

Maurice Chavelli

DÉCOUVREZ LE FRAMEWORK PHP

LARAVEL



OPENCLASSROOMS

EYROLLES

DÉCOUVREZ LE FRAMEWORK PHP

LARAVEL

Vous pratiquez PHP et vous savez créer des sites ? Vous avez l'impression de réécrire souvent les mêmes choses ? Vous vous posez des questions sur la meilleure façon de traiter une tâche particulière, comme créer des formulaires ou envoyer des e-mails ? Vous aimeriez disposer d'une boîte à outils toute prête pour tout le code laborieux ? Alors vous avez besoin d'un framework PHP et Laravel constitue actuellement ce qui se fait de mieux en la matière !

QU'ALLEZ-VOUS APPRENDRE ?

Les bases de Laravel

- Présentation générale
- Installation et organisation
- Routage et façades
- Les réponses
- Les contrôleurs
- Les entrées
- La validation
- Configuration et session
- L'injection de dépendances

Les bases de données

- Migrations et modèles
- Les ressources
- Ressources pour les utilisateurs et erreurs
- L'authentification
- Les relations 1:n et n:n
- Les commandes et les assistants
- Query Builder

Plus loin avec Laravel

- Environnement et déploiement
- Des vues propres, avec ou sans le conteneur de dépendances
- La localisation
- Ajax
- Les tests unitaires
- Événements et autorisations

À PROPOS DE L'AUTEUR

Maurice Chavelli a commencé l'informatique sur un Sinclair ZX en 1981. Développeur .NET et administrateur réseau, il devient rapidement un acteur très présent sur le Web. Il ouvre en 2013 un blog sur le framework PHP Laravel. Créeur de plusieurs sites web, il découvre Bootstrap lors de sa sortie en 2011 et l'adopte rapidement en pressentant tout son potentiel !

L'ESPRIT D'OPENCLASSROOMS

Des cours ouverts, riches et vivants, conçus pour tous les niveaux et accessibles à tous gratuitement sur notre plate-forme d'e-éducation : www.openclassrooms.com. Vous y vivrez une véritable expérience communautaire de l'apprentissage, permettant à chacun d'apprendre avec le soutien et l'aide des autres étudiants sur les forums. Vous profiterez des cours disponibles partout, tout le temps : sur le Web, en PDF, en eBook, en vidéo...

DÉCOUVREZ LE FRAMEWORK PHP

LARAVEL

DANS LA MÊME COLLECTION

R. DE VISSCHER. – **Découvrez le langage Swift.**

N°14397, 2016, 128 pages.

M. LORANT. – **Développez votre site web avec le framework Django.**

N°21626, 2015, 285 pages.

E. LALITTE. – **Apprenez le fonctionnement des réseaux TCP/IP.**

N°21623, 2015, 300 pages.

M. NEBRA, M. SCHALLER. – **Programmez avec le langage C++.**

N°21622, 2015, 674 pages.

SUR LE MÊME THÈME

R. GOETTER. – **CSS 3 Flexbox.**

N°14363, 2016, 152 pages.

W. MCKINNEY. – **Analyse de données en Python.**

N°14109, 2015, 488 pages.

E. BIERNAT, M. LUTZ. – **Data science : fondamentaux et études de cas.**

N°14243, 2015, 312 pages.

B. PHILIBERT. – **Bootstrap 3 : le framework 100 % web design.**

N°14132, 2015, 318 pages.

C. CAMIN. – **Développer avec Symfony2.**

N°14131, 2015, 474 pages.

S. PITTON, B. SIEBMAN. – **Applications mobiles avec Cordova et PhoneGap.**

N°14052, 2015, 184 pages.

C. DELANNOY. – **Le guide complet du langage C.**

N°14012, 2014, 844 pages.

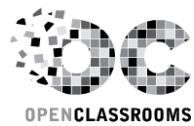
Retrouvez nos bundles (livres papier + e-book) et livres numériques sur
<http://izibook.eyrolles.com>

Maurice Chavelli

DÉCOUVREZ LE FRAMEWORK PHP

LARAVEL

Copyright © 2016 Eyrolles.



OPENCLASSROOMS

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre Français d'exploitation du droit de copie, 20, rue des Grands Augustins, 75006 Paris.

Avant-propos

Vous pratiquez PHP et vous savez créer des sites ? Vous avez l'impression de réécrire souvent les mêmes choses ? Vous vous posez des questions sur la meilleure façon de traiter une tâche particulière comme créer des formulaires ou envoyer des courriels ? Vous aimeriez disposer d'une boîte à outils toute prête pour tout le code laborieux ? Vous devez choisir une solution logicielle pour développer un site dynamique ?

Alors vous avez besoin d'un *framework* PHP. Et Laravel constitue actuellement ce qui se fait de mieux en la matière.

Laravel colle aux plus récentes avancées de PHP et surtout à son approche objet. Découvrir ce framework et plonger dans son code c'est prendre un cours de programmation et d'esthétique. C'est aussi disposer d'une boîte à outils simple et performante pour construire des applications web sans se soucier de l'intendance habituelle.

Laravel est un framework encore jeune qui connaît un succès très rapide, surtout aux États-Unis où il est devenu le plus populaire. La documentation existant actuellement étant essentiellement en langue anglaise, le présent ouvrage a pour objectif de présenter ce framework très prometteur au public français. Il s'adresse principalement aux développeurs et chefs de projets.

Cet ouvrage a été conçu en trois parties progressives qu'il convient de lire dans l'ordre.

- La première partie, particulièrement détaillée, est destinée à vous habituer au framework et à sa philosophie. Elle vous présente les notions essentielles.
- La deuxième partie est axée sur les bases de données, qui constituent la clé des applications dynamiques. Les autres notions seront évidemment développées et complétées au cours de cette partie qui vous demandera plus d'attention et d'expérimentation.
- La troisième partie consolide vos connaissances et détaille quelques spécificités comme la manière d'obtenir des vues épurées, l'utilisation d'Ajax et la localisation. Elle présente la façon de déployer une application Laravel selon le serveur de destination.

Le code est plus fourni et moins détaillé ; il vous faudra l'analyser et le mettre en pratique pour mettre à profit cet apprentissage.



Cet ouvrage nécessite d'avoir des connaissances correctes en PHP, HTML, CSS et JavaScript. Si vous avez des lacunes dans un de ces domaines, vous pouvez facilement les combler avec les nombreux cours à disposition sur le site OpenClassrooms (www.openclassrooms.com) et avec les nombreux ouvrages sur le sujet des éditions Eyrolles (www.editions-eyrolles.com) !

