nuova release e versione standard del framework GlisWeb

Questo file serve per orientarsi all'interno del framework GlisWeb; si tratta di una sorta di guida generale, da tenere sotto mano mentre si studia il framework.

descrizione dei file

In questa sezione tutti i file e le cartelle del framework sono riportati in ordine logico, per dare un'idea dell'insieme. Ogni file contiene poi i commenti dettagliati sul proprio funzionamento.

/.gitignore

Questo file (il cui contenuto cambia fra sviluppo del framework e sviluppo dei progetti) impedisce che vengano caricati nel repository Git del framework (o del progetto) file inutili o potenzialmente sensibili.

TODO nella vecchia versione il file .gitignore per i deploy è in /__usr/__deploy/__git/.gitignore e viene gestito tramite /__src/__sh/__gw.upgrade.sh; valutare se è il posto giusto o se va spostato ad es. in /__usr/__examples/__config/__git/.

/.htaccess

Questo file costituisce il punto di ingresso del framework, tutte le richieste in entrata vengono processate da questo file che poi le rimanda al file PHP opportuno. In pratica il file .htaccess costituisce il motore di routing principale del framework.

/composer.json e /composer.lock

Il file composer.json include le dipendenze del framework che vengono gestite tramite composer; la cartella di installazione per le librerie esterne è /__src/__lib/__ext/

strategia	libreria	versione	note
require	phpoffice/phpword	*	
require	phpoffice/phpspreadsheet	*	
require	phpoffice/phppresentation	*	
require	html2text/html2text	*	
require	phpmailer/phpmailer	6.*	
require	tecnickcom/tcpdf	*	
require	twig/twig	2.*	
require	predis/predis	*	
require	codeception/codeception	*	

strategia	libreria	versione	note
require	codeception/module-phpbrowser	*	
require	codeception/module-asserts	*	
suggest	twig/extra-bundle	*	

/__etc/__current.release e /__etc/__current.version

Il framework viene versionato con due diverse numerazioni, le release che seguono la classica notazione a tre stage (major.minor.bugfix) e le versioni che sono numerate progressivamente con una timestamp (ad es. 20240502225937). La ragione di questa distinzione è che le versioni vengono incrementate quotidianamente, mentre le release di rado, solo quando numerose versioni si sono accumulate.

L'aggiornamento della versione è fatto automaticamente tramite un git hook (/.git/hooks/pre-commit) ad ogni commit sul repository di sviluppo del framework:

```
#!/bin/bash

BRANCH=`git rev-parse --abbrev-ref HEAD`
VERS=$(date '+%Y%m%d%H%M%S')
GITNAME=`basename $(git remote get-url origin)`

echo "repository: "$GITNAME

if [ -n "$( echo $GITNAME | grep 'glisweb' )" ]; then

echo "branch: "$BRANCH
echo "version: "$VERS

    echo $VERS > _etc/_current.version
    git add _etc/_current.version

    echo "aggiornamento della versione effettuato con successo"

fi
```

La versione invece viene modificata a mano quando si crea una nuova release branch.

/__etc/__current.version

Vedi /_etc/_current.release.

/__etc/__common/__lorem.conf

Questo file contiene il testo di prova del framework; si è optato per il classico Lorem Ipsum (https://lipsum.com/) dal momento che la maggior parte dei grafici è già familiare con questo testo. Nel file /_src/_config/_420.pages.php viene processato il comando di una lettera "m" per inserire il Lorem Ipsum come testo della pagina.

/__etc/*dictionaries/*.-.conf

Questi file contengono i dizionari per la traduzione automatica dei microcontenuti. La traduzione automatica dei microcontenuti viene gestita tramite la macro Twig tr() dichiarata in /_src/_twig/_lib/_translation.twig. Un utilizzo tipico di tr() considerando che la variabile ietf contenga la lingua corrente e che la traduzione che si desidera è presente in $ct[‘tr’][‘generic’] è il seguente:

```
{% import '_lib/_translation.twig' as trn %}
{{ trn.tr({ 'w': '<chiave>', 'l': ietf, 'v': tr.generic }) }}
```

I file dei dizionari vengono importati in $cf nel file /_src/_config/_090.translation.php e $cf[’tr] viene collegato a $ct[’tr] in /_src/_config/_095.translation.php.

/__etc/__dictionaries/__generic.cs-CZ.conf

Dizionario generico per la lingua ceca.

/__etc/__dictionaries/__generic.de-DE.conf

Dizionario generico per la lingua tedesca.

/__etc/__dictionaries/__generic.en-GB.conf

Dizionario generico per la lingua inglese britannica.

/__etc/__dictionaries/__generic.en-US.conf

Dizionario generico per la lingua inglese statunitense.

/__etc/__dictionaries/__generic.es-ES.conf

Dizionario generico per la lingua spagnola.

/__etc/__dictionaries/__generic.fr-FR.conf

Dizionario generico per la lingua francese.

/__etc/__dictionaries/__generic.hr-HR.conf

Dizionario generico per la lingua croata.

/__etc/__dictionaries/__generic.hu-HU.conf

Dizionario generico per la lingua ungherese.

/__etc/__dictionaries/__generic.it-IT.conf

Dizionario generico per la lingua italiana.

/__etc/__dictionaries/__generic.ja-JP.conf

Dizionario generico per la lingua giapponese.

/__etc/__dictionaries/__generic.pl-PL.conf

Dizionario generico per la lingua polacca.

/__etc/__dictionaries/__generic.pt-BR.conf

Dizionario generico per la lingua portoghese brasiliana.

/__etc/__dictionaries/__generic.pt-PT.conf

Dizionario generico per la lingua portoghese.

/__etc/__dictionaries/__generic.ro-RO.conf

Dizionario generico per la lingua rumena.

/__etc/__dictionaries/__generic.ru-RU.conf

Dizionario generico per la lingua russa.

/__etc/__dictionaries/__generic.sv-SE.conf

Dizionario generico per la lingua svedese.

/__etc/__doxygen/__doxygen.conf

Questo è il file di configurazione utilizzato per compilare la documentazione del framework tramite Doxygen (https://www.doxygen.nl/). La compilazione della documentazione viene effettuata tramite lo script /__src/__sh/__doxygen.build.sh e i documenti compilati vengono salvati in /__usr/__docs/__html/ per la versione HTML e in /__usr/__docs/__pdf/ per la versione PDF.

La documentazione viene generata a partire dai commenti al codice e dai file dox presenti in /__usr/__docs/__dox, ed è disponibile via browser utilizzando il percorso /docs/index.html per l'HTML e /docs/pdf per il PDF.

TODO se è possibile, le varie opzioni di questo file andrebbero commentate una per una, magari facendo qui una tabella con il significato di ogni opzione.

/__etc/__robots/__deny.txt e /__etc/__robots/__robots.txt

Questi sono i file robots che vengono serviti richiedendo l'URL /robots.txt; nel file /.htaccess è presente un set di regole che in base al valore della variabile d'ambiente %{ENV:STATUS} eroga il file corretto (deny per DEV e TEST, robots per PROD). La variabile di ambiente viene settata nel file di configurazione dell'host di Apache per le varie configurazioni del sito:

```
<VirtualHost *:80>

    ...

    SetEnv STATUS DEV

    ...

</VirtualHost>
<IfModule mod_ssl.c>
<VirtualHost *:443>

    ...

    SetEnv STATUS DEV

    ...

</VirtualHost>
</IfModule>
```

Nella configurazione di PROD viene vietata l'indicizzazione di tutto il ramo /admin in quanto contiene le pagine del CMS, che non devono per ovvi motivi apparire nei risultati di ricerca.

