Structure des répertoires

Le logiciel est écrit sur un framework MVC-Ajax. C'est une base solide qui fournit des dispositifs aboutis, pour écrire facilement n'importe quelle sorte de requête.
Le principe est celui des Apps, qui sont autant de Classes où notamment certaines méthodes sont attendues.

La structure des fichiers :

- prog
- prod
    - app
        - admin
        - app
        - dev
        - pub
        - index.php (3)
    - core
    - css
    - js
    - lib.php
    - index.php (2)
- index.php (1)
- call.php
- api.php
- boot.php
- amt.php

Les indexes (1) : celui du navigateur, celui des requêtes Ajax, celui de l'Api, et une page nommée boot.php qui fournit le minimum requit pour l'exécution du framework.

Les dossiers prod et prog sont similaires. Ils contiennent tout le logiciel et ses applications.
Le dossier /prog sert pour le développement, et son contenu est ensuite poussé sur /prod.
(Les fichiers poussés sont dans conservés dans un dossier /bckp).

Détails du dossier /prod : deux sections :

- /prod/core
- /prod/app.

/core est un noyau de fonctionnalités disponibles pour toutes les Apps.
Au même niveau que /core, on trouve lib.php, le seul endroit où on s'autorise à avoir des fonctions de premier niveau (non encadrées par un objet). Ce sont des fonctions qui facilitent la rédaction des composants classiques, tels que les balises html, les requêtes Ajax et d'autres opérations fréquentes, des filtres pour les chaînes, etc…

L'index de ce niveau (2) est la première App affichée à l'ouverture du site.

Ce réglage par défaut est définit dans /cnfg/site.com.txt. (On l'a mit sur 'telex').

Détails du dossier /prod/app :
On y trouve 4 dossiers de base et 1 pour l'appli Télex :
/admin pour les composants de gestion
/pub pour les appendices accessoires
/dev pour les Apps en cours de développement,
/apps pour les applications abouties.
/telex

L'autoload trouve n'importe quel fichier situé dans /prod. (ou /prog, en mode dev). On peut regrouper des apps dans un nouveau dossier du fichier /prod/app.

Architecture logicielle

Le logiciel repose sur un framework MVC-Ajax.

Il permet de rendre routinières la plupart des actions de type Ajax et de centraliser leur fonctionnement. Bien sûr, comme dans tout framework, pour aller au-delà de ses limites il reste possible de revenir à des fonctionnement entièrement réécrit. Mais pour éviter cela, les fonctions natives sont devenues très souples, quoique relativement complexes. En effet on peut très bien devoir déclencher plusieurs actions, javascript ou ajax, avant et après l'envoi et le retour de la commande principale.

Tous les boutons ajax partent de la fonction de premier niveau aj(). D'autres fonctions supplétives peuvent occasionnellement servir.

Faire d'Ajax la colonne vertébrale du framework a conduit à concevoir le logiciel sous forme d'objets qui s'additionnent les uns aux autres. Chaque objet est capable d'avoir une "vie autonome" du reste du logiciel, et contient tous les requis pour son fonctionnement : installation des bases de données, javascript, css, et le code.

Un certain nombre de méthodes sont attendues par le framework, dont la principale 'content()', qui est en somme l'interface.

(N'importe quelle App peut servir de modèle pour en décliner une nouvelle).
Les méthodes destinées à être utilisées comme une interface appelable de l'extérieur sont reconnaissables par la variable unique qu'elle reçoit.

Dans la variable $prm, se trouvent les éléments envoyés du paramétrage autant que de la lecture sur la page des éléments. En prime, s'y trouvent quelques infos sur la transaction, comme le nom de l'App et de la méthode appelée, ou encore la taille de la fenêtre à ce moment-là.

Le choix s'est posé pour des méthodes statiques afin d'empêcher une inflation de complexité qui aurait rendu le logiciel impénétrable pour les plus jeunes développeurs. L'intérêt de la structuration logicielle est qu'on peut appeler bout à bout n'importe quelle App, une principale et ses satellites, de façon séquentielle (au lancement) aussi bien que de façon procédurale (une par une, en Ajax).

De cette manière la croissance en complexité du logiciel n'interfère jamais, au très rarement, avec les Apps déjà conçues. On peut facilement rajouter des fonctionnalités ou des dispositifs, qui peuvent être testés de façon unitaires, et s'attacher ensuite à des processus plus vastes.

C'est le framework qui garantit la stabilité, l'unicité du code, et sa capacité à évoluer rapidement tout en restant clair et facile à comprendre.