Table des matières

Première partie – Les bases de Laravel	1
1 Présentation générale	3
Un framework ?	3
<i>Approche personnelle</i>	3
<i>(Re)découvrir PHP</i>	4
<i>Définition et intérêt du framework</i>	4
Pourquoi Laravel ?	4
<i>Constitution de Laravel</i>	4
<i>Le meilleur de PHP</i>	5
<i>Documentation</i>	6
Définitions de MVC et POO	6
<i>MVC</i>	6
<i>POO</i>	7
En résumé	8
2 Installation et organisation	9
Composer	9
<i>Présentation</i>	9
<i>Installation</i>	10
<i>Fonctionnement</i>	10
Créer une application Laravel	11
<i>Prérequis</i>	11
<i>Installation avec Composer</i>	12
<i>Installation avec Laravel Installer</i>	13
<i>Autorisations</i>	14
<i>Serveur</i>	14
<i>URL propres</i>	14

L'organisation de Laravel	15
<i>Dossier app</i>	15
<i>Autres dossiers</i>	16
<i>Fichiers à la racine</i>	17
<i>Accessibilité</i>	17
<i>Environnement et messages d'erreur</i>	17
Le composant Html	19
En résumé	20
3 Routage et façades	21
Les requêtes HTTP	21
<i>Petit rappel</i>	21
<i>.htaccess et index.php</i>	22
Le cycle de la requête	22
Plusieurs routes et paramètres de route	24
Erreur d'exécution et contrainte de route	26
Une route nommée	27
Les façades	27
En résumé	29
4 Les réponses	31
Construire une réponse	31
<i>Codes des réponses</i>	31
<i>Vues</i>	32
La vue paramétrée	33
<i>URL</i>	33
<i>Route</i>	34
<i>Vue</i>	34
<i>Simplifier la syntaxe</i>	35
<i>Template</i>	36
Les redirections	38
En résumé	38
5 Les contrôleurs	39
L'utilité des contrôleurs	39
<i>Rôle</i>	39
<i>Constitution</i>	40
<i>Liaison avec les routes</i>	41
<i>Route nommée</i>	42
L'utilisation d'un contrôleur	42
En résumé	43
6 Les entrées	45
Scénario et routes	45

Le middleware	46
Le formulaire	47
Le contrôleur	49
La protection CSRF	51
En résumé	52
7 La validation	53
Scénario et routes	53
Les vues	55
<i>Template</i>	55
<i>Vue de contact</i>	55
<i>Vue de confirmation</i>	58
<i>Vue du courriel pour l'administrateur</i>	59
La requête de formulaire	59
Le contrôleur	62
Envoyer un courriel	63
En résumé	64
8 Configuration et session	65
La configuration	65
Les sessions	67
La requête de formulaire	68
Les routes et le contrôleur	69
Les vues	71
En résumé	74
9 L'injection de dépendances	75
Le problème et sa solution	75
<i>Problème</i>	75
<i>Solution</i>	76
La gestion	78
En résumé	81
Deuxième partie – Les bases de données	83
10 Migrations et modèles	85
Les migrations	85
<i>Configuration de la base</i>	85
<i>Artisan</i>	86
<i>Installer la migration</i>	86
<i>Créer la migration</i>	87
<i>Utiliser la migration</i>	88

Eloquent, un ORM très performant	89
Validation	90
Routes	92
Contrôleur	92
Vues	93
Fonctionnement	94
L'organisation du code	96
Première version	96
Deuxième version	97
Troisième version	98
En résumé	99
11 Les ressources	101
Les données	101
Une ressource	103
Création	103
Routes	105
Contrôleur	106
La validation	107
Création d'un utilisateur	107
Modification d'un utilisateur	108
En résumé	109
12 Ressources pour les utilisateurs et erreurs	111
Le gestionnaire de données (repository)	111
getPaginate	113
getByld	114
show	114
edit	114
update	115
store	116
destroy	116
Les vues	117
Template	117
Vue index	117
Vue show	119
Vue edit	120
Vue create	121
Une réflexion sur le code	124
Gestionnaire de base	124
Modèle	125
Contrôleur	126
Les erreurs	128
En résumé	132
13 L'authentification	133
La commande Artisan	133

Les tables	134
<i>users</i>	134
<i>password_reset</i>	135
Les middlewares	136
<i>Authenticate</i>	136
<i>RedirectIfAuthenticated</i>	137
Routes et contrôleurs	138
<i>Routes</i>	138
<i>Contrôleurs</i>	139
<i>AuthController</i>	139
<i>PasswordController</i>	141
Les vues	141
L'enregistrement d'un utilisateur	143
<i>Validation</i>	143
<i>Contrôleur</i>	144
<i>Vue auth.register</i>	146
Connexion et déconnexion	147
<i>Connexion</i>	147
<i>Vue auth.login</i>	152
<i>Déconnexion</i>	152
L'oubli du mot de passe	153
En résumé	159
14 La relation 1:n	161
Les données	161
<i>Migrations</i>	161
<i>Population</i>	164
La relation	167
Les modèles	168
Contrôleur et routes	170
<i>Contrôleur</i>	170
<i>Routes</i>	171
Le gestionnaire des articles	172
Les middlewares	173
La validation	174
Le fonctionnement	175
<i>Liste des articles</i>	175
<i>Ajout d'un article</i>	176
<i>Suppression d'un article</i>	176
Les vues	177
<i>Template</i>	177
<i>Vue pour afficher les articles</i>	178
<i>Vue pour créer un article</i>	181
En résumé	182

15 La relation n:n	183
Les données	183
<i>Migrations</i>	184
<i>Population</i>	187
Les modèles	189
La validation	190
La gestion	192
<i>Contrôleur et routes</i>	192
<i>Repositories</i>	193
Le fonctionnement	195
<i>Liste des articles</i>	195
<i>Nouvel article</i>	196
<i>Suppression d'un article</i>	198
<i>Recherche par mot-clé</i>	198
Les vues	199
<i>Template</i>	199
<i>Liste</i>	200
<i>Vue de création d'un article</i>	203
En résumé	204
16 Les commandes et les assistants	205
Améliorer une commande	205
Laravel Schema Designer	213
<i>Création des tables</i>	213
<i>Création des champs</i>	214
<i>Création des relations</i>	216
<i>Exportation des fichiers</i>	219
En résumé	222
17 Query Builder	223
Les données	223
<i>Migration</i>	223
<i>Population</i>	225
Les sélections	228
<i>Liste de tous les éditeurs</i>	228
<i>Tableau des valeurs d'une colonne</i>	229
<i>Ligne particulière</i>	229
<i>Colonne isolée</i>	230
<i>Lignes distinctes</i>	230
<i>Plusieurs conditions</i>	231
<i>Encadrer des valeurs</i>	232
<i>Prendre des valeurs dans un tableau</i>	233
<i>Ordonner et grouper</i>	233
Les jointures	234
<i>Trouver les titres des livres pour un éditeur dont on connaît l'identifiant.</i>	234
<i>Trouver les livres d'un auteur dont on connaît le nom</i>	235

