



# Liwe

LiWE - The Lightweight Web Environment



## Preparazione

Requisiti:

- Hg
- mysql client/server
- python mySQLdb

Dipendenze contenute nel tar.gz:

- os3pylib → <https://os3pylib.googlecode.com/hg/> os3pylib
- LIWE → <https://liwe.googlecode.com/hg/> liwe
- pyhp → <https://pyhp.googlecode.com/hg/> pyhp

Convenzioni:

- os3pylib → /home/username/src ( create sm on python path )
- pyhp folder ( tar.gz ) → /home/username/src ( create sm on python path )
- LIWE folder ( tar.gz ) → /home/username/src
- app folder → /home/web/appname

Struttura LIWE:

- os3jslib → gestione ajax
- os3wwl → gestione widget js
- tools → utility per creare una nuova app
- module → plug-in ( news, comment, staticpage, stars, user, tags ecc...)
- os3phplib → interfaccia con php
- pyhp → interfaccia con pyhp



# Liwe

LiWE - The Lightweight Web Environment



## Start

per iniziare una nuova web-app con LIWE i tools creano la struttura di base

```
>~/src/liwe/tools/liwe_start_website -p ~/web -n <appname> -l ~/src/liwe ( -c opzionale per windows )
```

## Anatomia della app

log

→ File di log

web

→ cache

→ modules link simbolici a moduli della liwe e moduli nuovi della app

→ os3jslib link simbolico

→ os3wwl link simbolico

→ os3 link simbolico

→ pyhpwsgi link simbolico

→ liwe link simbolico alla libreria pyhp

→ pages pagine del sito web ( index.html, admin.html )

→ site override dei moduli standard o del sito

→ liwe.cfg file di configurazione della app

→ wsgi.py

work file SQL per creare le tabelle del db

n.b.: os3, pyhpwsgi pyhp possono essere linkati simbolicamente nel python path

## Antomia di un modulo

Directory

- css → css del modulo
  - gfx → immagini del modulo
  - js → file js del modulo
  - templates → parti di codice HTML che saranno sostituite tramite richieste server(pyhp)/client(ajax) side
- work → file SQL per creare le tabelle del db

Files

- <module\_name>\_class.py → definizione della classe python operazioni CRUD e SELECT
- <module\_name>.py → init del modulo e funzioni python usabili nelle pagine html tramite i comandi fab
- ajax.py → interfaccia python con javascript

FIXME: Fabio controlla questa parte che sono sicuro di aver scritto delle boiate

## Modalità

Le LIWE permettono di sviluppare una web-app in tre modi differenti:

- mod\_pyhp → uso di apache
- estensione di mod\_wsgi → da usare per app che girano sul server
- ./wsgi → utilizzo stand-alone per develop/testing app



# Liwe

LiWE - The Lightweight Web Environment



## Configurazione iniziale

rendere eseguibile il file wsgi.py appena creato nella directory <progetto>/web

```
>chmod 775 wsgi.py
```

creazione di un database in mysql e creazione tabelle di default

```
>mysql -u root -p
```

```
>mysql>create database [<dbname>];
```

```
>quit;
```

```
>mysql -u root <dbname> < ~/src/liwe/modules/system/work/system.sql
```

```
>mysql -u root <dbname> < ~/src/liwe/modules/user/work/user.sql
```

```
>mysql -u root <dbname> < ~/src/liwe/modules/tags/work/tags.sql
```

configurare il file *liwe.cfg* per impostare la connessione la database

```
# =====
```

```
# Database Definitions
```

```
# =====
```

```
DB_KIND=mysql
```

```
DB_HOST=localhost
```

```
DB_USER=root
```

```
DB_NAME=<dbname>
```

```
DB_PASSWORD=
```

in modalita develop/testing lanciare il comando

```
>./wsgi.py
```

aprire un browser all' indirizzo

<http://localhost:8000/index.pyhp>

Il comando wsgi.py deve essere riavviato tutte le volte che modifichiamo qualsiasi .py



# Liwe

LiWE - The Lightweight Web Environment



## BACK-END FRONT-END

### Uso dei comandi fab

I comandi fab sono usati nelle pagine html vengono invocati come segue:

```
<?fab <module_name> <function_name> <p1> ... <p10> ?>
```

Per ogni modulo che vogliamo usare all'interno di una pagina html bisogna eseguire la sua init dopo il tag `<body>` con il comando seguente

```
<?fab <nomemodulo> init ?>
```