Convention de nommage

Le Framework utilise une convention fonctionnelle.
Ce n’est pas tant le contenu des variables qui est affiché que sa nature.

syntax	usage
_a	before current
_b	after current
_r	recursive
a	antecedent, first unknown value
b	second version, second unknown value
bt	button content
c	class
d	data (string)
e	element
f	file
g	get, go
h	height, hidden, hover
i	iteration
im	image
j	javascript
k	key
l	embeded link
m	method
n	number
o	option
p	param

q	query
r	array
ra, rb, rc	generations of arrays
ret	value to return
s	split
sq	sql code
sz	size
t	text, title
tit	title
txt	text
u	url
v	value (string)
w	width, where
wh	where
x	confiscated, eraser
z	trash

Les Connecteurs

Traitement de texte

Télex font circuler une information que seul le logiciel, ou un lecteur, peut mettre en forme.
Les balises html ne sont pas refusées mais remplacées par des connecteurs.
Les connecteurs sont des balises de template qui permettent de commander la fabrication de code HTML, ou de tout autre dispositif attendu.

C'est un compromis idéal pour normaliser et diminuer la quantité d'informations, et permettre d'insérer dans les message des objets logiciels tels qu'une référence à un autre télex, ou des Apps.

N'importe quelle App peut être appelée via le connecteur :
[param:myApp]

On peut ajouter une option au paramètre principal (qui est divisible en plusieurs) :
[param*option:myApp]
[param*option1-option2:myApp]

La syntaxe est voulue pour répondre à des usages simples avec 1 paramètre et éventuellement une option.
Pour faire des choses "compliquées" il faut les imbriquer.

Moteur principal

On peut faire des tests sur /app/connecteurs.
Deux élément du noyau gèrent les connecteurs :

- /core/Conn fabrique le rendu à partir de connecteur
- /core/Tran fabrique des connecteurs à partir d'une source HTML (ainsi nettoyée des balises inutiles).

Equivalences en connecteurs des balises HTML prises en charge

HTML	Connecteurs
h1	h1
h2	h2
h3	h3,h
h4	h4
b	b
i	i

u	u
ul	list
ol	numlist
sup	sup
sub	sub
small	small
big	big
strike	s
blockquote	q
span	span
div	div
a	a

Connecteurs par défaut

Connecteur	Usage
tag	n'importe quelle balise
url	renvoie un lien qui s'ouvre dans un nouvel onglet (attribut _blank) en n'affichant que le nom de domaine
img	renvoie une balise img en vérifiant que l'image existe
pic	renvoie un pictogramme d'après sa dénomination internationale
art	renvoie un lien vers un article en affichant son titre
apj	renvoie un commande d'appel sur place en ajax d'une App
app	renvoie le contenu d'une App
pop	renvoie un bouton pour ouvrir une App dans une popup
aj	renvoie un bouton ajax, à paramétrer
no	annule le retour du contenu

Connecteurs de Télex (supplantent ceux par défaut)

@	lien vers un utilisateur
#	lien vers une recherche sur ce tag
id	contenu d'un autre télex
link	lien utilisant le titre obtenu des métas
img	ouvre une image en tant qu'objet *
web	renvoie le contenu des métas du site visé : titre, image, et description : le tout mit en forme, en tant qu'objet *
video	renvoie un lecteur (approprié au fournisseur)
audio	renvoie un lecteur audio html5
mp4	renvoie un lecteur vidéo html5
gps	permet d'ouvrir une carte d'OpenStreetMap
article	ouvre un article en tant qu'objet *
chat	ouvre un module de chat en tant qu'objet *
app	ouvre n'importe quelle app en tant qu'objet *
open	ouvre une app sur place

* Les objet sont des boutons qui pointent vers l'ouverture d'une App. Pour certains (web, img) seul le premier est affiché en entier, les autres apparaissent dans une div de fichiers joints.

Les objets ont la particularité de pouvoir être sauvegardés sur le Desktop, et réutilisés ultérieurement.

En terme général, toute App (nommée myApp) est appelée sous forme d'objet : [param:myApp]

Les Langues

L'ensemble du logiciel, depuis sa création, n'affiche aucun message écrit en dur dans l'application. Ce ne sont que des références données par la fonction lang(), qui renvoient un contenu qui figure dans la table lang.

Ce contenu dépend du paramètre de langue obtenu dans la config générale pour celle choisie par l'utilisateur.

La table lang réunit des textes courts, qui sont mis en majuscules par défaut.
Elle peut être conjointe avec la table pictos qui leur associe des icônes.
De cette manière le logiciel profite de ces dispositifs quasi automatiques.

Ils facilitent le développement en reléguant à une console d'admin le choix du vocabulaire employé. C'est l'app admin_lang qui se charge des alertes, et l'app admin_icons des pictos.
On appelle un message lié à son picto avec la fonction langp().
langpi() place le message dans un 'title' actif au survol.
langph() fait que l'apparition du message dépend de la taille de l'écran.