Trouver les auteurs pour un éditeur dont on connaît l'identifiant	236
Attention aux requêtes imbriquées	237
En résumé	238
Troisième partie – Plus loin avec Laravel	239
18 Environnement et déploiement	241
L'environnement	241
Le déploiement	243
<i>Besoins de Laravel</i>	243
<i>Solution royale</i>	244
<i>Solution intermédiaire</i>	245
<i>Solution laborieuse</i>	246
<i>Mode maintenance</i>	246
En résumé	247
19 Des vues propres	249
Les macros	249
<i>Problème</i>	249
<i>Définition</i>	250
<i>Fournisseur de services</i>	252
Les templates	255
En résumé	257
20 Des vues propres avec le conteneur de dépendances	259
La nouvelle vue	259
L'organisation du code	260
Les constructeurs	261
<i>FormBuilder et HtmlBuilder</i>	261
<i>PanelBuilder</i>	262
Fournisseur de services et façade	264
En résumé	267
21 La localisation	269
Le principe	269
<i>Façade Lang et classe Translator</i>	269
<i>Fichiers de langue</i>	270
<i>Fonctionnement</i>	271
Le middleware	272
Les dates	275
Route et contrôleur	276
La réalisation de la localisation	278
Les vues	279

Les noms des contrôles	282
En résumé	283
22 Ajax	285
Les vues	285
Template	285
Vue login	288
JavaScript	292
Le traitement	293
Contrôleur	293
Middleware	295
En résumé	296
23 Les tests unitaires	297
L'intendance des tests	298
PHPUnit	298
Intendance de Laravel	298
Environnement de test	300
Construire un test en trois étapes	301
Assertions et appel de routes	302
Assertions	302
Appel de routes et test de réponse	302
Les vues et les contrôleurs	303
Vues	303
Contrôleurs	304
Isoler les tests	305
Simuler une classe	306
Tester une application	308
En résumé	309
24 Événements et autorisations	311
Les événements	311
Événements du framework	312
Fournisseur	313
Créer un observateur	317
Créer un événement	318
Les autorisations	319
Sécurité	319
Différentes autorisations	321
En résumé	324
Index	325

Première partie

Les bases de Laravel

Dans cette première partie, je vous propose de découvrir les bases de Laravel : où le trouver, comment l'installer, quelle configuration est nécessaire pour le faire fonctionner... Je vous présente aussi son organisation et sa philosophie : comment les requêtes sont aiguillées par des routes et dirigées vers des contrôleurs pour être traitées, et comment retourner ensuite une réponse appropriée.

Par ailleurs, vu qu'il est rare qu'une application web n'utilise pas de formulaire, je vous montre la manière d'en créer un, de valider les entrées saisies et de traiter les informations envoyées. La configuration de Laravel étant aussi une application avec ses propres nécessités et contraintes, nous verrons comment elle fonctionne.

Le protocole HTTP est performant mais il permet seulement des requêtes ponctuelles sans persistance. Or, lorsqu'on crée un site, on souhaite permettre aux utilisateurs de se connecter et d'avoir un environnement et des informations personnalisés. Il est donc nécessaire de gérer des sessions, et Laravel est bien équipé pour le faire.

Pour terminer cette partie, la notion d'injection de dépendance, qui est intensivement utilisée dans Laravel, sera présentée avec un exemple pratique d'application.

1

Présentation générale

Dans ce premier chapitre, je vais évoquer PHP, son historique et sa situation actuelle. J'expliquerai aussi l'intérêt d'utiliser un framework pour ce langage et surtout pourquoi j'ai choisi Laravel. J'évoquerai enfin le patron MVC et la programmation orientée objet (POO).

Un framework ?

Approche personnelle

PHP est un langage populaire et accessible. Il est facile à installer et présent chez tous les hébergeurs. C'est un langage riche et plutôt facile à aborder, surtout pour quelqu'un qui a déjà des bases en programmation. On peut réaliser rapidement une application web fonctionnelle grâce à lui. Toutefois, le revers de cette simplicité est que, bien souvent, le code créé est confus, complexe et sans aucune cohérence. Il faut reconnaître que PHP n'encourage pas à organiser son code et rien n'oblige à le faire.

Lorsqu'on crée des applications PHP, on finit par avoir des codes personnels réutilisables pour les fonctionnalités récurrentes, par exemple pour gérer des pages de façon dynamique. Une fois qu'on a créé une fonction ou une classe pour réaliser une tâche il est naturel d'aller la chercher lorsque la même situation se présente. Puisque c'est une bibliothèque personnelle et qu'on est seul maître à bord, il faut évidemment la mettre à jour lorsque c'est nécessaire et c'est parfois fastidieux.

En général, on a aussi une hiérarchie de dossiers à laquelle on est habitué et on la reproduit quand on commence le développement d'une nouvelle application. On se rend compte parfois que cette habitude a des effets pervers, parce que la hiérarchie qu'on met ainsi en place de façon systématique n'est pas forcément la plus adaptée.

En résumé, l'approche personnelle est plutôt du bricolage à la hauteur de ses compétences et de sa disponibilité.

(Re)découvrir PHP

Lorsque j'ai découvert PHP à la fin des années 1990, il en était à la version 3. C'était essentiellement un langage de script, en général mélangé au HTML, qui permettait de réaliser du *templating*, des accès aux données et du traitement. La version 4 en 2000 a apporté plus de stabilité et une ébauche de l'approche objet. Cependant, il a fallu attendre la version 5 en 2004 pour disposer d'un langage de programmation à la hauteur du standard existant pour les autres langages.

Cette évolution incite à perdre les mauvaises habitudes si on en avait. Un site comme <http://www.phptherightway.com> offre de bonnes pistes pour mettre en place de bonnes pratiques. Donc, si vous êtes un bidouilleur de code PHP, je vous conseille cette saine lecture qui devrait vous offrir un nouvel éclairage sur ce langage et, surtout, vous autoriser à vous lancer de façon correcte dans le code de Laravel.

Définition et intérêt du framework

D'après Wikipédia, un framework informatique est un « ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel » ; autrement dit, c'est une base cohérente avec des briques toutes prêtes à disposition. Il existe des frameworks pour tous les langages de programmation et en particulier pour PHP. En faire la liste serait laborieux tant il en existe !

L'utilité d'un framework est d'éviter de passer du temps à développer ce qui a déjà été fait par d'autres, souvent plus compétents, et qui a en plus été utilisé et validé par de nombreux utilisateurs. On peut imaginer un framework comme un ensemble d'outils à disposition. Par exemple, si je dois programmer du routage pour mon site, je prends un composant déjà tout prêt et qui a fait ses preuves et je l'utilise : gain de temps, fiabilité, mise à jour si nécessaire...

Il serait vraiment dommage de se passer d'un framework alors que le fait d'en utiliser un présente pratiquement uniquement des avantages.

Pourquoi Laravel ?

