Développer avec Symfony2

Le plus populaire des frameworks PHP

Clément Camin

EYROLLES

Développer avec Symfony2

C. Camin

Clément Camin est développeur freelance et formateur Symfony2. Il a accompagné des projets variés à différentes étapes de leur développement, de leur conception à leur réalisation en passant par leur maintenance. Il partage depuis de nombreuses années déjà sa passion de Symfony ainsi que des bonnes pratiques du développement logiciel sur son blog **http://blog.keiruaprod.fr.**

Symfony2, édité par le Français SensioLabs, est le framework PHP de référence des développeurs web. Il est à l'origine de nombreuses avancées qui permettent aujourd'hui aux développeurs de créer plus rapidement des applications web de meilleure qualité. Comme tous les outils à la pointe de la modernité, il n'en reste pas moins de prime abord complexe à utiliser, et nécessite un accompagnement pour concevoir des applications à la fois robustes et maintenables.

Un livre de référence pour les développeurs web

L'objectif premier de cet ouvrage est de découvrir les fonctionnalités clés du framework Symfony2, ainsi que les différents composants qui l'accompagnent. Au départ, il sera question de comprendre le fonctionnement des briques de base de toute application web afin d'arriver à développer une application complète : routage, contrôleurs, modèles de vues, gestion de la base de données, etc. L'accent sera ensuite mis sur l'utilisation de bonnes pratiques comme les tests automatisés ou le recours à un conteneur de services, pour concevoir des applications de meilleure qualité, plus faciles à maintenir et à faire évoluer.

Une étude de cas Symfony2 : créer un réseau social

Le développement logiciel s'appréhende par la pratique. Pour cela, vous réaliserez une application de réseau social au fil des chapitres et vous trouverez l'essentiel du code source dans ces pages. Vous irez donc plus loin que la seule utilisation du framework. Grâce à la création d'une application web moderne, vous découvrirez les techniques de développement web actuellement pratiquées. Vous deviendrez ainsi plus efficace dans la réalisation et l'évolution de vos projets, que vous soyez développeur professionnel ou amateur.

Au sommaire

Le choix du framework • Pourquoi utiliser un framework • Pourquoi choisir Symfony2 • **Installer Symfony2** • Installation de la distribution standard • Premiers pas dans notre projet • La notion d'environnement • Exercices • **Le fonctionnement du framework** • À l'intérieur d'une application • **Notre premier bundle** • Le bundle de démonstration • **Routeur et contrôleur** • Routage • Une première route • **Nos premières vues avec Twig** • Symfony2, une architecture MVC • Le moteur de vues Twig • **Faire le lien avec la base de données grâce à Doctrine** • ORM ? • Configurer l'application • Générer notre entité • Les événements de cycle de vie • Génération de CRUD • **Intégration d'un bundle externe** • Utiliser la force de l'open source • Un bundle externe pour les données factices • Notre premier écran « métier » • **Ajout de relations entre les entités** • Précisons notre modèle • Création d'un utilisateur • Lien entre les utilisateurs et leurs statuts • Afficher les utilisateurs dans la timeline • Création des commentaires • **Le dépôt** • Le dépôt (repository) • **La sécurité** • Authentification et autorisation • Installer le bundle FOSUserBundle • **Les formulaires** • La gestion des formulaires • Les différents types de champs • **La validation des données** • Le système des contraintes de validation • Les différentes contraintes • Créer ses propres contraintes • **Les ressources externes : JavaScript, CSS et images** • La problématique des ressources externes • Gestion des ressources avec Assetic • **L'internationalisation** • Le service translator • Traduction des pages statiques de l'application • **Services et injection de dépendances** • Les services dans Symfony • Mise en pratique des services • **Les tests automatisés** • Mise en pratique des tests unitaires et fonctionnels • **Déployer l'application** • Le cycle de développement d'une application web • Spécificités du déploiement d'une application Symfony.

À qui s'adresse cet ouvrage ?

