Università degli Studi di Salerno
Dipartimento di Informatica

Tesi di Laurea di I livello in
Informatica

Condivisione di immagini come Open Data

Relatore
Prof. Vittorio Scarano

Candidato
Roberta Guadagni

Anno Accademico 2015-2016

Dediche e ringraziamenti

Ringrazio i miei genitori che mi hanno sempre supportato e mi hanno permesso di intraprendere questo percorso e tutte le persone che come loro mi hanno aiutato e spinto ad andare avanti e terminare quello che sarà il primo passo verso il mio futuro.

Indice

Capitolo 1

Introduzione

Lo scopo di questa di tesi è permettere di inserire immagini all'interno di open datasets tramite la Piattaforma Sociale per gli Open Data (SPOD).
Gli open datasets sono tabelle che contengono open data, dati che possono essere liberamente utilizzati, riutilizzati e ridistribuiti da chiunque, soggetti eventualmente alla necessità di citarne la fonte e di condividerli con lo stesso tipo di licenza con cui sono stati originariamente rilasciati [1]. L'open data si richiama alla più ampia disciplina dell'open government, cioè una dottrina in base alla quale la pubblica amministrazione dovrebbe essere aperta ai cittadini, tanto in termini di trasparenza quanto di partecipazione diretta al processo decisionale, anche attraverso il ricorso alle nuove tecnologie dell'informazione e della comunicazione. Gli enti comunali o amministrativi hanno sempre più bisogno di condividere una grande quantità di open data con i loro cittadini; questi dati per essere compresi devono appunto essere strutturati in tabelle in cui vengono memorizzati i relativi metadati che agevolano la comprensione del dato stesso. In molti casi, come ad esempio quando i dati in questione rappresentano il patrimonio culturale del nostro paese, è importante che i dati possano essere rappresentati tramite delle immagini per essere appieno compresi e condivisi.
Il problema affrontato in questa tesi è quindi quello di permettere di visualizzare questi dati (quando possibile) tramite un'immagine e tutte le problematiche legate alla gestione di un'immagine all'interno di una piattaforma: l'upload, la visualizzazione, la condivisione e il riutilizzo, la modifica e la cancellazione dell'immagine.
Il lavoro si svolge all'interno della piattaforma SPOD poiché essa oltre che permettere le interazioni sociali tra gli utenti permette la condivisione di open datasets provenienti da fonti diverse, dette datasets provider. Gli utenti

iscritti alla piattaforma possono usufruire delle principali funzionalità offerte da un social network e di numerose novità tra cui quella di creare delle stanze pubbliche per le discussioni, creare una visualizzazione dei datasets dette Datalet oppure creare delle stanze private dette CoCreation Knowledge room o CoCreation Data room, in quest'ultimo caso gli utenti possono creare un dataset all'interno delle stanze, manipolandone uno già esistente all'interno della piattaforma, oppure possono crearne uno nuovo. ll lavoro di questa tesi si concentra in particolar modo su questa funzionalità della piattaforma poiché nel momento di creazione del dataset vengono aggiunte le immagini.
Per ottenere questi obiettivi è stato modificato un plugin già esistente in SPOD, riutilizzando e modificando, per adattarlo alle nostre necessità, il plugin Photo di Oxwall, piattaforma per lo sviluppo di social network, che fornisce funzionalità standard per la gestione delle immagini all'interno di una piattaforma.
Nei prossimi capitoli verrà presentato il sistema Oxwall con particolare attenzione alla struttura di un plugin, verranno presentate le novità aggiunte a SPOD dal lavoro di questa tesi e come sono state implementate, infine una panoramica finale sugli obiettivi raggiunti e sui possibili sviluppi futuri.

Capitolo 2

Oxwall

Oxwall è una piattaforma open source per lo sviluppo di social network [2], sviluppata dalla società commerciale Skalfa LLC nel 2010 per poi successivamente donarne i diritti alla Oxwall Fondation [3], facendola diventare a tutti gli effetti una piattaforma open source. La sua particolarità è la flessibilità dovuta alla sua struttura: le principali funzionalità sono implementate tutte tramite plugin: [1] questo permette, istallando e disinstallando i plugins, di aggiungere ed eliminare funzionalità senza andare a modificare la struttura del software.

2.1 Le Motivazioni

Si è scelto di utilizzare un plugin di Oxwall poiché come vedremo in modo più approfondito nel prossimo paragrafo, Oxwall fornisce già le principali funzionalità di un social network, tra cui le funzionalità legate alle immagini, implementate nel plugin Photo. Si è tenuto anche conto che SPOD è stato implementato con l'ausilio di Oxwall ereditandone la struttura a plugin ed è risultato quindi semplice riutilizzare alcune funzionalità di Oxwall all'interno di un plugin di SPOD.

2.2 Funzionalità di base

L'ultima versione di Oxwall rilasciata il 26 luglio 2016 è la 1.8.4. Insieme al download dell'ultima versione del software vengono rilasciati diversi plugins, ognuno dei quali corrisponde ad una pagina diversa (figura 2.1):

[1] Un plugin è un blocco di software non autonomo che va ad aggiungere o a modificare funzionalità presenti in un programma di base.

- *Dashboard* , dove sono elencante tutte le attività proprie e degli amici;
- *Main*, dove sono elencate le attività di tutti gli utenti iscritti al sito;
- *Members*, dove si possono visualizzare tutti gli utenti del sito ordinati per data di iscrizione, visualizzare gli utenti online, i loro compleanni, o effettuare una ricerca;
- *Forum*, dove si possono visualizzare e ricercare i forum del sito;
- *Groups*, dove si possono visualizzare, ricercare e creare gruppi;
- *Photo*, dove le foto possono essere caricate, visualizzate e ricercate[2];
- *Video*, dove si possono caricare e visualizzare i video di tutti gli utenti;
- *Events*, dove si possono creare e visualizzare eventi;
- *Blogs*, dove si possono visualizzare i post ordinati con diversi criteri o creare nuovi post.

Oltre queste funzionalità accessibili tramite le varie pagine, Oxwall permette agli utenti di poter chattare tra loro, implementa un sistema di notifica per avvertirli di tutte le attività che li riguardano e ogni utente ha una propria pagina del profilo, dove sono contenute tutte le sue informazioni personali che può modificare in qualsiasi momento. L'amministratore del sito inoltre ha a disposizione un'altra dashboard (figura 2.2) che serve a gestire le impostazioni del sito: l'admin può da qui facilmente cambiare la grafica e il tema delle pagine, i permessi degli utenti, disinstallare/disattivare e istallare/attivare plugins e cambiare le impostazioni e la grafica della versione mobile del sito.

2.2.1 Funzioni del plugin Photo

In questa pagina è possibile accedere a tutte le funzionalità messe a disposizione dal plugin Photo.
All'interno della pagina è possibile caricare un nuovo album di cui bisogna indicare necessariamente un titolo, una descrizione opzionale e la privacy, che indica chi può visualizzare le immagini caricate al suo interno. Ci sono quattro possibili opzioni: "public", le immagini possono essere visualizzate da tutti gli iscritti al sito; "all friends" , l'immagine può essere visualizzata

[2]Tutte le funzionalità del plugin Photo sono descritte dettagliatamente nel prossimo paragrafo.

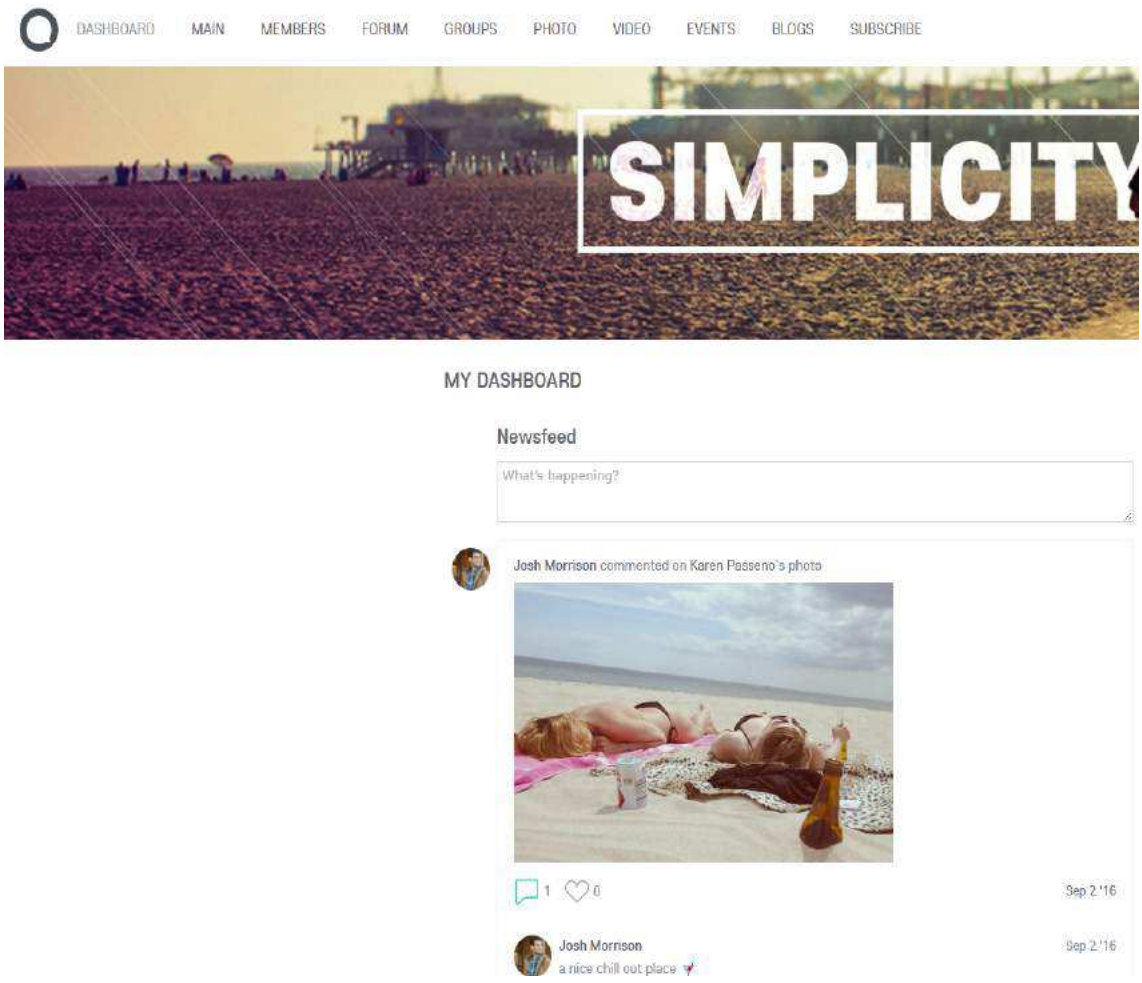


Figura 2.1: Pagina principale di Oxwall, di default è selezionata la pagina Dashboard, si possono notare in alto anche le altre tutte le pagine del sito.

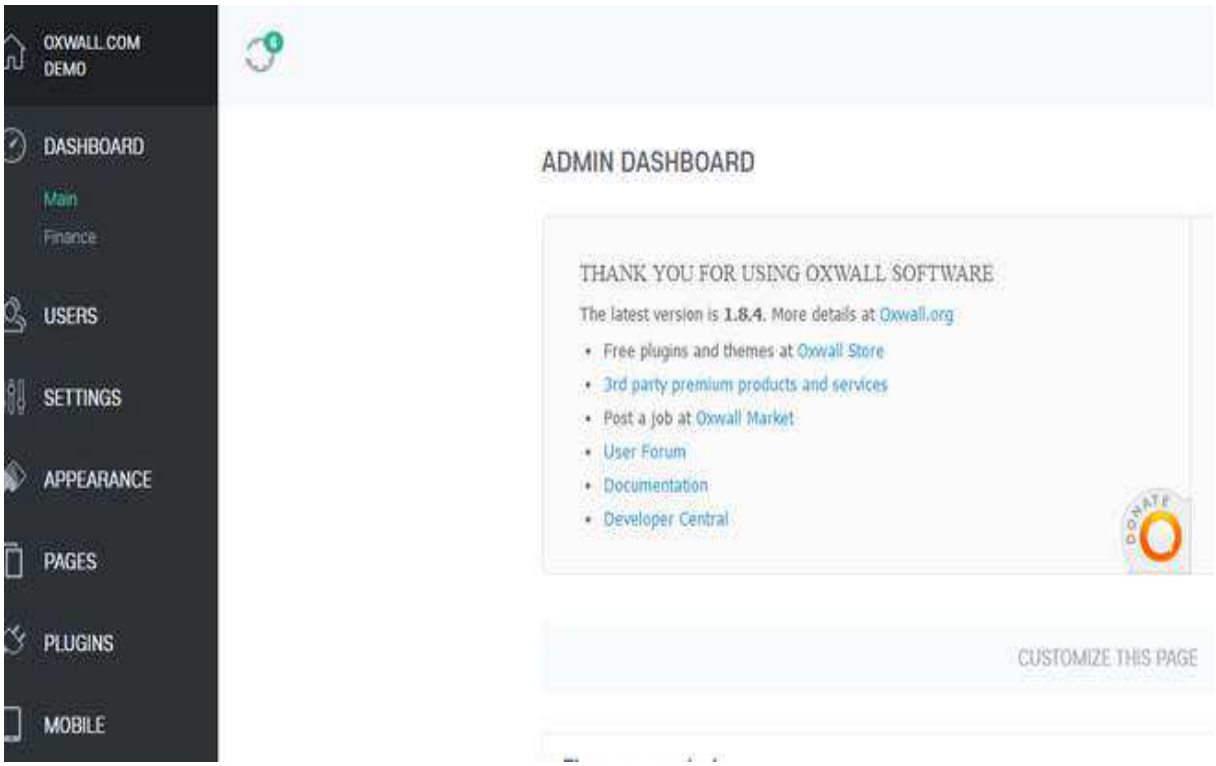


Figura 2.2: Dashboard dell'admin da cui può svolgere le azioni di amministrazione del sito.