Constitution de Laravel

Laravel, créé par Taylor Otwell, initie une nouvelle façon de concevoir un framework en utilisant ce qui existe de mieux pour chaque fonctionnalité. Par exemple, toute application web a besoin d'un système qui gère les requêtes HTTP. Plutôt que de réinventer quelque chose, le concepteur de Laravel a tout simplement utilisé celui de **Symfony** en l'étendant pour créer un système de routage efficace. De la même manière, l'envoi

des courriels se fait avec la bibliothèque *SwiftMailer*. En quelque sorte, Otwel a fait son marché parmi toutes les bibliothèques disponibles. Nous verrons dans cet ouvrage comment cela est réalisé. Néanmoins, Laravel n'est pas seulement le regroupement de bibliothèques existantes ; c'est aussi un ensemble de nombreux composants originaux et surtout une orchestration de tout cela.

Vous allez trouver dans Laravel :

- un système de routage perfectionné (*RESTFul* et ressources) ;
- un créateur de requêtes SQL et un ORM performants ;
- un moteur de templates efficace ;
- un système d'authentification pour les connexions ;
- un système de validation ;
- un système de pagination ;
- un système de migration pour les bases de données ;
- un système d'envoi de courriels ;
- un système de cache ;
- un système d'événements ;
- un système d'autorisations ;
- une gestion des sessions...
- et bien d'autres choses encore que nous allons découvrir ensemble.

Il est probable que certains éléments de cette liste ne vous évoquent pas grand-chose, mais ce n'est pas important pour le moment ; tout cela deviendra plus clair au fil des chapitres.

Le meilleur de PHP

Plonger dans le code de Laravel, c'est recevoir un cours de programmation tant le style est clair et élégant et le code merveilleusement organisé. La version actuelle de Laravel est la 5.2 et nécessite au minimum la version 5.5.9 de PHP. Pour aborder de façon efficace ce framework, il est souhaitable de se familiariser avec les notions suivantes.

- **Les espaces de noms** : c'est une façon de bien ranger le code pour éviter des conflits de nommage. Laravel utilise cette possibilité de façon intensive. Tous les composants sont rangés dans des espaces de noms distincts, de même que l'application créée.
- **Les fonctions anonymes** : ce sont des fonctions sans nom (souvent appelées *closures*) qui améliorent le code. Les utilisateurs de JavaScript y sont habitués, ceux de PHP un peu moins parce qu'elles y sont plus récentes. Laravel les utilise aussi de façon systématique.
- **Les méthodes magiques** : ce sont des méthodes qui n'ont pas été explicitement décris dans une classe, mais qui peuvent être appelées et résolues.
- **Les interfaces** : une interface est un contrat de constitution des classes. En programmation objet, c'est le sommet de la hiérarchie. Tous les composants de Laravel

sont fondés sur des interfaces. La version 5 a même vu apparaître un lot de contrats pour étendre de façon sereine le framework.

- **Les traits :** c'est une façon d'ajouter des propriétés et méthodes à une classe sans passer par l'héritage, ce qui aide à passer outre certaines limitations de l'héritage simple proposé par défaut par PHP.



Un framework n'est pas fait pour remplacer la connaissance d'un langage, mais pour assister celui (ou celle) qui connaît déjà bien ce langage. Si vous avez des lacunes, il vaut mieux les combler pour profiter pleinement de Laravel.

Documentation

Quand on s'intéresse à un framework, il ne suffit pas qu'il soit riche et performant ; il faut aussi que la documentation soit à la hauteur. C'est le cas pour Laravel. Vous trouverez la documentation sur le site officiel <https://laravel.com/>, mais il existe de plus en plus d'autres sources, dont voici les principales :

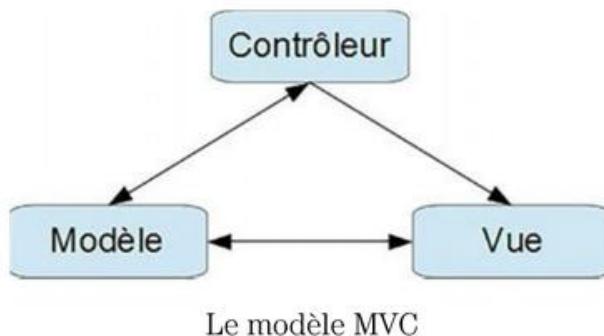
- <https://laravel.fr/> : site d'entraide francophone avec un forum actif ;
- <http://laravel.io/> : le forum officiel ;
- <http://laravel.sillo.org/> : mon blog créé début 2013 et toujours actif, qui constitue une initiation progressive complémentaire du présent ouvrage ;
- <http://cheats.jesse-obrien.ca/> : une page bien pratique qui résume toutes les commandes ;
- <http://www.laravel-tricks.com/> : un autre site d'astuces ;
- <http://packalyst.com/> : le rassemblement de tous les *packages* pour ajouter des fonctionnalités à Laravel ;
- <https://laracasts.com/> : de nombreux tutoriels vidéo en anglais, dont un certain nombre en accès gratuit, notamment une série complète pour Laravel 5 : <https://laracasts.com/series/laravel-5-fundamentals>.

Il existe aussi de bon livres, mais ils sont tous en anglais.

Définitions de MVC et POO

MVC

On peut difficilement parler d'un framework sans évoquer le patron *Modèle-Vue-Contrôleur*. Pour certains, il s'agit de la clé de voûte de toute application rigoureuse ; pour d'autres, c'est une contrainte qui empêche d'organiser judicieusement son code. De quoi s'agit-il ? Voici un petit schéma pour y voir clair.



C'est un modèle d'organisation du code.

- Le *modèle* est chargé de gérer les données.
- La *vue* est chargée de la mise en forme pour l'utilisateur.
- Le *contrôleur* est chargé de gérer l'ensemble.

En général, on résume en disant que le modèle gère la base de données, la vue produit les pages HTML et le contrôleur fait tout le reste. Plus précisément, détaillons cette organisation dans Laravel.

- Le modèle correspond à une table d'une base de données. C'est une classe qui étend la classe `Model`, qui gère simplement et efficacement les manipulations de données et l'établissement automatisé de relations entre tables.
- Le contrôleur se décline en deux catégories : contrôleur classique et contrôleur de ressource (je détaillerai évidemment tout cela dans cet ouvrage).
- La vue est soit un simple fichier avec du code HTML, soit un fichier utilisant le système de templates `Blade` de Laravel.

Laravel propose ce modèle mais ne l'impose pas. Nous verrons d'ailleurs qu'il est parfois judicieux de s'en éloigner, parce qu'il y a plusieurs choses qu'on n'arrive pas à caser dans ce modèle. Par exemple, si je dois envoyer des emails, où vais-je placer mon code ? En général, ce qui se produit est l'inflation des contrôleurs auxquels on demande des rôles pour lesquels ils ne sont pas faits.