Nelle pagine di back-end la init del modulo deve essere invoca con il parametro `admin="1"`

### Il Nostro BACK-END

OS3 ha sviluppato un back-end per la gestione di tutti i moduli esistenti e futuri la pagina html di partenza è `admin.html` ed è così strutturata

```
<body>
  <?fab system init admin="1" ?>
  <?fab user init admin="1" ?>
  <?fab tags init admin="1" ?>
  <?fab site init admin="1" ?>
</body>
```

*system* inizializza la struttura del back-end, lista i moduli, controlla le permission dell'utente

*user* effettua tutte le operazioni sull'utente

*tags* come altri moduli i tags sono strutturati in modo da poter essere associati ad altri moduli ad esempio una news può contenere più tags così come un prodotto, tutti i tag creati nel sito sono presenti in unica struttura dati che di volta in volta viene associata ad un modulo.

Aggiungiamo il modulo News al nostro environment.

*News* dipende da *Static\_Page* che a loro volta dipende da *Page Manager*, *Admin\_Manager*, *Stats* e *Media\_Manager* bisogna creare un link simbolico per tutti questi moduli, creare le tabelle usando gli script sql contenuti nella directory work del modulo e inizializzare correttamente la pagina `admin.html`

```
<body>
  <?fab system init admin="1" ?>
  <?fab user init admin="1" ?>
  <?fab tags init admin="1" ?>
  <?fab staticpage init admin="1" ?>
  <?fab admin_manager init admin="1" ?>
  <?fab media_manager init admin="1" ?>
  <?fab stats init admin="1" ?>
  <?fab news init admin="1" ?>
  <?fab site init admin="1" ?>
</body>
```



# Liwe

LiWE - The Lightweight Web Environment



n.b.: E' importante che la init di site sia messa per ultima, cosi facendo il modulo site aggiungerà per ultimo i file .js e .css nella head del foglio html.

accendiamo al Back-End

<http://localhost:8000/index.pyhp?/page/admin/>

user = root

password = liwe

## User

user tags staticpage news comment

Lista Utenti | Creazione utente |

Risultati: 1 - Pagina: 1 di 1

<input type="checkbox"/>	UID	Login / e-mail	Data creazione	Abilitato	Last log				
<input type="checkbox"/>	1	root info@example.com	15-06-2011	SI	16-06-2011				

Risultati: 1 - Pagina: 1 di 1

Benvenuto: root Log out

## Tags

user tags staticpage news comment

Aggiungi tag | Lista tag | Search | Add Meta | List Meta |

ID Object:

tags:

Tags disponibili

Benvenuto: root Log out





user



tags



staticpage



news



comment

Nuova pagina | Lista Pagine |

Nome pagina:



Titolo:

HTML:



M



HTML

Visibile: ☒

Salva

Benvenuto: root

 Log out



# Liwe

LiWE - The Lightweight Web Environment



## Il Vostro FRONT-END

Passo 1 → Visualizziamo le News

Modifichiamo la pagina html per poter usare i moduli

```
<body>
  <?fab user init ?>
  <?fab tags init ?>
  <?fab staticpage init ?>
  <?fab news init ?>
  <?fab site init ?>
  ...
  ...
  ...
</body>
```

il modulo news ha un comando fab definito come funzione dentro *news.py* che crea l'html per la lista delle news inserite usiamo questo comando per vedere le news nel block main della pagina *index.html*

```
<div id = "block_main">
    Main content
    <?fab news list ?>
</div>
```

Passo 2 → Aggiungiamo i commenti

I commenti hanno una dipendenza con il Page\_Manager che è già stato importato ci basterà quindi linkare il modulo comment e eseguire lo script SQL sotto il folder Work. Aggiungiamo il comando *<?fab ?>* di inizializzazione nella pagina di admin e nell'index.

Il modulo News quando crea la lista delle news inserisce una chiamata ad una funzione ajax:

```
news.show ( <id_news>, <dest_div> );
```

La funzione news.show è associabile ad una chiamata di call-back, possiamo usare questa caratteristica per estendere il risultato e aggiungere il box dei commenti.

Nel folder modules c'è il modulo dedicato ad questo tipo di operazioni: site. Il modulo site è come tutti gli altri moduli è strutturato come segue: CSS e GFX contengono fogli di stile e file di tipo grafico, il file site.py contiene l'inizializzazione del modulo che aggiunge nella *<head>* i riferimenti ai css e ai javascript, che sono contenuti nel folder js, per il momento non sono presenti i file per la classe e per le chiamate ajax.



