Framework PrototypePHP

Comment l'utiliser ?

Éric Quinton

16 septembre 2016

Table des matières

Présentation

Historique

Au début des années 2000, PHP commençait à être largement utilisé pour créer des applications web. Certains frameworks étaient déjà présents, mais ils présentaient souvent des difficultés pour les appréhender et n'étaient pas forcément adaptés aux besoins de l'époque (performance souvent insuffisante en raison d'un chargement systématique de toutes les classes, fonctionnement exclusivement objet, etc.). De plus, ils ne permettaient que difficilement de remplacer certains composants par d'autres.

Des outils comme Smarty, un moteur de templates qui permet de séparer le code HTML du code PHP commençaient à se faire une place. On trouvait également des bibliothèques assez élaborées comme PHPGACL pour gérer les droits de manière particulièrement pertinente.

La gestion des bases de données n'était pas des plus optimales, et un souvent un peu trop conceptuelle.

PrototypePHP a été créé pour assembler divers outils disponibles, selon la conception qu'en avait l'auteur à l'époque. Il était loin d'être parfait et a évolué de multiples fois, pour intégrer une approche mvc, puis des contraintes de sécurité, etc. Toutefois, les fondements de départ sont restés quasiment identiques, même si certaines évolutions ont été intégrées :

- des actions décrites dans un fichier xml, qui est utilisé pour générer le menu en fonction des droits détenus par l'utilisateur ;
- une gestion des droits basée sur PHPGACL. Si le produit initial a été abandonné, sa philosophie a été conservée ;
- une séparation du code PHP et HTML avec l'utilisation de SMARTY ;
- un accès aux tables de la base de données réalisé par l'intermédiaire d'une classe dédiée à cet usage, ObjetBDD, qui contient des fonctions très simples à manipuler, comme ecrire($data), lire($id), supprimer($id). La connexion à la base de données, à l'époque réalisée en utilisant la bibliothèque ADODB, a été remplacée par PDO ;

— un support de l'identification selon quatre modalités : base de données, annuaire LDAP, annuaire LDAP puis base de données, et connexion via un serveur CAS ;
— un souci permanent de la performance, lié au passé de son concepteur [1].

La première version publiée l'a été en 2008, dans sourceforge (`https://sourceforge.net/projects/prototypephp/`). Depuis quelques années, elle est disponible dans github (`https://github.com/equinton/prototypephp`), la branche active étant la branche *bootstrap*, créée au moment du basculement de l'affichage en utilisant les fonctionnalités de ce produit.

Si le principe général d'une conception MVC a prévalu depuis plusieurs années, des améliorations récentes, notamment dans la gestion des vues, a été apportée. À partir de septembre 2016, une meilleure gestion des droits a été implémentée, notamment dans les contextes de travail avec un annuaire d'entreprise LDAP. Il n'est pas impossible également que le support de Shibboleth puisse être intégré dans le futur, notamment quand des bibliothèques prêtes à l'emploi seront disponibles.

Gestion des versions

Le framework est mis à jour en parallèle aux développements de logiciels bâtis à partir de celui-ci. Le code disponible reflète donc les retranscriptions des modifications apportées au gré des évolutions envisagées par son concepteur.

Il n'existe ainsi plus depuis plusieurs mois de gestion de version : le plus simple est de se référer à la date du commit, en utilisant la branche *bootstrap*, qui est celle de travail actuel.

plugins utilisés

Les bibliothèques suivantes sont installées dans le framework :
— pour le code PHP :
  — ObjetBDD (conçu par le développeur du framework), qui gère l'interface avec la base de données ;
  — SMARTY (`http://www.smarty.net`), le moteur de templates ;
  — phpCAS (`https://wiki.jasig.org/display/CASC/phpCAS`), pour la connexion par l'intermédiaire d'un serveur CAS ;
  — et d'autres bibliothèques disponibles dans le framework, mais utilisées uniquement si nécessaire, comme tcpdf (`https://sourceforge.net/`

1. il a commencé sa carrière à une époque où les ressources informatiques étaient rares, chères, et dont la puissance était limitée

projects/tcpdf/files/), odtphp (https://sourceforge.net/projects/odtphp/), openoffice_generation, phpExcelReader (sourceforge.net/projects/phpexcelreader/)...