POO

Laravel est fondamentalement orienté objet. La POO est un *design pattern* qui s'éloigne radicalement de la programmation procédurale. Avec la POO, tout le code est placé dans des classes qui découlent d'interfaces établissant des contrats de fonctionnement. Avec la POO, on manipule des objets.

Avec la POO, la responsabilité du fonctionnement est répartie dans des classes alors que dans l'approche procédurale tout est mélangé. Le fait de répartir la responsabilité évite la duplication du code qui est le lot presque forcé de la programmation procédurale. Laravel pousse au maximum cette répartition en utilisant l'injection de dépendances.

L'utilisation de classes bien identifiées dont chacune a un rôle précis, le pilotage par des interfaces claires et l'injection de dépendances, tout cela crée un code élégant, efficace, lisible, facile à maintenir et à tester. C'est ce que Laravel propose. Alors, vous

pouvez évidemment greffer là-dessus votre code approximatif, mais vous pouvez aussi vous inspirer des sources du framework pour améliorer votre style de programmation.



L'injection de dépendances est destinée à éviter de rendre les classes dépendantes et à privilégier une liaison dynamique plutôt que statique. Le résultat est un code plus lisible, plus facile à maintenir et à tester. Nous verrons ce mécanisme à l'œuvre dans Laravel.

En résumé

- Un framework fait gagner du temps et donne l'assurance de disposer de composants bien codés et fiables.
- Laravel est un framework novateur, complet, qui utilise les possibilités les plus récentes de PHP et qui est impeccamment codé et organisé.
- La documentation de Laravel est complète, précise et de plus en plus de tutoriels et exemples apparaissent sur la toile.
- Laravel adopte le patron MVC, mais ne l'impose pas. Il est totalement orienté objet.

2

Installation et organisation

Dans ce chapitre, nous allons faire connaissance avec le gestionnaire de dépendances Composer, présenter la création d'une application Laravel et expliquer comment le code est organisé dans cette application.

Avertissement

Pour utiliser Laravel et suivre ce chapitre et l'ensemble de l'ouvrage, vous aurez besoin d'un serveur équipé de PHP avec au minimum la version 5.5.9 et aussi de MySQL. Il existe plusieurs applications « tout-en-un » faciles à installer : wampserver (<http://www.wampserver.com/>), xampp (<https://www.apachefriends.org/fr/index.html>), easypHP (<http://www.easypHP.org/>)... Personnellement, j'utilise wamp, qui répond sans problème à toutes mes attentes et permet de basculer entre les versions de PHP et de MySQL en un simple clic.

Une solution toute prête, Homestead (<https://laravel.com/docs/5.0/homestead>), est facile à mettre en œuvre sous Linux, mais beaucoup moins conviviale avec Windows. Pour ce dernier, il existe une autre possibilité bien pensée : Laragon (<https://laragon.org/>).

Quelle que soit l'application que vous utilisez, vérifiez que vous avez la bonne version de PHP (minimum 5.5.9). En outre, les extensions PDO, Tokenizer, OpenSSL et Mbstring de PHP doivent être activées.

Composer

Présentation

Je vous ai dit que Laravel utilise des composants d'autres sources. Plutôt que de les incorporer directement, il utilise un gestionnaire de dépendances : Composer. D'ailleurs, les composants de Laravel sont aussi traités comme des dépendances. De quoi s'agit-il ?

Imaginez que vous créez une application PHP et que vous utilisez des composants issus de différentes sources : Carbon pour les dates, Redis pour les données... Une

méthode laborieuse consiste à aller chercher tout cela de façon manuelle, et vous allez être confrontés à des difficultés :

- télécharger tous les composants dont vous avez besoin et les placer dans votre structure de dossiers ;
- traquer les éventuels conflits de nommage entre les bibliothèques ;
- mettre à jour manuellement les bibliothèques quand c'est nécessaire ;
- prévoir le code pour charger les classes à utiliser...

Tout cela est évidemment faisable, mais avouez qu'il serait bien agréable d'automatiser ces procédures. C'est justement ce que fait un gestionnaire de dépendances !

Installation

Laravel utilise Composer comme gestionnaire de dépendances. Il vous faut donc commencer par l'installer sur votre ordinateur. Selon votre système, la procédure est différente ; je vous renvoie donc au site <https://getcomposer.org/> pour obtenir tous les renseignements sur le sujet.

Pour Windows, il suffit de télécharger un installateur (<https://getcomposer.org/download/>), qui fait tout très proprement et renseigne aussi la variable d'environnement PATH, ce qui rend Composer utilisable depuis n'importe quel emplacement. En revanche, l'installateur vous demandera où se trouve `php.exe` et vous devrez répondre, car Composer est un fichier PHP et a besoin d'être exécuté.

Pour les autres systèmes, en particulier Linux, le plus simple est d'utiliser `curl`. Il suffit de suivre les instructions détaillées sur le site.



Pour aller plus loin avec Composer, vous pouvez lire l'article suivant : <http://laravel.sillo.org/jouer-avec-composer/>.

Fonctionnement

Pour comprendre le fonctionnement de Composer, il faut connaître le format JSON (*JavaScript Object Notation*). Un fichier JSON a pour but de contenir des informations de type étiquette-valeur. Regardez cet exemple élémentaire :

```
{  
    "nom": "Durand",  
    "prénom": "Jean"  
}
```

Les étiquettes sont "nom" et "prénom" ; les valeurs correspondantes sont "Durand" et "Jean". Les valeurs peuvent être aussi des tableaux ou des objets. Regardez ce second exemple :

```
{  
    "identité1": {  
        "nom": "Durand",  
        "prénom": "Jean"  
    },  
    "identité2": {  
        "nom": "Dupont",  
        "prénom": "Albert"  
    }  
}
```

Composer a besoin d'un fichier `composer.json`, qui contient les instructions nécessaires : les dépendances, les classes à charger automatiquement... Voici un extrait de ce fichier pour Laravel :

```
{  
    "name": "laravel/laravel",  
    "description": "The Laravel Framework.",  
    "keywords": ["framework", "laravel"],  
    "license": "MIT",  
    "type": "project",  
    "require": {  
        "php": ">=5.5.9",  
        "laravel/framework": "5.2.*"  
    },  
    ...  
}
```

Créer une application Laravel

Prérequis

Composer fonctionne en ligne de commande. Vous avez donc besoin de la console (nommée Terminal ou Konsole sur OS X et Linux). Les utilisateurs de Linux sont très certainement habitués à l'utilisation de la console, mais il n'en est généralement pas de même pour les adeptes de Windows. Pour trouver la console sur ce système, il faut chercher l'invite de commande.