– Aux développeurs et chefs de projet web qui souhaitent découvrir et mieux utiliser ce framework
– Aux développeurs et administrateurs de sites et d'applications web ou mobiles
– Aux étudiants en informatique souhaitant appréhender les techniques du Web

Téléchargez le code source de tous les exemples du livre sur le site www.editions-eyrolles.com

Code éditeur : G14131
ISBN : 978-2-212-14131-3

Conception : Nord Compo

Développer avec Symfony2

Le plus populaire des frameworks PHP

Dans la même collection

S. Pittion, B. Siebman. – **Applications mobiles avec Cordova et PhoneGap.**
N°14052, 2015, 184 pages.

H. Giraudel, R. Goetter. – **CSS 3 : pratique du design web.**
N°14023, 2015, 372 pages.

C. Delannoy. – **Le guide complet du langage C.**
N°14012, 2014, 844 pages.

K. Ayari. – **Scripting avancé avec Windows PowerShell**.
N°13788, 2013, 358 pages.

W. Bories, O. Mirial, S. Papp. – **Déploiement et migration Windows 8**.
N°13645, 2013, 480 pages.

W. Bories, A. Laachir, D. Thiblemont, P. Lafeil, F.-X. Vitrant. – **Virtualisation du poste de travail Windows 7 et 8 avec Windows Server 2012**.
N°13644, 2013, 218 pages.

J.-M. Defrance. – **jQuery-Ajax avec PHP**.
N°13720, 4e édition, 2013, 488 pages.

L.-G. Morand, L. Vo Van, A. Zanchetta. – **Développement Windows 8 - Créer des applications pour le Windows Store**.
N°13643, 2013, 284 pages.

Y. Gabory, N. Ferrari, T. Petillon. – **Django avancé.**
N°13415, 2013, 402 pages.

P. Roques. – **Modélisation de systèmes complexes avec SysML.**
N°13641, 2013, 188 pages.

Sur le même thème

R. Rimelé, R. Goetter. – **HTML 5 – Une référence pour le développeur web**.
N°13638, 2e édition, 2013, 752 pages.

E. Daspet, C. Pierre De Geyer. – **PHP5 avancé**.
N°13435, 6e édition, 2012, 870 pages.

S. Pollet-villard. – **Créer un seul site pour toutes les plates-formes**.
N°13986, 2014, 144 pages.

E. Marcotte. – **Responsive web design**.
N°13331, 2011, 160 pages.

F. Draillard. – **Premiers pas en CSS 3 et HTML 5**.
N°13944, 6e édition, 2015, 472 pages.

Développer avec Symfony2

Le plus populaire des frameworks PHP

Clément Camin

EYROLLES

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

Table des matières

CHAPITRE 1
Introduction **1**

Les objectifs de ce livre 1
L'application fil rouge 3
Un réseau social 3
Les fonctionnalités de l'application 3
Les utilisateurs 3
La timeline 3
Les amis 4
Les commentaires 4
Les différents écrans 4
La page d'accueil 4
L'écran de connexion 4
La timeline d'un utilisateur quelconque 5
La timeline de l'utilisateur connecté 5
La timeline des amis d'un utilisateur 5
Le modèle de données 6
À propos des exemples de code de ce livre 6

CHAPITRE 2
Le choix du framework **7**

Pourquoi utiliser un framework 7
Ne pas réinventer la roue 7
Les clients veulent du code métier 8
La qualité de la base de code 8
L'art des bonnes pratiques 8
Des tests automatisés 9
Des mises à jour et de la maintenance 9
Harmoniser la structure du code 9
Des outils 10
La documentation 10
Une communauté 10
Développer plus vite et mieux 11
Pourquoi choisir Symfony2 11
Il n'y a pas de mauvais choix de framework 11
Symfony2 est un très bon framework 11
Une bonne réputation 12

Les avantages d'un framework mais aussi des composants indépendants 12
Fiable et stable 13
De l'innovation 13
Symfony2 met l'accent sur la sécurité. 13
Les fonctionnalités proposées par Symfony2 14
Des ressources et de l'assistance 14
Une communauté en ébullition 14
Documentation. 15
De l'aide 16

CHAPITRE 3
Installer Symfony2 17

Installation de la distribution standard 17
Prérequis 17
Note sur les installations 18
PHP. 18
Configurer le serveur web et les outils. 19
Git 19
Composer 19
Plusieurs manières de récupérer la distribution standard 21
Distribution standard ? 21
Composer 21
L'installeur. 22
Télécharger une archive de la distribution standard 22
Quelle version choisir ? 23
Installer notre projet 24
Tester l'installation de l'application 25
La configuration du serveur 25
Configurer les permissions d'écriture. 27
La page de démonstration. 28
Mettre à jour Symfony 28
Mettre à jour une version patch 29
Mettre à jour vers une version mineure 29
Premiers pas dans notre projet 30
Application, bundles et bibliothèques 30
Structure de répertoires de l'application 31
À la racine 31
Les fichiers composer.json et composer.lock. 32
Le répertoire vendor 33
Le répertoire web 33
Le répertoire app 33
Le répertoire src 34
La configuration 34
Le format Yaml 34
Fichiers de paramètres et fichiers de configuration 36
Fichier dist 37
La notion d'environnement 37
Contrôleurs de façade 37
La configuration 38