# Liwe

LiWE - The Lightweight Web Environment



definiamo la call-back dentro la init di site

```
//site.js
liwe.AJAX.url = "/ajax.pyhp";
var site = {};
site.init = function ()
{
    console.debug ( "SITE INIT" );
    liwe.dom.tableize ();
    liwe.AJAX.cbacks [ 'serialize' ] = function ( s )
    {
        if ( typeof ( s ) != 'string' ) return s;
        return s.htmlEntities ();
    };
    liwe.history.init ();
    news.cbacks [ 'show' ] = site._fmt_news;
};
```

Inseriamo la funzione per renderizzare la news con il box dei commenti

```
//site.js
site._fmt_news = function ( data, id_news, dest_div, permalink, pos )
{
    console.debug ( "qui il nostro prossimo codice" );
};
```



# Liwe

LiWE - The Lightweight Web Environment



tra le variabili passate alla call-back la variabile `data` contiene una serie di informazioni tra cui tutte le informazioni legate alla news selezionata

_img	""
abstract	"bla bla bla "
categ	"test"
created	"2011-06-15 16:52:25"
descr	"ri bla bla bla bla "
pub_date	"2011-06-15 16:50:00"
title	"prova 01"
views	25

Prima di scrivere del codice dentro la funzione `_fmt_news` bisogna dare qualche premessa sulle funzioni javascript che sono state implementate.

*funzione get*

```
var dct = [];  
var res = dct.get ( 'test', 'notest' );  
console.debug ( res );  
  
>>>notest
```

la `get` javascript funziona come la `get` di python quindi se `dct` non contiene la chiave `'test'` `res` è impostato a `'notest'`

*funzione \$*

```
$( <dest_div>, <out> );
```

Questa riga di codice in particolare ricerca l'elemento con id uguale a `dest_div` e sostituisce il suo contenuto con `out`. La funzione `$` non è da confondere con `jquery`.



# Liwe

LiWE - The Lightweight Web Environment



funzione *formatDict*

```
var string_to_format = "<div class='%(class_name)s'>%(testo)s</div>";
var dct = [];
dct [ 'class_name' ] = 'test';
dct [ 'testo' ] = 'testo';
var res = String.formatDict ( string_to_format, dct );
console.debug ( res );

>>><div class='test'>testo</div>
```

corrisponde alla funzione di format di python in questo caso i valori del “dizionario” di dct sostituiranno i segnaposto presenti nella stringa *string\_to\_format*.

*n.b.: le chiavi in esubero nel dizionario non verranno considerate*

Strettamente connessi a questa funzione sono i templates delle liwe. Ogni modulo ha un folder templates che, normalmente, contiene un file chiamato *<nome\_modulo>.txt* il contenuto di questo file è un dictionary di python, ad esempio se guardiamo nel folder templates del modulo news troviamo tutti i template che vengono usati per creare la struttura delle news: show, list, ecc...

```
#news.txt
{ "NEWS_SHOW" : ""
<div class="row">
  <div class="box_img">
    <a href="#">%( _img)s</a>
  </div>
  <span class="title">%(title)s</span>
  <div class="abstract">
    %(abstract)s
  </div>
  <div class="descr">
    %(descr)s
  </div>
  <div class="categ">
    Categoria: <a %( _HREF_LIST)s %( _ACTION_LIST)s>%(categ)s</a>
  </div>
  <div class="info">
    News postata: %(created)s - Visualizzazioni: %(views)s
  </div>
  <div class="tags">%( _html_tags)s</div>
  <div id="navi_news"></div>
</div>"" }
```

si nota la corrispondenza dei segnaposto con i valori presenti nella variabile data passata alla call-back.

I template di un modulo possono essere chiamati in un punto qualunque della nostra applicazione, ma



# Liwe

LiWE - The Lightweight Web Environment



vengono usati soprattutto da javascript quindi per ottenere il valore di una chiave del template si usa la funzione seguente

```
news.templates [ 'NEWS_SHOW' ]
```

i template possono essere sovrascritti ricostruendo la struttura dei folder a partire dal modules dentro il folder site contenuto dentro web. Se volessimo ricostruire il template appena analizzato possiamo quindi creare sotto site la seguente struttura : *modules/news/templates/news.txt* e inserire il nostro nuovo template

```
//site/modules/news/templates/news.txt
{"NEWS_SHOW" : ""
<div class="row">
  <div class="box_img">
    <a href="#">%(_img)s</a>
  </div>
  <span class="title">%(title)s</span>
  <div class="abstract">
    %(abstract)s
  </div>
  <div class="descr">
    %(descr)s
  </div>
  <div class="info">
    News postata: %(created)s - Visualizzazioni: %(views)s
  </div>
  <div class="tags">%(_html_tags)s</div>
  <div id="navi_news"></div>
</div>
%(comment_box)s""
}
```