— pour l'affichage et la conception des pages web, le recours au javascript est omniprésent :
  — JQuery, JQueryUI, et des plugins pour les sélections des dates ;
  — DataTables et ses plugins ;
  — OpenLayers pour l'affichage des cartes ;
  — bootstrap pour la prise en compte de l'affichage sur le mode *responsive* ;
  — ...

Elles sont mises à jour régulièrement, mais il est préférable de vérifier si de nouvelles versions sont disponibles avant de procéder à une mise en production.

Modèle MVC

Le framework est basé sur un modèle MVC, qui présente les caractéristiques suivantes :

— le contrôleur est unique, les actions et les droits associés sont décrits dans un fichier unique ;
— les vues sont héritées d'une classe non instanciable, avec des classes dédiées à l'usage (html via Smarty, ajax, csv pour le moment) ;
— le modèle est constitué de deux types d'objets : des classes héritées d'ObjetBDD pour gérer les échanges avec la base de données, et des fichiers de script exécutant les modules (ou actions) demandés.

Le framework n'a pas une philosophie « tout objet », comme peuvent l'être d'autres, pour tirer parti de la souplesse du php. De nombreuses fonctions permettent de faciliter et limiter le code à écrire.

Quelques classes génériques sont utilisées (une classe Message, par exemple), et l'application recourt fortement aux variables de session.

Terminologie

Dans le framework, l'appel à une action, depuis le navigateur, se nomme un module. Un module se caractérise par un fichier appelé, des droits nécessaires à son exécution, et une action particulière (list, display, change...) qui est traitée spécifiquement dans le fichier appelé.

Cette approche présente des avantages de clarté au niveau du code (relativement peu de fichiers), mais aussi une moindre visibilité au niveau des éditeurs comme

Eclipse : comme les actions ne sont pas déclarées comme des fonctions de classe, la navigation dans le code est parfois une peu plus complexe.

