseriamo quindi il codice Javascript per gestire la modifica e il revert dei valori inseriti.

Andiamo quindi ad analizzare lo stile ricevuto, che la taglib dovrà gestire, vale a dire riconoscere e predisporre di conseguenza gli input.

Innanzitutto, gli attributi *use-custom* diventeranno dei *radio button* con due opzioni:

1. usa un unico attributo;
2. usa 4 attributi custom diversi.

Per i restanti attributi verranno mostrati campi di input e/o menu dropdown, a seconda del contento degli array di costanti che abbiamo opportunamente predisposto. Se l'attributo è un colore verrà visualizzato il `ColorPicker` di jQuery. Se invece è un'immagine apparirà l'elenco delle immagini disponibili per quel tema.

La modifica di un attributo verrà registrata da un'apposito input *hidden*. Inoltre, la modifica di un attributo causerà la visualizzazione di un pulsante di *revert*, che, se premuto, annullerà le modifiche apportate su quell'elemento.

Di default, i campi di input visualizzeranno il valore *attuale* dell'attributo, ed è per questo che importiamo e gestiamo opportunamente i parametri dello stile.

L'interfaccia

L'interfaccia si gestisce mediante il codice JSP dei file nella directory `/docroot/html/theme_wizard/`. Vediamo in dettaglio di cosa abbiamo bisogno.

INIT.JSP In questo file si definiscono le dipendenze importando le librerie e gli oggetti necessari, e si creano alcune variabili usate in ogni punto del portlet. Viene importato in ogni altro file JSP del portlet.

```
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %
    >
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<%@ taglib uri="http://liferay.com/tld/portlet" prefix="liferay-
    portlet" %>
<%@ taglib uri="http://liferay.com/tld/theme" prefix="liferay-theme
    " %>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>
<%@ taglib uri="http://liferay.com/tld/util" prefix="liferay-util"
    %>
<%@ taglib uri="http://prjmoon.com/tld/ui" prefix="prjmoon-ui" %>

<%@ page import="it.prjmoon.portlet.themewizard.model.TWTheme"%>
<%@ page import="it.prjmoon.portlet.themewizard.model.TWThemeImage"
    %>
<%@ page import="it.prjmoon.portlet.themewizard.service.
    TWThemeLocalServiceUtil"%>
<%@ page import="it.prjmoon.portlet.themewizard.service.
    TWThemeImageLocalServiceUtil"%>
```

```
<%@ page import="it.prjmoon.portlet.themewizard.service.impl.
    TWThemeImageLocalServiceImpl"%>
<%@ page import="it.prjmoon.portlet.themewizard.util.TWConstants"%>

<%@ page import="com.liferay.portal.kernel.dao.search.
    SearchContainer"%>
<%@ page import="com.liferay.portal.kernel.dao.search.SearchEntry"%
    >
<%@ page import="com.liferay.portal.kernel.dao.search.ResultRow"%>
<%@ page import="com.liferay.portal.kernel.portlet.
    LiferayWindowState"%>
<%@ page import="com.liferay.portal.kernel.util.ParamUtil"%>
<%@ page import="com.liferay.portal.util.PortalUtil"%>
<%@ page import="com.liferay.portal.kernel.util.StringPool"%>
<%@ page import="com.liferay.portal.kernel.util.WebKeys"%>

<%@ page import="java.util.ArrayList"%>
<%@ page import="java.util.List"%>

<%@ page import="javax.portlet.PortletURL"%>
<%@ page import="javax.portlet.WindowState"%>

<%@ page import="javax.portlet.PortletMode"%>
<%@ page import="javax.portlet.PortletRequest" %>
<%@ page import="javax.portlet.PortletPreferences" %>
<%@ page import="javax.portlet.PortletURL" %>
<%@ page import="javax.portlet.WindowState" %>


<portlet:defineObjects />
<liferay-theme:defineObjects />

<%
String currentURL = PortalUtil.getCurrentURL(request);
WindowState windowState = renderRequest.getWindowState();
%>
```

VIEW.JSP È il file "base", quello che determina il comportamento della schermata iniziale del portlet.

```
<%@ page import="com.ext.portlet.partners.service.
    PartnerWorkerLocalServiceUtil"%>
<%@ include file="/html/theme_wizard/init.jsp" %>

<%
String names1 = "user–themes";
boolean isWorker = false;

if (PartnerWorkerLocalServiceUtil.containsUser(themeDisplay.
    getUserId())) {
   isWorker = true;
}
```

```
if(isWorker) {
   names1 += ",partner-themes";
}

String tabs1 = ParamUtil.getString(request, "tabs1", "user-themes")
    ;
PortletURL portletURL = renderResponse.createRenderURL();

portletURL.setParameter("tabs1", tabs1);
portletURL.setWindowState(WindowState.MAXIMIZED);
pageContext.setAttribute("portletURL", portletURL);

%>

<liferay-portlet:renderURLParams varImpl="portletURL" />

<liferay-ui:tabs
   names="<%= names1 %>"
   url="<%= portletURL.toString() %>"
/>
<%= tabs1 %>
<c:choose>
   <c:when test="<%= tabs1.equals(\"user-themes\") %>">
      <%@ include file="/html/theme_wizard/view_theme.jspf" %>
   </c:when>
   <c:otherwise>
      Hallo, partner
   </c:otherwise>
</c:choose>
```