Exercices . . . 39
En résumé . . . 39

CHAPITRE 4
Le fonctionnement du framework . . . **41**
À l'intérieur d'une application . . . 41
Plusieurs bundles . . . 41
Plusieurs environnements . . . 42
Plusieurs outils pour faciliter le développement . . . 42
La barre de débogage . . . 42
La console . . . 43
Le cache interne . . . 44
Les fichiers de logs . . . 45
Un framework basé sur HTTP . . . 46
Fonctionnement du protocole HTTP . . . 46
Requête et réponse dans Symfony2 . . . 48
Le composant HttpFoundation . . . 48
À l'intérieur d'une requête . . . 50
Un moteur événementiel . . . 51
Événement ? . . . 52
La transformation principale, de la requête à la réponse . . . 52
kernel.request . . . 52
kernel.controller . . . 53
kernel.view . . . 53
kernel.response . . . 53
kernel.terminate . . . 53
kernel.exception . . . 53
En résumé . . . 54

CHAPITRE 5
Notre premier bundle . . . **55**
Le bundle de démonstration . . . 55
Le système de bundles . . . 55
Que contient un bundle ? . . . 56
Un bundle vit en autonomie . . . 56
Notre premier bundle . . . 57
Le bundle de démonstration AcmeDemoBundle . . . 57
Nettoyage préliminaire . . . 57
Création du bundle . . . 58
Génération automatisée du bundle avec l'application Console . . . 58
Structure de répertoires . . . 59
Activation du bundle . . . 60
Création manuelle de bundle . . . 61
La classe Bundle . . . 62
Espace de noms et autochargement . . . 62
Enregistrer le bundle . . . 62
Et ensuite ? . . . 63
En résumé . . . 63

CHAPITRE 6

Routeur et contrôleur 65

Routage 65
L'importance d'avoir de belles adresses 66
Localisation des routes 66
Le routage global de l'application 67
Configuration des routes 68
Importance de l'ordre des routes 70
Déboguer les routes 71
Optimiser les routes 72
Une première route 73
Le routage de l'application 73
Structure par défaut du bundle 73
Le contrôleur par défaut 74
Le routage dans notre bundle 74
Le contrôleur 75
Contrôleur 75
HttpFoundation, les bases de Request/Response 75
L'objet Request 76
Manipuler la réponse 78
Le contrôleur de base 80
Générer une vue 81
Gérer les URL 82
Rediriger 82
Gérer le conteneur de services 84
Quelques accesseurs utiles 85
Gérer les erreurs 86
Vérifier les droits de l'utilisateur 86
Manipuler les formulaires. 88

CHAPITRE 7

Nos premières vues avec Twig 91

Symfony2, une architecture MVC 91
Le moteur de vues Twig 92
Une surcouche qui présente de nombreux avantages 93
Trois délimiteurs 93
La page d'accueil 93
Le contrôleur 94
La vue 94
Premier test 95
Passage de paramètres à la vue 95
Structure à trois niveaux 96
Un premier niveau de vues, l'applicatif 96
Un second niveau, le bundle 98
Dernier niveau : la page 99
Validation 100
La page « À propos » 100
Ajout de la barre de navigation 101

Les fonctionnalités de Twig . . . 103
Les opérations de base . . . 103
Afficher des variables . . . 103
Tags . . . 104
La structure de contrôle if . . . 104
Les opérateurs . . . 104
La structure de contrôle for . . . 106
Les tags de gestion de fichier . . . 106
Les tags liés à l'affichage . . . 108
Les macros . . . 109
Les fonctions . . . 110
cycle . . . 110
date . . . 110
min/max . . . 111
random . . . 111
range . . . 111
parent . . . 112
Fonctions spécifiques à Symfony . . . 113
Les filtres . . . 113
Les filtres sur les chiffres . . . 113
Les filtres sur les chaînes . . . 115
Les filtres sur les dates . . . 118
Les filtres sur les tableaux . . . 119
Les filtres divers . . . 123
En résumé . . . 124