Première partie

Le contrôleur

Chapitre 1

Fonctionnement général

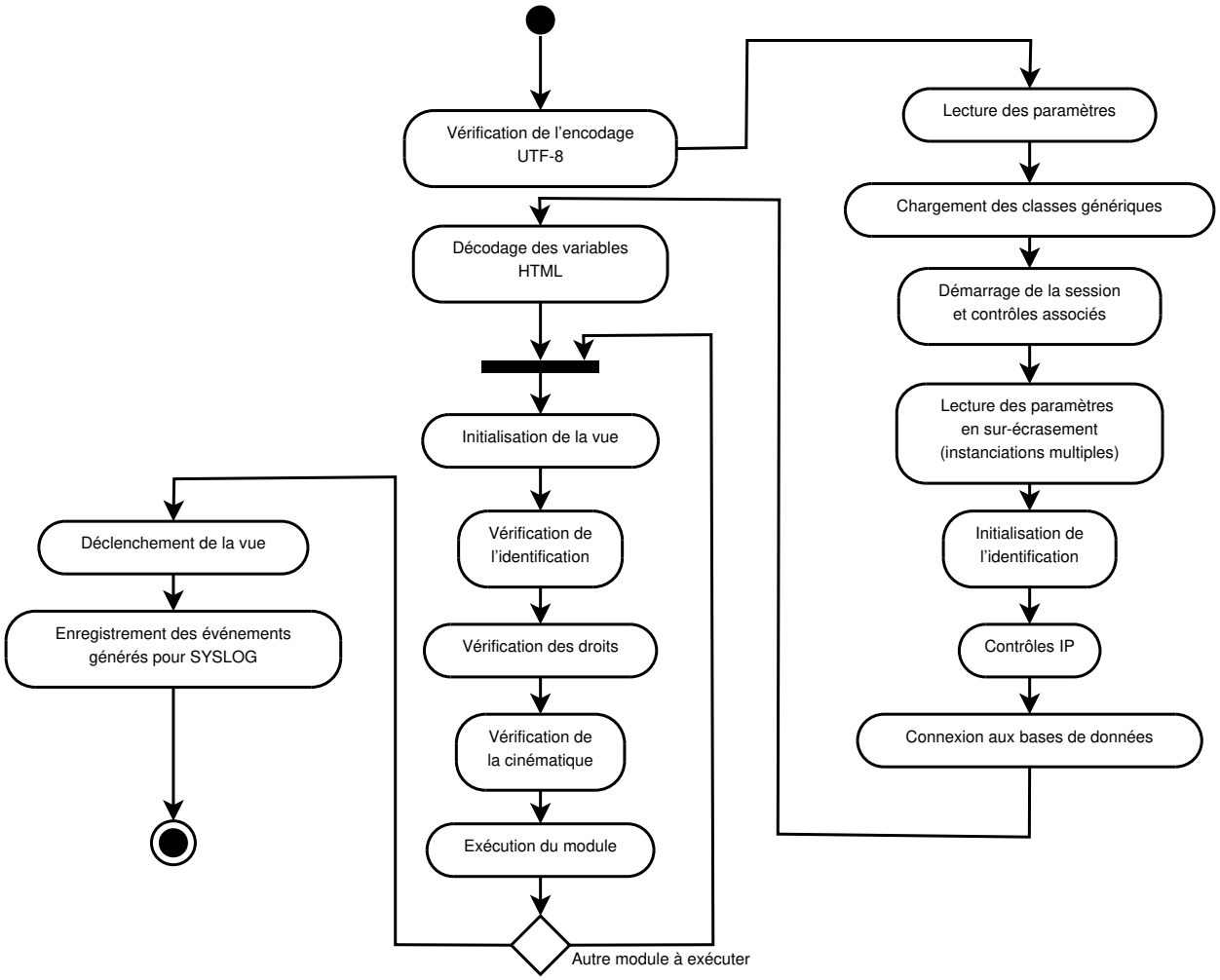

FIGURE 1.1 – Synopsis général de fonctionnement du contrôleur

Synopsis

L'appel de toute page dans l'application passe nécessairement par l'ensemble de ces étapes :

- vérification que l'encodage des caractères transmis respecte bien l'encodage utf-8
- lecture des paramètres
- chargement des classes génériques utilisées systématiquement
- démarrage de la session, et ajout de contrôles (durée de la session ouverte...)
- lecture des paramètres en sur-écrasement, ce qui permet des implémentations multiples avec le même code
- initialisation de l'identification
- contrôles de cohérence IP (vérification que, pour une même session, l'adresse IP ne change pas)
- lancement des connexions aux bases de données (par défaut, deux connexions : une pour la base des droits, l'autre pour les données applicatives)
- décodage des variables HTML encodées (protection contre les attaques de type XSS)
- traitement du module demandé :
  - initialisation, le cas échéant, de la vue associée
  - vérification de l'identification, ou déclenchement des procédures d'identification
  - vérification des droits nécessaires pour accéder au module
  - vérification, le cas échéant, de la cinématique : les opérations de modification ne devraient être possibles que si l'opération précédente correspond à l'affichage du formulaire de saisie
  - exécution du module
  - analyse du code de retour du module, et enchaînement le cas échéant sur un autre module
- déclenchement de la vue
- enregistrement, le cas échéant, des messages destinés à SYSLOG (messages systèmes)

Organisation des dossiers

Les fichiers sont organisés selon cette arborescence :

- **database** : dossier de travail contenant la description de la base de données, la documentation pour les développeurs, les scripts. Le dossier doit être supprimé lors de la mise en production

- **display** : le seul dossier accessible. Il contient tous les fichiers nécessaires pour gérer l'affichage :
  - **CSS** : les feuilles de style
  - **images** : les icônes et images utilisées dans l'affichage des pages
  - **javascript** : l'ensemble des librairies Javascript utilisées
  - **templates** : les modèles de documents utilisés par Smarty (cf. 7, *La vue Smarty*, page 25)
  - **templates_c** : dossier utilisé par Smarty pour compiler les templates. Ce dossier doit être accessible en écriture par le serveur Web
- **doc** : ancien dossier, contenant un mécanisme de gestion de la documentation en ligne. N'est plus utilisé actuellement, mais pourrait être employé le cas échéant
- **framework** : le code de base du framework. Il comprend :
  - **droits** : dossier permettant de gérer les droits
  - **identification** : gestion de la connexion des utilisateurs
  - **import/import.class.php** : classe créée il y a quelques années pour gérer les imports (obsolète en grande partie)
  - **ldap/ldap.class.php** : connexion à un annuaire LDAP et récupération d'informations
  - **navigation** : programmes utilisés pour générer le menu et décoder les actions demandées à partir du fichier XML les contenant
  - **translateId/translateId.class.php** : classe permettant de transcoder les identifiants des enregistrements de la base de données, pour éviter les attaques par forçage de clé
  - de nombreux fichiers utilisés par le framework, dont le contrôleur (controller.php), des fonctions génériques (fonctions.php)...
  - **vue.class.php** : les classes utilisées pour les vues (cf. 6 *Les vues - fonctionnement général*, page 23)
- **install** : contient des scripts d'installation de la base de données (normalement à déplacer dans *database*), et le fichier **readme.txt**, décrivant les dernières nouveautés
- **locales** : dossier contenant les fichiers de langue (fr.php et en.php)
- **modules** : dossier contenant le code spécifique de l'application. Il est organisé ainsi :
  - **classes** : les classes nécessaires pour l'application
  - **example** : des exemples de codage
  - les autres dossiers sont libres et contiennent les modules de l'application
  - **beforeDisplay.php** : fichier appelé systématiquement avant l'affichage des pages HTML