solo dagli amici dell'utente che ha caricato l'album, memorizzato come proprietario dell'album e quindi anche di tutte le immagini al suo interno; "individual friends", solo alcuni amici selezionati possono vedere le immagini; "password protected", l'immagine viene protetta da una password scelta dal proprietario. L'utente ha inoltre la possibilità di caricare immagini all'interno di un album già esistente di cui è proprietario, indicando opzionalmente una descrizione per le immagini caricate.
La pagina è divisa in due sezioni: "EXPLORE" e "MY PHOTOS". Nella sezione "EXPLORE" (figura 2.3) sono visualizzate le immagini caricate da tutti gli utenti del sito di cui viene visualizzata un'anteprima, la sua descrizione se è stata inserita, il nome dell'utente che l'ha caricata, il nome dell'album a cui appartiene e il numero dei commenti e dei voti che ha ricevuto. Ognuna di esse può essere scaricata, votata, commentata, visualizzata a schermo intero o eventualmente segnalata se il suo contenuto è ritenuto offensivo. Le foto sono ordinate in ordine decrescente in base a tre criteri: la data di caricamento, il numero di voti e il numero di commenti. All'interno della sezione è fornita anche una funzionalità di ricerca che permette di ricercare tutte le immagini caricate da un determinato utente, tutte le immagini contrassegnate da un determinato hashtag o tutte le immagini la cui descrizione contiene una determinata frase.
Nella sezione "MY PHOTOS" sono invece visualizzate tutte le immagini caricate dall'utente loggato, divisa in due sottosezioni: "PHOTOS" mostra le immagini singolarmente con i relativi metadati descritti prima e "ALBUMS" (figura 2.4) elenca gli album caricati dall'utente loggato di cui viene visualizzata la copertina, il titolo e il numero di immagini che contiene. Tutte le immagini contenute in questa sezione possono essere modificate, modificando o aggiungendo la descrizione, possono essere spostate in un altro album appartenente allo stesso utente proprietario, possono essere utilizzate come immagine del suo profilo e possono essere eliminate. Ugualmente gli album possono essere modificati modificandone titolo, descrizione, privacy e immagine di copertina, e possono essere eliminati.
Oxwall permette agli utenti anche di caricare immagini all'interno della pagina *Dashboard*, il sistema automaticamente inserisce queste immagini all'interno di un album "public" a cui è associato di default il titolo "Neewsfeed Photos" che non può essere modificato, la privacy invece può essere successivamente modificata.
L'utente con il ruolo di admin, in quanto amministratore del sito, accede a funzionalità in più rispetto ad un normale utente iscritto al sito. In primo luogo egli può modificare ed eliminare qualsiasi immagine caricata sul sito (non solo quelle caricate da lui stesso), inoltre può decidere di mettere

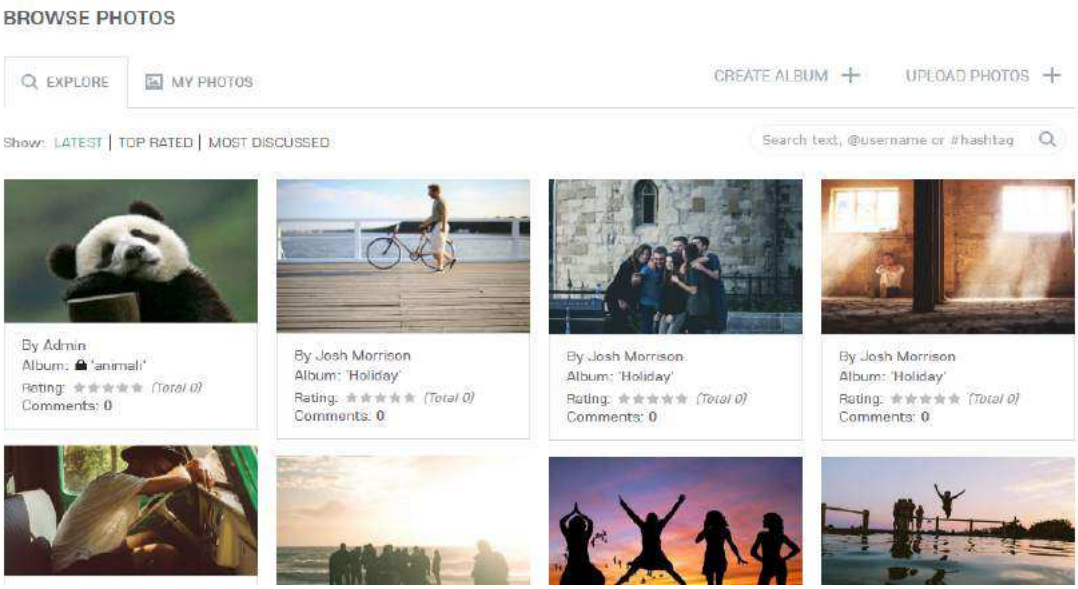


Figura 2.3: Sezione *EXPLORE* della pagina Photo di Oxwall.

in primo piano alcune immagini contrassegnandole come "featured", queste immagini verranno inserite nella sottosezione omonima di "EXPLORE" creata nel momento in cui viene contrassegnata la prima immagine.
Oxwall quindi fornisce una serie di funzionalità legate alle immagini che vengono caricate sul sito, come vedremo nel prossimo capitolo alcune di queste sono state ereditate in SPOD mentre altre invece sono state eliminate poiché non utili agli obiettivi prefissi per questa tesi.

2.3 Plugin

La maggior parte del lavoro di questa tesi si concentra sulla modifica di un plugin, è importante quindi comprenderne la struttura.
I plugins in Oxwall seguono il pattern Model-View-Controller che divide il plugin in tre componenti interconnesse tra loro: il model si occupa della memorizzazione e gestione dei dati in base ai comandi che riceve dal controller, il controller invia comandi al model in base al quale i dati devono essere modificati e modifica la view in base ai cambiamenti apportati ai dati, la view è l'interfaccia con l'utente da cui viene preso l'input e mostrato

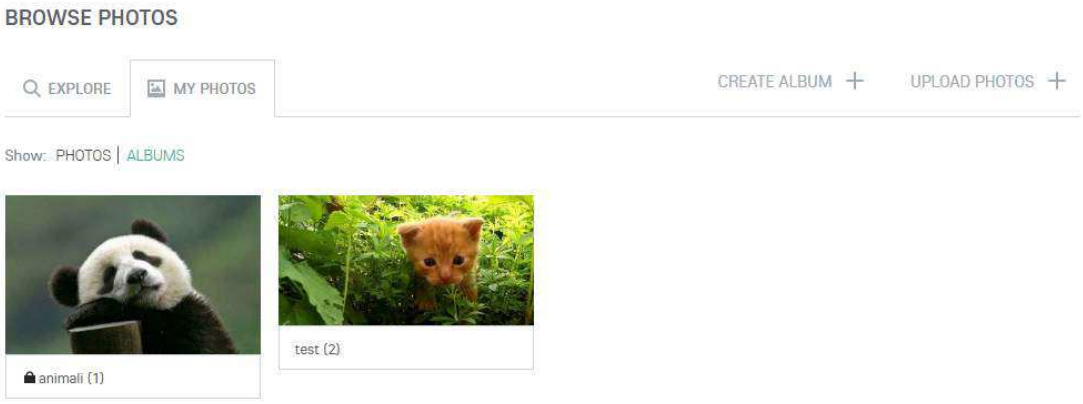


Figura 2.4: Sottosezione *ALBUMS* di *MY PHOTOS* della pagina *Photo* di Oxwall.

l'output dopo le azioni effettuate dal controller. Il model è gestito dal BOL (Business Object Layer) che gestisce l'interazione con il database, in questo caso database MYSQL, i controller sono sviluppati utilizzando il linguaggio PHP ad oggetti e le API fornite da Oxwall, mentre la view sfrutta il template engine Smarty [5] che separa la logica dalla presentazione. [3]

2.3.1 Ciclo di vita

Un plugin quando viene aggiunto ad Oxwall, viene inserito nella lista degli *Available plugins* , a quel punto deve essere istallato e sarà quindi aggiunto alla lista dei plugins istallati. L'istallazione è gestita dal file *install.php*, quando il plugin viene istallato vengono create le tabelle nel database, importato il file zip contenente le stringhe localizzate, create le opzioni di configurazione del plugin, aggiunta una pagina delle impostazioni visibile agli admin e gestite le autorizzazioni. Una volta istallato il plugin è automaticamente attivato (l'attivazione è gestita dal file *active.php*) ma l'amministratore può disattivarlo (*deactive.php*): quando viene disattivato il plugin rimane nell'elenco dei plugins istallati ma le sue funzioni vengono disattivate quindi vengono eliminati widget al sito e link a pagine, in qualsiasi momento può

[3] Lo sviluppo di queste tre componenti verrà approfondito nei paragrafi 2.3.3, 2.3.4, 2.3.6.

essere riattivato e le sue funzioni sono nuovamente visibili. Le tabelle del database vengono eliminate solo quando viene disinstallato (*unistall.php*), la disinstallazione implica la disattivazione e in tal caso il plugin ritorna nella lista degli *Available plugins*, per rimuoverlo anche dall'elenco dei plugins disponibili bisogna eliminarlo.

2.3.2 Struttura

Per sviluppare un plugin è necessario seguire determinate regole in modo che Oxwall possa interpretare i file in maniera corretta. Un plugin in Oxwall deve essere necessariamente contenuto in una cartella a sua volta contenuta nella cartella *ow_plugins* e contenere il file *plugin.xml* che contiene alcune informazioni fondamentali su di esso: il nome, una descrizione, il nome dello sviluppatore, la plugin key, una stringa che identifica univocamente il plugin (il nome della cartella del plugin deve corrispondere alla plugin key), e la developer key, cioè la stringa che identifica univocamente lo sviluppatore. Le chiavi univoche sono necessarie per la pubblicazione nello store di Oxwall.
All'interno della cartella del plugin [4] troviamo oltre il file *plugin.xml* e i files del ciclo di vita anche il file *init.php*. Questo file verrà eseguito ogni volta che il plugin è attivo, al suo interno vengono inizializzate nuove classi, nuove rotte e associati handler agli eventi. Per capire il concetto di rotta è necessario comprendere che ad ogni pagina è associato il metodo di un controller, ad ogni controller può essere associata una rotta ovvero un oggetto della classe **OW_Route** che alla URL del metodo associa un percorso più facile da ricordare. Mostriamo un esempio:

```
OW::getRouter()->addRoute(new OW_Route('contactus.index',
        'contact', "CONTACTUS_CTRL_Contact", 'index'));
```

L'oggetto OW_Route prende quattro parametri: il nome identificativo della rotta appena creata, di solito segue lo schema <plugin keyname><name>, l'URL abbreviata tramite la quale si può accedere alla pagina associata al metodo del controller, gli ultimi due parametri sono appunto il nome del controller e il nome del metodo.
Oltre i file appena elencati la cartella di un plugin deve contenere determinate sottocartelle:

[4] Ricordiamo che la cartella porta il nome della plugin key

- la cartella */bol* che contiene i file che compongono il BOL,[5] tutte le classi in questa cartella devono seguire il formato PLUGINKEY_BOL_NOME;
- la cartella */classes* che contiene delle classi di utility, le classi contenute in questa cartella devono seguire il formato PLUGINKEY_CLASS_NOME;
- la cartella */components* che contiene tutte le componenti del plugin, tutte le classi in questa cartella devono seguire il formato PLUGINKEY_CMP_NOME ;
- la cartella */controllers* che contiene tutti i controller del plugin, tutte le classi in questa cartella devono seguire il formato PLUGINKEY_CTRL_NOME;
- la cartella */mobile* che contiene tutti i file che servono alla versione mobile del sito;
- la cartella */static* che contiene i file statici del plugin, ovvero i file CSS e Javascript che andranno quindi inseriti rispettivamente nella cartella */static/css* e nella cartella */static/js*. Per aggiungere un file CSS o Javascript al documento che si sta generando, è possibile utilizzare vari metodi della classe **OW_HtmlDocument**, la cui istanza si ottiene chiamando il metodo **OW::getDocument()**. Tale classe offre metodi come **addStyleSheet**, **addScript** o **addOnLoadScript** per includere fogli di stile o script nella pagina HTML in creazione.
- la cartella */update* serve a specificare azioni da eseguire nel momento in cui si sta aggiornando il plugin, all'interno di tale cartella deve essere creata una nuova sottocartella ogni volta che un aggiornamento richiede l'esecuzione di uno script PHP;
- la cartella */views* che contiene tutte le view del plugin divise in due sottocartelle */components* per le views delle componenti e */controllers* per le views dei controllers.

2.3.3 Bol

Il Business Object Layer svolge il ruolo della componente Model del pattern MVC e quindi come abbiamo detto precedentemente dell'interazione con il database.

[5] Ricordiamo che il BOL è il Business Object Layer che si occupa dell'interazione con il database.