/__etc/__robots/__robots.txt

Vedi /_etc/_robots/_deny.txt.

/__etc/__security/__banned.words.conf

Questo file contiene una lista di parole vietate negli URL, che viene gestita da /_src/_inc/_macro/_security.php. Si tratta di un filtro un po' grezzo ma funzionale, che blocca molti tipi di attacchi basati su URL.

/__etc/__security/__common.passwords.conf

Questo file contiene un piccolo dizionario di password vietate per l'utente root. Il controllo viene effettuato in /_src/_api/_status/_framework.php nella sezione sicurezza.

/__src/__config.php

Questo file costituisce il kernel del framework; è ampiamente documentato quindi si rimanda al sorgente per gli approfondimenti, in breve comunque il suo compito è quello di includere tutti i componenti del framework per renderli disponibili al successivo codice sorgente. Qualsiasi file PHP nel quale si desidera utilizzare il framework deve iniziare o comunque contenere la riga:

```
require '<percorso>_config.php';
```

Laddove è il percorso necessario a raggiungere il file /_src/_config.php.

/__src/__api/__bookmarks.php

Questa API si occupa di gestire la memorizzazione di elementi nello spazio di lavoro della sessione. Il meccanismo del "pin" di elementi in sessione è molto comodo per portarsi dietro delle informazioni da una maschera all'altra, dove previsto dalla maschera stessa. Per ulteriori informazioni su questo meccanismo si veda anche la documentazione dei file /_src/_config/_710.session.php e /_src/_config/_715.session.php, e dei file /_src/_config/_770.bookmarks.php e /_src/_config/_775.bookmarks.php.

/__src/__api/__cron.php

Questa API si occupa di eseguire i task periodici; solitamente viene richiamata dal cron di sistema tramite un file appositamente creato nella cartella /etc/cron.d/. La creazione di questo file può essere eseguita in maniera semi automatica tramite lo script /_src/_sh/_crontab.install.sh. La gestione dei task periodici è piuttosto complessa, per ulteriori dettagli si veda la documentazione del file /_src/_api/_cron.php.

/__src/__api/__download.php

Questo file si occupa di erogare i download, verificando che l'utente sia autorizzato a scaricare il file che sta richiedendo; riceve le richieste dalle regole del file .htaccess e fatti i dovuti controlli restituisce il file richiesto. Agisce in pratica come un guardiano della cartella /var che contiene ordinati in sottocartelle i file caricati tramite il CMS. Si noti che alcune sotto cartelle di /var (come /var/log) sono protette e l'accesso è impedito da regole apposite all'inizio di /.htaccess.

/__src/__api/__job.php

Questa API consente l'esecuzione dei job in foreground; per ulteriori informazioni si veda la documentazione del file stesso oltre a quella del file /_src/_api/_cron.php.

/__src/__api/__pages.php

Questo file ha lo scopo di renderizzare e erogare le pagine. Svolge numerose funzioni ed è ampiamente documentato, quindi si rimanda al sorgente per i dettagli. In sintesi, riceve le richieste di pagina in base alle regole del file /.htaccess e le soddisfa tramite le informazioni in suo possesso.

/__src/__api/__user.php

Questa API consente il login dell'utente, è utilizzata per le integrazioni e per il dialogo con app e altri sistemi esterni. Tramite il meccanismo di login è possibile ottenere un'API key temporanea per fare più rapidamente le chiamate successive. Per ulteriori dettagli sul meccanismo di login tramite API key si vedano i commenti a questo file e ai file dev/__src/__config/__210.auth.php e dev/__src/__config/__220.auth.php.

/__src/__api/__job/__test.job.php

Questo è un job di test.

/__src/__api/__report/__cookie.php

Questo report restituisce l'elenco di tutti i cookie presenti nel browser per il dominio corrente indicando se sono gestiti o meno dal framework, il loro scopo e altre informazioni utili.

/__src/__api/__report/__import.php

Questo report innesca l'importazione dei file presenti in /var/spool/import/todo/*/ e /var/spool/import, e restituisce un report dettagliato delle operazioni svolte. Normalmente l'importazione dei file è svolta dall'API cron, ma questo report può essere fondamentale per lo sviluppo, il test e il debug dei nuovi tracciati di importazione.

/__src/__api/__status/__cf.php

Questa API di stato restituisce il contenuto, navigabile, dell'array $cf. Tutti i dati sensibili sono censurati tramite la funzione core array2censored() per evitare problemi di sicurezza. Tramite una regola di /.htaccess l'accesso a questa API è possibile tramite l'URL speciale /cf.

/__src/__api/__status/__framework.php

Questa API restituisce un report di auto diagnostica del framework, indicando lo stato corrente della piattaforma ed eventuali errori o warning. Si tratta di un file piuttosto complesso, al cui sorgente si rimanda per approfondimenti. Una regola di /.htaccess rende disponibile questa API all'URL speciale /status.

/__src/__api/__task/__framework.setup.php

TODO Questo task è ancora da implementare.

/__src/__api/__task/__geografia.importazione.php

Questo task, piuttosto semplice, scarica i dati relativi alla geografia dalla versione corrente del framework:

- https://dataserver.istricesrl.com/geografia/01.update.stati.csv
- https://dataserver.istricesrl.com/geografia/02.update.regioni.csv
- https://dataserver.istricesrl.com/geografia/03.update.provincie.csv
- https://dataserver.istricesrl.com/geografia/04.update.comuni.csv

e li copia nella cartella DIR_VAR_SPOOL_IMPORT per sfruttare il normale sistema di importazione automatica dei file CSV del framework (dev/_src/_config/_740.controller.php) e aggiornare le informazioni relative alla geografia per il deploy corrente.

/__src/__api/__task/__memcache.clean.php

Questo task si occupa semplicemente di svuotare la memoria di memcache.

/__src/__api/__task/__mysql.patch.php

Questo task si occupa di installare e aggiornare il database MySQL del deploy corrente. La logica di aggiornamento del database è piuttosto complessa ma funzionale allo scopo di tenere aggiornati tutti i database di tutti i deploy senza dover eseguire le query a mano. Le componenti del sistema sono:

- l'API /_src/_api/_task/_mysql.patch.php
- i file di patch presenti in /_usr/_database/_patch

Il meccanismo di patch del database è illustrato nel dettaglio nel file /_src/_api/_task/_mysql.patch.php al quale si rimanda per approfondimenti.

/__src/__api/__task/__test.cron.php

Questo è un semplice task di test, utile per verificare il funzionamento del sistema dei task ricorrenti; non fa altro che scrivere su un file di log quando viene eseguito, in questo modo è facile fare dei test e del debug sul meccanismo dei task.

/__src/__api/__task/__test.job.start.php

Questo task avvia un job di test.

/__src/__config/__000.debug.php

Questo file di configurazione inizializza l'array $cf['debug'] e setta i default per le sue chiavi principali.

/__src/__config/__005.debug.php

Questo file di configurazione integra $cf[‘debug’] con $cx[‘debug’] e collega $cf[‘debug’] a $ct[‘debug’].

/__src/__config/__010.site.php

In questo file viene inizializzato l’array $cf[‘sites’] e viene definito il sito di default.