EDIT_THEME.JSP In questo file si predispongono i campi di inserimenti per i dati dei temi quando questi vengono creati.

```
<%
long twThemeId = ParamUtil.getLong(request, "twThemeId");
String redirect = ParamUtil.getString(request, "redirect");
%>
<script type="text/javascript">
   function <portlet:namespace />themeAction() {
      submitForm(document.<portlet:namespace />fm,
               '<portlet:actionURL name="updateTheme" />');
   }
</script>

<form name="<portlet:namespace />fm" method="post" onsubmit="<
    portlet:namespace />themeAction(); return false;">
<input type="hidden" name="<portlet:namespace />redirect" value="
    <%= redirect %>" />
<input type="hidden" name="<portlet:namespace />twThemeId" value="
    <%= twThemeId %>" />
<table>
```

```
<tr>
   <td>
      <liferay-ui:message key="name" />
   </td>
   <td>
      <liferay-ui:input-field model="<%= TWTheme.class %>" bean="
          <%= twThemeId %>" field="name" defaultValue="<%=twThemeId
           > 0 ? TWThemeLocalServiceUtil.getTheme(twThemeId).
          getName() : \"\" %>" />
   </td>
</tr>
<tr>
   <td>
      <liferay-ui:message key="description" />
   </td>
   <td>
      <liferay-ui:input-field model="<%= TWTheme.class %>" bean="
          <%= twThemeId %>" field="description" defaultValue="<%=
          twThemeId > 0 ? TWThemeLocalServiceUtil.getTheme(
          twThemeId).getDescription() : \"\" %>" />
   </td>
</tr>

<tr>
   <td>
      <liferay-ui:message key="limitedGroupId" />
   </td>
   <td>
      <liferay-ui:input-field model="<%= TWTheme.class %>" bean="
          <%= twThemeId %>" field="limitedGroupId" defaultValue="
          <%=twThemeId > 0 ? TWThemeLocalServiceUtil.getTheme(
          twThemeId).getLimitedGroupId() : \"\" %>" />
   </td>
</tr>
<tr>
   <td></td>
   <td><input type="submit" value="<liferay-ui:message key="save"
       />">
      <c:if test="<%= (twThemeId == 0) %>" >
         <input type="reset" value="<liferay-ui:message key="reset"
             />">
      </c:if>
      <input type="button" value="<liferay-ui:message key="cancel"
          />"
                        onclick="history.back();"/>
   </td>
</tr>
</table>
</form>
```

VIEW_THEME.JSP Elenca i temi creati da un partner, e ne consente la modifica.

```
<%@ include file="/html/theme_wizard/init.jsp" %>
<%@ page import="com.liferay.portal.kernel.dao.search.SearchEntry"%
    >

<form name="<portlet:namespace />fm" method="post" onsubmit="<
    portlet:namespace />addValues(); return false;">

<%
   List headerNames = new ArrayList();

   headerNames.add("name");
   headerNames.add("description");
   headerNames.add(StringPool.BLANK);

   PortletURL portletURL = renderResponse.createRenderURL();

   SearchContainer searchContainer = new SearchContainer(
         renderRequest, null, null,
         SearchContainer.DEFAULT_CUR_PARAM,
         SearchContainer.DEFAULT_DELTA, portletURL, headerNames,
         "no-requests-were-found");

   // We list alll the themes created by the user as a user

   long partnerId = 0;

   List results = TWThemeLocalServiceUtil.getThemes(partnerId,
         themeDisplay.getUserId(), searchContainer.getStart(),
         searchContainer.getEnd());

   int total = results.size();

   searchContainer.setTotal(total);

   searchContainer.setResults(results);
%>

<br /><br />

<%
   List resultRows = searchContainer.getResultRows();

   for (int i = 0; i < results.size(); i++) {
      TWTheme twTheme = (TWTheme) results.get(i);
      ResultRow row = new ResultRow(twTheme, twTheme.getThemeId(),
          i);
      PortletURL rowURL = renderResponse.createRenderURL();

      rowURL.setWindowState(WindowState.MAXIMIZED);
```

```
      rowURL.setParameter("jspPage",
            "/html/theme_wizard/edit_theme.jsp");
      rowURL.setParameter("twThemeId",
            String.valueOf(twTheme.getThemeId()));
      rowURL.setParameter("redirect", currentURL);

      row.addText(twTheme.getName(), rowURL);      // Name
      row.addText(twTheme.getDescription(), rowURL);  //
          Description
      row.addJSP("right", SearchEntry.DEFAULT_VALIGN,
            "/html/theme_wizard/theme_action.jsp", application,
            request, response);           // Actions
      resultRows.add(row);   // Add result row
   }
%>

<liferay-ui:search-iterator searchContainer="<%= searchContainer %>
    " />

<%
   PortletURL createURL = renderResponse.createRenderURL();
   createURL.setParameter("jspPage",
         "/html/theme_wizard/edit_theme.jsp");
   createURL.setParameter("twThemeId", String.valueOf(0));
   createURL.setParameter("redirect", currentURL);
   createURL.setWindowState(WindowState.MAXIMIZED);
%>

<input type="button" value="<liferay-ui:message key="create" />"
    onclick="location.href ='<%=createURL.toString()%>'" />
</form>
```

THEME_ACTION.JSP Imposta la destinazione dell'azione del pulsante di editing di tema, che nel nostro caso è il file `work_on_theme.jsp`.

```
<%
ResultRow resultRow = (ResultRow)request.getAttribute(WebKeys.
    SEARCH_CONTAINER_RESULT_ROW);
TWTheme twTheme = (TWTheme)resultRow.getObject();
%>
<liferay-ui:icon-menu>
   <portlet:renderURL windowState="<%= WindowState.MAXIMIZED.
       toString() %>" var="editURL">
      <portlet:param name="jspPage" value="/html/theme_wizard/
          work_on_theme.jsp" />
      <portlet:param name="redirect" value="<%= currentURL %>" />
      <portlet:param name="twThemeId" value="<%= String.valueOf(
          twTheme.getThemeId()) %>" />
   </portlet:renderURL>
   <liferay-ui:icon image="edit" url="<%= editURL %>" />
</liferay-ui:icon-menu>
```