Ad ogni tabella corrisponde una classe il cui nome segue lo schema predefinito, il nome specifico della classe deve seguire la notazione camelcase e deve essere tutto minuscolo a parte la lettera iniziale, ad esempio PLUGINKEY_BOL_MyCamelCaseName, mentre il file si chiamerà my_camel_case_name.php. La classe, se rappresenta una tabella del database, dovrà estendere necessariamente la classe **OW_Entity** essa è un DTO (Data Transfer Object). Vediamo un esempio di un DTO che rappresenta una tabella dipartimento in cui sono memorizzate le email dei dipendenti di quel dipartimento:

```
class CONTACTUS_BOL_Department extends OW_Entity
{

public $email;
}
```

Alla variabile $email corrisponde l'attributo *email* della tabella *Department*, l'id non deve essere specificato; è importante che ad ogni classe che estende la classe **OW_Entity** corrisponda una tabella del database nel file *istall.php* che come abbiamo detto si occupa della creazione delle tabelle nel database:

```
$sql = "CREATE TABLE `" . OW_DB_PREFIX . "contactus_department`(
`id` INT(11) NOT NULL AUTO_INCREMENT,
`email` VARCHAR(200) NOT NULL,
PRIMARY KEY (`id`)
)
ENGINE=MyISAM
ROW_FORMAT=DEFAULT";

OW::getDbo()->query($sql);
```

Il nome della tabella segue lo schema <pluginKey><tablename> in modo da evitare che ci siano conflitti con tabelle di altri plugins con lo stesso nome. L'oggetto **OW::getDbo** si riferisce all'oggetto **OW_Database** che è l'oggetto di più basso livello che interagisce con il database, permettendo tramite il metodo *query* di eseguire una query SQL passata come parametro di tipo stringa al metodo.
Dopo aver creato tutti i DTO, per ognuno di essi bisogna creare una classe DAO (Data Access Object) che si occupa di svolgere il mapping tra oggetto PHP e database. Una classe DAO deve necessariamente estendere la classe **OW_BaseDao**.

```
class CONTACTUS_BOL_DepartmentDao extends OW_BaseDao
{

        /**
        * Constructor.
        *
        */
        protected function __construct()
        {
                parent::__construct();
        }
        /**
        * Singleton instance.
        *
        * @var CONTACTUS_BOL_DepartmentDao
        */
        private static $classInstance;

        /**
        * Returns an instance of class (singleton pattern implementation).
        *
        * @return CONTACTUS_BOL_DepartmentDao
        */
        public static function getInstance()
        {
                if ( self::$classInstance === null )
                {
                        self::$classInstance = new self();
                }

                return self::$classInstance;
        }

        /**
        * @see OW_BaseDao::getDtoClassName()
        *
        */
        public function getDtoClassName()
        {
```

```
        return 'CONTACTUS_BOL_Department';
    }

    /**
    * @see OW_BaseDao::getTableName()
    *
    */
    public function getTableName()
    {
        return OW_DB_PREFIX . 'contactus_department';
    }
```

Come possiamo vedere questa classe è un singleton e contiene vari metodi per lavorare con il database, in particolare contiene sempre due metodi: quello che restituisce il nome della tabella (linee 48-51) e quello che restituisce il nome della classe DAO (linee 39-42).
Oltre la classe DAO solitamente viene creata un'altra classe che aggiunge logica di business e viene chiamata *service.php*.

```
class CONTACTUS_BOL_Service
{
    /**
    * Singleton instance.
    *
    * @var CONTACTUS_BOL_Service
    */
    private static $classInstance;

    /**
    * Returns an instance of class (singleton pattern implementation).
    *
    * @return CONTACTUS_BOL_Service
    */
    public static function getInstance()
    {
        if ( self::$classInstance === null )
        {
            self::$classInstance = new self();
        }

        return self::$classInstance;
```

```
}

private function __construct()
{

}

public function getDepartmentLabel( $id )
{
        return OW::getLanguage()->text('contactus',
        $this->getDepartmentKey($id));
}

public function addDepartment( $email, $label )
{
        $contact = new CONTACTUS_BOL_Department();
        $contact->email = $email;
        CONTACTUS_BOL_DepartmentDao::getInstance()->save($contact)

        BOL_LanguageService::getInstance()->addValue(
        OW::getLanguage()->getCurrentId(),'contactus',
        $this->getDepartmentKey($contact->id),trim($label));
}

public function deleteDepartment($id )
{
        $id = (int)$id;
        if ( $id > 0 )
        {
        $key = BOL_LanguageService::getInstance()->
        findKey('contactus',
        $this->getDepartmentKey($id));
        BOL_LanguageService::getInstance()->
        deleteKey($key->id, true);
        CONTACTUS_BOL_DepartmentDao::getInstance()->
        deleteById($id);
        }
}

private function getDepartmentKey( $name )
```

```
        {
                return 'dept_' . trim($name);
        }

        public function getDepartmentList()
        {
                return CONTACTUS_BOL_DepartmentDao::
                getInstance()->findAll();
        }
```

La differenza tra la classe service e la classe DAO è che la classe service si serve della classe DAO per eseguire logica di business ma la classe DAO non contiene logica di business, solo query al database.

2.3.4 Controller

Un controller in Oxwall svolge il compito dell'omonima componente del pattern MVC. Esso è una classe il cui nome segue il formato descritto precedentemente ed estende la classe **OW_ActionController** che a sua volta estende la classe **OW_Renderable**. Ad ogni controller corrisponde una vista con lo stesso nome nella cartella */views/controllers*, eventualmente si possono definire viste diverse per ogni metodo del controller quindi il nome della vista sarà nomeController_nomeMetodo. Quando viene chiamata L'URL corrispondente ad un metodo di un controller o la rotta settata per esso, viene mostrata la view corrispondente. Mostriamo un esempio di controller:

```
class CONTACTUS_CTRL_Contact extends OW_ActionController
{
        public function index()
        {
                $this->setPageTitle("Contact Us");
                $this->setPageHeading("Contact Us");
        }

}
```

Il controller semplicemente definisce il titolo e il page heading della pagina utilizzando i metodi della classe **OW_ActionController** a cui accedere tramite la variabile $this.

2.3.5 Component

Una componente è un elemento reindirizzabile del plugin che estende la classe **OW_Component**, ad ogni componente deve corrispondere una view con lo stesso nome nella cartella */views/components*. Nelle componenti è spesso utilizzato il metodo:

```
$this -> assign ( 'name' , 'value' );
```

Questo metodo prende due parametri un nome e un valore associato a quel nome. Questa coppia chiave-valore verrà inviata al template smarty della view della componente, dove può essere utilizzata con il nome che abbiamo definito, il cui valore sarà ovviamente il secondo parametro che abbiamo passato al metodo *assign*. Tale metodo è un metodo della classe **OW_Renderable** di conseguenza può essere utilizzato anche nei controllers.

2.3.6 View

Una view svolge i compiti dell'omonima componente del pattern MVC e quindi definisce l'interfaccia con l'utente. Come abbiamo visto sono definite sia le viste per i controllers che per le componenti. Le views sono definite tramite il template engine smarty che si differenzia da un normale documento HTML poiché il codice HTML viene utilizzato senza la necessità di dover inserire la direttiva DOCTYPE, i tag <html>, <head>, <body> e si può accedere alle variabili passate al template, tramite il metodo *assign*, con la notazione {$nomevariabile}, si possono svolgere cicli e istruzioni condizionali tramite {if},{else},{foreach} , tutto ciò permette di separare la logica di business dalla presentazione.

2.3.7 Form

Oxwall fornisce delle API per costruire agevolmente una form e inserirla indifferentemente all'interno di un controller, di una componente o di una classe di utility. Vediamo subito un esempio di form:

```
$form = new Form('contact_form');
$fieldTo = new Selectbox('to');
foreach ( $contactEmails as $email => $label )
{
        $fieldTo->addOption($email, $label);
}
$fieldTo->setRequired();
$fieldTo->setHasInvitation(false);
```

```
$fieldTo->setLabel($this->text('contactus', 'form_label_to'));
$form->addElement($fieldTo);

$fieldFrom = new TextField('from');
$fieldFrom->setLabel($this->text('contactus', 'form_label_from'));
$fieldFrom->setRequired();
$fieldFrom->addValidator(new EmailValidator());
$form->addElement($fieldFrom);

$fieldSubject = new TextField('subject');
$fieldSubject->setLabel($this->text('contactus',
'form_label_subject'));
$fieldSubject->setRequired();
$form->addElement($fieldSubject);

$fieldMessage = new Textarea('message');
$fieldMessage->setLabel($this->text('contactus',
'form_label_message'));
$fieldMessage->setRequired();
$form->addElement($fieldMessage);

$fieldCaptcha = new CaptchaField('captcha');
$fieldCaptcha->setLabel($this->text('contactus',
'form_label_captcha'));
$form->addElement($fieldCaptcha);

$submit = new Submit('send');
$submit->setValue($this->text('contactus',
'form_label_submit'));
$form->addElement($submit);
```

Prima di tutto viene creato un oggetto di tipo Form a cui viene assegnato un nome che corrisponde al valore dell'attributo *name* del corrispondente tag HTML. Successivamente vengono creati una Selectbox, due TextField, una TextArea, un CaptchField e un bottone di tipo submit. Ad ognuno di essi viene assegnato un nome che ugualmente alla form corrisponde all'attributo *name* del corrispondente tag HTML, viene creata una label tramite il metodo **setLabel** e aggiunto alla form tramite il metodo **addElement**. Possiamo notare altri metodi ad esempio il metodo **setRequired** che indica che l'input deve essere compilato obbligatoriamente; **setValue** setta

il valore dell'attributo *value* del tag; il metodo **addValidator** aggiunge un validator per l'input, in questo caso un email validator; i metodi **setHasInvitation** e **setInvitation** stabiliscono se e quale frase far comprarire come suggerimento per il l'input.
Per recuperare i valori inseriti dall'utente nella form quando andrà a cliccare sul pulsante di submit, si inserisce il seguente blocco di codice dopo il codice che va a definire la form:

```
if ( OW::getRequest() -> isPost () )
 {
        if ( $form -> isValid ( $_POST ) )
        {
                $data = $form -> getValues ();

                $mail = OW::getMailer () -> createMail ();
                $mail -> addRecipientEmail ( $data ['to']);
                $mail -> setSender ( $data ['from']);
                $mail -> setSubject ( $data ['subject']);
                $mail -> setTextContent ( $data ['message']);
                 OW :: getMailer () -> addToQueue ( $mail );

                 OW :: getSession () ->set( 'contactus.dept' ,
                  $contactEmails [$data ['to']]) ;
                  $this -> redirectToAction('sent');
                }
         }
```

Poiché non abbiamo selezionato una action per il form, il browser invierà una richiesta HTTP POST alla stessa URL che ha generato il form. Quindi all'interno dello stesso metodo controlliamo se la richiesta è di tipo POST e se i valori passati tramite la richiesta POST sono validi. Dopodiché estraiamo tali valori tramite il metodo **getValues** della classe Form e li utilizziamo, in questo caso, per inviare un'email e per impostare un cookie di sessione, infine generiamo un redirect al metodo sent dello stesso controller.
Per visualizzare la form dobbiamo aggiungere alcune variabili all'interno del template smarty:

```
{form name='contact_form'}
<table class="ow_table_1 ow_form ow_automargin ow_superwide">
<tr class="ow_alt1">
<td class="ow_label">{label name='to'}</td>
<td class="ow_value">{input name='to'}{error name='to'}</td>
```

```
</tr>
<tr class="ow_alt2">
<td class="ow_label">{label name='from'}</td>
<td class="ow_value">{input name='from'}{error name='from'}</td>
</tr>
<tr class="ow_alt1">
<td class="ow_label">{label name='subject'}</td>
<td class="ow_value">{input name='subject'}{error name='subject'}</td>
</tr>
<tr class="ow_alt2">
<td class="ow_label">{label name='message'}</td>
<td class="ow_value">{input name='message'}{error name='message'}</td>
</tr>
<tr class="ow_alt1">
<td class="ow_label">{label name='captcha'}</td>
<td class="ow_value ow_center">{input name='captcha'}{error name='captcha'}
</tr>
<tr>
<td class="ow_center" colspan="2">{submit name='send'
class='ow_button ow_ic_mail'}</td>
</tr>
</table>
{/form}
```

Indichiamo l'inizio della form con la variabile {forma name=" "} (dove name è l'attributo name che abbiamo indicato nel codice php quando abbiamo creato la form) e la fine con {/form}. Per inserire un input usiamo la variabile {input name" "}, se vogliamo inserire una label per un input si usa la notazine {label name=" "} dove name è il name dell'input a cui si riferisce (verrà visualizzato il valore che abbiamo settato nel codice php con il metodo **setLabel**), e infine con {error name=" "} possiamo aggiungere un messaggio di errore di validazione per l'input associato al name specificato. Gli input possono essere inseriti in una qualsiasi struttura HTML in questo caso sono stati aggiunti in una tabella.

Capitolo 3

SPOD

3.1 Introduzione

Come abbiamo accennato nell'Introduzione, SPOD una è Piattaforma Sociale per gli Open Data che permette le tradizionali interazioni sociali tra utenti: ognuno di essi ha un profilo con le proprie informazioni personali, essi possono aggiungere altri utenti come loro amici, possono chattare tra loro e ricevere notifiche per le attività che li riguardano. Le pagine principali di SPOD sono (figura 3.1):

- *Neewsfeed*, all'interno della quale gli utenti possono condividere un post, allegati di vario genere oppure una Datalet [1];
- *Membri*, che ha le medesime funzionalità della pagina omonima di Oxwall;
- *Spazio privato*, dove gli utenti possono creare *card* (link, testo, Datalet, albero di decisione) visibili solo a loro stessi;
- *Agora*, dove possono creare delle stanze di discussione pubbliche;
- *CoCreation*, dove possono creare delle stanze private;
- *Eventi*, che ha le stesse funzionalità dell'omonima pagina di Oxwall.