CHAPITRE 8
Faire le lien avec la base de données grâce à Doctrine..... 125

ORM ? . . . 125
Doctrine . . . 126
De gros modèles et des contrôleurs légers . . . 127
Configurer l'application . . . 128
parameters.yml et parameters.yml.dist . . . 128
Configurer l'application et la base de données . . . 128
Générer notre entité . . . 130
Qu'est-ce qu'une entité ? . . . 130
L'entité Status . . . 131
Créer une entité avec l'application console . . . 132
Annotations . . . 135
D'autres formats de configuration de l'entité . . . 136
L'entité . . . 136
Informations d'association au format Yaml . . . 136
Informations d'association au format XML . . . 137
Informations d'association en PHP . . . 137
Quel format de configuration choisir ? . . . 138
Modification manuelle de l'entité . . . 138
Mettre à jour le schéma de base de données . . . 140
Attention ! . . . 141
Les événements de cycle de vie . . . 141

Modification de l'entité 142
Mise à jour de l'entité 143
Ajout de deux événements 144
Mise à jour du schéma de base de données 144
Génération de CRUD 145
Qu'est-ce que le CRUD ? 145
La force des générateurs 145
Une force dont nous allons nous passer 148

CHAPITRE 9
Intégration d'un bundle externe 151
Utiliser la force de l'open source 151
Beaucoup de fonctionnalités ont déjà été codées ! 151
Les avantages de l'open source 152
Les inconvénients 153
Comment choisir les composants ? 154
Un bundle externe pour les données factices 155
Utiliser des données factices 155
Le problème 155
Notre solution, DoctrineFixturesBundle 156
De nombreux avantages 156
DoctrineFixturesBundle 157
Mise en place de DoctrineFixturesBundle 157
Installation du bundle 157
Activer le bundle 158
Notre classe d'import des données 159
Le gestionnaire d'entités (entity manager) 160
Charger nos deux données factices 161
De meilleures données factices 161
Faker, la bibliothèque de génération de données aléatoires 162
Notre premier écran « métier » 163
La route 164
Le contrôleur 164
La vue 165

CHAPITRE 10
Ajout de relations entre les entités 167
Relations entre entités 167
One To Many et son inverse Many To One 168
Le modèle du pays 168
Le modèle de la ville 169
Code SQL associé 170
Many To Many 170
Le modèle du client 170
Le modèle du code de réduction 171
Code SQL associé 172
One To One 173
Le modèle de l'entreprise 173

Le modèle pour l'adresse . . . 174
Code SQL associé . . . 175
Manipuler les entités liées . . . 175
Précisons notre modèle . . . 176
Création d'un utilisateur . . . 177
Création de l'entité . . . 177
Lien entre les utilisateurs et leurs statuts . . . 178
Création de la relation One To Many dans l'entité utilisateur . . . 178
Création de la relation Many To One dans l'entité Status . . . 179
Mettre à jour le schéma de base de données . . . 180
Modification des données factices . . . 181
Modification des utilisateurs de test . . . 181
Modification des statuts de test . . . 182
Afficher les utilisateurs dans la timeline . . . 183
Timeline d'un seul utilisateur . . . 188
Les amis . . . 191
Modification de l'entité utilisateur . . . 191
Relation unidirectionnelle . . . 191
Relation bidirectionnelle . . . 192
Relation unidirectionnelle ou bidirectionnelle ? . . . 193
Modification du constructeur . . . 193
Mise à jour des entités . . . 194
Ajout d'une méthode utilitaire dans l'entité . . . 195
Mise à jour du schéma de base de données . . . 196
Création de données factices pour les amis . . . 196
Modification de la timeline utilisateur . . . 198
Création des commentaires . . . 199
Création de l'entité . . . 199
Ajout des informations d'association dans l'entité . . . 200
Modification des associations dans les autres entités . . . 201
Mise à jour des entités . . . 202
Ajout d'événements de cycle de vie . . . 202
Mise à jour du schéma de base de données . . . 203
Création des données factices pour les commentaires . . . 204
Affichage des commentaires dans la timeline d'un utilisateur . . . 205

CHAPITRE 11
Le dépôt . . . 207

Le dépôt (repository) . . . 207
Le principe du dépôt . . . 207
Le dépôt par défaut . . . 208
Lazy loading et eager loading . . . 208
Le lazy loading (chargement retardé) . . . 208
L'eager loading (chargement précoce) . . . 209
Création et utilisation d'un dépôt dédié . . . 209
Modifier les contrôleurs . . . 210
Modification de l'entité . . . 211
Le dépôt des statuts . . . 211