WORK_ON_THEME.JSP È la maschera che nell'analisi abbiamo chiamato "concentratore". Contiene i pulsanti per agire sui vari aspetti del tema, mediante finestre specifiche, per le quali esiste un file `jsp` ciascuna.

```
<%
long twThemeId = ParamUtil.getLong(request, "twThemeId");
String redirect = ParamUtil.getString(request, "redirect");
%>
<script type="text/javascript">
function <portlet:namespace />preview() {
   var previewWindow = window.open('<portlet:renderURL
      windowState="<%=LiferayWindowState.POP_UP.toString()%>">
      <portlet:param name="jspPage" value="/html/theme_wizard/
          preview.jsp" />
      <portlet:param name="twThemeId" value="<%=String.valueOf(
          twThemeId)%>" />
      </portlet:renderURL>',
      'preview', 'directories=no,height=600,location=no,menubar=no,
          resizable=yes,
      scrollbars=yes,status=no,toolbar=no,width=1020');
   void('');
   previewWindow.focus();
}

function <portlet:namespace />publish() {
   <portlet:actionURL name="publishVersion" var="publishVersionURL"
       >
      <param name="twThemeId" value="<%= twThemeId %>" />
      <param name="redirect" value="<%= redirect %>" />
   </portlet:actionURL>
   submitForm(document.<portlet:namespace />fm, "<%=
       publishVersionURL.toString() %>");
}

function <portlet:namespace />saveVersion() {
   submitForm(document.<portlet:namespace />fm);
}
</script>

<portlet:actionURL name="saveVersion" var="saveVersion">
   <param name="twThemeId" value="<%= twThemeId %>" />
</portlet:actionURL>


<form name="<portlet:namespace />fm" method="post" onsubmit="<
    portlet:namespace />saveVersion();" action="<%= saveVersion %>"
    >
<input type="hidden" name="<portlet:namespace />redirect" value="
    <%=currentURL%>" />
<input type="hidden" name="<portlet:namespace />twThemeId" value="
    <%=twThemeId%>" />
<table>
```

```
<tr>
   <td><input type="button" class="button"
      value="<liferay–ui:message key="edit-images" />"
      onClick="var imagesWindow = window.open('<
          portlet:renderURL
         windowState="<%= LiferayWindowState.POP_UP.toString() %
             >">
         <portlet:param name="jspPage" value="/html/theme_wizard
             /images_window.jsp" />
         <portlet:param name="redirect" value="<%= currentURL %>
             " />
         <portlet:param name="twThemeId" value="<%= String.
             valueOf(twThemeId) %>" />
         </portlet:renderURL>',
         'images',
         'directories=no,height=640,location=no,menubar=no,
             resizable=yes,
         scrollbars=yes,status=no,toolbar=no,width=680');
         void('');
         imagesWindow.focus();" />
   </td>
</tr>
<tr>
   <td><input type="button" class="button"
      value="<liferay–ui:message key="edit-general" />"
      onClick="var generalWindow = window.open('<
          portlet:renderURL
         windowState="<%= LiferayWindowState.POP_UP.toString() %
             >">
         <portlet:param name="jspPage" value="/html/theme_wizard
             /edit_general.jsp" />
         <portlet:param name="twThemeId" value="<%= String.
             valueOf(twThemeId) %>" />
         </portlet:renderURL>',
         'general', 'directories=no,height=640,location=no,
             menubar=no,resizable=yes,
         scrollbars=yes,status=no,toolbar=no,width=680');
         void('');
         generalWindow.focus();" />
   </td>
</tr>
<tr>
   <td><input type="button" class="button"
      value="<liferay–ui:message key="edit-navigation" />"
      onClick="var navigationWindow = window.open('<
          portlet:renderURL
         windowState="<%= LiferayWindowState.POP_UP.toString() %
             >">
         <portlet:param name="jspPage" value="/html/theme_wizard
             /edit_navigation.jsp" />
         <portlet:param name="twThemeId" value="<%= String.
             valueOf(twThemeId) %>" />
```

```
            </portlet:renderURL>',
            'navigation', 'directories=no,height=640,location=no,
                menubar=no,resizable=yes,
            scrollbars=yes,status=no,toolbar=no,width=680');
            void('');
            navigationWindow.focus();" />
      </td>
   </tr>
   <tr>
      <td><input type="button" class="button"
         value="<liferay-ui:message key="edit-portlet" />"
         onClick="var portletWindow = window.open('<
             portlet:renderURL
            windowState="<%= LiferayWindowState.POP_UP.toString() %
                >">
            <portlet:param name="jspPage" value="/html/theme_wizard
                /edit_portlet.jsp" />
            <portlet:param name="twThemeId" value="<%= String.
                valueOf(twThemeId) %>" />
            </portlet:renderURL>',
            'portlet', 'directories=no,height=640,location=no,
                menubar=no,resizable=yes,
            scrollbars=yes,status=no,toolbar=no,width=680');
            void('');
            portletWindow.focus();" />
      </td>
   </tr>
   <tr>
      <td>
         <input type="button"
            value="<liferay-ui:message key="preview" />"
            onclick="<portlet:namespace />preview();" />
         <input type="submit" value="<liferay-ui:message key="save"
              />" />
         <input type="button"
            value="<liferay-ui:message key="publish" />"
            onclick="<portlet:namespace />publish();" />
         <input type="button"
            value="<liferay-ui:message key="cancel" />"
            onclick="location.href='<%= redirect %>';" />
      </td>
   </tr>
</table>
</form>
```

IMAGES_WINDOW.JSP Permette il caricamento delle immagini, e visualizza le immagini associate al tema che si sta editando.