[1] Una Datalet è una visualizzazione di un dataset

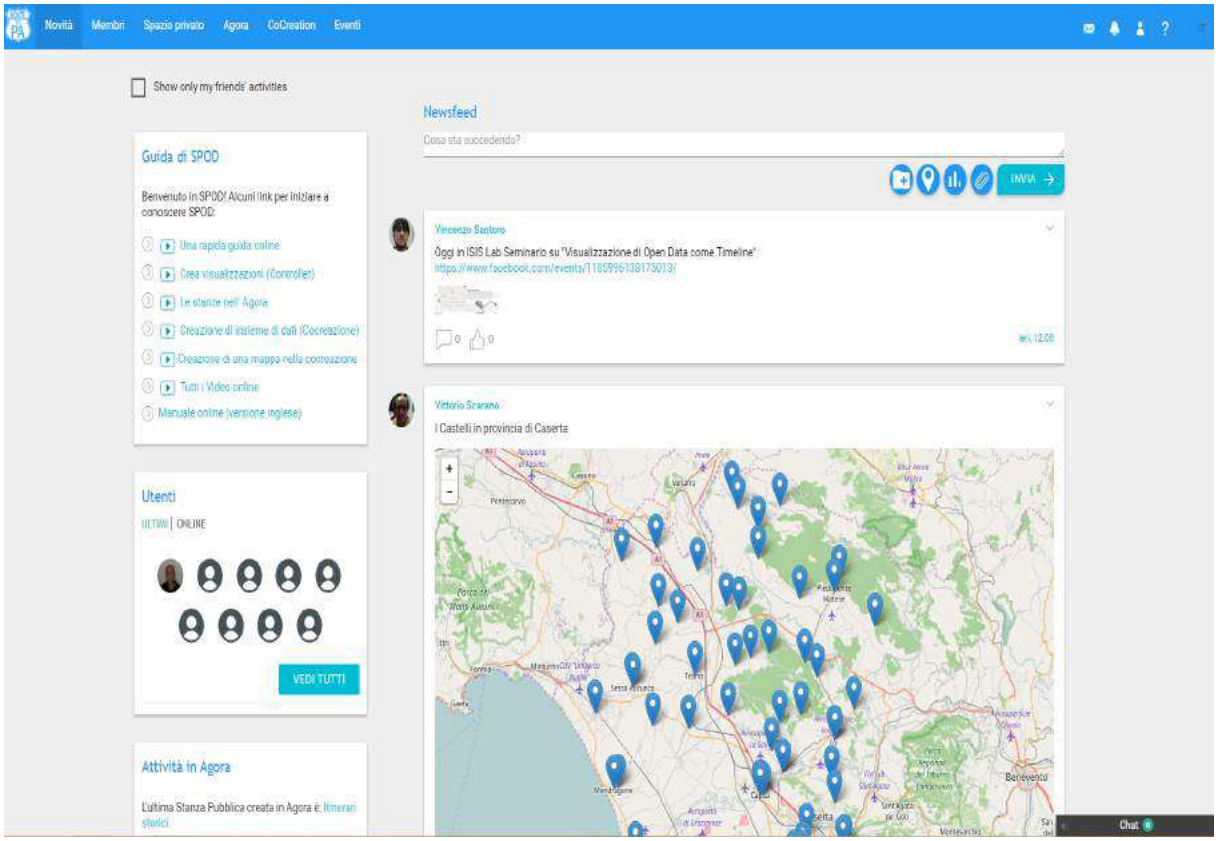


Figura 3.1: Pagina principale di SPOD, di default è selezionata la pagina *Neewsfeed*, si possono notare in alto anche le altre pagine del sito.

3.2 CoCreation

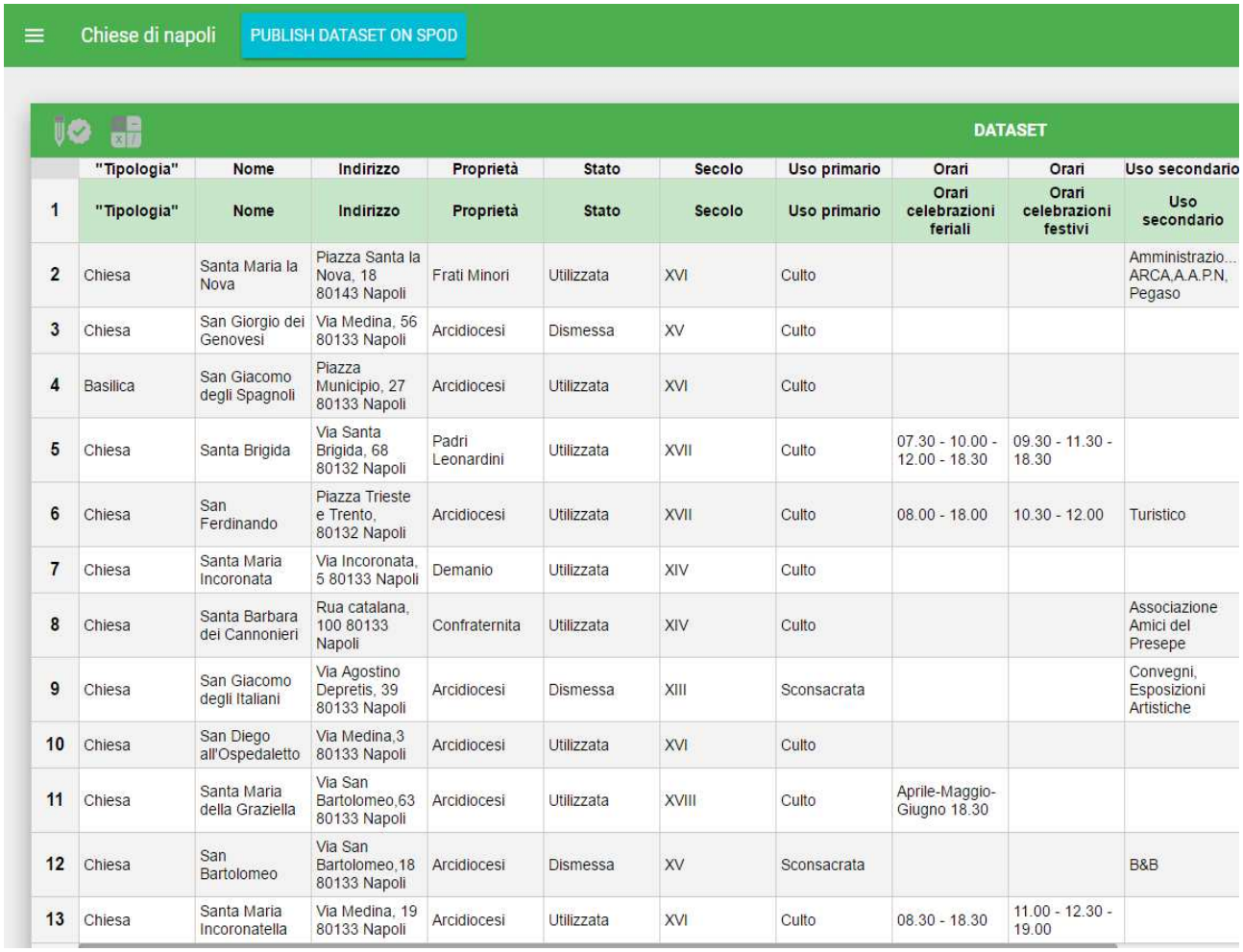
Le CoCreation rooms sono un spazio collaborativo, ovvero gli utenti all'interno di queste stanze possono collaborare e lavorare contemporaneamente per creare un nuovo prodotto. Esse sono stanze private in quanto a differenza della stanze nell'*Agora*, non sono visibili e accessibili a tutti; possono accedere alle CoCreation rooms solo gli utenti invitati (che accettano l'invito) dell'utente che ha creato la stanza, detto *owner*, che potrebbe anche non invitare nessun altro utente nella sua stanza[2]. Le CoCreation rooms si dividono in due tipologie: CoCreation Knownledge room e CoCreation Data room. Le CoCreation Knownledge rooms includono strumenti che permettono ai membri della stanza di collaborare per creare un *template document* e di annotare tramite post-it le Datalates da discutere, mentre le CoCreation Data rooms forniscono ai membri della stanza gli strumenti per creare un dataset.
All'interno della pagina *CoCreation*, un utente può visualizzare tutte le stanze di cui è membro, visualizzare la lista di tutti i datasets pubblicati che, tra le altre opzioni che ha a disposizione, può esportare in formato CSV [3] per poi modificarlo all'interno di una stanza per creare un nuovo datasets; infine può creare una nuova CoCreation room di cui deve compilare obbligatoriamente vari campi: Nome, soggetto, descrizione, periodo di validità, obiettivo e testo di invito della stanza.

3.2.1 CoCreation Data room

Una CoCreation Data room è un foglio di lavoro in cui i membri possono lavorare contemporaneamente, tutte le modifiche sono immediatamente visibili a tutti (figura 3.2). Come abbiamo già specificato i membri possono creare un dataset partendo dal nulla o importando un dataset già esistente, inoltre hanno a disposizione diverse funzioni: possono aggiungere numerosi metadati al dataset, aggiungere note tramite un documento di testo condiviso esattamente come il foglio di lavoro, creare e argomentare una discussione sul dataset e infine creare una Datalet. Una Datalet è la visualizzazione di un dataset ed esistono diverse tipologie disponibili, a seconda dei dati che il dataset contiene (SPOD automaticamente suggerisce agli utenti le tipologie di Datalet adatte). Quando il dataset contiene delle immagini automatica-

[2] L'owner può aggiungere nuovi membri anche in un secondo momento.

[3] In questo formato, ogni riga della tabella è normalmente rappresentata da una linea di testo, che a sua volta è divisa in campi (le singole colonne) separati da un apposito carattere separatore, ciascuno dei quali rappresenta un valore.



	"Tipologia"	Nome	Indirizzo	Proprietà	Stato	Secolo	Uso primario	Orari	Orari	Uso secondario
1	"Tipologia"	Nome	Indirizzo	Proprietà	Stato	Secolo	Uso primario	Orari celebrazioni feriali	Orari celebrazioni festivi	Uso secondario
2	Chiesa	Santa Maria la Nova	Piazza Santa la Nova, 18 80143 Napoli	Frati Minori	Utilizzata	XVI	Culto			Amministrazio... ARCA,A.A.P.N, Pegaso
3	Chiesa	San Giorgio dei Genovesi	Via Medina, 56 80133 Napoli	Arcidiocesi	Dismessa	XV	Culto			
4	Basilica	San Giacomo degli Spagnoli	Piazza Municipio, 27 80133 Napoli	Arcidiocesi	Utilizzata	XVI	Culto			
5	Chiesa	Santa Brigida	Via Santa Brigida, 68 80132 Napoli	Padri Leonardini	Utilizzata	XVII	Culto	07.30 - 10.00 - 12.00 - 18.30	09.30 - 11.30 - 18.30	
6	Chiesa	San Ferdinando	Piazza Trieste e Trento, 80132 Napoli	Arcidiocesi	Utilizzata	XVII	Culto	08.00 - 18.00	10.30 - 12.00	Turistico
7	Chiesa	Santa Maria Incoronata	Via Incoronata, 5 80133 Napoli	Demanio	Utilizzata	XIV	Culto			
8	Chiesa	Santa Barbara dei Cannonieri	Rua catalana, 100 80133 Napoli	Confraternita	Utilizzata	XIV	Culto			Associazione Amici del Presepe
9	Chiesa	San Giacomo degli Italiani	Via Agostino Depretis, 39 80133 Napoli	Arcidiocesi	Dismessa	XIII	Sconsacrata			Convegni, Esposizioni Artistiche
10	Chiesa	San Diego all'Ospedaletto	Via Medina,3 80133 Napoli	Arcidiocesi	Utilizzata	XVI	Culto			
11	Chiesa	Santa Maria della Graziella	Via San Bartolomeo,63 80133 Napoli	Arcidiocesi	Utilizzata	XVIII	Culto	Aprile-Maggio-Giugno 18.30		
12	Chiesa	San Bartolomeo	Via San Bartolomeo,18 80133 Napoli	Arcidiocesi	Dismessa	XV	Sconsacrata			B&B
13	Chiesa	Santa Maria Incoronatella	Via Medina, 19 80133 Napoli	Arcidiocesi	Utilizzata	XVI	Culto	08.30 - 18.30	11.00 - 12.30 - 19.00	

Figura 3.2: Esempio di una CoCreation Data room.

mente viene suggerita, oltre la classica tipologia *Table*, la tipologia *Map* che è l'unica in cui le immagini vengono visualizzate[4]. La Datalet *Map* richiede l'inserimento nel dataset di coordinate geografiche a cui verrà associato un balloon che rappresenterà la posizione sulla mappa; per visualizzare l'immagine è necessario solamente inserirla nel *Balloon content*, eventualmente insieme ad altri metadati. Le Datalets una volta create sono visibili solo all'interno della stanza nell'apposita voce del menù, i membri della stanza possono però importarla nel proprio spazio privato e da lì pubblicarla. Il dataset, così come per le Datalets, una volta creato è visibile solo all'interno della stanza, ma può essere pubblicato su un nuovo provider, detto SPOD provider, e sarà quindi a quel punto condiviso con tutti gli utenti del sito e visibile nella lista dei datasets pubblicati.

3.2.2 Immagini all'interno delle CoCreation Data rooms

Come spiegato nell'Introduzione il problema affrontato è quello di inserire le immagini all'interno dei datasets durante la CoCreazione, con le conseguenti problematiche di dover gestire tali immagini all'interno della piattaforma. Per realizzare questi obiettivi sono state aggiunte alle Cocreation Data rooms due funzionalità: Una pagina *Browse Photo* a cui si accede tramite il menù delle stanze, e una componente a cui si accede tramite il foglio di lavoro.

3.2.2.1 Pagina *Browse Photo*

La pagina è divisa in due sezioni: "EXPLORE" dove è possibile accedere a tutte le immagini caricate nella piattaforma[5] e "ROOM'S PHOTOS" dove invece si accede alle immagini relative alla stanza corrente (figura 3.3). Quando viene creata una stanza, automaticamente viene creato un album il cui nome, descrizione e proprietario corrispondono al nome, alla descrizione e all'*owner* della stanza, e al suo interno verranno caricate tutte le immagini relative ad essa. I membri della stanza possono quindi visualizzare il contenuto dell'album e caricare altre immagini di cui possono opzionalmente indicare una descrizione e decidere quali immagini inserire nel foglio di lavoro. Una volta caricate, infatti, le immagini sono contenute all'interno dell'album associato alla stanza ma per inserirle nel dataset, bisogna inserire l'URL dell'immagine all'interno del foglio di lavoro. Questo è possibile

[4]Nelle altre tipologie come ad esempio la tipologia *Table* viene visualizzato solamente l'URL dell'immagine.

[5]Questa sezione ha le medesime funzionalità dell'omonima sezione del plugin Photo descritte nel paragrafo 2.2.1.