/__src/__config/__015.site.php

In questo file l’array $cf[‘sites’] viene integrato con $cx[‘sites’] e $cf[‘sites’] viene collegato a $ct[‘sites’]. Viene inoltre elaborato l’URL corrente per capire in quale sito ci si trova, e popolato di conseguenza l’array $cf[‘site’].

/__src/__config/__020.debug.php

In questo file vengono applicati i settaggi relativi al debug impostati precedentemente.

/__src/__config/__025.site.php

In questo file vengono elaborate diverse impostazioni di $cf[‘site’] ricavate dai dati settati in precedenza; viene anche integrato l’array $cf[‘site’] con $cx[‘site’] e infine $cf[‘site’] viene collegato a $ct[‘site’].

/__src/__config/__030.common.php

In questo file vengono settate diverse variabili di utilità generale in $cf[‘common’] fra cui i codici di stato HTTP e il Lorem Ipsum visto sopra; vengono inoltre recuperati i numeri di versione e release correnti e quelli del deploy corrente.

/__src/__config/__035.common.php

In questo file l’array $cf[‘common’] viene integrato con $cx[‘common’] e collegato successivamente a $ct[’common0].

/__src/__config/__040.cache.php

In questo file vengono definite le configurazioni per le cache in uso sul sito. GlisWeb supporta Memcache, Redis, APCU e caching su disco.

/__src/__config/__045.cache.php

In questo file vengono integrati i dati da $cx per le varie cache, effettuate le connessioni ai server di cache, collegati i profili e gli array $cf a $ct relativamente alle varie cache. Da questo runlevel in poi il caching è disponibile.

/__src/__config/__050.session.php

In questo file viene configurata la sessione PHP per utilizzare come backend in ordine di preferenza Redis, Memcache e il disco fisso.

/__src/__config/__055.session.php

In questo file viene avviata la sessione PHP, quindi a partire da questo runlevel la sessione è disponibile. Vengono anche collegati $_SESSION a $cf[‘session’] e $cf[‘session’] a $ct[‘session’].

/__src/__config/__060.privacy.php

TODO Questo runlevel e questa factory sono da riprogettare e migliorare.

/__src/__config/__065.privacy.php

In questo file l’array $cf[‘privacy’] viene integrato con $cx[‘privacy’], con $cf[‘site’][‘privacy’] e con $_COOKIE[‘privacy’]; viene inoltre collegato a $ct[‘privacy’]. Questo file gestisce inoltre l’invio dei moduli di consenso cookie $_REQUEST[‘**cookie**’] salvando le preferenze dell’utente in $cf[‘privacy’][‘cookie’].

/__src/__config/__070.cache.php

Questo file si occupa di verificare se la risorsa richiesta dall’utente è presente nella cache statica dei contenuti, e nel caso la restituisce, interrompendo di fatto l’esecuzione del framework. Questo è un importante meccanismo di risparmio delle risorse server, in quanto evita che vengano svolte operazioni potenzialmente onerose e di fatto inutili, come la connessione al database.

/__src/__config/__080.localization.php

In questo file viene creato e configurato l’array $cf[‘localization’], che contiene le informazioni relative alla localizzazione del framework. GlisWeb è nativamente multilingua e il supporto per la localizzazione e la traduzione fa parte delle sue caratteristiche base.

/__src/__config/__085.localization.php

In questo file l’array $cf[‘localization’] viene integrato con $cx[‘localization’] e collegato a $ct[‘localization’], dopodiché viene individuata la lingua corrente tramite varie strategie.

/__src/__config/__090.translation.php

In questo file viene creato l’array $cf[‘tr’] e vengono importati i dizionari presenti in /_etc/_dictionaries.

/__src/__config/__095.translation.php

In questo file l'array $cf['tr'] viene integrato con $cx['tr'] e collegato a $ct['tr'].

/__src/__config/__100.security.php

Questo file è vuoto e serve per le customizzazioni.

/__src/__config/__110.google.php

In questo file vengono definiti i profili per i servizi Google sotto $cf['google']['profiles'].

/__src/__config/__115.google.php

In questo file l'array $cf['google'] viene integrato con $cx['google'] e collegato a $ct['google']; inoltre viene collegato il profilo Google per lo status corrente a $cf['google']['profile'].

/__src/__config/__120.mysql.php

In questo file vengono definiti i server e i profili MySQL; viene inizializzato l'array $cf['mysql'].

/__src/__config/__125.mysql.php

In questo file l'array $cf['mysql'] viene integrato con $cx['mysql'] e collegato a $ct['mysql']; viene inoltre collegato il profilo MySQL corrente a $cf['mysql']['profile'], e infine vengono effettuate tutte le connessioni. Da questo runlevel in poi la connessione al database è disponibile tramite la chiave $cf['mysql']['connection'].

/__src/__config/__130.redirect.php

In questo file viene inizializzato l'array $cf['redirect'] dopodiché viene popolato con i dati eventualmente presenti nel file FILE_REDIRECT e nella vista redirect_view del database.

/__src/__config/__135.redirect.php

In questo file i redirect prelevati da filesystem e quelli prelevati da database vengono indicizzati in un unico array $cf['redirect']['index']. L'array $cf['redirect'] viene integrato con $cx['redirect']; infine, viene verificato se l'URL corrente corrisponde a un redirect, e nel caso la redirezione viene applicata tramite header http. In caso di redirect, l'esecuzione del framework termina qui.

/__src/__config/__140.session.php

In questo file vengono gestiti i tag UTM in $cf['session']['utm'] e inizializzate le impostazioni per l'anti spam in $cf['session']['spam'].

/__src/__config/__180.privacy.php

In questo file vengono lette dal database le impostazioni di privacy dei moduli e vengono riportate su $cf[‘privacy’][‘moduli’]. Si noti che la parte relativa alla privacy è tuttora in costante sviluppo.

/__src/__config/__190.localization.php

In questo file le informazioni di localizzazione vengono integrate con le informazioni presenti sul database.

/__src/__config/__195.localization.php

In questo file vengono applicate le impostazioni relative alla lingua corrente.

/__src/__config/__200.auth.php

In questo file vengono inizializzati gli array che servono per l’autenticazione sul framework; in particolare vengono aggiunti gli utenti a $cf[‘auth’][‘accounts’] e i gruppi a $cf[‘auth’][‘groups’]. I privilegi vengono inseriti in $cf[‘auth’][‘privileges’]. I profili di creazione degli account vengono inseriti in $cf[‘auth’][‘profili’]. Secondariamente, viene inizializzato il salt per JWT. Ulteriori dettagli sul funzionamento del sistema di autenticazione del framework possono essere reperite proprio nei commenti di questo file.

/__src/__config/__205.auth.php

In questo file l’array $cf[‘auth’] viene integrato con $cx[‘auth’].

/__src/__config/__210.auth.php

In questo file vengono svolte le operazioni di login. Le opzioni per identificare e autenticare gli utenti nel framework sono diverse e comprendono:

- basic HTTP auth
- token JWT
- bearer token
- username e password

Gli utenti e tutte le relative informazioni possono essere letti dal database o dai file di configurazione. È una prassi comune e raccomandata definire tramite file di configurazione l’utente root, in modo da poter accedere al framework anche in caso di malfunzionamento del database.

/__src/__config/__220.auth.php

In questo file vengono gestiti il timeout della sessione e il logout.