Trouver la console dans Windows



Cet ouvrage a été créé avec la version 5.2.* de Laravel. Lorsque vous créez une nouvelle application, que ce soit avec `composer create-project` ou avec l'installateur, vous obtenez la dernière version stable. Sur OpenClassrooms.com, je m'efforcerai de garder ce cours en phase avec l'évolution de Laravel, mais il y aura toujours un délai entre la sortie d'une nouvelle version et cet ouvrage. Si vous rencontrez des différences de fonctionnement avec les exemples utilisés, vous pouvez toujours, en attendant la mise à niveau de l'ouvrage installer la version précédente de Laravel. Il suffit d'utiliser la commande `create-project` en spécifiant la version comme troisième argument (voir la documentation complète sur <https://getcomposer.org/doc/03-cli.md#create-project>).

Installation avec Composer

Il y a plusieurs façons de créer une application Laravel. Celle qui me semble la plus simple consiste à utiliser la commande `create-project` de Composer. Par exemple, si je veux créer une application dans un dossier `laravel5` à la racine de mon serveur, voici la syntaxe à utiliser :

```
| composer create-project --prefer-dist laravel/laravel laravel5
```

L'installation démarre et je n'ai plus qu'à attendre quelques minutes pour que Composer fasse son travail jusqu'au bout. Une liste de téléchargements s'affiche, pour finalement se retrouver avec l'architecture suivante.



Architecture des dossiers de Laravel

On peut vérifier que tout fonctionne bien avec l'URL <http://localhost/laravel5/public>. Normalement, on doit obtenir cette page très épurée :

Laravel 5

Page d'accueil de Laravel

Sous Windows avec Wamp, il est possible d'avoir un souci pour afficher la page d'accueil. Si c'est votre cas, il y a trois solutions.



- N'utilisez pas Wamp mais par exemple Laragon (<http://laragon.org>).
- Créez un hôte virtuel.
- Suivez ce qui est préconisé sur la page <http://stackoverflow.com/questions/15607132/laravel-route-not-working-with-wamp>.

Pour les mises à jour ultérieures, il suffit d'utiliser encore Composer, mais avec la commande update :

```
| composer update
```

Installation avec Laravel Installer

Une autre solution consiste à utiliser l'installateur de Laravel. Il faut commencer par installer globalement l'installateur avec Composer :

```
| composer global require "laravel/installer"
```

Il faut ensuite informer la variable d'environnement path de l'emplacement du dossier `.../composer/vendor/bin`.

Pour créer une application, il suffit de taper :

```
| laravel new monAppli
```

Laravel sera alors installé dans le dossier `monAppli`.

Si vous installez Laravel en téléchargeant directement les fichiers sur Github et en utilisant la commande `composer install`, il vous faut effectuer deux actions complémentaires. Dans ce cas en effet, il ne sera pas automatiquement créé de clé de sécurité et vous allez tomber sur une erreur au lancement. Il faut donc la créer avec la commande `php artisan key:generate`. De plus, vous aurez à la racine le fichier `.env.example` que vous devrez renommer en `.env` pour que la configuration fonctionne.



Autorisations

Au niveau des dossiers de Laravel, le seul qui ait besoin de droits d'écriture par le serveur est `storage`.

Serveur

Pour fonctionner correctement, Laravel a besoin de PHP :

- version $\geq 5.5.9$;
- extension `PDO` ;
- extension `Mbstring` ;
- extension `OpenSSL` ;
- extension `Tokenizer`.

URL propres

Pour un serveur Apache, il est prévu dans le dossier public un fichier `.htaccess` avec ce code :

```
<IfModule mod_rewrite.c>
    <IfModule mod_negotiation.c>
        Options -MultiViews
    </IfModule>

    RewriteEngine On

    # Redirect Trailing Slashes If Not A Folder...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)/$ /$1 [L,R=301]

    # Handle Front Controller...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]

    # Handle Authorization Header
    RewriteCond %{HTTP:Authorization} .
    RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
</IfModule>
```

Le but est d'éviter d'avoir `index.php` dans l'URL. Cependant, pour que cela fonctionne, il faut activer le module `mod_rewrite`.



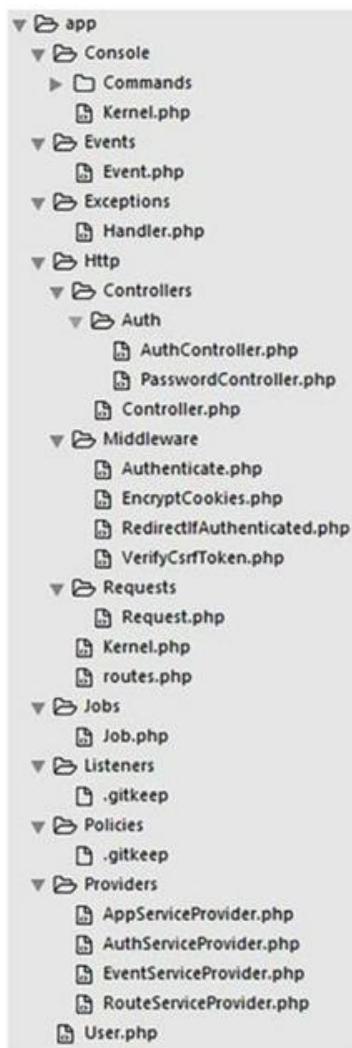
Une autre façon d'obtenir un Laravel sur mesure est d'utiliser mon outil en ligne <http://laravel-designer.sillo.org/>, décrit à l'adresse <http://laravel.sillo.org/laravel-designer/>.

L'organisation de Laravel

Maintenant qu'on a un Laravel tout neuf et qui fonctionne, voyons un peu ce qu'il contient.

Dossier app

Ce dossier contient les éléments essentiels de l'application.



Dossier App

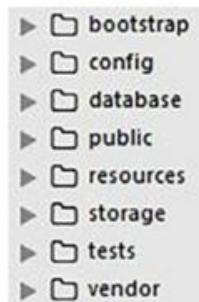
- Console/Commands : toutes les commandes en mode console (la commande Inspire sert d'exemple).
- Jobs : commandes concernant les tâches que doit effectuer l'application (nouveauté de la version 5 que je n'aborderai pas dans cet ouvrage).

- **Events et Listeners** : événements et écouteurs nécessaires pour l’application.
- **Http** : tout ce qui concerne la communication : contrôleurs, routes, *middlewares* (il y a quatre *middlewares* de base) et requêtes.
- **Providers** : tous les fournisseurs de services (quatre au départ), qui servent à initialiser les composants.
- **Policies** : une évolution récente qui facilite la gestion des droits d’accès.
- **Exceptions** : tout ce qui concerne les erreurs d’exécution.

On trouve également le fichier `User.php`, qui est un modèle d’utilisateur pour la base de données.

Évidemment, tout cela doit vous paraître assez nébuleux pour le moment, mais nous verrons en détail la plupart de ces sections au fil de l’ouvrage.