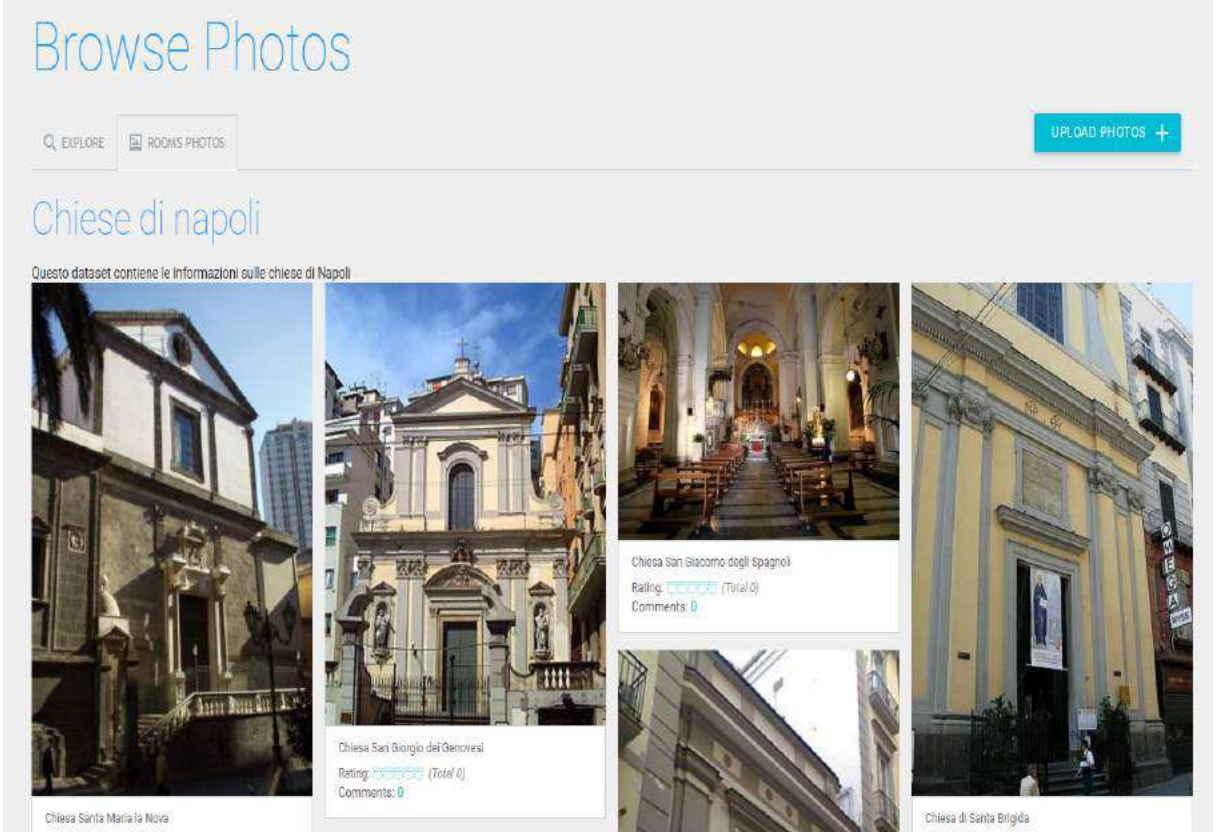


Figura 3.3: Esempio della pagina *Browse Photo* della stanza "Chiese di Napoli".

tramite la funzionalità *Inserisci nella stanza*, disponibile per le immagini di entrambe le sezioni[6], che fornisce all'utente l'URL dell'immagine che può copiare e incollare nella cella che preferisce a seconda di come il dataset è stato costruito.
L'*owner* della stanza ha inoltre la possibilità di modificare la descrizione delle immagini oppure eliminarle mentre l'album, in quanto legato alla stanza, non può essere nè eliminato nè tanto meno i suoi metadati possono essere modificati.

[6]Può quindi essere caricata nel dataset una qualsiasi immagine caricata sul sito, non solo quelle contenute nell'album relativo alla stanza di creazione.

3.2.2.2 Componente *Aggiungi un'immagine*

All'interno del foglio di lavoro, l'utente ha la possibilità di caricare un'immagine e inserirla direttamente nel foglio senza accedere alla pagina *Browse Photo*. Tramite il tasto destro accede alla voce *Aggiungi un'immagine* che apre un frame nel quale può selezionare un'immagine dal suo file system, tramite il classico tasto *Scegli file* o tramite la tecnica di *drag and drop* [7] (figura 3.4) . L'immagine selezionata sarà visualizzata in anteprima, l'utente può selezionare e visualizzare diverse immagini e opzionalmente inserire una descrizione, finchè non sceglie quella da caricare nel dataset. L'immagine scelta è caricata nell'album associato alla stanza [8] e la sua URL è automaticamente inserita nella cella selezionata.

3.3 Implementazione

Tutta l'implementazione del lavoro di tesi avviene all'interno del plugin Cocreation di SPOD, in cui sono state modificate alcune classi, mentre altre sono state aggiunte o riutilizzate dal plugin *Photo* [9]. Dividiamo la spiegazione dello sviluppo del lavoro in due parti relative alle due principali funzionalità.

3.3.1 Pagina *Browse Photo*

3.3.1.1 Creazione dell'album

Per quanto riguarda l'implementazione della pagina *Browse Photo* all'interno delle CoCreation Data rooms, per prima cosa è stato fatto in modo che quando un utente crea una CoCreation room, automaticamente viene creato un album corrispondente a quella stanza. Per ottenere questo legame tra una stanza e un album è stata modificata la classe COCREATION_BOL_Room che è l'entità che rappresenta la corrispondente tabella nel database.

```
1  <?php
2
3  class COCREATION_BOL_Room extends OW_Entity
4  {
5      public $type;
6      public $ownerId;
```

[7] L'utente può trascinare e rilasciare l'immagine all'interno dell'apposito riquadro.
[8] Essa sarà quindi visibile nella pagina *Browse Photo*
[9] di cui abbiamo parlato nel paragrafo 2.2.1.

Figura 3.4: Esempio della componente *Aggiungi un'immagine*.

```
7       public $name;
8       public $subject;
9       public $description;
10      public $from;
11      public $to;
12      public $goal;
13      public $invitationText;
14      public $isOpen;
15      public $timestamp;
16      public $albumId;
17  }
```

A questa classe è stata aggiunto il campo *albumId* che rappresenta appunto l'id dell'album associato a quella stanza, ovviamente alla modifica dell'entità corrisponde una modifica al database quindi è stata modificata anche la dichiarazione della tabella in MYSQL nel file *istall.php*[10].

```
1    'CREATE TABLE IF NOT EXISTS `` . OW_DB_PREFIX . 'cocreation_room` (
2     `id` int(11) NOT NULL AUTO_INCREMENT,
3     `type` varchar(255),
4     `ownerId` int(11) NOT NULL,
5     `albumId` int(11),
6     `name` varchar(255),
7     `subject` varchar(255),
8     `description` varchar(255),
9     `from` date,
10    `to` date,
11    `goal` varchar(255),
12    `invitationText` varchar(255),
13    `isOpen` tinyint(1),
14    `timestamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
15    PRIMARY KEY (`id`)
16  ) ENGINE=MyISAM  DEFAULT CHARSET=utf8 ;
```

La form della creazione di una CoCreation room è implementata dalla componente *create_room.php*, l'action della form è il metodo **createRoom** del Controller *ajax.php* che prende i valori inseriti nella form e crea un'istanza della classe COCREATION_BOL_Room; crea l'album corrispondente alla stanza e invia un messaggio ai membri invitati dall'*owner*. Vediamo la parte di codice relativa alla creazione dell'album:

[10] Ricordiamo che quando il plugin viene istallato vengono create le tabelle.

```
1  $clean = ODE_CLASS_InputFilter::getInstance()->sanitizeInputs($_REQUEST);
2
3   if ($clean == null){
4
5   OW::getFeedback()->info(OW::getLanguage()->text('cocreation', 'insane_user_email_
6   exit;
7   }
8
9   $album = new PHOTO_BOL_PhotoAlbum();
10  $album->name = $clean['name'];
11  $album->userId =  OW::getUser()->getId();
12  $album->entityType = 'user';
13  $album->createDatetime = time();
14  $album->description =  $clean['description'];
15
16  $albumId =  PHOTO_BOL_PhotoAlbumService::getInstance()->addAlbum($album);
17
18  $room = COCREATION_BOL_Service::getInstance()->addRoom(
19  OW::getUser()->getId(),
20  $clean['name'],
21  $clean['subject'],
22  $clean['description'],
23  $clean['data_from'],
24  $clean['data_to'],
25  $clean['goal'],
26  $clean['invitation_text'],
27  empty($clean['is_open']) ? 0 : 1,
28  implode("#######", $clean['users_value']),
29  $clean['room_type'],
30  $albumId
31  );
```

La variabile $clean contiene l'input inserito dall'utente che ottiene tramite $_REQUEST [11] (riga 1), viene controllato che la variabile non sia null, in tal caso è lanciato un messaggio di errore (righe 3-7); viene creato un nuovo album, istanza della classe PHOTO_BOL_PhotoAlbum [12] , a cui

[11] $_REQUEST è un array associativo che contiene le variabili di una richiesta HTTP, ovvero il contenuto di $_GET, $_POST e $_COOKIE.

[12] la classe si trova nell'appendice sezione A.1.1, assumiamo che tutte le classi la cui plugin key è PHOTO appartengono al plugin Photo.

viene assegnato il nome e la descrizione inserite per la stanza, ottenute dalla variabile $clean, il proprietario dell'album, rappresentato dal campo *userId* è l'*owner* della stanza, ovvero l'utente corrente ottenuto tramite la funzione **OW::getUser()->getId()**. Viene poi aggiunto l'album appena creato al database tramite la funzione **addAlbum** della classe PHOTO_BOL_PhotoAlbumService che si occupa della logica di business relativa all'entità PhotoAlbum (righe 9-16). A partire da riga 18 in poi viene creata e aggiunta una nuova stanza nell'apposita tabella nel database, assegnandole gli input inseriti dall'utente, e al campo *albumId* viene assegnato l'id dell'album appena creato.
Una volta che la stanza è stata creata, le CoCreation Data rooms sono gestite dal Controller *data_room.php* che contiene solo il metodo *index*. Al suo interno tramite il metodo **UTIL_JsGenerator::composeJsString**, vengono memorizzati alcuni valori relativi alla stanza corrente, che tramite il nome che gli assegniamo possiamo utilizzare in qualunque script javascript esterno alle views. Vediamo i valori memorizzati che ci serviranno in seguito:

```
1  $js = UTIL_JsGenerator::composeJsString('
2  COCREATION.roomId                          = {$roomId}
3  COCREATION.ownerId                         = {$ownerId}
4  COCREATION.album                           = {$album}
5  ', array(
6   'roomId'                                 => $params['roomId'],
7   'ownerId'                                => $room->ownerId,
8   'album'                                  => $room->albumId,
9   ));
```

Possiamo notare che sono stati memorizzati l'id della stanza, l'id del *owner* della stanza e quindi anche dell'album e l'id dell'album, tramite le label: *COCREATION.roomId, COCREATION.ownerId, COCREATION.album.*

3.3.1.2 Aggiunta della pagina *Browse Photo* alle CoCreation data rooms

All'interno della view corrispondente al controller che gestisce le CoCreation Data rooms *data_room_index.html* è stata aggiunta una voce al menù per aggiungere il collegamento alla pagina *Browse Photo* (figura 3.5):

```
<paper-item onclick="room.handleSelectUIMode('Photo')">
<iron-icon></iron-icon>
<paper-item-body two-lines title='{text key="cocreation+room_cc_datalet"}'>
```

```
<div>Photo</div>
</paper-item-body>
</paper-item>
```

La funzione **room.handleSelectedUIMode** è definita all'interno del file *cocreation-data.js* :

```
1  room.handleSelectUIMode = function(mode){
2   switch(mode){
3   case 'Photo':
4   window.location='/cocreation/album/'+ COCREATION.ownerId +'/' + COCREATION.album;
5   break;
6  }
7  };
```

La funzione prende come parametro un valore che indica quale voce del menù è stata selezionata, in base a questo parametro svolge funzioni diverse; nel nostro caso abbiamo passato il parametro "Photo" a cui è associata una funzione che semplicemente apre la nostra pagina *Browse Photo*. La pagina è gestista dal Controller *photo.php* [13]aggiunto al plugin Cocreation, che contiene due metodi: *explore* che gestisce l'omonima sezione della pagina e *userAlbum* che gestisce la sezione "ROOM'S PHOTOS". A tutti e due i metodi del controller sono state associate delle rotte [14]:

```
1  OW::getRouter()->addRoute(new OW_Route('cocreation.photo',
2  'cocreation/photo/:user/:albumId',"COCREATION_CTRL_Photo",'explore'));
3  OW::getRouter()->addRoute(new OW_Route('cocreation.album',
4  'cocreation/album/:user/:albumId', "COCREATION_CTRL_Photo", 'userAlbum'));
```

I valori preceduti da ":" sono valori che devono essere passati come parametri quando la rotta viene chiamata. Nell'estratto di codice precedente abbiamo richiamato la rotta corrispondente al metodo *userAlbum* passandogli l'id dell'*owner* e l'id dell'album associato alla stanza, in modo che il controller mostrerà il contenuto dell'album corrispondente alla stanza.

3.3.1.3 Funzionalità della pagina

All'interno della pagina, i membri della stanza che vi possono accedere, possono caricare delle immagini tramite il tasto *UPLOAD PHOTOS* che lancia la componente *upload_photos.php*, che a sua volta lancia la classe

[13]vedi appendice sezione A.1.2.