/__src/__config/__250.auth.php

In questo file viene dichiarato l'array $cf['auth']['permissions'] che definisce quali permessi hanno gli utenti su tutte le entità gestite dal framework.

/__src/__config/__255.auth.php

In questo file se c'è un login in corso vengono applicati i permessi all'account che si è appena connesso, in base a quanto stabilito dal runlevel _250.auth.php.

/__src/__config/__300.pages.php

In questo file viene verificato se i contenuti del sito sono già presenti in cache. Se lo sono, la chiave $cf['contents']['cached'] viene impostata a false e il successivo runlevel _310.pages.php viene eseguito per intero. Viceversa, il grosso delle elaborazioni del runlevel _310.pages.php viene saltato se i contenuti sono presenti in cache. I contenuti che vengono salvati in cache sono:

- $cf['contents']['cached']
- $cf['contents']['updated']
- $cf['contents']['pages']
- $cf['contents']['tree']
- $cf['contents']['index']
- $cf['contents']['reverse']
- $cf['contents']['shortcuts']

Per ulteriori dettagli si vedano i commenti al codice.

/__src/__config/__310.pages.php

In questo file vengono letti tutti i file di configurazione delle pagine, popolando $cf['contents']['pages'] che viene poi successivamente integrato da $cx['contents'] e da $cf['site']['contents'] se presenti. Al termine di questo runlevel tutti i dati delle pagine sono caricati e pronti per l'indicizzazione e l'elaborazione.

/__src/__config/__320.pages.php

In questo file l'elenco delle pagine viene elaborato per popolare alcuni indici necessari al funzionamento del sito, che sono:

- $cf['contents']['tree']
- $cf['contents']['index']
- $cf['contents']['shortcuts']
- $cf['contents']['reverse']

Questi indici sono necessari per la decodifica dell'URL richiesto e per la creazione dell'albero delle pagine, sul quale poggiano diversi elementi di navigazione, fra cui i menù e le briciole di pane.

/__src/__config/__330.pages.php

In questo file tutti i dati elaborati finora sui contenuti vengono salvati in cahce (Memcache) per utilizzi successivi.

/__src/__config/__340.sms.php

In questo file vengono definiti i template SMS utilizzati dal framework. I template vengono definiti come array PHP e integrati con i template presenti nel database.

/__src/__config/__350.mail.php

In questo file vengono definiti i template mail utilizzati dal framework. I template vengono definiti come array PHP e integrati con i template presenti nel database.

/__src/__config/__355.mail.php

In questo file $cf[‘mail’] viene integrato con $cx[‘mail’].

/__src/__config/__360.image.php

In questo file vengono definiti i formati immagine supportati dal framework. Il framework implementa un meccanismo di auto scalamento delle immagini per ottimizzare la banda e supportare i tag HTML5 responsivi; i dettagli riguardanti gli orientamenti e le dimensioni di scalatura sono definiti in questo file.

/__src/__config/__365.image.php

In questo file l’array $cf[‘image’] viene integrato con $cx[‘image’] e collegato a $ct[‘image’].

/__src/__config/__380.twig.php

In questo file vengono definiti i profili di funzionamento del template manager Twig.

/__src/__config/__385.twig.php

In questo file l’array $cf[‘twig’] viene integrato con $cx[‘twig’]; inoltre viene definito il profilo corrente $cf[‘twig’][‘profile’] come link a $cf[‘twig’][‘profiles’][$cf[‘site’][‘status’]]. Infine viene verificato che la cartella della cache di Twig, se necessaria, esista.

/__src/__config/__400.rewrite.php

In questo file viene fatto il parsing dell’URL richiesto dal client e determinata la pagina corrente. Questo è un file cruciale per il funzionamento del framework e dev’essere studiato assieme a /_src/_api/_pages.php.

/__src/__config/__420.pages.php

Questo runlevel è dedicato alle elaborazioni specifiche relative alla pagina corrente; fra le varie cose, vengono create le shortcut $ct[‘pages’] come link a $cf[‘contents’][‘pages’] e $ct[‘page’] come link a $cf[‘contents’][‘page’]. Vengono inoltre effettuate varie elaborazioni specifiche della pagina, fra cui l’elaborazione del menù a schede (se presente).

In questo file vengono inoltre gestiti i comandi di una lettera, fondamentali per il debug. I comandi gestiti qui sono:

comando	effetto
t	modifica il template della pagina (es. ?t=minerva)
s	modifica lo schema della pagina (es. ?s=schema-prova)
c	modifica il tema della pagina (es. ?c=natale)
m	inserisce del lorem ipsum in $ct[‘page’][‘content’][$cf[‘localization’][‘language’][‘ietf’]] (es. ?m=5)

/__src/__config/__430.security.php

File inserito per retrocompatibilità e customizzazione.

/__src/__config/__510.smtp.php

In questo file vengono definiti i server e i profili SMTP.

/__src/__config/__515.smtp.php

In questo file l’array $cf[‘smtp’] viene integrato con $cx[‘smtp’] e con $cf[‘site’][‘smtp’]. Viene inoltre salvato il profilo SMTP attivo in $cf[‘smtp’][‘profile’] e il server SMTP di default in $cf[‘smtp’][‘server’].

/__src/__config/__520.mapquest.php

In questo file vengono definiti i server e i profili Mapquest.

/__src/__config/__525.mapquest.php

In questo file l’array $cf[‘mapquest’] viene integrato con $cx[‘mapquest’] e collegato tramite puntatore a $ct[‘mapquest’]. Vengono inoltre definiti il profilo corrente in $cf[‘mapquest’][‘profile’] e il server corrente in $cf[‘mapquest’][‘server’].

/__src/__config/__540.sms.php

In questo file vengono definiti i server e i profili SMS.

/__src/__config/__545.sms.php

In questo file l'array $cf['sms'] viene integrato con $cx['sms']. Vengono inoltre definiti il profilo corrente in $cf['sms']['profile'] e il server corrente in $cf['sms']['server'].

/__src/__config/__550.slack.php

In questo file vengono definiti i server e i profili Slack.

/__src/__config/__555.slack.php

In questo file l'array $cf['slack'] viene integrato con $cx['slack'] e $cf['site']['slack'] e collegato tramite puntatore a $ct['slack']. Viene inoltre definito il profilo corrente in $cf['slack']['profile'].

/__src/__config/__560.archivium.php

In questo file vengono definiti i server e i profili Archivium.

/__src/__config/__565.archivium.php

In questo file l'array $cf['archivium'] viene integrato con $cx['archivium'] e $cx['site']['archivium']; viene inoltre collegato $cf['archivium'] a $ct['archivium'] e viene definito il profilo corrente in $cf['archivium']['profile'].

/__src/__config/__570.openai.php

In questo file vengono definiti i server e i profili OpenAI.

/__src/__config/__575.openai.php

In questo file viene integrato l'array $cf['openai'] con $cx['openai']; viene inoltre collegato $cf['openai'] a $ct['openai'] tramite puntatore. Infine vengono impostati il profilo e il server corrente rispettivamente in $cf['openai']['profile'] e $cf['openai']['server'].

/__src/__config/__580.ftp.php

In questo file vengono definiti i server e i profili FTP.

/__src/__config/__585.ftp.php

In questo file l'array $cf['ftp'] viene integrato con $cx['ftp'] e $cx['site']['ftp']. Vengono inoltre definiti il profilo corrente in $cf['ftp']['profile'] e il server corrente in $cf['ftp']['server'].