La prima parte del file è l'uploader, mentre la seconda parte fa uso del `SearchContainer` per recuperare ed elencare le immagini del tema, associando a ciscuna le informazioni richieste.

```
<%
```

```
long twThemeId = ParamUtil.getLong(request, "twThemeId");
String redirect = ParamUtil.getString(request, "redirect");
%>

<script type="text/javascript">
function <portlet:namespace />imgAction() {
   submitForm(document.<portlet:namespace />imgForm, '<
       portlet:actionURL name="addImage" />');
}
</script>

<form name="<portlet:namespace />imgForm" enctype="multipart/form-
    data" method="post" onsubmit="<portlet:namespace />imgAction();
     return false;">
<input type="hidden" name="<portlet:namespace />redirect" value="
    <%=currentURL%>" />
<input type="hidden" name="<portlet:namespace />twThemeId" value="
    <%=twThemeId%>" />
<table>
   <tr>
      <td>upload an image </td>
      <td><input name="<portlet:namespace />img" size="50" type="
          file" /></td>
   </tr>
   <tr>
      <td>
         <input type="submit"
            value="<liferay-ui:message key="ok" />" />
         <input type="button"
            value="<liferay-ui:message key="close" />"
            onclick="window.close();" />
      </td>
   </tr>
</table>
</form>

<%
String tabs1 = ParamUtil.getString(request, "tabs1", "user-themes")
    ;
PortletURL portletURL = renderResponse.createRenderURL();
portletURL.setParameter("tabs1", tabs1);
portletURL.setWindowState(WindowState.MAXIMIZED);
pageContext.setAttribute("portletURL", portletURL);
List headerNames = new ArrayList();
headerNames.add("name");
headerNames.add(StringPool.BLANK);

/*PortletURL*/ portletURL = renderResponse.createRenderURL();

SearchContainer searchContainer = new SearchContainer(
      renderRequest, null, null,
      SearchContainer.DEFAULT_CUR_PARAM,
```

```
      SearchContainer.DEFAULT_DELTA, portletURL, headerNames,
      "no_images_were_found");

// We list all the images of a theme
long partnerId = 0;
List results = TWThemeLocalServiceUtil.getImages(twThemeId);
int total = results.size();
searchContainer.setTotal(total);
searchContainer.setResults(results);
%>
<br /><br />
<%
List resultRows = searchContainer.getResultRows();
for (int i = 0; i < results.size(); i++) {
   TWThemeImage twThemeImage = (TWThemeImage) results.get(i);
   ResultRow row = new ResultRow(twThemeImage,twThemeImage.
       getThemeId() i);
   PortletURL rowURL = renderResponse.createRenderURL();
   rowURL.setWindowState(WindowState.MAXIMIZED);
   row.addText(String.valueOf(twThemeImage.getName())); // name
   row.addText((twThemeImage.getIsUsed() ? "yes" : "no")); // is
       used?
   resultRows.add(row); // Add result row
}
%>
<liferay-ui:search-iterator searchContainer="<%= searchContainer %>
    " />
```

EDIT_GENERAL.JSP Consente di modificare l'aspetto generale del portlet, ovvero sfondi, header e footer, colori e stili del testo.

I file `edit_general.jsp`, `edit_general.jsp`, `edit_general.jsp` sono identici nel comportamento, l'unico aspetto che li differenzia sono i selettori che le diverse finestre vanno a mostrare. Verrà quindi spiegato il funzionamento solo di questa maschera.

Vengono in primo luogo inizializzati i parametri che verranno passati alla taglib `css-settings` che abbiamo creato più sopra. In particolare, gli ID delle immagini associate al tema vengono recuperati e inseriti in un'unica stringa. Viene inoltre recuperata la draft del tema, se non esiste la si crea.

Quindi si scorrono i selettori associati alla maschera e i relativi attributi, e per ciascun attributo si eseguono le seguenti operazioni:

1. si recupera il corrispondente stile nella draft;
   - se lo stile è presente nella draft come modificato, i suoi valori vengono settati come parametri base per la taglib;
   - se lo stile non è stato modificato nella corrente sessione di lavoro, lo si recupera dalla versione genitrice, impostandone i valori come parametri per la taglib;
2. si importa la taglib, passandole i parametri richiesti.