Écriture de requêtes dans le dépôt 212
Plusieurs manières d'écrire les requêtes 212
DQL 212
QueryBuilder 215
Ajout de paramètres dans les requêtes 216
Obtenir la timeline d'un utilisateur 218
La timeline des amis d'un utilisateur 221

CHAPITRE 12
La sécurité 225

Authentification et autorisation 225
Authentifier l'utilisateur 226
Comprendre le fichier security.yml 226
Configuration initiale 226
Créer les premières routes 227
Forcer l'authentification pour accéder à la page sécurisée 229
Configurer les utilisateurs 230
Se déconnecter 231
Autoriser l'accès 233
Obtenir l'utilisateur courant 233
Gérer les règles d'accès dans les vues 234
Gérer les règles d'accès dans les contrôleurs 235
Sécuriser les objets 236
Installer le bundle FOSUserBundle 237
Télécharger le code et activer le bundle 237
Activer le système de traduction 238
Créer la classe d'utilisateur 238
Configurer la sécurité 239
Configurer l'application 241
Mettre à jour le schéma de la base de données 241
Inclure les routes 242
Vérifier que tout fonctionne bien 243
Créer des données de test 243
Mettre à jour le menu utilisateur 245
Faire pointer les liens vers les bonnes pages 247
Tester si l'utilisateur est connecté 247
Afficher des liens différents si l'utilisateur est connecté 248
Surcharger les templates de FOSUserBundle 250
Gérer les rôles des utilisateurs 252
Se faire passer pour un utilisateur 252

CHAPITRE 13
Les formulaires 255

La gestion des formulaires 255
Un composant dédié 256
La sécurité 256
Une manipulation en quatre étapes 256
Étape 1 - Créer la classe de formulaire 257

Étape 2 - Contrôleur pour l'affichage . . . 257
Étape 3 - Un modèle pour l'affichage . . . 258
Étape 4 - Traiter le formulaire . . . 258
Création de l'objet formulaire . . . 259
Création de formulaire depuis le contrôleur . . . 259
Utilisation d'une classe de description de formulaire . . . 260
Une entité n'est pas forcément stockée dans la base de données . . . 261
Les différents types de champs . . . 262
Spécifier le type de champ . . . 262
Configurer les différents types de champs . . . 263
Champs de texte . . . 263
Champs de choix . . . 265
Champs de type date et heure . . . 266
Champs de type Groupe . . . 267
Boutons . . . 267
Champs divers . . . 268
Affichage du formulaire . . . 268
Affichage brut du formulaire . . . 268
Affichage élément par élément . . . 268
Gestion fine de l'affichage des différentes propriétés des champs . . . 269
Les variables de formulaire . . . 271
Suggestion de fonctionnement . . . 271
Validation de formulaire . . . 272
Deux types de validation . . . 272
Validation côté client . . . 272
Validation côté serveur . . . 273
Ajout de la possibilité de poster un statut . . . 273
Création de l'objet formulaire . . . 273
Afficher le formulaire . . . 275
Gérer la soumission de formulaire . . . 278

CHAPITRE 14
La validation des données . . . 281
Le système des contraintes de validation . . . 281
Des données invalides de plusieurs manières . . . 281
Les contraintes de validation, garantes des règles métier . . . 282
Valider une entité . . . 283
Depuis un formulaire . . . 283
Sans formulaire . . . 284
Les différentes contraintes . . . 285
Contraintes de base . . . 285
NotBlank . . . 285
NotNull . . . 285
True . . . 286
False . . . 286
Type . . . 286
Contraintes sur les nombres . . . 286
EqualTo, NotEqualTo, IdenticalTo, NotIdenticalTo . . . 286