[14]Ricordiamo che le rotte sono definite nel file *init.php*

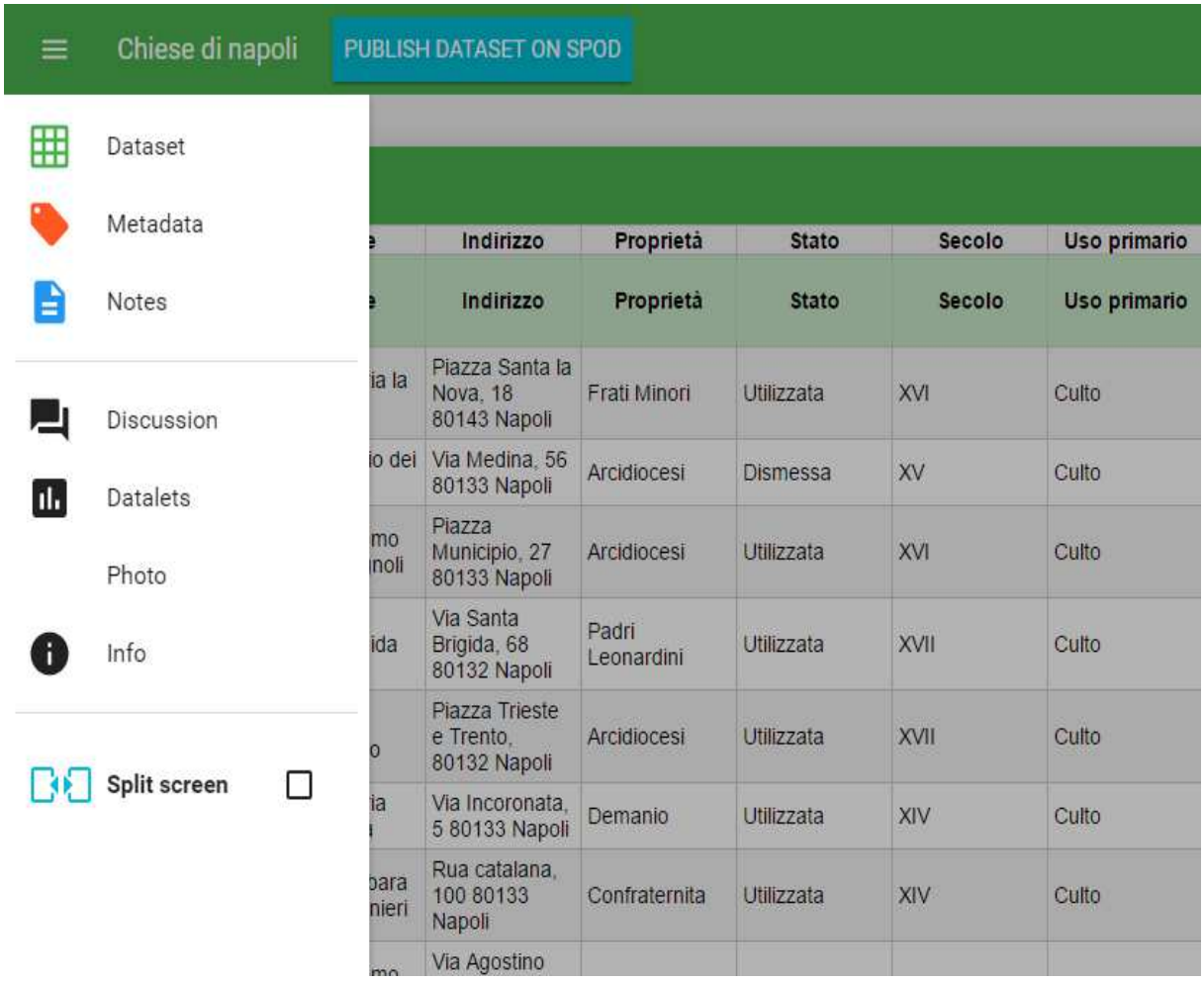


Figura 3.5: Menù delle CoCreation Data rooms.

upload_photos_form.php[15]. L'utente può caricare una o più immagini, indicare opzionalmente una descrizione e quando effettua il submit, le immagini vengono caricate nell'album e quindi visualizzate nella pagina. Queste classi sono le stesse classi del plugin Photo con piccole modifiche dovute al fatto che le immagini devono essere caricate solo nell'album corrispondente alla stanza e non in altri album [16]. Rispetto al plugin Photo, è stata aggiunta invece la possibilità di ottenere l'URL dell'immagine per poterla inserire nel foglio di lavoro. Vediamo l'implementazione di questa funzionalità che si trova all'interno del file *browse_photo.js*:

```
1   InserisciNellaStanza: function (slot) {
2
3   var photoId = slot.id;
4   $.post("/cocreation/getPhoto", {id: photoId}, function (data, status){
5   var check=  window.prompt("Copia l'URL con CTRL+C e incollalo con CTRL+V
6   nella cella della stanza che preferisci",data);
7   if(check!=null){
8   $.post("/cocreation/getRoom", {albumId: slot.albumId},
9   function (data, status) {
10  window.location="/cocreation/data-room/"+data;
11       });}
12   });
13  }
14
```

Su ogni immagine compare un menù a tendina in cui ci sono varie opzioni tra cui *Inserisci nella stanza*, a cui è collegata la funzione mostrata nel frammento di codice. La funzione prende come parametro l'oggeto *slot* che è il riferimento all'immagine selezionata, tramite questo oggetto possiamo accedere alle proprietà dell'immagine, in questo caso al suo id memorizzato nella variabile photoId (riga 3). Tramite la funzione **$.post** [8] svolgiamo una chiamata di tipo POST al metodo **getPhotoUrl** del controller *ajax.php* a cui è associata la rotta "/cocreation/getPhoto"; il secondo parametro della funzione è l'id dell'immagine che passiamo al metodo, il terzo è una funzione che viene eseguita quando la chiamata POST è terminata, essa a sua volta prende due parametri: il risultato (data) e lo stato (status) della chiamata. Mostriamo il codice del metodo **getPhotoUrl** per comprendere cosa restituisce:

[15] le due classi sono mostrate nella sezione A.1.3 dell'appendice.

[16] Queste modifiche sono di poca importanza quindi non sono state inserite.

```
1   public function getPhotoUrl(){
2   $clean = ODE_CLASS_InputFilter::getInstance()->sanitizeInputs($_REQUEST);
3   if ($clean == null){
4   OW::getFeedback()->info(OW::getLanguage()->text('cocreationes', 'insane_values'));
5   exit;
6   }
7   $photoId=$clean['id'];
8   $photoService=PHOTO_BOL_PhotoService::getInstance();
9   $url=$photoService->getPhotoUrl($photoId);
10  echo $url;
11  exit;
12  }
```

Le righe 2-4 servono per inizializzare e svolgere i controlli sulla variabile $clean come abbiamo già visto in precedenza; otteniamo l'id dell'immagine che gli abbiamo passato, tramite il metodo **getPhotoUrl** dell'oggetto **PhotoService** otteniamo l'URL dell'immagine, la restituiamo e terminiamo il metodo (riga 7-11).
Tornando al codice javascript, il terzo parametro di **$.post** lancia una finestra tramite window.prompt dove viene inserito l'URL dell'immagine ottenuto dalla chiamata POST; memorizza nella variabile check se l'utente ha confermato o annullato l'operazione (linee 5-6), se ha confermato l'operazione (cioè ha copiato l'URL e premuto su OK), viene effettuata una chiamata POST al metodo **getRoom**, la cui rotta è "cocreation/getRoom", dello stesso controller, nel medesimo modo di prima, passando stavolta come parametro l'id dell'album, ottenuto sempre grazie all'oggetto *slot* (linea 8).
Mostriamo il metodo **getRoom**:

```
1   public function getRoomId(){
2   $clean = ODE_CLASS_InputFilter::getInstance()->sanitizeInputs($_REQUEST);
3   if ($clean == null){
4   OW::getFeedback()->info(OW::getLanguage()->text('cocreationes', 'insane_values'));
5   exit;
6   }
7   $albumId=$clean['albumId'];
8   $room=COCREATION_BOL_Service::getInstance()->getRoomIdByAlbumId($albumId);
9   echo $room;
10  exit;
11  }
```

Il metodo ottiene l'id dell'album tramite $clean e chiama il metodo **getRoomIdByAlbumId** dell'oggetto COCREATION_BOOL_Service che si occu-

pa della logica di business relativa alle tabelle della Cocreation, il metodo restituisce appunto l'id della stanza a cui associato quell'album: .

```
1   public function getRoomIdByAlbumId($albumId){
2   $example = new OW_Example();
3   $example->andFieldEqual('albumId', $albumId);
4   return COCREATION_BOL_RoomDao::getInstance()->findIdByExample($example);
5   }
```

In questo metodo grazie all'oggetto **OW_Example** abbiamo settato una condizione di uguaglianza, e tramite il metodo **findIdByExample**, otteniamo l'id della stanza a cui è associato l'id dell'album passato come parametro.
Quindi tornando di nuovo alla funzione javascript, la chiamata al metodo **getRoom** restituisce l'id della stanza, così quando l'utente conferma che ha copiato l'URL, la pagina viene reindirizzata nella stanza corrente dove può copiare l'URL nella cella che desidera (linee 9-11); se clicca su annulla, la finestra si chiude e rimane sulla pagina *Browse Photo*.

3.3.2 Componente *Aggiungi un'immagine*

Quando l'utente accede ad una stanza di cui è membro, cliccando con il tasto destro su una cella qualsiasi del foglio di lavoro, avrà la possibilità di aggiungere un'immagine direttamente nella cella tramite l'apposita voce nel menù che compare, a cui è collegata la componente *add_photo.php* che è stata aggiunta al plugin Cocreation.
Il foglio di lavoro di una CoCreation Data room è implementato tramite Ethersheet[17], per aggiungere una voce al menù del foglio sono state quindi apportate delle modifiche a Ethersheet, mostriamo prima queste modifiche e poi l'implementazione della componente *add_photo.php*.

3.3.2.1 Ethersheet

All'interno del file *cell_menu.jst* è stata aggiunta una nuova voce:

```
1   <li
2   id="es-menu-add-newphoto"
3   class="es-menu-button i18n"
4   data-action="add_new_photo"
```

[17]Ethersheet è uno strumento open source per collaborare realtime su uno foglio di calcolo in modo facile, veloce e sicuro

```
5   data-i18n="add_new_photo">
6   Add a new photo
7   </li>
```

La voce ha come valore "Add a new photo", tradotto in tutte le lingue in cui SPOD è disponibile, ha un id univoco e una action che servirà a definire l'azione legata al pulsante nel file *cell_menu.js*:

```
1    onButtonClick: function(e){
2       console.log("CLICKLICKLCIK")
3       var action = $(e.currentTarget).data('action');
4       switch(action){
5       case 'add_geo_point':
6       this.addGeoPoint();
7       break;
8       case 'add_new_photo':
9       this.addNewPhoto();
10      break;
11      };
12  },
13
14   addNewPhoto:function () {
15   top.postMessage("open-select-photo_event",
16   'http://' + window.location.hostname);
17    }
```

La funzione associata all'azione del pulsante semplicemente lancia l'evento "open-select-photo_event". All'interno del file *cocreation-data.js* [18] viene gestito l'evento:

```
1   case 'open-select-photo_event':
2   ODE.pluginPreview = "cocreation";
3   previewFloatBox = OW.ajaxFloatBox('COCREATION_CMP_AddPhoto',
4   {albumId: COCREATION.album} ,
5   {width:'100%', height:'60vh', iconClass:'ow_ic_lens', title:''});
6   break;
7   }
```

Quando l'evento viene catturato, viene lanciata la componente **COCREATION_CMP_AddPhoto** (linea 3) a cui passiamo come parametro l'id

[18] che ricordiamo non si trova all'interno di Ethersheet.

dell'album della stanza corrente[19] (linea 4), all'interno di una ajaxFloatBox di cui settiamo le dimensioni (linea 5), che la farà visualizzare all'interno della stessa pagina che l'ha lanciata.

3.3.2.2 Componente *add_photo.php*

```
1   class COCREATION_CMP_AddPhoto extends OW_Component {
2
3    public function __construct($albumId){
4
5      $this->assign('albumId',$albumId);
6
7      $form=new Form('add_photo_form');
8
9      $form->setAction(OW::getRouter()->urlFor('
10     COCREATION_CTRL_AjaxUpload', 'upload') );
11     $form->setId('addphoto');
12     $form->setEnctype(Form::ENCTYPE_MULTYPART_FORMDATA);
13
14     $input=new FileField('file');
15     $input->setId("photo");
16     $form->addElement($input);
17
18     $desc=new TextField('desc');
19     $desc->setLabel('Descrizione');
20     $desc->setId('desc');
21     $form->addElement($desc);
22
23     $scegli=new Button('scegli');
24     $scegli->setId('scegli');
25     $scegli->setValue('Scegli file');
26     $form->addElement($scegli);
27
28     $carica=new Submit('carica');
29     $carica->setId('carica');
30     $carica->setValue('Carica');
31     $form->addElement($carica);
32
```

[19] ricordiamo che nel controller *data_room.php* abbiamo salvato l'id dell'album associato alla stanza corrente nella variabile *COCREATION.album*.

```
33      $this->addForm($form);
34
35
36       }}
```

La componente prende come parametro l'id dell'album associato alla stanza corrente e lo invia al template Smarty tramite il metodo **assign**[20] (linea 5). Dopo questo passaggio viene creata la form, settiamo il metodo **upload** del controller *ajax_upload.php*[21] come action della form (linee 9-10), settiamo l'enctype della form come "multipart-form-data" per consentire il caricamento di files (linea 12), dopodichè creiamo un input di tipo file, un input di tipo text e la sua label, due button di cui uno di tipo submit per sottomettere la form di cui andiamo a definire i corrispondenti valori. Per tutti gli elementi definiamo un id, e li aggiungiamo alla form (linee 14-31), infine aggiungiamo la form alla componente. Mostriamo ora la view della componente *add_photo.html*:

```
1   <div id="cont" >
2   <div id="immagine" >
3   <p> Trascina qui </p>
4   </div>
5   <div id="tasti">
6   {form name='add_photo_form'}
7   <table>
8   <tr>
9   <td hidden>{input name='file'}</td>
10  <td>{label name='desc'}</td><td>{input name='desc'}</td>
11  </tr><tr>
12  <td>{submit name='scegli'}</td>
13  <td>{submit name='carica'}</td></tr>
14  </table>
15  {/form}
16  </div>
17  </div>
```

Nella view ci sono tre tag : uno rappresenta il riquadro per l'immagine, uno è il contenitore per la form ed infine l'ultimo è il contenitore che contiene gli altri due. La form è stata strutturata tramite una tabella. Nella prima riga abbiamo due elementi ognuno contenuto in una cella diversa:

[20] di cui abbiamo spiegato il funzionamento nel paragrafo 2.3.5.
[21] di cui spiegheremo dopo il funzionamento.

l'input di tipo file che però è nascosto poichè la sua funzione è simulata dal button il cui valore è "Scegli file" [22], e l'input di tipo testo all'interno del quale l'utente può eventualmente inserire una descrizione per l'immagine. Nella seconda riga ci sono invece i due button[23]. All'interno della view c'è un tag <script> che contiene il codice javascript che gestisce la componente. Mostriamo per prima cosa la gestione degli eventi:

```
1   $(document).ready(function(){
2   var file;
3
4    $("#photo").on('change',function () {
5    file=this.files[0];
6    upload(file);
7    });
8
9    $("#scegli").click(function () {
10   $("#photo").click();
11   });
12
13   $('#cont').on(
14   'dragover',
15   function(e) {
16   e.preventDefault();
17   e.stopPropagation();
18   });
19
20   $('#cont').on(
21  'dragenter',
22   function(e) {
23   e.preventDefault();
24   e.stopPropagation();
25   });
26
27   $('#cont').on(
28   'drop',
29    function(e){
```

[22] In questo modo è stato possibile personalizzare lo stile del button, poichè l'input di tipo file segue delle regole CSS di default difficili da modificare.