Autres dossiers



Autres dossiers

Voici une description du contenu des autres dossiers :

- `bootstrap` : scripts d’initialisation de Laravel pour le démarrage de l’application, le chargement automatique des classes et la fixation de l’environnement et des chemins ;
- `public` : tout ce qui doit apparaître dans le dossier public du site (images, CSS, scripts, etc.) ;
- `vendor` : tous les composants de Laravel et de ses dépendances ;
- `config` : toutes les configurations (application, authentification, cache, base de données, espaces de noms, emails, systèmes de fichiers, session, etc.) ;
- `database` : migrations et populations ;
- `resources` : vues, fichiers de langage et *assets* (par exemple les fichiers LESS ou Sass) ;
- `storage` : données temporaires de l’application (vues compilées, caches, clés de session, etc.) ;
- `tests` : fichiers de tests unitaires.

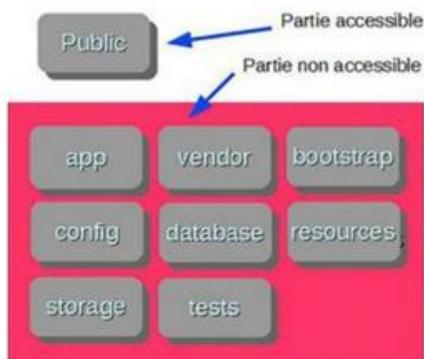
Fichiers à la racine

Il y a un certain nombre de fichiers à la racine. En voici les principaux :

- artisan : outil en ligne de Laravel pour des tâches de gestion ;
- composer.json : fichier de référence de Composer ;
- phpunit.xml : fichier de configuration de phpunit (pour les tests unitaires) ;
- .env : fichier pour spécifier l'environnement d'exécution.

Accessibilité

Pour des raisons de sécurité sur le serveur, seul le dossier public doit être accessible.

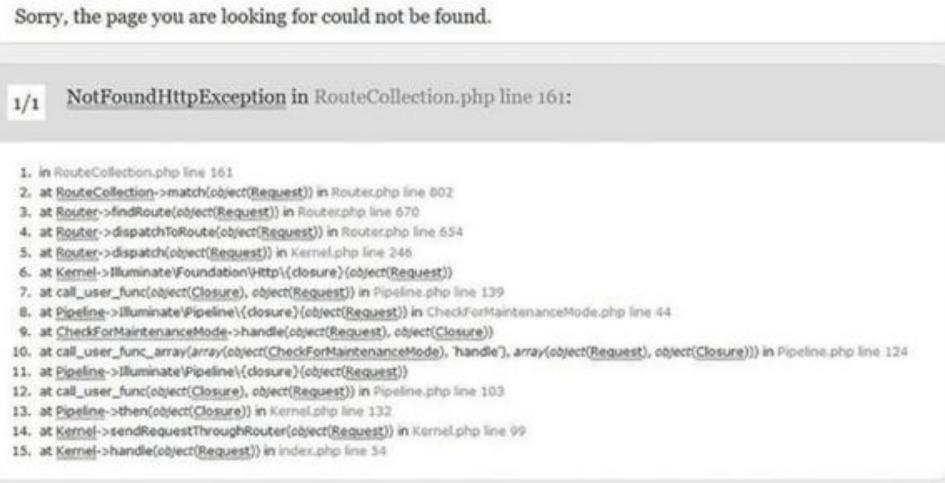


Le dossier public est le seul accessible

Cette configuration n'est pas toujours possible sur un serveur mutualisé. Il faut alors modifier un peu Laravel pour que cela fonctionne ; j'en parlerai dans le chapitre sur le déploiement.

Environnement et messages d'erreur

Par défaut, lorsque vous installez Laravel, celui-ci est en mode *debug*. Concernant l'affichage des erreurs, si vous entrez une URL qui n'est pas prévue, vous obtenez quelque chose comme ceci.

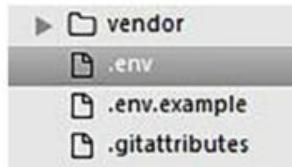


Un message d'erreur en mode debug

Pendant la phase de développement, on a besoin d'obtenir des messages explicites pour traquer les erreurs qu'on commettra inévitablement. En mode *production* au contraire, il faudra cacher tous ces détails. Ouvrez le fichier config/app.php et trouvez la ligne suivante :

```
'debug' => env('APP_DEBUG', false),
```

Autrement dit, on cherchera la valeur dans l'environnement. Regardez à la racine des dossiers, vous y trouvez un fichier .env.



Le fichier de l'environnement

En voici le contenu :

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=base64:/qnv1yuwcCDFJuki9lgc7LBtIzRkJgFxusIX2x1wwUM=
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
```

```
DB_USERNAME=homestead  
DB_PASSWORD=secret  
  
CACHE_DRIVER=file  
SESSION_DRIVER=file  
...
```

Vous remarquez que, dans ce fichier, la variable APP_DEBUG a la valeur true. On la conserve ainsi pour être en mode *debug*, avec affichage de messages d'erreur détaillés. Pour passer en mode *production*, il faut indiquer false (ou supprimer la variable). Avec une URL non prévue, vous obtenez alors juste ce qui suit.

Sorry, the page you are looking for could not be found.

Un message d'erreur en mode production



Il ne faudra évidemment pas laisser la valeur true lors d'une mise en production ! On en reparlera lorsqu'on verra la gestion de l'environnement. Vous ne risquerez ainsi plus d'oublier de changer cette valeur parce que Laravel saura si vous êtes sur votre serveur de développement ou sur celui de production.



La valeur de APP_KEY qui sécurise les informations est automatiquement générée lors de l'installation avec `create-project`.

Le composant Html

Dans la version 4 de Laravel, le composant Html facilitait la création des formulaires et offrait un lot de fonctions dites *helpers* pour faciliter l'écriture du HTML. Dans la version 5, ce composant n'est pas chargé par défaut. Comme nous en aurons besoin, une fois que vous avez réussi à installer une application toute neuve de Laravel, modifiez comme suit le fichier `composer.json` :

```
"require": {  
    "php": ">=5.5.9",  
    "laravel/framework": "5.2.*",  
    "laravelcollective/html": "5.2.*"  
},
```

On demande ainsi à Composer de charger le composant laravelcollective/html (<https://laravelcollective.com/docs/5.2/html>). Lancez alors une mise à jour (attention de bien vous positionner dans le dossier racine de l'application) :

```
| composer update
```

Attendez la fin du chargement. Il faut ensuite modifier comme suit le fichier config/app.php (ajout des lignes contenant Collective\Html) :

```
<?php
/*
 * Application Service Providers...
 */
App\Providers\AppServiceProvider::class,
App\Providers\AuthServiceProvider::class,
App\Providers\EventServiceProvider::class,
App\Providers\RouteServiceProvider::class,
Collective\Html\HtmlServiceProvider::class,
...