  - **beforesession.inc.php** : fichier appelé systématiquement avant le démarrage de la session. Il permet de déclarer les librairies qui sont nécessaires pour instancier des classes stockées en variables de session
  - **common.inc.php** : fichier appelé systématiquement avant le traitement des modules
  - **fonctions.php** : fonctions déclarées par le programmeur et disponibles dans toute l'application
  - **postLogin.php** : script exécuté uniquement quand un utilisateur s'est identifié
- **param** : dossier contenant les paramètres de l'application :
  - **actions.xml** : fichier contenant la description de l'ensemble des modules utilisables, avec les droits associés et le type de vue à utiliser
  - **menu.xml** : description du menu qui sera généré
  - **param.default.inc.php** : les paramètres par défaut
  - **param.inc.php** : paramètres en écrasement, spécifiques de l'implémentation. Ce fichier n'est jamais livré lors des mises à jour, pour éviter la suppression des paramètres de base de données, par exemple
  - **param.inc.php.dist** : fichier d'exemple de *param.inc.php*, à renommer et à mettre à jour lors de l'installation d'une nouvelle implémentation
- **plugins** : dossier contenant les bibliothèques tierces, comme Smarty, ObjetBDD (maintenant intégré au framework)...
- **temp** : dossier de stockage temporaire, qui doit être accessible en écriture au serveur web. Les fichiers présents dans celui-ci ont une durée de vie de 24 heures (suppression lors de la connexion d'un utilisateur)
- **test** : dossier utilisé pour réaliser certains tests. Doit être systématiquement supprimé lors de la mise en production

Seuls le fichier index.php, à la racine, les dossiers display et test sont accessibles directement. Les autres dossiers sont protégés par des fichiers .htaccess.

Paramètres

Les paramètres utilisés dans l'application sont gérés avec 3 fichiers différents :

- **param/param.default.inc.php** : contient l'ensemble des paramètres utilisés ;
- **param/param.inc.php** : contient ceux issus du fichier précédent, qui sont adaptés à l'implémentation ;
- **param.ini** : fichier contenant les paramètres spécifiques du nom DNS de l'application (par exemple, schéma particulier associé au nom du site). Pour plus d'informations sur ce point, consultez le chapitre 10.2 *Travailler avec plusieurs applications différentes à partir du même code*, page 33.

Voici la description de l'ensemble des paramètres :

Paramètres généraux

Variable	Signification
APPLI_version	Numéro de version de l'application
APPLI_versiondate	Date de la version
language	Langue par défaut
DEFAULT_formatdate	Format par défaut d'affichage des dates
navigationxml	nom du fichier XML contenant la description des modules exécutables
APPLI_session_ttl	durée de la session, en secondes
APPLI_cookie_ttl	durée de vie par défaut des cookies, en secondes
APPLI_path_stockage_session	obsolète
LOG_duree	Durée de conservation des traces des actions réalisées, en jours
APPLI_mail	Adresse pour déclarer les incidents (mail ou non)
APPLI_titre	Nom de l'application qui sera affiché (cas où le code est utilisé par plusieurs entrées différentes)
APPLI_code	Code interne de l'application. Utilisé dans certains cas
APPLI_fds	Feuille de style utilisée par défaut (obsolète)
APPLI_address	Adresse DNS de l'application. Utilisée en cas d'identification CAS (adresse de retour)
APPLI_modeDeveloppement	si à true, certaines opérations sont réalisées dans un contexte de développement (affichage de messages, recalcul systématique du menu...)
APPLI_notSSL	utilisé en développement, si l'application ne fonctionne pas en mode SSL (déconseillé)
APPLI_utf8	systématiquement à true (plus de support des autres encodages)
APPLI_menufile	nom du fichier XML contenant la description du menu
APPLI_temp	nom du dossier utilisé pour stocker les fichiers temporaires
APPLI_moduleDroitKO	nom du module appelé en cas de refus d'accès pour un problème de droits