/__src/__config/__600.common.php

In questo file vengono definiti i profili di funzionamento per le integrazioni con TeamSystem e Zucchetti.

/__src/__config/__605.common.php

In questo file vengono integrati con la configurazione da file i profili di funzionamento per le integrazioni con TeamSystem e Zucchetti, e vengono definiti i profili correnti.

/__src/__config/__610.paypal.php

In questo file vengono definiti i profili di funzionamento di PayPal. Questi profili sono utilizzati dal modulo pagamenti per gestire il saldo dei pagamenti, e non sono collegati al modulo e-commerce.

/__src/__config/__615.paypal.php

In questo file i dati di funzionamento di PayPal vengono integrati con le direttive da file di configurazione, generali e per sito. Vengono inoltre definite le scorciatoie e collegato $cf[‘paypal’] a $ct[‘paypal’].

/__src/__config/__620.amazon.php

In questo file vengono definiti i profili di funzionamento di Amazon.

/__src/__config/__625.amazon.php

In questo file i dati di Amazon vengono integrati con le direttive presenti nei file di configurazione generali e per sito; vengono create le scorciatoie e l’array $cf[‘amazon’] viene collegato a $ct[‘amazon’].

/__src/__config/__640.facebook.php

In questo file vengono definiti i profili di funzionamento di Facebook.

/__src/__config/__645.facebook.php

In questo file i dati di Facebook vengono integrati con le direttive di configurazione da file, generali e per sito. Vengono definite le scorciatoie e l’array $cf[‘facebook’] viene collegato a $ct[‘facebook’].

/__src/__config/__680.hotjar.php

In questo file vengono definiti i profili di funzionamento di Hotjar.

/__src/__config/__685.hotjar.php

In questo file i dati di Hotjar vengono integrati con le configurazioni da file, generiche e per sito. Vengono configurate le scorciatoie e l'array $cf['hotjar'] viene collegato a $ct['hotjar'].

/__src/__config/__710.session.php

In questo file vengono inizializzati gli array $_SESSION['**view**'], $_SESSION['**work**'], $_REQUEST['**err**'] e _REQUEST['**info**'].

/__src/__config/__715.session.php

In questo file gli array $_SESSION['**view**'] e $_SESSION['**work**'] vengono integrati e collegati a $_REQUEST['**view**'], $_REQUEST['**work**'].

/__src/__config/__720.privacy.php

In questo file i cookie vengono indicizzati per ID.

/__src/__config/__730.controller.php

In questo file vengono inclusi gli eventuali parser di pagina. I parser sono un meccanismo tramite il quale il framework pre elabora i dati presenti in $_REQUEST per prepararli al lavoro dei runlevel successivi.

/__src/__config/__740.controller.php

Questo file gestisce l'importazione di dati in batch. Il meccanismo è molto potente ma anche complesso, e si rimanda alla documentazione del file per i dettagli.

/__src/__config/__750.controller.php

Questo file si occupa di gestire i blocchi dati in entrata e passarli alla controller(). Questo è il meccanismo con cui il framework gestisce la maggior parte dei blocchi data in ingresso.

/__src/__config/__760.controller.php

Questo file nel framework base è inserito solo per consentirne la customizzazione. Questo è il punto dove svolgere tutte le operazioni successive all'esecuzione della controller.

/__src/__config/__770.bookmarks.php

Questo file imposta i gruppi dell'area di lavoro per cui possono essere pinnati degli elementi. Il meccanismo dei bookmarks, o elementi pinnati,

è dettagliatamente illustrato nei commenti di questo file e del successivo /_src/_config/_775.bookmarks.php. Si vedano anche i commenti all'API /_src/_api/_bookmarks.php; si veda anche la documentazione dei file /_src/_config/_710.session.php e /_src/_config/_715.session.php.

/_src/_config/_775.bookmarks.php

Questo file integra la configurazione di $cf['bookmarks'] con $cx['bookmarks'] e imposta il collegamento a puntatore fra $cf['bookmarks'] e $ct['bookmarks'].

/_src/_config/_790.job.php

Questo file si occupa di selezionare i job in foreground e renderli disponibili in $cf['jobs']['foreground'] e tramite link simbolico in $ct['jobs']['foreground'].

/_src/_config/_920.privacy.php

Questo file è a disposizione per la customizzazione.

/_src/_config/_940.session.php

In questo file vengono salvate diverse informazioni utili sulla sessione, fra cui la timestamp dell'ultimo utilizzo.

/_src/_config/_980.sitemap.php

Questo file si occupa di generare, se necessario, le sitemap. Per ulteriori informazioni si veda la documentazione del file stesso.

/_src/_config/_990.debug.php

Questo file serve per il debug generale dei runlevel, in quanto conclude l'esecuzione del kernel space del framework. Da qui in poi l'esecuzione passa alle macro di pagina e dev'essere debuggata di conseguenza.

/_src/_css/_back2top.css

In questo file viene definito lo stile CSS per il tasto "torna su".

/_src/_css/_main.css

Questo file contiene gli stili di base validi per tutti i template.

/_src/_css/_selectbox.css

Questo file contiene gli stili per la tendina intelligente (combobox).

/__src/__css/__terminale.css

Questo file contiene gli stili per il terminale (va verificato che siano ancora necessari perché risalgono alla versione precedente del framework).

/__src/__img/__favicon.ico

Questa è la favicon di default del framework. Riguardo alla favicon, si devono tenere presenti diversi file primo fra tutti il file /.htaccess che effettua il routing da /favicon.ico a /__src/__img/__favicon.ico a meno che /favicon.ico non esista. Questo è un modo deprecato di customizzare la favicon dato che è disponibile un meccanismo molto più avanzato che consente di fornire al browser formati diversi di favicon.

Le favicon custom vanno collocate nella cartella /img/favicons/ (oppure /img/favicons//) dove vengono cercate dal file dev/__src/__api/__pages.php e in particolare vengono cercati questi file:

```
- android-icon-36x36.png
- android-icon-48x48.png
- android-icon-72x72.png
- android-icon-96x96.png
- android-icon-144x144.png
- android-icon-192x192.png
- apple-icon.png
- apple-icon-57x57.png
- apple-icon-60x60.png
- apple-icon-72x72.png
- apple-icon-76x76.png
- apple-icon-114x114.png
- apple-icon-120x120.png
- apple-icon-144x144.png
- apple-icon-152x152.png
- apple-icon-180x180.png
- apple-icon-precomposed.png
- favicon.ico
- favicon-16x16.png
- favicon-32x32.png
- favicon-96x96.png
- ms-icon-70x70.png
- ms-icon-144x144.png
- ms-icon-150x150.png
- ms-icon-310x310.png
```

Dal momento che la creazione di così tante icone può risultare tediosa, è possibile avvalersi di strumenti come https://www.favicon-generator.org/ in attesa che il framework implementi una propria gestione della scalatura delle favicon.

/__src/__inc/__macro/__app.php

Questa è la macro di pagina di default della pagina app. In standard non prevede particolari funzionalità, ma è pensata per essere customizzata.

/__src/__inc/__macro/__security.php

Questo file implementa il firewall applicativo del framework ed è quindi cruciale per la sua sicurezza. Viene incluso da /_src/_config.php e si occupa di filtrare le richieste potenzialmente dannose. Per i dettagli del suo funzionamento si vedano i commenti al codice.