Une autre table existe pour les alertes plus longues, contenues dans les bulles d'aides, c'est la table help.
Son admin est l'app admin_help.
elle peut renvoyer un contenu brut ou entouré d'une div avec une classe.

Ainsi, l'intégralité du "discours" du logiciel est situé dans ces deux tables lang et help, et peut être décliné en n'importe quelle langue.

Gestionnaire Mysql

Dans toutes les apps qui utilisent une base de donnée, figure la methode install(). C'est un dispositif permettant de créer une table à la volée, et de modifier rapidement les colonnes en cours de dev.

Sql::create('testable',['uid'=>'int','tit'=>'var','txt'=>'text'],1);

où int, var et txt sont des commandes pour int, varchar(255) et longtext.

Toute table est entourée par défaut d'une colonne 'id' pour les indexes, et d'une colonne 'up' qui contient la date du dernier accès en écriture.

Quand on modifie la structure d'une table, un backup de l'ancienne est créé et on tente d'y copier les colonnes reconnues. (De cette manière le développement se fait sans avoir à toucher phpMyAdmin, sauf petites erreurs).

Quand le logiciel est terminé, on met en commentaire la fonction install().
Lors d'une installation, toutes les méthodes install() de toutes les classes sont interrogées, et les tables créées.

(Certains tables nécessitent un contenu mit à jour, ce qui peut être fait via app/apisql.)

Les requêteurs

Bien que n'importe quelle requête puisse être lancée depuis Sql::query, on le déconseille. insert() et update() servent à sécuriser le formatage des données.

insert2() et updates() acceptent des données de plusieurs lignes en même temps.
read() est la plus utilisée, et avec read_inner (jointure simple) ils utilisent un dispositif de formatage des réponses :

read($d,$b,$p,$q='',$z='');

où :
$d = colonnes
$b = nom de la table
$p = paramètre de traitement
$q = requête where (peut contenir les jointures)
$z = verbose de la requête construite

Les paramètres de traitement :

(aucun) : renvoie le contenu du fetch_row.
rq = ne retourne aucun traitement (il faut encore la fetch)
ry = retourne le fetch_array

ra = retourne le fetch_assoc
rw = retourne le fetch_row
v = retourne une chaîne (une seule colonne appelée)
rr = lit et retourne le fetch_assoc

```
k = $ret[$r[0]]=1;
rv = $ret=$r[0];
kv = $ret[$r[0]]=$r[1];
kr = $ret[$r[0]]=$r[1];
kk = $ret[$r[0]][$r[1]]=1
kkc = $ret[$r[0]][$r[1]]=1;
vv = $ret=array($r[0],$r[1]);
kkv = $ret[$r[0]][$r[1]]=$r[2];
kkr = $ret[$r[0]][$r[1]]=$r[2];
kkk = $ret[$r[0]][$r[1]][$r[2]]=1;
kvv = $ret[$r[0]]=array($r[1],$r[2]);
id = $k=array_shift($r); $ret[$k]=$r; //première colonne comme clef
```

Premiers pas pour développer une App

Prérequis

Pour développer, il vaut mieux disposer des droits maximaux. Seul la personne ayant accès aux bases de données Mysql peut opérer manuellement la modification du niveau de permission dans la colonne 'auth' de la table 'login'.

Ensuite il faut passer en mode dev, via l'url /?dev==.

Quand on est logués à Télex, un nouveau menu apparaît et informe si on est en dev ou en prod.

Être en dev signifie que vous travaillez sur les fichiers du répertoire/prog. Quand vous avez fini, vous pouvez pousser les fichiers modifiés sur le répertoire /prod ; celui auquel accèdent les visiteurs.

Ce dispositif de pré-prod permet surtout le débogage rapide, sinon il vaut mieux travailler sur un autre serveur.

Environnement lexical

lang
Avant de commencer, un petit tour vers les bases lexicales.
Le logiciel est multilingue, et tout ce qui est affiché à l'écran (textes, icônes) provient de tables telles que 'lang', 'helps', 'icons'.
La fonction lang('example') permet d'afficher l'équivalent de 'example' dans la langue utilisateur. Si le terme n'existe pas, il sera créé. Il faut se rendre sur un nouvel onglet, sur la page /app/admin_lang. Les nouveaux termes non définis sont placés en haut. Ensuite il faut informer les autres langues.
la fonction langp() associe un picto avec le terme. (/app/admin/icon).
Ensuite, langpi() place le texte dans un attribut 'title', et langh() efface le texte sur les petits écrans.

help
Comme 'lang', un autre dispositif nommé 'help' (avec /app/admin_help', et la fonction help()), est prévu pour les textes longs, tels que les bulles d'aide ou des descriptifs.
Le dispositif 'help' interprète les connecteurs, ce qui permet aussi d'en faire des menus, qui seront éditables sur place.
hlpbt() affiche une bulle. hlpxt() affiche le texte brut, là où help('text','class') le place dans une div, en option avec une classe.