Infine, si inseriscono i due pulsanti, `submit` e `cancel`.

```
<script type="text/javascript">
```

```
function <portlet:namespace />saveNav() {
   submitForm(document.<portlet:namespace />fm);
}
</script>

<%
long twThemeId = ParamUtil.getLong(request, "twThemeId");
String redirect = ParamUtil.getString(request, "redirect");
TWTheme twTheme = null;
TWVersion draft = null;

int base_T = 0;
double base_V = 0;
long base_B = 0;

try {
   twTheme = TWThemeLocalServiceUtil.getTheme(twThemeId);
   draft = TWVersionLocalServiceUtil.getDraftVersion(
         twThemeId,
         themeDisplay.getUserId());
} catch (Exception e) {}

List<TWThemeImage> images = TWThemeLocalServiceUtil.getImages(
    twThemeId);
long[] imageIds = new long[images.size()];
for (int j = 0; j < images.size(); j ++) {
   imageIds[j] = images.get(j).getThemeImageId();
}
String imgs = StringUtil.merge(imageIds, ",");
%>

<c:if test="<%= Validator.isNotNull(twTheme) && Validator.isNotNull
    (draft) %>">

<%
TWVersion parentVersion = null;
if (draft.getParentVersionId() > 0) {
   parentVersion = TWVersionLocalServiceUtil.getVersion(
                  draft.getParentVersionId());
}
%>

<portlet:actionURL name="saveDraft" var="saveURL">
   <param name="twThemeId" value="<%= twThemeId %>" />
</portlet:actionURL>
<form name="<portlet:namespace />fm" method="post" onsubmit="<
    portlet:namespace />saveNav();" action="<%= saveURL %>">
   <input type="hidden" name="<portlet:namespace />twThemeId" value
       ="<%= twThemeId %>" />

   <%
```

```
for(int iterS = 0; iterS < TWConstants.EDITABLE_PROPERTIES[0].
    length ; iterS++) {
   if (Validator.isNotNull(TWConstants.EDITABLE_PROPERTIES[0][
       iterS])) {
      %>
      <liferay-ui:tabs names="<%= TWConstants.SELECTORS[
          TWConstants.EDITABLE_PROPERTIES[0][iterS]] %>" />
      <dl>
         <%
         for(int iterA = 0 ; iterA < TWConstants.
             SELECTORS_ATTRIBUTES[TWConstants.
             EDITABLE_PROPERTIES[0][iterS]].length ; iterA++) {
            int iterSA=TWConstants.SELECTORS_ATTRIBUTES[
                TWConstants.EDITABLE_PROPERTIES[0][iterS]][iterA
                ];

            TWStyle style = TWStyleLocalServiceUtil.getStyle(
                           draft.getVersionId(),
                           TWConstants.EDITABLE_PROPERTIES[0][
                               iterS],
                           iterSA
                        );

            if(style != null) {
               base_B = style.getBackground();
               base_V = style.getValue();
               base_T = style.getType();
            } else if (parentVersion != null) {
               style = TWStyleLocalServiceUtil.getStyle(
                         parentVersion.getVersionId(),
                         TWConstants.EDITABLE_PROPERTIES[0][iterS
                             ],
                         iterSA
                       );
               if(style != null) {
                  base_B = style.getBackground();
                  base_V = style.getValue();
                  base_T = style.getType();
               } else {
                  base_T = TWConstants.ATTRIBUTE_TYPES[iterSA
                      ][0];
               }
            }
            %>
            <prjmoon-ui:css-settings selector="<%= TWConstants.
                EDITABLE_PROPERTIES[0][iterS] %>"
               attribute="<%= iterSA %>"
               type="<%= base_T %>"
               value="<%= base_V %>"
               background="<%= base_B %>"
               images = "<%= imgs %>"
            />
```

```
                <%
            }
          %></dl>
          <%
       }
   }
   %>

   <%-- SALVATAGGIO --%>
   <input type="submit" value="<liferay-ui:message key="apply" />"
       />
   <input type="button" value="<liferay-ui:message key="cancel" />"
            onclick="location.href='<%= redirect %>';" />
</form>
</c:if>
```

EDIT_NAVIGATION.JSP È il file in cui si definisce il comportamento della maschera per la modifica delle proprietà della barra di navigazione.

```
<script type="text/javascript">
function <portlet:namespace />saveNav() {
   submitForm(document.<portlet:namespace />fm);
}
</script>

<%
long twThemeId = ParamUtil.getLong(request, "twThemeId");
String redirect = ParamUtil.getString(request, "redirect");
TWTheme twTheme = null;
TWVersion draft = null;

int base_T = 0;
double base_V = 0;
long base_B = 0;

try {
   twTheme = TWThemeLocalServiceUtil.getTheme(twThemeId);
   draft = TWVersionLocalServiceUtil.getDraftVersion(
         twThemeId,
         themeDisplay.getUserId());
} catch (Exception e) {}

List<TWThemeImage> images = TWThemeLocalServiceUtil.getImages(
    twThemeId);
long[] imageIds = new long[images.size()];
for (int j = 0; j < images.size(); j ++) {
   imageIds[j] = images.get(j).getThemeImageId();
}
String imgs = StringUtil.merge(imageIds, ",");
%>
```

```
<c:if test="<%= Validator.isNotNull(twTheme) && Validator.isNotNull
    (draft) %>">

<%
TWVersion parentVersion = null;
if (draft.getParentVersionId() > 0) {
   parentVersion = TWVersionLocalServiceUtil.getVersion(
                draft.getParentVersionId());
}
%>

<portlet:actionURL name="saveDraft" var="saveURL">
   <param name="twThemeId" value="<%= twThemeId %>" />
</portlet:actionURL>
<form name="<portlet:namespace />fm" method="post" onsubmit="<
    portlet:namespace />saveNav();" action="<%= saveURL %>">
   <input type="hidden" name="<portlet:namespace />twThemeId" value
       ="<%= twThemeId %>" />

   <%
   for(int iterS = 0; iterS < TWConstants.EDITABLE_PROPERTIES[0].
       length ; iterS++) {
      if (Validator.isNotNull(TWConstants.EDITABLE_PROPERTIES[0][
          iterS])) {
         %>
         <liferay-ui:tabs names="<%= TWConstants.SELECTORS[
             TWConstants.EDITABLE_PROPERTIES[0][iterS]] %>" />
         <dl>
            <%
            for(int iterA = 0 ; iterA < TWConstants.
                SELECTORS_ATTRIBUTES[TWConstants.
                EDITABLE_PROPERTIES[0][iterS]].length ; iterA++) {
               int iterSA=TWConstants.SELECTORS_ATTRIBUTES[
                   TWConstants.EDITABLE_PROPERTIES[0][iterS]][iterA
                   ];

               TWStyle style = TWStyleLocalServiceUtil.getStyle(
                             draft.getVersionId(),
                             TWConstants.EDITABLE_PROPERTIES[0][
                                 iterS],
                             iterSA
                          );

               if(style != null) {
                  base_B = style.getBackground();
                  base_V = style.getValue();
                  base_T = style.getType();
               } else if (parentVersion != null) {
                  style = TWStyleLocalServiceUtil.getStyle(
                            parentVersion.getVersionId(),
                            TWConstants.EDITABLE_PROPERTIES[0][iterS
                                ],
```

```
                        iterSA
                    );
                if(style != null) {
                    base_B = style.getBackground();
                    base_V = style.getValue();
                    base_T = style.getType();
                } else {
                    base_T = TWConstants.ATTRIBUTE_TYPES[iterSA
                        ][0];
                }
            }
            %>
            <prjmoon-ui:css-settings selector="<%= TWConstants.
                EDITABLE_PROPERTIES[o][iterS] %>"
                attribute="<%= iterSA %>"
                type="<%= base_T %>"
                value="<%= base_V %>"
                background="<%= base_B %>"
                images = "<%= imgs %>"
            />

            <%
        }
    %></dl>
    <%
  }
}
%>

<%-- SALVATAGGIO --%>
<input type="submit" value="<liferay-ui:message key="apply" />"
    />
<input type="button" value="<liferay-ui:message key="cancel" />"
        onclick="location.href='<%= redirect %>';" />
</form>
</c:if>
```