/__src/__inc/__pages/__app.it-IT.php

Questo file contiene la dichiarazione delle pagine della web app standard del framework.

/__src/__inc/__pages/__dashboard.it-IT.php

Questo file contiene la dichiarazione delle pagine della dashboard del CMS.

/__src/__inc/__pages/__null.it-IT.php

Questo file contiene la dichiarazione della pagina NULL utilizzata per l'errore HTTP 404.

/__src/__inc/__pages/__site.it-IT.php

Questo file è vuoto in modo che possa essere facilmente customizzato.

/__src/__js/__main.js

Questa libreria Javascript contiene le funzioni di utilità generale del framework, nonché le operazioni da eseguire al caricamento del DOM.

/__src/__lib/__acl.utils.php

Questa libreria contiene le funzioni di utilità per la gestione dei permessi degli utenti.

/__src/__lib/__apcu.tools.php

Questa libreria contiene le funzioni per l'utilizzo della cache APCU.

/__src/__lib/__array.tools.php

Questa libreria contiene una collezione di funzioni per la manipolazione degli array.

/__src/__lib/__controller.tools.php

Questa libreria contiene la funzione controller() e le sue funzioni di appoggio.

/__src/__lib/__cryptography.tools.php

Questa libreria contiene alcuni strumenti per la gestione della crittografia.

/__src/__lib/__csv.tools.php

Questa libreria contiene una collezione di strumenti per la manipolazione dei file e dei dati in formato CSV.

/__src/__lib/__filesystem.tools.php

Questa libreria contiene una collezione di funzioni per la gestione dell'I/O sul filesystem.

/__src/__lib/__fsv.tools.php

Questa libreria è ancora da implementare e dovrebbe contenere una collezione di funzioni per la gestione dei file a larghezza fissa.

/__src/__lib/__ftp.tools.php

Questa libreria è ancora da implementare e dovrebbe contenere una collezione di funzioni per la gestione del protocollo FTP.

/__src/__lib/__jwt.tools.php

Questa libreria contiene le funzioni per la gestione dei token JWT.

/__src/__lib/__localization.tools.php

Questa libreria contiene funzioni utili per la localizzazione.

/__src/__lib/__log.utils.php

Questa libreria è inserita solo per garantire la retrocompatibilità con il vecchio sistema di log del framework.

/__src/__lib/__memcache.tools.php

Questa libreria contiene funzioni per la gestione della cache su Memcache.

/__src/__lib/__menu.utils.php

Questa libreria contiene funzioni per la generazione dei menu di navigazione.

/__src/__lib/__mysql.tools.php

Questa libreria contiene le funzioni necessarie alla gestione del database MySQL.

/__src/__lib/__mysql.utils.php

Questa libreria contiene funzioni di varia utilità basate su MySQL.

/__src/__lib/__output.tools.php

Questa libreria contiene funzioni per l'output.

/__src/__lib/__random.tools.php

Questa libreria contiene funzioni utili per la generazione di dati casuali.

/__src/__lib/__recaptcha.tools.php

Questa libreria contiene una collezione di funzioni per la gestione di Google reCaptcha.

/__src/__lib/__redis.tools.php

Questa libreria contiene funzioni utili per gestire la cache di Redis.

/__src/__lib/__rest.tools.php

Questa libreria contiene strumenti utili per la gestione delle chiamate REST.

/__src/__lib/__rewrite.tools.php

Questa funzione contiene strumenti per il supporto alla gestione dell'URL rewriting.

/__src/__lib/__string.tools.php

Questa libreria contiene una collezione di funzioni per la manipolazione delle stringhe.

/__src/__lib/__xml.tools.php

Questa libreria contiene funzioni per la gestione dell'XML.

/__src/__sh/__backup.run.sh

Questo script crea un backup del sito nella cartella genitore della document root.

/__src/__sh/__codeception.init.sh

Questo script inizializza le cartelle e il codice per i test. TODO va riordinato e documentato.

/__src/__sh/__codeception.run.sh

Questo script esegue i test di accettazione del framework; i test si basano sugli esempi contenuti in _usr/_examples/; per ulteriori dettagli sul funzionamento dei test di accettazione del framework fare riferimento alla documentazione presente in dev/_usr/_docs/_dox/_test.dox.

/__src/__sh/__composer.update.sh

Questo file esegue l'aggiornamento delle librerie esterne tramite composer; può inoltre eseguire una pulizia delle librerie attualmente installate se lanciato in modalità hard, questo è utile per risolvere problemi di aggiornamento di composer.

/__src/__sh/__crontab.install.sh

Questo file installa il file crontab necessario a far funzionare le operazioni pianificate del framework. Il file viene posizionato in /etc/cron.d/ in modo da sfruttare il cron di sistema. Per ulteriori informazioni sul sistema delle operazioni pianificate del framework si veda la documentazione delle API /_src/_api/_cron.php e dev/_src/_api/_job.php.

/__src/__sh/__deploy.run.sh

Questo script esegue il deploy dell'installazione corrente su un target indicato come argomento. L'argometo che specifica il target deve corrispondere al nome del file di configurazione da utilizzare per il deploy, fra quelli disponibili nella cartella /etc/deploy/; il nome del file va specificato al netto dell'estensione .properties:

```
./_src/_sh/_deploy.run.sh stable
```

/__src/__sh/__doxygen.build.sh

Questo script compila la documentazione tramite Doxygen. Il framework è ampiamente documentato con commenti che Doxygen è in grado di trasformare in documentazione HTML e PDF, e grazie a questo meccanismo è possibile risparmiare molto tempo sulla scrittura di manualistica.

/__src/__sh/__folders.check.sh

Questo script controlla che esistano le cartelle custom solitamente necessarie al funzionamento corretto del framework.

/__src/__sh/__gw.clean.sh

Questo script effettua una pulitura dei file superflui del framework; può essere chiamato in modalità soft o hard a seconda di quanto si vuole cancellare.

/__src/__sh/__lamp.permissions.open.sh

Questo script resetta i permessi della document root del sito in modo che i file siano accessibili a più utenti possibile, solitamente è una modalità utilizzata per sviluppo e debug; si noti che con i permessi aperti il framework non gira per motivi di sicurezza.

/__src/__sh/__lamp.permissions.secure.sh

Questo script resetta i permessi della document root del sito in modo che siano il più restrittivi possibile. Questa è la modalità di funzionamento predefinita del framework.

/__src/__sh/__lamp.setup.sh

Questo script fa il setup dell'ambiente LAMP necessario a far girare il framework.

/__src/__sh/__password.hash.sh

Questo script genera l'hash di una password in modo che possa essere utilizzata nella configurazione o nel database del framework.

/__src/__sh/__test.import.sh

Questo script genera dei file di test in /var/spool/import/todo e /var/spool/import per il test del sistema di importazione file; si veda /_src/_config/_740.controller.php per i dettagli e /_src/_api/_report/_import.php per i test del sistema.

/__src/__sh/__lib/__functions.sh

Questa libreria viene utilizzata dagli script shell del framework per svolgere alcuni compiti base come la gestione degli argomenti da linea di comando.

/__src/__twig/__inc/__analytics.head.twig

Questo file include il codice per Google Analytics posto che l'utente abbia prestato il consenso oppure che Analytics sia configurato in modalità anonimizzazione IP.