Environnement logiciel

lib

Pendant le développement, il est opportun de garder sous la main la librairie des principaux éléments de rédaction du code. En premier la page prog/lib. Elle vous expliquera ce que sont les éléments les plus souvent utilisés.

/core

Ensuite, il y a le répertoire /core, qui contient les principaux moteurs logiciels, comme notamment la classe Sql.

commentaires

Pour ce qui est des notes et commentaires, on a prit le parti de ne pas le faire figurer dans le code, premièrement parce qu'ils sont trop périssables, unilangues, lourds (oui le poids ça compte !) et que ça donne de l'arthrite à force de scroller :p
On peut faire figurer les commentaires et explications dans la table sys, via /app/admin_sys, qui permet de stocker tout le code. (Ce dispositif sera utilisé pour le travail à plusieurs, il permettra de recompiler le code à partir des versions les plus récentes de chaque méthode de chaque classe.)

Décliner une app à partir d'un modèle

Pour créer une nouvelle App, il faut partir de l'App 'prod/ap/model'. On peut tout aussi bien décliner une app existante, par exemple app/petitions, pour en faire une variante.
A partir de là, il faut chercher-remplacer toutes les occurrences de 'model' par le nom de la nouvelle App, puis enregistrer sous [nouvelleApp].php.

Ensuite, il faut effacer les méthodes dont on n'aura pas besoin. Les méthodes figurant sur le modèle sont celles qui sont attendues par le framework.

Méthodes d'une App :

```
<?php

/*
To open an App :
- on window, use the url /app/model
- in code, use App::open('model',['method'=>'com','prm1'=>1]);
- using Ajax, use aj(). ex: aj('popup|model,com|prm1=1,prm2=2','click!');
*/

class _model{
#set this auth level required to acceed to this App
#0=public,1:no edition,2:loged user,3,4,5,6:admin,7:superadmin
static $private='0';
#name of mysql table
static $db='model';

#js to append to the header of the parent page (who call this App by Ajax)
//to do that, add the 4th indicator,,,1 in aj()
```

```
//aj('popup,,,1|model,com|prm1=1','click!');
static function injectJs(){
return '';
}

#header to display with the App
//to do that, add param headers=1
//aj('popup|model,com|headers=1,prm1=1','click!');
static function headers(){
Head::add('csscode','.board{
color:#424242; background-color:#dbdbdb; border:1px solid #aaa;
padding:7px 10px; border-radius:2px; box-shadow: 2px 2px 4px #aaa;
}');
Head::add('jscode',self::injectJs());
Head::add('meta',array('attr'=>'property','prop'=>'description','content'=>'object _model for Ph1'));
}

#menus to add to the admin of /app or of popup
static function admin(){//see core/Menus
$r=array('','j','popup|_model,content','plus',lang('open'));
return $r;
}

#install table
//you can update your table here while development
static function install(){
Sql::create(self::$db,array('mid'=>'int','mname'=>'var'),0);}//1=update

#titles to display in popup for each method
static function titles($d){
$d=val($p,'appMethod');
$r['content']='welcome';
$r['build']='model';
if(isset($r[$d]))return lang($r[$d]);//vocable
}

#here is the real code of your app
static function read(){
#Sql called with methode 'v' (simple value)
#note ses('uid') is the loged user.
$r=Sql::read('mname',self::$db,'v','where mid='.ses('uid'));
return $r;
}

#called by telex
static function call($p){
$ret=self::read($p);
return $ret;
}

#interface with other Apps
static function com(){
return $p['msg'].': '.$p['inp1'];
}
```

```
#content
//url: 'app/_model/p1=val1,p2=val2
static function content($p){$ret='';
//self::install();
$p1=val($p,'p1');
$p2=val($p,'p2');
$p['rid']=randid('md');
$p['p1']=val($p,'param',val($p,'p1'));//unamed param
//$ret=hlpbt('underbuilding');
$ret=input('inp1','value1','','1');
$ret.=aj('popup|_model,com|msg=text|inp1',lang('send'),'btn');
$ret=div($ret,'board');
return $ret;}
}
```

Pour appeler une App depuis une autre, utiliser la méthode App::open('model',) permet d'y ajouter les headers qui y sont associés, avant le rendu final.

Dans Télex, les Apps en lecture appellent la méthode call().
En édition, c'est la méthode com() qi est appelée.
Enfin, les Apps utilisées par Télex figurent dans /app/telex/tlxcall, dans le tableau de la méthode menuapps().