EDIT_PORTLET.JSP In questo file invece si definisce lo stile dei portlet.

```
<script type="text/javascript">
function <portlet:namespace />saveNav() {
   submitForm(document.<portlet:namespace />fm);
}
</script>

<%
long twThemeId = ParamUtil.getLong(request, "twThemeId");
String redirect = ParamUtil.getString(request, "redirect");
TWTheme twTheme = null;
TWVersion draft = null;

int base_T = 0;
```

```
double base_V = 0;
long base_B = 0;

try {
   twTheme = TWThemeLocalServiceUtil.getTheme(twThemeId);
   draft = TWVersionLocalServiceUtil.getDraftVersion(
         twThemeId,
         themeDisplay.getUserId());
} catch (Exception e) {}

List<TWThemeImage> images = TWThemeLocalServiceUtil.getImages(
    twThemeId);
long[] imageIds = new long[images.size()];
for (int j = 0; j < images.size(); j ++) {
   imageIds[j] = images.get(j).getThemeImageId();
}
String imgs = StringUtil.merge(imageIds, ",");
%>

<c:if test="<%= Validator.isNotNull(twTheme) && Validator.isNotNull
    (draft) %>">

<%
TWVersion parentVersion = null;
if (draft.getParentVersionId() > 0) {
   parentVersion = TWVersionLocalServiceUtil.getVersion(
                draft.getParentVersionId());
}
%>

<portlet:actionURL name="saveDraft" var="saveURL">
   <param name="twThemeId" value="<%= twThemeId %>" />
</portlet:actionURL>
<form name="<portlet:namespace />fm" method="post" onsubmit="<
    portlet:namespace />saveNav();" action="<%= saveURL %>">
   <input type="hidden" name="<portlet:namespace />twThemeId" value
       ="<%= twThemeId %>" />

   <%
   for(int iterS = 0; iterS < TWConstants.EDITABLE_PROPERTIES[0].
       length ; iterS++) {
      if (Validator.isNotNull(TWConstants.EDITABLE_PROPERTIES[0][
          iterS])) {
         %>
         <liferay-ui:tabs names="<%= TWConstants.SELECTORS[
             TWConstants.EDITABLE_PROPERTIES[0][iterS]] %>" />
         <dl>
            <%
            for(int iterA = 0 ; iterA < TWConstants.
                SELECTORS_ATTRIBUTES[TWConstants.
                EDITABLE_PROPERTIES[0][iterS]].length ; iterA++) {
```

```
            int iterSA=TWConstants.SELECTORS_ATTRIBUTES[
                TWConstants.EDITABLE_PROPERTIES[0][iterS]][iterA
                ];

            TWStyle style = TWStyleLocalServiceUtil.getStyle(
                           draft.getVersionId(),
                           TWConstants.EDITABLE_PROPERTIES[0][
                               iterS],
                           iterSA
                       );

            if(style != null) {
               base_B = style.getBackground();
               base_V = style.getValue();
               base_T = style.getType();
            } else if (parentVersion != null) {
               style = TWStyleLocalServiceUtil.getStyle(
                         parentVersion.getVersionId(),
                         TWConstants.EDITABLE_PROPERTIES[0][iterS
                             ],
                         iterSA
                       );
               if(style != null) {
                  base_B = style.getBackground();
                  base_V = style.getValue();
                  base_T = style.getType();
               } else {
                  base_T = TWConstants.ATTRIBUTE_TYPES[iterSA
                      ][0];
               }
            }
            %>
            <prjmoon-ui:css-settings selector="<%= TWConstants.
                EDITABLE_PROPERTIES[0][iterS] %>"
               attribute="<%= iterSA %>"
               type="<%= base_T %>"
               value="<%= base_V %>"
               background="<%= base_B %>"
               images = "<%= imgs %>"
            />

            <%
         }
      %></dl>
      <%
   }
}
%>

<%-- SALVATAGGIO --%>
<input type="submit" value="<liferay-ui:message key="apply" />"
    />
```

```
   <input type="button" value="<liferay-ui:message key="cancel" />"
          onclick="location.href='<%= redirect %>';" />
</form>
</c:if>
```

PREVIEW.JSP È il file che inietta dinamicamente il tema creato in una pagina `home.html` fissa, rappresentativa degli elementi modificabili.

```
<%long twThemeId = ParamUtil.getLong(request, "twThemeId");%>

<style>
<%= ThemeWizardUtil.createPreviewTheme(twThemeId) %>
</style>

<%@ include file="/html/theme_wizard/home.html" %>
```

Nelle figure 17 nella pagina successiva, 18 a fronte e 19 nella pagina 114 vediamo alcuni screenshot del prodotto finale.