# Liwe

LiWE - The Lightweight Web Environment



possiamo tornare ora alla funzione `_fmt_news` caricare il template delle news aggiungere a data i commenti e sostituire il segnaposto `comment_box`

```
//site.js
site._fmt_news = function ( data, id_news, dest_div, permalink, pos )
{
    if ( ! data ) data = [];
    dest_div = "block_main";

    tpl = news.templates [ 'NEWS_SHOW' ];
    data [ 'comment_box' ] = comment.box ( data [ 'id' ], 'news', '', true );
    var out = String.formatDict ( tpl , data );
    $( dest_div, out );
    comment.list ( data [ 'id' ], 'news', '', comment.render );
};
```

```
tpl = news.templates [ 'NEWS_SHOW' ];
```

il nostro nuovo template

```
data [ 'comment_box' ] = comment.box ( data [ 'id' ], 'news', '', true );
```

la function `comment.box` crea la form per aggiungere i commenti che viene inserite come valore di `'comment_box'` dentro `data`

```
var out = String.formatDict ( tpl , data );
```

ora `out` contiene il nostro template con i segnaposto sostituiti dai valori delle chiavi contenuti in `data`

```
$( dest_div, out );
```

carichiamo dentro il div `'block_main'` `out`

```
comment.list ( data [ 'id' ], 'news', '', comment.render );
```

questa funzione carica la lista dei commenti dentro a un div che è presente nel DOM grazie alla funzione `comment.box`



# Liwe

LiWE - The Lightweight Web Environment



## Test

possiamo notare che al nostro sito mancano:

- un link alla home page
- se non effettui il login i commenti non vengono caricati

Il primo bug si risolve inserendo un `<a href="http://localhost:8000/index.pyhp">Home</a>` nella nostra `index.html` dato che ci siamo aggiungiamo anche il link alla pagina di admin.

```
<!-- index.html-->
<div id="leftbar">
  <ul>
    <li><a href="http://localhost:8000/index.pyhp">Home</a></li>
    <li><a href="http://localhost:8000/index.pyhp?page/admin/">Admin</a></li>
  </ul>
</div>
```

Il secondo bug ci permette di introdurre alcuni concetti.

aggiungiamo alla pagina `index.html` un comando fab per visualizzare la login box

```
<!-- index.html-->
<div id="rightbar">
  <?fab user show_login ?>
</div>
```

se non abbiamo effettuato il login appare un link [Accedi/Registrati](#) che non funziona ancora se lo clicchiamo la console di firebase riporta un errore legato a lighbox non presente.

Lighbox è un .js delle liwe che deve essere aggiunto durante il caricamento del sito per far questo dobbiamo modificare il file `modules/site/site.py`

```
# modules/site/site.py
def init ( l, admin = 0 ):
    from liwe import parts
    parts.add ( l.layman, [ "ext" ] )
...
```

questa modifica aggiunge alcuni .js nella head del nostro documento html tra cui anche `lighbox.js`



# Liwe

LiWE - The Lightweight Web Environment



se ora clicchiamo su [Accedi/Registrati](#) si apre la form di login come quella dell'amministrazione

Accesso utente

Login:

Password:

Accedi

Hai dimenticato la password?

Tua e-mail:

Richiedi

Registrazione nuovo utente

La registrazione a questo sito è totalmente gratuita!

Vuoi registrarti al sito? [Clicca qui!](#)

dopo aver effettuato l'accesso si noterà che il link [Accedi/Registrati](#) non è scomparso, perchè il DOM deve essere ricaricato per permettere alla funzione fab di modificare la scritta con l'id dell utente che si è identificato.