/__usr/__database/__patch/__010000999999.tables.sql

Questo file contiene le patch base necessarie alla creazione delle tabelle nel database del framework; per ulteriori informazioni sul funzionamento del sistema di patch si vedano i commenti al file /_src/_api/_task/_mysql.patch.php.

FAQ

domande generali

quali sono le operazioni da svolgere per pubblicare una nuova release? Per pubblicare una nuova release è necessario:

- incrementare il numero di versione
- creare una release branch da develop
- effettuare il debug e i test sulla release branch
- fare il merge della release branch su master e su develop

quali sono i comandi di una lettera disponibili nel framework? Il framework supporta diversi comandi di una lettera che possono essere passati nell'URL per ottenere determinati effetti, solitamente utili agli sviluppatori e ai tester.

comando	implementato in	effetto
b	/_src/_config/_220.auth.php	URL di ritorno
c	/_src/_config/_420.pages.php	forza il tema della pagina
j	/_src/_config/_210.auth.php	innesca il login via JWT
m	/_src/_config/_420.pages.php	inserisce il Lorem Ipsum come testo della pagina
s	/_src/_config/_420.pages.php	forza lo schema della pagina
t	/_src/_config/_420.pages.php	forza il template della pagina
u	/_src/_api/_pages.php	crea un commento HTML con il dump dell'array $ct a partire dal nodo indicato

come creo pagine statiche senza bisogno di configurare il framework? È possibile creare contenuti statici che vengono serviti tramite il framework inserendoli nella cartella /usr/pages; è anche possibile creare delle sottocartelle. Una pagina HTML creata in /usr/pages/test.html sarà quindi raggiungibile, tramite un'apposita regola di /.htaccess, all'URL /test.html e una pagina creata in /usr/pages/prova/test.html sarà raggiungibile all'URL /prova/test.html.

Questo meccanismo consente di importare nel framework interi siti statici, in modo da poter utilizzare le funzioni del framework che si desidera senza la necessità di riscrivere tutto.

voglio creare un nuovo template, come faccio? Comincia studiando la documentazione su come sono strutturati i template in /_usr/_docs/_dox/_templates.dox.

che cos'è esattamente un'entità nel gergo del framework? Un'entità è l'astrazione nel contesto del framework di un insieme di oggetti o concetti della

vita reale. Solitamente corrisponde a una tabella nel database, e gli utenti hanno diversi tipi di permessi di interazione con essa.

come aggiungo un nuovo metodo di pagamento al framework? Dopo aver studiato il nuovo metodo di pagamento occorre capire come si può interfacciare al processo di pagamento del framework. Tipicamente si dovrà creare una libreria per le funzioni di supporto, e almeno un listener per gestire le comunicazioni in ingresso dal server del gestore del pagamento. A livello di configurazione è necessario aggiungere il metodo di pagamento all'array dei metodi di pagamento. Prima di iniziare il processo di integrazione studiare bene tutti i commenti al codice delle integrazioni già esistenti.

come definisco quali cookie utilizza il sito? Per ogni cookie o gruppo di cookie che utilizzi devi definire un paragrafo di configurazione in $cf[‘privacy’][‘cookie’], e precisamente:

- i cookie propri tecnici vanno sotto $cf[‘privacy’][‘cookie’][’propri][‘tecnici’]
- i cookie propri di profilazione vanno sotto $cf[‘privacy’][‘cookie’][’propri][‘analitici’]
- i cookie di terze parti tecnici vanno sotto $cf[‘privacy’][‘cookie’][’terzi][‘tecnici’]
- i cookie di terze parti di profilazione vanno sotto $cf[‘privacy’][‘cookie’][’terzi][‘analitici’]

Per una panoramica delle chiavi da inserire si vedano i commenti al codice del file /_src/_config/_060.privacy.php mentre per comprendere come questi dati vengono letti e scritti sui cookie si veda /_src/_config/_065.privacy.php. Le impostazioni dei cookie vengono poi lette dai file twig che compongono le pagine per decidere se inserire o meno il codice che genera i cookie.

come creo una pagina tramite file di configurazione PHP? Per creare una pagina tramite file di configurazione PHP è necessario creare o editare un file in /src/inc/pages, ad esempio /src/inc/pages/site.it-IT.php e aggiungere il relativo paragrafo di configurazione. Un tipico paragrafo di configurazione per una pagina avrà all'incirca questo aspetto:

```
$p['pagina.test'] = array(
    'id_sito'       => 1,
    'sitemap'       => true,
    'cacheable'     => true,
    'title'         => array( $l        => 'pagina di test' ),
    'description'   => array( $l        => 'questa è una pagina di test' ),
    'h1'            => array( $l        => 'ciao sono la pagina di prova' ),
    'template'      => array( 'path'    => '_src/_tpl/_aurora/', 'schema' => 'default.twig'
    'parent'        => array( 'id'      => NULL ),
    'menu'          => array( 'main'    => array( '' => array( 'label' => array( $l => 'pagi
);
```

Per un approfondimento sui valori da assegnare alle varie chiavi, e per l'elenco delle chiavi disponibili, si faccia riferimento alla documentazione del file

/_src/_config/_310.pages.php.

Se stai personalilzzando pagine standard, copia in custom il file standard al quale vuoi aggiungere una pagina (ricordati di togliere l'underscore iniziale); a questo punto puoi aggiungere la pagina. Ad esempio se vuoi aggiungere una pagina al file /_src/ _inc/_pages/_app.it-IT.php devi customizzarlo in /src/inc/pages/app.it-IT.php.

come creo una pagina tramite file di configurazione JSON o YAML?
Per creare una pagina tramite la configurazione estesa JSON/YAML è sufficiente aggiungere alla configurazione una chiave per la pagina, con le relative sottochiavi. Ad esempio in JSON:

```
{
    [...]
    "contents": {
        "pages": {
            [...]
            "prova.pagina.json": {
                "id_sito": 1,
                "sitemap": true,
                "cacheable": true,
                "title": {
                    "it-IT": "pagina di prova da file JSON"
                },
                "description": {
                    "it-IT": "pagina di prova da file JSON"
                },
                "h1": {
                    "it-IT": "pagina di prova da file JSON"
                },
                "template": {
                    "path": "_src/_tpl/_aurora/",
                    "schema": "default.twig"
                },
                "parent": {
                    "id": null
                },
                "menu": {
                    "main": {
                        "": {
                            "label": {
                                "it-IT": "prova JSON"
                            },
                            "priority": 190
                        }
```

```
                    }
                }
            },
            [...]
        }
    }
    [...]
}
```

Ovviamente la stessa struttura appare molto più sintetica in YAML ma la sostanza è uguale:

```
[...]
contents:
  pages:
    [...]
    prova.pagina.yaml:
      id_sito: 1
      sitemap: true
      cacheable: true
      title:
        it-IT: pagina di prova da file YAML
      description:
        it-IT: pagina di prova da file YAML
      h1:
        it-IT: pagina di prova da file YAML
      template:
        path: _src/_tpl/_aurora/
        schema: default.twig
      parent:
        id: null
      menu:
        main:
          "":
            label:
              it-IT: prova YAML
            priority: 290
    [...]
[...]
```

come definisco staticamente i contenuti di una pagina già presente nell'array delle pagine? Il framework controlla la cartella src/inc/contents/ per contenuto relativo a una data pagina; puoi creare un file in src/inc/contents/ con il nome uguale all'ID della pagina cui vuoi settare il contenuto ad es. .xx-XX.html. Per ulteriori dettagli su questo meccanismo si veda la documentazione del file /_src/_api/_pages.php.