Variable	Signification
APPLI_moduleErrorBefore	nom du module appelé en cas de problème lié à la cinématique de l'application
APPLI_moduleNoLogin	nom du module appelé en cas d'échec d'identification
paramIniFile	nom du fichier contenant les paramètres spécifiques liés au DNS utilisé (*cf.* 10.2 *Travailler avec plusieurs applications différentes à partir du même code*, page 33)
SMARTY_param	Paramètres utilisés par le moteur de templates SMARTY
SMARTY_variables	variables systématiquement transmises à SMARTY et utilisées lors de l'affichage général
ERROR_display	Affiche les erreurs à l'écran (mode développement)
OBJETBDD_debugmode	0 : pas d'affichage de message d'erreur, 1, affichage des messages d'erreur, 2 : affichage de toutes les commandes SQL générées par ObjetBDD
ADODB_debugmode	obsolète

TABLE 1.1: Variables générales de l'application

Identification

Variable	Signification
ident_type	Type d'identification supporté. L'application peut gérer **BDD** (uniquement en base de données),**LDAP** (uniquement à partir d'un annuaire LDAP) **LDAP-BDD** (d'abord identification en annuaire LDAP, puis en base de données), et **CAS** (serveur d'identification *Common Access Service*)
CAS_plugin	Nom du plugin utilisé pour une connexion CAS
CAS_address	Adresse du serveur CAS
CAS_port	Systématiquement 443 (connexion chiffrée)
LDAP	tableau contenant tous les paramètres nécessaires pour une identification LDAP

Variable	**Signification**
privateKey	clé privée utilisée pour générer les jetons d'identification
pubKey	clé publique utilisée pour générer les jetons d'identification
tokenIdentityValidity	durée de validité, en secondes, des jetons d'identification

TABLE 1.2: Variables utilisées pour paramétrer l'identification

Voici le contenu des variables du tableau LDAP :

Variable	**Signification**
address	adresse de l'annuaire
port	389 en mode non chiffré, 636 en mode chiffré
rdn	compte de connexion, si nécessaire
basedn	base de recherche des utilisateurs
user_attrib	nom du champ contenant le login à tester
v3	toujours à *true*
tls	*true* en mode chiffré
groupSupport	**true** si l'application recherche les groupes d'appartenance du login dans l'annuaire
groupAttrib	Nom de l'attribut contenant la liste des groupes d'appartenance
commonNameAttrib	Nom de l'attribut contenant le nom de l'utilisateur
mailAttrib	Nom de l'attribut contenant l'adresse mail de l'utilisateur
attributgroupname	Attribut contenant le nom du groupe lors de la recherche des groupes (cn par défaut)
attributloginname	attribut contenant les membres d'un groupe
basedngroup	base de recherche des groupes

TABLE 1.3: Variables utilisées pour paramétrer l'accès à l'annuaire LDAP

Connexions aux bases de données

Deux connexions sont systématiquement implémentées : l'une à la base de données contenant la gestion des droits, et l'autre à celle contenant les données propres à l'application.

Variable	**Signification**
BDD_login	compte de connexion à la base de données
BDD_passwd	mot de passe associé
BDD_dsn	adresse de la base de données sous forme normalisée
BDD_schema	schéma utilisé (plusieurs schémas peuvent être décrits, en les séparant par une virgule - fonctionnement propre à Postgresql)
GACL_dblogin	compte de connexion à la base de données des droits
GACL_dbpasswd	mot de passe associé
GACL_dsn	adresse normalisée
GACL_schema	schéma utilisé
GACL_aco	nom du code de l'application utilisé dans la gestion des droits (*cf.* 3 *Identification des utilisateurs et gestion des droits*, page 13)