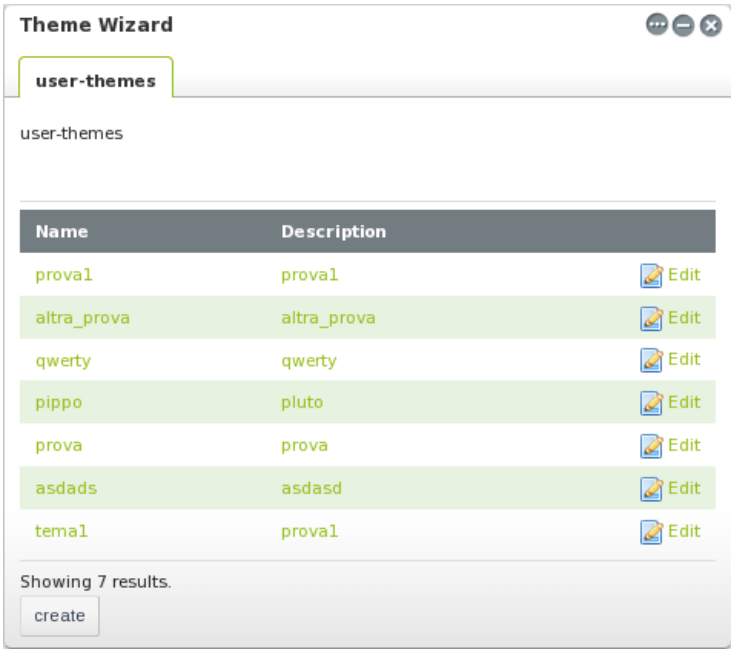


Figura 17: La finestra iniziale con l'elenco dei temi

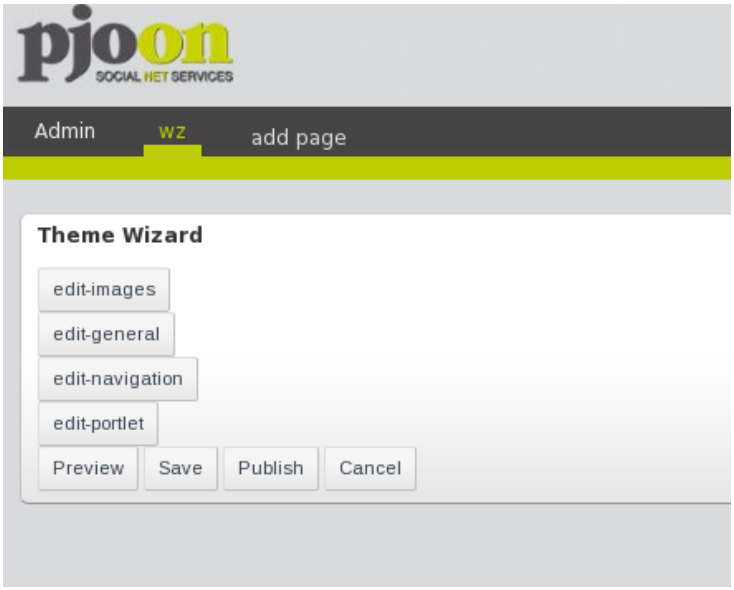


Figura 18: Il concentratore

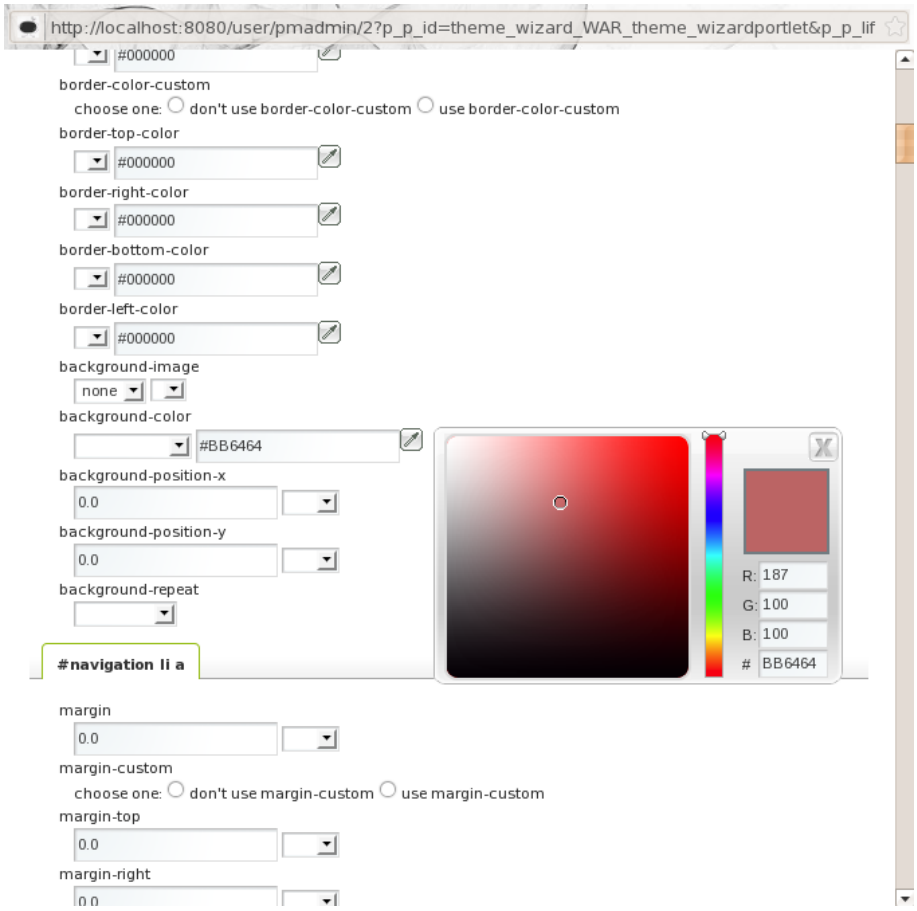


Figura 19: La finestra di editing della navigazione

5 CONCLUSIONI

INDICE

5.1 PUNTO DI ARRIVO E SVILUPPI FUTURI

5.1.1 Il sito terranostra.pjoon.com

Il sito terranostra.pjoon.com, per il quale si sono predisposte le strutture e i modelli presentati in questa relazione, ha soddisfatto il cliente, ed è ora online e perfettamente funzionante.