[23] Possiamo vedere come la form è stata costruita seguendo le regole descritte nel paragrafo 2.3.7.

```
30      if(e.originalEvent.dataTransfer){
31      if(e.originalEvent.dataTransfer.files.length) {
32      e.preventDefault();
33      e.stopPropagation();
34      file=e.originalEvent.dataTransfer.files[0];
35      upload(file);
36      }}});
```

All'input di tipo file, il cui id è "photo", viene associato il gestore all'evento *change* che si scatena quando l'utente sceglie un'immagine. Alla variabile file viene assegnata l'immagine selezionata, a cui si accede tramite la proprietà files che rappresenta la lista di tutti i files caricati dall'utente, in questo caso prendiamo il primo file della lista in quanto è anche l'unico, poichè l'utente può scegliere una sola immagine per volta. La variabile file viene poi passata come parametro alla funzione upload, che verrà spiegata successivamente in questo paragrafo (linee 4-7).
Al button "Scegli file", il cui id è "scegli", associamo il comportamento di default dell'input di tipo file (linee 9-11); al div che contiene gli altri due, il cui id è "cont", dobbiamo associare vari eventi per permettere il drag and drop dell'immagine. Gli eventi sono tre: *dragover* e *dragenter* che sono scatenati quando l'utente trascina l'immagine sul riquadro ma non la rilascia, e *drop* che si scatena quando invece l'immagine viene rilasciata nel div che rappresenta l'area droppable, per ognuno di essi bisogna fare in modo che non venga eseguito il comportamento di default associato all'evento tramite le funzioni **e.preventDefault** e **e.stopPropagation**, perchè altrimenti l'immagine verrebbe caricata come link nel browser. Per quanto riguarda l'evento *drop* facciamo in modo che, quando si scatena, alla variabile file sia assegnata l'immagine trascinata, grazie al metodo **e.originalEvent.dataTransfer.files** che accede alla lista dei files rilasciati in un'area droppable, come prima prendiamo il primo e unico file della lista, e passiamo la variabile come parametro alla funzione upload (linee 13-63). Vediamo ora la funzione upload:

```
1     var check=false;
2
3     function upload(file){
4     var allowed = ["jpeg","png","jpg","gif"];
5     var found = false;
6     allowed.forEach(function(extension) {
7     if (file.type.match('image/'+extension)) {
8     found = true;
9     }
```

```
10    });
11    if(!found || file.size>2 * 1048576)
12    alert("Non è possibile caricare questo file");
13    else{
14    var cont=document.getElementById("cont");
15    var immagine;
16    if(!check){
17     check=true;
18     immagine=document.createElement("img");
19     immagine.id="imag";
20     var div =document.getElementById("immagine");
21     div.hidden=true;
22     cont.appendChild(immagine);
23     }else{
24     immagine=document.getElementById("imag");}
25
26     var reader = new FileReader();
27     reader.onload = function(event) {
28     var dataURL = event.target.result;
29     immagine.src=dataURL;
30     };
31     reader.readAsDataURL(file);
32     }};
```

La funzione prende il file che gli è stato passato come parametro, svolge i controlli lato client sulla dimensione ed estensione del file in modo da poter avvertire l'utente se ha scelto un'immagine che non può essere caricata (linee 4-12), dopodichè fa in modo che l'immagine sia visualizzata in anteprima nel riquadro (div con id "immagine") e che l'utente possa scegliere e visualizzare varie immagini (ma una sola per volta) prima di caricare quella definitiva all'interno della stanza. Viene creato un tag <img> a cui viene assegnato l'id "imag" e inserito al posto del div "immagine", ovvero quest'ultimo viene nascosto, mentre il tag <img> viene inserito come figlio del div "cont". La variabile booleana check controlla che il tag <img> sia già stato creato (l'utente ha già visualizzato un'immagine in anteprima), in tal caso ottiene il riferimento al tag e non ne crea uno nuovo (linee 13-24). Ottenuto il riferimento, all'attributo src del tag assegniamo l'URL dell'immagine selezionata, ottenuta tramite l'oggetto **FileReader**. L'oggetto **FileReader** consente l'accesso in lettura del file scelto dall'utente e ad esso viene associato il gestore all'evento *onload* (quando l'utente sceglie il file): si accede

all'oggetto che ha scatenato l'evento tramite la proprietà result di target dell'oggetto event, l'oggetto tramite la funzione **readAsDataURL** è interpretato come un data URL e può essere associato all'attributo src del tag <img> (linee 26-32).
Passiamo ora ad affrontare la parte relativa alla sottomissione della form, come abbiamo detto precedentemente in questo paragrafo l'action è il metodo **upload** del controller *ajax_upload.php*. Questo metodo è il medesimo metodo definito nell'omonimo controller nel plugin Photo, che è stato leggermente modificato per far in modo che l'immagine non fosse solo caricata sul server, ma anche memorizzata nel database e inserita nell'album della stanza corrente. Vediamo solo l'estratto di codice relativo alle modifiche:

```
1   if ( ($id = PHOTO_BOL_PhotoTemporaryService::getInstance()->
2   addTemporaryPhoto($_FILES['file']['tmp_name'],
3   OW::getUser()->getId(), $order)) )
4   {
5   $photoTmpService = PHOTO_BOL_PhotoTemporaryService::getInstance();
6   $photoService=PHOTO_BOL_PhotoService::getInstance();
7   $photo = $photoTmpService->moveTemporaryPhoto($id, $_POST['albumId'],
8   $_POST['desc'] , NULL, 0);
9   $photoTmpService->deleteTemporaryPhoto($id);
10  $fileUrl=$photoService->getPhotoUrl($photo->id);
11  $this->returnResponse(array(
12  'status' => self::STATUS_SUCCESS,
13  'fileUrl' => $fileUrl,
14  'id' => $photo->id
15   ));
16   }
```

L'immagine viene caricata sul server e aggiunta alle immagini temporanee tramite il metodo **addTemporaryPhoto** dell'oggetto PhotoTemporaryService (linee 1-3) che restituisce l'id dell'immagine. Tramite il metodo **moveTemporaryPhoto** l'immagine viene spostata dalle immagini temporanee a quelle definitive e inserita nell'album; il metodo prende infatti tra i parametri l'id dell'immagine temporanea, l'id dell'album e la descrizione dell'immagine. Il metodo restituisce un oggetto di tipo PHOTO_BOL_Photo e una volta spostata, l'immagine è eliminata dalla tabella PhotoTemporary. Tramite il metodo **getPhotoUrl** a cui passiamo l'id dell'oggetto Photo, otteniamo l'URL dell'immagine, che a sua volta passiamo come parametro al metodo **returnResponse** insieme all'id dell'immagine e allo stato. Il metodo **returnResponse** semplicemente codifica i dati in formato JSON e

li restituisce.
Tornando alla componente *addPhoto* , quando la form viene sottomessa, di default verrebbe eseguito il metodo upload che abbiamo appena visto, in quanto action della form, ma abbiamo associato all'evento *submit* un gestore che fa in modo che anzichè eseguire l'action della form, quando l'utente clicca sul tasto "CARICA" (il button di tipo submit) l'URL dell'immagine sia automaticamente inserita nel foglio di lavoro.

```
1   $('#addphoto').submit(function (event) {
2   event.preventDefault();
3   var desc=document.getElementById("desc").value;
4   var formData = new FormData();
5   formData.append('albumId',"{$albumId}");
6   formData.append('file', file);
7   formData.append('desc', desc);
8   $.ajax({
9   url : this.action,
10  type : 'POST',
11  data : formData,
12  processData: false,
13  contentType: false,
14  success : function(data) {
15  var dati=JSON.parse(data);
16  $("iframe")[0].contentWindow.postMessage(dati.fileUrl,
17  'http://' + window.location.hostname + ":8001");
18  previewFloatBox.close();
19  },
20  error:function () {
21  alert("Errore nell'invio della richiesta");
22  }
23  });
24  });
```

Per prima cosa viene impedito che venga eseguito il comportamento di default (linea 2), otteniamo la descrizione che l'utente ha inserito per l'immagine (linea 3) e costruiamo un oggetto **FormData** la cui funzionalità è di compilare un'insieme di coppie chiave-valore da inviare tramite un XMLHttpRequest; i valori inviati sono l'id dell'album, l'immagine scelta e la descrizione, da notare che l'id dell'album è ottenuto tramite la variabile {$albumId} che abbiamo passato al template Smarty con il metodo **assign** (linee 4-7). Tramite la funzione **$.ajax** [7] svolgiamo una richiesta (ajax)

HTTP asincrona all'action della form, specifichiamo che la chiamata è di tipo POST, e altri due parametri: *processData* settato a false, indica che i dati vengono processati non come stringa ma con il loro formato originale e *contentType* anche esso settato a false, serve ad indicare che la form è di tipo multipart/form-data (linee 8-13). Se la chiamata ha avuto successo viene eseguita la funzione associata al parametro *success* , il cui parametro *data* rappresenta i dati restituiti dal Server. La funzione decodifica i dati, che ricordiamo sono in formato JSON, e tramite il metodo **postMessage** [6] invia l'URL dell'immagine all'oggetto su cui il metodo è chiamato, che può essere un frame o una finestra, in questo caso è il primo frame contenuto nel documento HTML da cui la componente è stata lanciata, cioè il frame in cui si trova il foglio di calcolo Ethersheet. Il secondo parametro del metodo serve a specificare l'origine della finestra su cui il metodo è chiamato per evitare che i dati siano intercettati, nel nostro caso è stato inserito l'indirizzo del servizio Ethersheet; infine la floatBox viene chiusa (linee 14-18). L'ultimo parametro della funzione **$.ajax** è *error* che stabilisce cosa deve essere eseguito se la chiamata è fallita, in questo caso l'utente viene avvertito del fallimento tramite una alert (linee 20-21).

Capitolo 4

Conclusioni

L'obiettivo di questa tesi era quello di permettere agli utenti iscritti a SPOD, di poter inserire immagini all'interno di open datasets. L'obiettivo è stato raggiunto in quanto gli utenti, grazie alle funzionalità che sono state aggiunte, possono caricare diverse immagini sulla piattaforma che sono visibili in un'apposita sezione all'interno delle CoCreation Data rooms, in cui le immagini sono raggruppate in album legati alle stanze di creazione per rendere agli utenti la loro visualizzazione e la ricerca più chiara. Ereditando alcune funzionalità del plugin Photo, le immagini possono essere visualizzate a schermo intero, votate, commentate, segnalate, ricercate e visualizzate in base a vari criteri ed inoltre possono essere modificate ed eliminate dall'*owner* della stanza. Tutte le immagini caricate sulla piattaforma sono a disposizione di tutti gli utenti e possono essere inserite all'interno dei datasets, con la possibilità di essere visualizzate nella Datalet di tipo *Map*.
Il lavoro potrebbe essere ampliato in quanto prende in considerazione solo la fase di Cocreazione di un dataset, e non la pubblicazione dello stesso. Quando un dataset è pubblicato, bisognerebbe tenere traccia delle immagini contenute al suo interno, in quanto queste non potranno più essere eliminate dal server e magari potrebbero essere inserite in una sezione a parte, differenziando quindi le immagini che vengono caricate in una stanza di Cocreazione per essere eventualmente inserite nel datasets e quelle che sono state effettivamente inserite. Bisogna tener conto inoltre che l'utente può scegliere di inserire nel dataset anche un'immagine caricata in un album diverso da quello associato alla sua stanza di creazione, quindi inserendo le immagini in una sezione separata ci sarebbe molta più chiarezza. Infine potrebbe essere estesa la funzionalità di modifica ed eliminazione delle immagini non solo all'*owner* della stanza, ma a tutti i suoi membri, trasfor-

mando l'ambiente delle immagini in un'ambiente realtime esattamente come quello delle CoCreation rooms (tutti i membri possono apportare modifiche contemporaneamente, esse saranno visibili a tutti immediatamente).