TABLE 1.4: Variables utilisées pour paramétrer les connexions

Chapitre 2

Décrire les actions

Chapitre 3

Identification des utilisateurs et gestion des droits

Deuxième partie

Le modèle

Chapitre 4

ObjetBDD - accéder aux bases de données

Chapitre 5

Décrire les actions

Troisième partie

Les vues

Chapitre 6

Les vues - fonctionnement général

Chapitre 7

La vue Smarty

Chapitre 8

Les autres vues

La vue Ajax

La vue CSV

La vue PDF

Quatrième partie

Sécurité et implémentation

Chapitre 9

Mécanismes de sécurité et mise en production

Intégrer le transcodage des clés

Chapitre 10

Mise en production

Configuration et installation générale

Configuration du serveur web

Nettoyage de l'application et contrôles à réaliser

Installation de la base de données des droits

Nettoyage des comptes par défaut

Travailler avec plusieurs applications différentes à partir du même code

Dans certains cas, l'application réalisée doit permettre de travailler avec des bases de données différentes selon le contexte, pour éviter de mélanger les informations. La première solution consiste à créer autant de copies que nécessaire du logiciel.

La seconde consiste à n'utiliser qu'un seul code, mais en paramétrant les informations spécifiques à chaque base de données.

Voici le principe général (*cf.* schéma 10.2) :

Dans le paramétrage de l'alias DNS (en principe, dans **/etc/apache2/sites-available**), l'application pointe vers le dossier **/var/www/appliApp/appli1/bin**. */var/www* correspond à la racine du site web, *appliApp* au dossier racine de l'application, *appli1* au dossier spécifique de l'alias DNS.

Ce dossier *appli1* ne contient que deux fichiers : **param.ini**, qui contient les paramètres spécifiques, et **bin**, qui est un lien symbolique vers le dossier **../bin**.

Le dossier *../bin* (donc, dans */var/www/appliApp*) est lui aussi un alias qui pointe vers le code réel de l'application, ici **code_appli**.

Le fichier **param.inc.php** décrit l'entrée suivante :

```
$paramIniFile = "../param.ini";
```

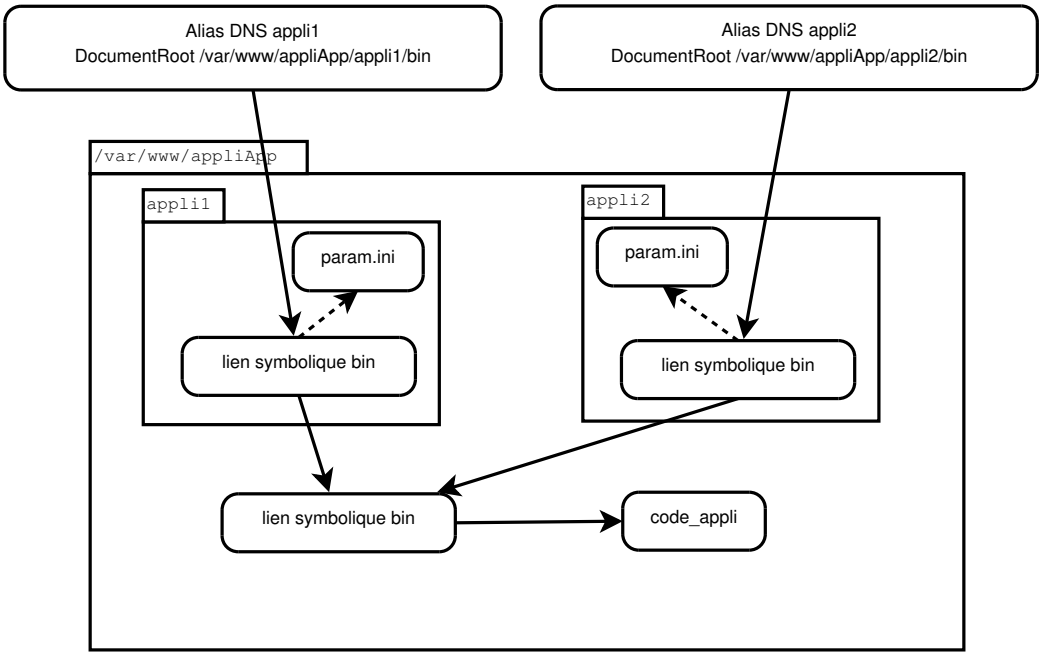

FIGURE 10.1 – Schéma général d'implémentation pour utiliser le même code avec des noms d'application et des jeux de données différents

Le fichier **param.ini** sera cherché dans le dossier parent du code de l'application, c'est à dire soit dans *appli1*, soit dans *appli2* dans cet exemple.

Il suffit qu'il contienne les paramètres adéquats pour rendre l'application utilisable dans des contextes différents à partir du même code initial.