Altre strutture e modelli predisposti per clienti ma non trattati in questa relazione sono online su diversi siti sviluppati dall'azienda nel periodo di questo tirocinio.

Tutto questo materiale è a disposizione del cliente, facilmente ampliabile, adattabile a diverse esigenze che possono sorgere nel corso del tempo, ed è possibile apportarvi eventuali correzioni con molta semplicità, nel caso si verificasse la presenza di errori o malfunzionamenti. Al momento della scrittura di questa relazione il committente sta seguendo degli incontri di formazione, e sta apprendendo a gestire autonomamente il sito.

5.1.2 Il wizard

Per quanto riguarda il wizard, lo sviluppo della prima versione è quasi completato, la prosecuzione dell'attività proseguirà con gli ultimi test, la correzione degli ultimi bug e l'integrazione all'interno del portale.

I partner saranno così a breve in grado di creare e usare temi da loro prodotti per i loro clienti. Per motivi di tempo sono state tralasciate funzionalità minori, in alcuni casi peraltro già analizzate, che non compromettono in alcun modo l'attività dei partner.

Lo sviluppo proseguirà nei prossimi mesi con l'aggiunta delle altre funzionalità previste ma non ancora implementate, come ad esempio

- controllo sui nomi dei temi;
- struttura dei temi;
- schemi colore base;
- automatizzazione del processo di validazione e deployment del tema;
- ottimizzazione del codice e aumento delle prestazioni;
- eventuale implementazione di altre funzionalità inizialmente non previste.

Le prime tre funzionalità erano inizialmente previste per la prima versione. Essendo state comunque trattate in fase di analisi, è noto come procedere.

Sono stati inoltre già individuati alcuni punti su cui intervenire per migliorare le prestazioni, innanzitutto la gestione delle versioni, che al momento occupano molto spazio nel database. Inoltre è stata proposta la variante di inserire le 4 maschere di inserimento modifiche in tab diverse della stessa pagina, invece che in finestre pop-up. Tale proposta verrà presa in considerazione nella prossima fase di analisi. Sono infine stati individuati alcuni interventi da effettuare per pulire il codice, in particolar modo per marcare ulteriormente la separazione tra le componenti *model* e *controller*.

Mi è stato chiesto di occuparmi della prosecuzione dello sviluppo, rimanendo in Projectmoon nei prossimi mesi.

5.2 VALUTAZIONE DEL TIROCINIO

Concetti appresi

Durante questo tirocinio ho avuto innanzitutto modo di applicare concetti appresi nel corso degli studi quali progettazione in UML e progettazione di basi di dati, portando le mie conoscenze nella metodologia di lavoro adottata in Projectmoon/SMC.

Ho inoltre ripreso e ampliato le mie conoscenze pregresse di HTML, CSS e Javascript, linguaggi di cui avevo conoscenze base ma che non avevo mai avuto modo di applicare e di conseguenza coltivare. Per la prima volta ho usato JSP e framework come Hibernate e Spring.

Ho avuto quindi modo di apprendere concetti di cui prima avevo solo sentito parlare, ma non avevo mai applicato, quali pattern di sviluppo model-view-controller. Ho infine lavorato per la prima volta con una piattaforma molto vasta e completa quale Projectmoon System, a sua volta basata su un prodotto molto complesso quale è Liferay, sia dal punto di vista dello sviluppo sia dal punto di vista dell'utilizzo avanzato (ovvero le strutture e i modelli per gli articoli).

Difficoltà riscontrate

Le maggiori difficoltà che ho riscontrato sono state causate dal dover sviluppare in breve tempo un prodotto, formandomi sulla piattaforma man mano che lo sviluppo procedeva. A causa di motivi logistici non è stato infatti possibile portare avanti una formazione sufficiente prima dell'inizio dello sviluppo, e questo ha rallentato lo sviluppo e obbligato a tralasciare le funzionalità sopra menzionate. Gli ostacoli maggiori in fase di sviluppo sono stati riscontrati principalmente nella prima fase dello sviluppo, e sono stati causati dalla limitata conoscenza della piattaforma e dalla necessità di applicare contemporaneamente concetti di Java, Javascript e JSP, cosa che ha reso difficoltosa soprattutto la correzione degli errori.

Il tirocinio si è svolto con reciproca soddisfazione mia e dell'azienda, e si è concluso con la proposta di proseguire il rapporto continuando lo sviluppo del wizard e formandomi maggiormente sulla piattaforma, per divenire maggiormente coinvolto nello sviluppo di applicazioni.

A BIBLIOGRAFIA

CAY HORSTMANN Concetti di informatica e fondamenti di Java, III edizione
ed. Apogeo

MICHAEL T. GOODRICH, ROBERTO TAMASSIA Data structures & algorithms in Java, IV edizione
ed. Wiley

PAOLO ATZENI, STEFANO CERI, STEFANO PARABOSCHI, RICCARDO TORLONE Basi di dati - Modelli e linguaggi di interrogazione, II edizione
ed. McGraw-Hill

RAMEZ A. ELMASRI, SHAMKANT B. NAVATHE Sistemi di basi di dati - Fondamenti, V edizione
ed. Paravia

ROGER S. PRESSMAN Principi di ingegneria del software, V Edizione
ed. McGraw-Hill

MARTIN FOWLER UML distilled, III edizione
Pearson-Addison Wesley

ADRIANO COMAI http://analisi-disegno.com

- Linee guida – UML – Modelli Architetturali
- Linee guida – UML – Casi d'uso
- Linee guida – Requisiti by example

http://docs.liferay.com

http://velocity.apache.org

http://www.w3schools.com