come definisco il contenuto di una pagina direttamente dal file di configurazione (PHP, JSON o YAML)? Per definire il contenuto della pagina direttamente dalla configurazione è sufficiente valorizzare la chiave content e la relativa sotto chiave per lingua. In PHP ad esempio questo si tradurrà in:

```
$p['pagina.test'] = array(
    'id_sito'       => 1,
    'sitemap'       => true,
    'cacheable'     => true,
    'title'         => array( $l        => 'pagina di test' ),
    'description'   => array( $l        => 'questa è una pagina di test' ),
    'h1'            => array( $l        => 'ciao sono la pagina di prova' ),
    'content'       => array( $l        => 'ciao sono il contenuto della pagina di prova' ),
    'template'      => array( 'path'    => '_src/_tpl/_aurora/', 'schema' => 'default.twig'
    'parent'        => array( 'id'      => NULL ),
    'menu'          => array( 'main'    => array( '' => array( 'label' => array( $l => 'pagi
);
```

Analogamente, in JSON e in YAML si procederà aggiungendo la chiave con le relative sotto chiavi per lingua.

è possibile specificare delle configurazioni di pagina solo per uno specifico sito? Sì certo, la chiave contents è valida anche come sotto chiave di ogni singolo sito, questo consente ad esempio di modificare il template di una pagina presente in più siti a seconda del sito corrente.

come faccio a testare il meccanismo dei task ricorrenti? Per prima cosa sincerati che il task di test funzioni correttamente se eseguito manualmente; ad esempio chiama l'URL /task/test.cron e sincerati che l'esecuzione del task avvenga senza problemi. Una volta controllato questo, puoi procedere con il controllo dell'API cron.

Per verificare che il task venga eseguito anche dall'API cron, devi verificare che il task sia pianificato, ovvero controllare sul database (SELECT * FROM `task`) che sia presente una riga per _src/_api/_task/_test.cron.php. Se la riga manca, aggiungila (vedi /_src/_api/_cron.php per i dettagli).

Una volta controllato che la tabella dei task contenga il task di test, puoi lanciare a mano l'API cron chiamando l'endpoint /api/cron, dopodiché verifica che il file di test sia stato scritto correttamente nella cartella dei log.

Una volta che il funzionamento del sistema è stato verificato in modalità manuale puoi utilizzare lo script _crontab.install.sh per installare il file di cron nella cartella /etc/cron.d/; a questo punto, il file di log dovrebbe venire scritto automaticamente ogni minuto.

come faccio a testare il meccanismo dei job? Per testare un job devi innanzitutto inserire la riga relativa nella tabella dei job; puoi farlo manualmente

oppure creare un task di avvio che lo faccia in modo semi automatico (esistono un job di test e il relativo task di avvio già pronti nel framework). Come primo step, testa il job in foreground.

Per testare un job in foreground è necessario inserirlo nel database con il flag se_foreground settato; in questo caso, il job verrà ignorato dall'API cron e sarà possibile eseguirlo solo manualmente tramite l'API job, il che è ottimo per il debug. Sincerati di aver inserito il job in modalità foreground e chiama l'API job con l'ID del job da testare per controllare che tutto funzioni regolarmente.

Una volta che il test manuale è andato a buon fine, puoi mandare il job in background settando a NULL il campo se_foreground, e chiamare manualmente l'API cron per verificare che venga effettivamente eseguito senza errori.

Se anche il test manuale tramite l'API cron va a buon fine, puoi attivare il cron di sistema e controllare che il lavoro del job venga incrementato ogni minuto in modo automatico.

come funziona la distribuzione del framework GlisWeb tramite container Docker? Il container della versione di sviluppo del framework GlisWeb è disponibile su https://hub.docker.com/repository/docker/istricesrl/glisdev/general e può essere liberamente scaricata e lanciata. Per effettuare un test minimale è possibile predisporre un piccolo file di configurazione e lanciare il container mappando la cartella in cui lo si è salvato. Si supponga ad esempio di creare il seguente file YAML in una cartella chiamata ./conf/:

```
sites:
  1:
    __label__: "test Docker"
    name:
      it-IT: "test Docker"
    protocols:
      DEV: "https"
    hosts:
      DEV: ""
    domains:
      DEV: "localhost"
```

A questo punto è possibile lanciare il container con:

```
docker run -dit -p 8080:80 -v ./conf/:/var/www/html/src/config/ext/ istricesrl/glisdev:$1
```

Aprendo un browser su http://localhost:8080/status si dovrebbe vedere correttamente configurato il framework. Per fermare il container in esecuzione è necessario conoscerne l'ID:

```
docker container list
```

e successivamente fermarlo con:

```
docker container stop <containerID>
```

Per entrare in un container ed esplorare i file (utile per vedere i file di log) utilizzare:

```
docker exec -it <containerId> bash
```

come creo una patch per il database? Una patch per il database è un file contenente uno o più comandi SQL, opportunamente commentati per far capire al framework come deve considerarli. Per mantenere la consistenza fra database e applicazione è necessario attenersi strettamente alle seguenti regole ogni volta che si desidera modificare il database del framework:

- per prima cosa, modificare opportunamente i file di patch base in modo che la modifica si propaghi ai NUOVI database
- secondariamente, creare i file di patch necessari a propagare la modifica ai database ESISTENTI

Per creare un file di patch finalizzato al deploy di una modifica ai database esistenti, è necessario rispettare le seguenti regole:

- il file va inserito nella cartella /_usr/_database/_patch/
- il nome del file deve corrispondere alla timestamp corrente seguita da quattro nove al posto dell'orario (YYYYMMDD9999)
- i comandi all'interno del file vanno separati da un progressivo formato dalla timestamp corrente seguita da quattro cifre (vedi sotto)
- i simboli dei commenti e i pipe sono significativi e vengono utilizzati dal framework per decodificare il file, attenersi all'esempio

Per comprendere la struttura del file di patch, si osservi il seguente codice:

```
--
-- PATCH
-- aggiornamento della tabella matadati 2022/10/07
-- file 202210079999.sql
--

-- | 202210070010
ALTER TABLE `metadati`
    ADD COLUMN `id_tipologia_todo` int(11) DEFAULT NULL AFTER `id_pianificazione`,
    ADD UNIQUE KEY `unica_tipologia_todo` (`id_lingua`,`id_tipologia_todo`,`nome`),
    ADD KEY `id_tipologia_todo` (`id_tipologia_todo`);

-- | 202210070020
ALTER TABLE `metadati`
    ADD CONSTRAINT `metadati_ibfk_27` FOREIGN KEY (`id_tipologia_todo`) REFERENCES `tipologi

-- | 202210070030
CREATE OR REPLACE VIEW `metadati_view` AS
```

```
    SELECT
        [...]
    FROM metadati
        LEFT JOIN lingue ON lingue.id = metadati.id_lingua
;

-- | FINE FILE
```

Come si vede, il file inizia con una testatina commentata (in forma libera) nella quale è possibile inserire note e commenti alla patch. Seguono tre sezioni introdotte dalla sequenza – | seguita dalla timestamp più il progressivo, e infine la chiusura – | FINE FILE. Ogni comando SQL è contenuto in una sezione separata.

Per ulteriori informazioni sul funzionamento del sistema di patch del database si faccia riferimento alla documentazione del file /_src/_api/_task/_mysql.patch.php.