'View'=>Illuminate\Support\Facades\View::class,
'Form'=>Collective\Html\FormFacade::class,
'Html'=>Collective\Html\HtmlFacade::class,
```

Ainsi, vous allez disposer de ce composant bien utile !



Le composant utilisé est dérivé de Illuminate/html, qui ne sera plus suivi.

En résumé

- Pour son installation et sa mise à jour, Laravel utilise le gestionnaire de dépendances Composer.
- La création d'une application Laravel se fait à partir de la console avec une simple ligne de commande.
- Laravel est organisé en plusieurs dossiers.
- Le dossier public est le seul qui doive être accessible pour le client.
- L'environnement est fixé à l'aide du fichier .env.
- Le composant Html n'est pas prévu par défaut ; il faut le charger indépendamment.

3

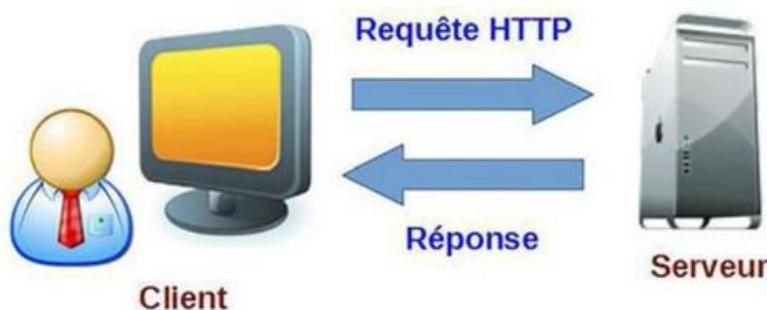
Routage et façades

Dans ce chapitre, nous allons nous intéresser au devenir d'une requête HTTP qui arrive dans notre application Laravel. Nous allons voir l'intérêt d'utiliser un fichier `.htaccess` pour simplifier les URL. Nous présenterons aussi le système de routage pour trier les requêtes.

Les requêtes HTTP

Petit rappel

Commençons par un petit rappel sur ce qu'est une requête HTTP.



Les requêtes HTTP

HTTP (*Hypertext Transfer Protocol*) est un protocole de communication entre un client et un serveur. Le client demande une page au serveur en envoyant une *requête* et le serveur retourne une *réponse*, en général une page HTML.

La requête du client comporte un certain nombre d'informations, mais nous allons nous intéresser pour le moment seulement à deux d'entre elles :

- la *méthode* : `get`, `post`, `put`, `delete` ;
- l'*URL* : c'est l'adresse de la page demandée sur le serveur.

Notre application Laravel doit savoir interpréter ces informations et les utiliser de façon pertinente pour renvoyer ce que demande le client.

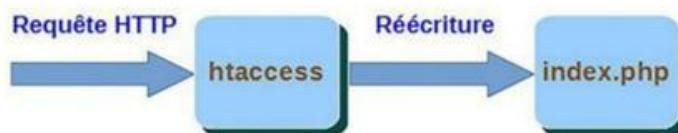
.htaccess et index.php

On veut que toutes les requêtes aboutissent obligatoirement sur le fichier `index.php` situé dans le dossier `public`. Pour y arriver, on peut utiliser une URL de ce genre :

```
http://monsite.fr/index.php/mapage
```

Cependant, ce n'est pas très esthétique avec cet `index.php` au milieu. Si vous avez un serveur Apache, lorsque la requête du client arrive sur le serveur où se trouve votre application Laravel, elle passe en premier par le fichier `.htaccess`, s'il existe, qui fixe des règles pour le serveur. Il y a justement un tel fichier dans le dossier `public` de Laravel, qui contient une règle de réécriture fournissant des URL simplifiées :

```
http://monsite.fr/mapage
```



La réécriture des URL



Pour que cela fonctionne, il faut que le serveur Apache ait le module `mod_rewrite` activé.

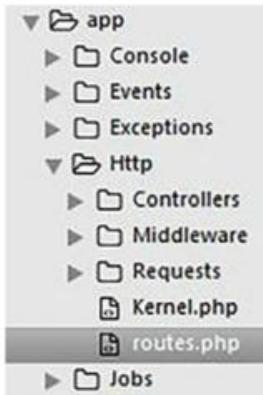


Si vous n'utilisez pas Apache mais Nginx, il faut utiliser la directive suivante :

```
location / {  
    try_files $uri $uri/ /index.php?$query_string;  
}
```

Le cycle de la requête

Lorsque la requête atteint le fichier `public/index.php`, l'application Laravel est créée et configurée, puis l'environnement est détecté. Nous reviendrons plus tard en détail sur ces étapes. Ensuite, le fichier `routes.php` est chargé.



Les fichiers des routes

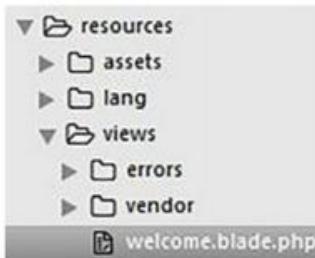
C'est avec ce fichier que la requête sera analysée et dirigée. Regardons ce qu'on y trouve au départ :

```
<?php  
Route::get('/', function () {  
    return view('welcome');  
});
```

Comme Laravel est explicite, vous pouvez déjà deviner à quoi sert ce code :

- `Route` : on utilise le routeur ;
- `get` : on regarde si la requête a la méthode `get` ;
- `' / '` : on regarde si l'URL comporte uniquement le nom de domaine ;
- dans la fonction anonyme, on retourne (`return`) une vue (`view`) à partir du fichier `welcome`.

Ce fichier `welcome` se trouve bien rangé dans le dossier des vues.



La vue `welcome` dans le dossier des vues

C'est ce fichier comportant du code HTML qui génère le texte d'accueil obtenu au démarrage initial de Laravel.



Laravel propose plusieurs fonctions *helpers* qui simplifient la syntaxe, facilitant et accélérant le codage. Il y a par exemple `view` pour la classe `View` comme on l'a vu dans le code précédent.

Visualisons le cycle de la requête sur la figure suivante.



Le cycle de la requête



Sur votre serveur local, vous n'avez pas de nom de domaine et vous allez utiliser une URL de la forme `http://localhost/tuto/public` en admettant que vous ayez créé Laravel dans un dossier `www/tuto`. Vous pouvez aussi créer un hôte virtuel pour avoir une situation plus réaliste.

Plusieurs routes et paramètres de route

À l'installation, Laravel connaît une seule route, qui correspond à l'URL de base composée uniquement du nom de domaine. Voyons maintenant comment créer d'autres routes. Imaginons que nous ayons trois pages qui doivent être affichées avec les URL suivantes :

- `http://monsite.fr/1` ;
- `http://monsite.fr/2` ;
- `http://monsite.fr/3`.

J'ai fait apparaître en gras la partie spécifique de l'URL pour chaque page. Il est facile de réaliser cela avec le code suivant :

```
<?php  
Route::get('1', function() {return 'Je suis la page 1 !';});  
Route::get('2', function() {return 'Je suis la page 2 !';});  
Route::get('3', function() {return 'Je suis la page 3 !';});
```