# Liwe

LiWE - The Lightweight Web Environment



Modifichiamo il file modules/site/js/site.js

```
//modules/site/js/site.js

site.init = function ()
{
    console.debug ( "SITE INIT" );
    liwe.dom.tableize ();

    liwe.AJAX.cbacks [ 'serialize' ] = function ( s )
    {
        if ( typeof ( s ) != 'string' ) return s;
        return s.htmlEntities ();
    };
    liwe.history.init ();
    news.cbacks [ 'show' ] = site._fmt_news;
    user.events [ 'login' ] = site._refresh;
    user.events [ 'logout' ] = site._refresh;
};
...
...
site._refresh = function ()
{
    document.location.reload ( true );
};
```

aggiungiamo alla chiamata di login e di logout del modulo user una function per ricaricare la pagina

Ora funziona tutto manca solo il link per effettuare il log-out. Per far questo bisogna aggiungere dentro al div della login-box un link per effettuare il logout, questo link deve apparire solo se abbiamo effettuato l'accesso.



# Liwe

LiWE - The Lightweight Web Environment



Per prima cosa abbiamo bisogno di mettere nei templates il nostro nuovo link di log-out, aggiungiamo il folder modules/site/templates e creiamo il file site.txt

```
#modules/site/templates/site.txt
{
    'EX-loguot' : ""<a href="javascript: user.logout ()">logout</a>""
}
```

i templates però devono essere caricati per far questo abbiamo bisogno di altri due file:

```
#modules/site/site_class.py

#!/usr/bin/env python

import sys
from os3.utils.utils import md5crypt
from liwe.module import LiweModule

class Site ( LiweModule ):
    def __init__ ( self, liwe ):
        super ( Site, self ).__init__ ( liwe, "site" )
```

```
#modules/site/ajax.py

#!/usr/bin/env python

from site_class import Site

def get_templates ( liwe ):
    s = Site ( liwe )
    return { "templates" : s.get_templates () }
```

Il primo file ereditando LiweModule si occupa di caricare i templates, il secondo è il ponte al primo file per la chiamata ajax che viene fatta da un file di tipo .js nel nostro caso sarà site.js



# Liwe

LiWE - The Lightweight Web Environment



Modifichiamo il file modules/site/js/site.js

```
//modules/site/js/site.js

site.init = function ()
{
    site.load_templates ( function ()
    {
        console.debug ( "SITE INIT" );
        liwe.dom.tableize ();

        liwe.AJAX.cbacks [ 'serialize' ] = function ( s )
        {
            if ( typeof ( s ) != 'string' ) return s;
            return s.htmlEntities ();
        };
        liwe.history.init ();
        news.cbacks [ 'show' ] = site._fmt_news;
        user.events [ 'login' ] = site._refresh;
        user.events [ 'logout' ] = site._refresh;
    } );
};
```

Analizziamo il codice:

```
site.load_templates ( function ()
{
...
...
    site._chek_user();
...
...
});
```

con la load\_template carichiamo i templates di site nel nostro caso *EX-logout* la funzione non eseguirà altro codice finche non finisce la load templates. Se proviamo da console ad eseguire site.templates [ 'EX-logout' ] il risultato è il valore del nostro template.



# Liwe

LiWE - The Lightweight Web Environment



prima di passare alla fase seguente, è necessario controllare se nel folder del modulo user è presente qualche funzione, che mi permette di sapere se il nostro utente è già collegato oppure no.

Nel file modules/user/ajax.py è presente una funzione get\_info, effettiamo il login e da console scriviamo:

```
>>> user.ajax ( { action: "user.ajax.get_info" } );
```

la response della post contiene un oggetto js

```
{"uid": 1, "created": "2011-06-15 15:24:05", "err_descr": "", "enabled": 1, "login": "root", "last_log": "2011-06-17 14:55:03", "email": "info@example.com", "err_code": 0};
```

se facciamo la stessa cosa non connessi, otteniamo solo err\_code e err\_descr. Possiamo quindi creare una funzione che controlla se l'utente è loggato oppure no e se l'esito è positivo, possiamo aggiungere il nostro template nell'html



# Liwe

LiWE - The Lightweight Web Environment



Il file modules/site/site.js lo possiamo modificare come segue

```
//modules/site/js/site.js

site.init = function ()
{
    site.load_templates ( function ()
    {
        ...
        ...
        site._chek_user();
    } );
};
...
...
site._chek_user = function ()
{
    user.ajax ( { action: "user.ajax.get_info" }, function ( res )
    {
        if ( res.get ( 'uid', 0 ) != 0 )
        {
            var new_rb = $( "rightbar" ).innerHTML;
            var log_outbox = site.templates [ 'EX-loguot' ];
            new_rb += log_outbox;
            $( "rightbar", new_rb)
        };
    });
};
```