Bibliografia

[1] http://opendefinition.org/ 1

[2] "Sito web di Oxwall" http://www.oxwall.org/ 2

[3] "Oxwall Foundation." https://developers.oxwall.com/foundation 2

[4] "Plugin Development Crash Course" https://wiki.oxwall.com/dev:begin:crash-course

[5] "Sito web di Smarty" http://www.smarty.net/ 2.3

[6] https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage 3.3.2.2

[7] http://api.jquery.com/jquery.ajax/ 3.3.2.2

[8] http://www.w3schools.com/jquery/ajax_post.asp 3.3.1.3

[9] "Gioele Ciaparrone, Transparent Behaviours' Injection in Oxwall, tesi in informatica, Relatore: Prof. Vittorio Scarano, Anno Accademico 2014-2015"

Appendice A

Appendice

A.1 Listati

A.1.1 photoAlbum.php

```
1   class PHOTO_BOL_PhotoAlbum extends OW_Entity
2   {
3       /**
4        * @var integer
5        */
6       public $userId;
7       /**
8        * @var string
9        */
10      public $entityType = 'user';
11      /**
12       * @var integer
13       */
14      public $entityId = null;
15      /**
16       * @var string
17       */
18      public $name;
19
20      public $description;
21
22      /**
23       * @var integer
```

```
24        */
25       public $createDatetime;
26
27   }
```

A.1.2 photo.php

```
1    class COCREATION_CTRL_Photo extends OW_ActionController
2
3    {
4
5    private $photoService;
6    private $photoAlbumService;
7
8    public function __construct()
9    {
10   parent::__construct();
11   $this->photoService = PHOTO_BOL_PhotoService::getInstance();
12   $this->photoAlbumService = PHOTO_BOL_PhotoAlbumService::getInstance();
13   }
14
15   public function explore( array $params)
16   {
17
18   $listType = isset($params['listType']) ? $params['listType'] : 'latest';
19   $owerId=$params['user'];
20   $albumId=$params['albumId'];
21   $event = new BASE_CLASS_EventCollector('photo.collectPhotoList');
22   OW::getEventManager()->trigger($event);
23   $validLists = array_merge($event->getData(),
24   array('featured', 'latest', 'toprated', 'tagged', 'most_discussed'));
25
26    if (!in_array($listType, $validLists)) {
27    $this->redirect(OW::getRouter()->urlForRoute('view_photo_list',
28    array('listType' => 'latest')));
29           }
30
31    $this->assign('listType', $listType);
32    $this->assign('owerId', $owerId);
33    $this->assign('albumId', $albumId);
```

```
34
35    $language = OW::getLanguage();
36    foreach ($validLists as $type) {
37    $language->addKeyForJs('photo',
38   'meta_title_photo_' . $type);
39    }
40
41    $params = array(
42    "sectionKey" => "photo",
43    "entityKey" => "photoList",
44    "title" => "photo+meta_title_photo_list",
45    "description" => "photo+meta_desc_photo_list",
46    "keywords" => "photo+meta_keywords_photo_list",
47    "vars" => array("list_type" =>
48    $language->text("photo", "list_type_label_" . $listType))
49           );
50
51    OW::getEventManager()->trigger(new OW_Event("base.provide_page_meta_info",
52     $params));
53    OW::getDocument()->setTitle($language->text('photo',
54   'meta_title_photo_' . $listType));
55    OW::getDocument()->setDescription($language->text('photo',
56   'meta_description_photo_' . $listType));
57       }
58
59
60       public function userAlbum( array $params )
61       {
62           if (empty($params['user']) ||
63              ($userDto = BOL_UserService::getInstance()->findByUsername
64              (BOL_UserService::getInstance()->getUserName($params['user'])))
65               === null || empty($params['albumId']) || ($album =
66              $this->photoAlbumService->findAlbumById($params['albumId']))
67               === null)
68           {
69               throw new Redirect404Exception();
70           }
71
72           OW::getEventManager()->trigger(new OW_Event('photo.user_album_view',
73                array('album' => $album)));
```

```
74
75          OW::getDocument()->setTitle(
76              OW::getLanguage()->text('photo', 'meta_title_photo_useralbum', array(
77                  'displayName' => BOL_UserService::getInstance()->
78                   getDisplayName($userDto->id),
79                  'albumName' => $album->name
80              ))
81          );
82
83          $owerId=$params['user'];
84          $isOwner = $album->userId == OW::getUser()->getId();
85          $isModerator = OW::getUser()->isAuthorized('photo');
86          $albumId=$params['albumId'];
87
88          $this->assign('isModerator', $isModerator);
89          $this->assign('isOwner', $isOwner);
90          $this->assign('album', $album);
91          $this->assign('albumId',$albumId);
92          $this->assign('owerId', $owerId);
93
94
95              $form = new PHOTO_CLASS_AlbumEditForm($album->id);
96              $this->addForm($form);
97              $this->assign('extendInputs', $form->getExtendedElements());
98
99              $exclude = array($album->id);
100             $newsfeedAlbum = PHOTO_BOL_PhotoAlbumService::getInstance()->
101             getNewsfeedAlbum($album->userId);
102
103             if ( !empty($newsfeedAlbum) )
104             {
105                 $exclude[] = $newsfeedAlbum->id;
106             }
107
108             $albumNameList = $this->photoAlbumService->
109             findAlbumNameListByUserId($userDto->id, $exclude);
110             $this->assign('albumNameList', $albumNameList);
111
112       OW::getDocument()->addScriptDeclarationBeforeIncludes(
113      UTIL_JsGenerator::composeJsString(';window.albumParams =
```

```
114       Object.freeze({$params});',
115       array(
116      'params' => array(
117      'url' => OW::getRouter()->urlFor('PHOTO_CTRL_Photo', 'ajaxResponder'),
118      'album' => $album,
119      'isClassic' => (bool)OW::getConfig()->getValue('photo',
120       'photo_list_view_classic'),
121       'albumNameList' => array_values($albumNameList)
122                         )
123                  ))
124              );
125
126      OW::getDocument()->addScript(OW::getPluginManager()->getPlugin('photo')->
127       getStaticJsUrl() . 'album.js');
128      OW::getDocument()->addOnloadScript(';window.photoAlbum.init();');
129      $lang = OW::getLanguage();
130
131      $lang->addKeyForJs('photo', 'move_to_new_album');
132      $lang->addKeyForJs('photo', 'no_photo_selected');
133      $lang->addKeyForJs('photo', 'crop_photo_title');
134      $lang->addKeyForJs('photo', 'set_as_cover_label');
135      $lang->addKeyForJs('photo', 'are_you_sure');
136      $lang->addKeyForJs('photo', 'confirm_delete_photos');
137      $lang->addKeyForJs('photo', 'photo_deleted');
138      $lang->addKeyForJs('photo', 'photos_deleted');
139      $lang->addKeyForJs('photo', 'photo_album_updated');
140      $lang->addKeyForJs('photo', 'to_small_cover_img');
141      $lang->addKeyForJs('photo', 'album_name_error');
142      $lang->addKeyForJs('photo', 'newsfeed_album');
143      $lang->addKeyForJs('photo', 'photo_success_moved');
144
145      OW::getDocument()->addOnloadScript("$(document).ready(function () {
146      $(\"#add-new-photo-album\").click(function () {
147      previewFloatBox=OW.ajaxFloatBox('COCREATION_CMP_UploadPhotos',
148      {albumId : $albumId}, {
149      title: 'UPLOAD PHOTOS',
150      width: 500
151       });
152      });
153
```

```
154
155     });  ");
156
157          // meta info
158          $vars = BOL_SeoService::getInstance()->getUserMetaInfo($userDto);
159          $vars["album_name"] = $album->name;
160
161          $params = array(
162              "sectionKey" => "photo",
163              "entityKey" => "userAlbum",
164              "title" => "photo+meta_title_user_album",
165              "description" => "photo+meta_desc_user_album",
166              "keywords" => "photo+meta_keywords_user_album",
167              "vars" => $vars,
168              "image" => BOL_AvatarService::getInstance()->
169               getAvatarUrl($userDto->getId(), 2)
170          );
171
172          OW::getEventManager()->trigger(new OW_Event
173          ("base.provide_page_meta_info", $params));
174
175
176
177      }
178
179
180  }
```

A.1.3 upload_photos

```
1   class COCREATION_CMP_UploadPhotos extends OW_Component
2   {
3       public function __construct( $albumId = null, $albumName = null,
4       $albumDescription = null, $url = null, $data = null )
5       {
6           parent::__construct();
7
8           if ( !OW::getUser()->isAuthorized('photo', 'upload') )
9           {
10              $this->setVisible(false);
```

```
11
12              return;
13          }
14
15          $userId = OW::getUser()->getId();
16          $document = OW::getDocument();
17
18          PHOTO_BOL_PhotoTemporaryService::getInstance()->
19          deleteUserTemporaryPhotos($userId);
20
21          $plugin = OW::getPluginManager()->getPlugin('photo');
22
23          $document->addStyleSheet($plugin->getStaticCssUrl() . 'photo_upload.css');
24          $document->addScript($plugin->getStaticJsUrl() . 'codemirror.min.js');
25          $document->addScript($plugin->getStaticJsUrl() . 'upload.js');
26
27          $document->addScriptDeclarationBeforeIncludes(
28          UTIL_JsGenerator::composeJsString(';window.ajaxPhotoUploadParams =
29          Object.freeze({$params});',
30          array(
31         'params' => array(
32         'actionUrl' => OW::getRouter()->urlForRoute('photo.ajax_upload'),
33         'maxFileSize' => PHOTO_BOL_PhotoService::getInstance()->
34          getMaxUploadFileSize(),
35         'deleteAction' => OW::getRouter()->urlForRoute('photo.ajax_upload_delete')
36                    )
37                )
38              )
39          );
40          $document->addOnloadScript(';window.ajaxPhotoUploader.init();');
41
42          $album=PHOTO_BOL_PhotoAlbumService::getInstance()->findAlbumById($albumId)
43
44          $form = new COCREATION_CLASS_UploadPhotosForm('user',$userId, $albumId,
45          $album->name, $album->description , $url, $data);
46          $this->addForm($form);
47          $this->assign('extendInputs', $form->getExtendedElements());
48          $this->assign('albumId', $albumId);
49          $this->assign('userId', $userId);
50
```

```
51
52          $language = OW::getLanguage();
53          $language->addKeyForJs('photo', 'not_all_photos_uploaded');
54          $language->addKeyForJs('photo', 'size_limit');
55          $language->addKeyForJs('photo', 'type_error');
56          $language->addKeyForJs('photo', 'dnd_support');
57          $language->addKeyForJs('photo', 'dnd_not_support');
58          $language->addKeyForJs('photo', 'drop_here');
59          $language->addKeyForJs('photo', 'please_wait');
60          $language->addKeyForJs('photo', 'create_album');
61          $language->addKeyForJs('photo', 'album_name');
62          $language->addKeyForJs('photo', 'album_desc');
63          $language->addKeyForJs('photo', 'describe_photo');
64          $language->addKeyForJs('photo', 'photo_upload_error');
65      }
66
67
68
69  }
```

```
1   class COCREATION_CLASS_UploadPhotosForm extends PHOTO_CLASS_AbstractPhotoForm
2
3
4
5   {
6       const FORM_NAME = 'ajax-upload';
7       const ELEMENT_ALBUM = 'album';
8       const ELEMENT_ALBUM_NAME = 'album-name';
9       const ELEMENT_DESCRIPTION = 'description';
10      const ELEMENT_ID = 'album-id';
11
12      public function __construct( $entityType, $entityId, $albumId = null,
13      $albumName = null, $albumDescription = null, $url = null, $data = null )
14      {
15          parent::__construct(self::FORM_NAME);
16
17          $this->setAjax(true);
18          $this->setAjaxResetOnSuccess(false);
19          $this->setAction(OW::getRouter()->urlForRoute('cocreation.submit'));
20          $this->bindJsFunction(self::BIND_SUCCESS,
```

```
21          UTIL_JsGenerator::composeJsString('function( data )
22          {
23              if ( data )
24              {
25                  if ( !data.result )
26                  {
27                      if ( data.msg )
28                      {
29                          OW.error(data.msg);
30                      }
31                      else
32                      {
33                          OW.getLanguageText("photo", "photo_upload_error");
34                      }
35                  }
36                  else
37                  {
38                      var url = {$url};
39
40                      if ( url )
41                      {
42                          window.location.href = url;
43                      }
44                      else if ( data.url )
45                      {
46                          window.location.href = data.url;
47                      }
48                  }
49              }
50              else
51              {
52                  OW.error("Server error");
53              }
54          }', array(
55              'url' => $url
56          )));
57
58          $language = OW::getLanguage();
59
60          $albumField = new TextField(self::ELEMENT_ALBUM);
```

```
61
62          $albumField->addAttribute(FormElement::ATTR_CLASS,
63          'ow_dropdown_btn ow_inputready ow_cursor_pointer');
64          $albumField->addAttribute('autocomplete', 'off');
65          $albumField->addAttribute(FormElement::ATTR_READONLY);
66          $albumField->setValue($albumName);
67
68          $albumIdField=new TextField(self::ELEMENT_ID);
69          $albumIdField->setValue($albumId);
70          $this->addElement($albumIdField);
71
72          $albumNameField = new TextField(self::ELEMENT_ALBUM_NAME);
73
74          $albumNameField->addValidator(new PHOTO_CLASS_AlbumNameValidator(false));
75          $albumNameField->addAttribute('placeholder', $language->text('photo',
76          'album_name'));
77          $this->addElement($albumNameField);
78
79          $desc = new Textarea(self::ELEMENT_DESCRIPTION);
80          $desc->addAttribute('placeholder', $language->text('photo', 'album_desc'))
81          $desc->setValue(!empty($albumDescription) ? $albumDescription : null);
82          $this->addElement($desc);
83
84
85
86
87          $this->addElement($albumField);
88
89          $submit = new Submit('submit');
90          $submit->addAttribute('class', 'ow_ic_submit ow_positive');
91          $this->addElement($submit);
92
93          $this->triggerReady(array(
94              'entityType' => $entityType,
95              'entityId' => $entityId,
96              'albumId' => $albumId,
97              'albumName' => $albumName,
98              'albumDescription' => $albumDescription,
99              'url' => $url,
100             'data' => $data
```

```
101             ));
102         }
103
104         public function getOwnElements()
105         {
106             return array(
107                 self::ELEMENT_ALBUM,
108                 self::ELEMENT_ALBUM_NAME,
109                 self::ELEMENT_DESCRIPTION
110             );
111         }
112
113
114
115
116     }
```