LessThan, LessThanOrEqual, GreaterThan, GreaterThanOrEqual 287
Range 287
Contraintes sur les dates 288
Date 288
DateTime 288
Time 288
Contraintes sur les chaînes de caractères 288
Length 288
Regex 289
Email 289
Url 290
Ip 290
Contraintes de localisation 291
Locale 291
Language 291
Country 291
Contraintes financières 292
CardScheme 292
Luhn 292
Iban 293
Contraintes sur des identifiants standardisés 293
Isbn 293
Issn 294
Contraintes sur les collections 294
Choice 294
Count 295
UniqueEntity 295
Contraintes sur les fichiers 296
File 296
Image 297
Contraintes inclassables 297
Callback 297
UserPassword 297
Valid 298
All 298
Collection 298
Créer ses propres contraintes 298
Créer une nouvelle contrainte 299
Créer le validateur pour la contrainte 299
Utiliser la nouvelle contrainte dans l'application 301
Créer une contrainte de validation pour une classe 302
Mise en pratique dans notre application 303
Les entités Status et Comment 303
L'entité User 304

CHAPITRE 15
Les ressources externes : JavaScript, CSS et images 307

La problématique des ressources externes 307
Ressources de développement et ressources déployées 308
Exposition des ressources 308

Utilisation des ressources avec la fonction asset . . . 310
Gestion des ressources avec Assetic . . . 310
Inclusion des ressources . . . 310
Inclusion de fichier CSS . . . 310
Inclusion de fichier JavaScript . . . 311
Inclusion d'image . . . 311
Nom du fichier de sortie . . . 311
Le cache busting . . . 312
Les filtres d'Assetic . . . 313
Automatiser la transformation entre fichiers . . . 314
Réécriture d'URL dans les fichiers CSS . . . 314
Utilisation de filtres Node.js . . . 315
Utilisation de filtres Java . . . 317
Utilisation de filtres PHP . . . 318
Filtres et développement . . . 320
Génération à la volée ou non . . . 320
Export des ressources pour la production . . . 321
Mise en pratique . . . 321
Gestion des fichiers CSS dans notre bundle . . . 321
Mise à jour des vues . . . 321
Publication des ressources du bundle . . . 323
Création du style . . . 324
Inclusion du style . . . 324
Activation d'Assetic . . . 325
JavaScript : ajout d'un utilisateur comme ami . . . 326
Ajout du bouton . . . 326
L'API d'ajout/suppression d'un ami . . . 328
Exposition des routes côté JavaScript . . . 329
Exposition des fichiers JavaScript . . . 330
Inclusion du fichier de routage JavaScript dans la vue . . . 330
Exposition des routes . . . 331
Les ressources nommées . . . 332

CHAPITRE 16
L'internationalisation . . . 335

Le service translator . . . 335
Activation du système de traduction . . . 335
Fonctionnement de la traduction . . . 336
Les clés de traduction . . . 336
Algorithme de traduction et langue de secours . . . 336
Les dictionnaires . . . 337
Différents formats . . . 337
Les clés de traduction . . . 339
Génération des traductions . . . 340
Localisation des fichiers de traduction . . . 341
Utilisation du service de traduction . . . 341
Traduction simple . . . 341
Traduction avec paramètres . . . 342
Pluralisation . . . 343
Utilisation du domaine . . . 344

Traduction dans les contrôleurs . . . 345
Les locales . . . 346
Obtention de la locale courante . . . 346
Locale de fallback . . . 346
Traduction des pages statiques de l'application . . . 346
Modification des vues . . . 347
Création des dictionnaires . . . 348
Test . . . 350
Traduction du menu . . . 351
Modification des vues . . . 351
Traduction au niveau applicatif . . . 352
Localisation des routes . . . 353
Modification de toutes les routes pour y inclure la locale . . . 353
Installation de JMSI18nRoutingBundle . . . 353
Configuration de la sécurité . . . 355
Test . . . 356
Recherche des traductions manquantes . . . 356

CHAPITRE 17
Services et injection de dépendances . . . 357

Les services dans Symfony . . . 357
Les services . . . 358
Conteneur de services . . . 358
L'injection de dépendances . . . 358
Conteneur d'injection de dépendances . . . 359
Enregistrement des services dans le conteneur . . . 359
Le choix du format . . . 359
Notre service d'exemple . . . 360
Structure du fichier de configuration . . . 360
Déclaration d'un service . . . 361
Arguments du constructeur . . . 361
Injection de l'accesseur set . . . 362
Déclaration de paramètres de configuration . . . 363
Utilisation des paramètres de configuration . . . 364
Configuration finale du service . . . 365
Quelques bonnes pratiques autour de l'injection de dépendances . . . 366
Quelques services courants . . . 367
templating, le service de rendu des vues . . . 367
request_stack, la pile de requêtes . . . 368
doctrine.orm.entity_manager . . . 369
mailer . . . 369
logger . . . 369
router . . . 370
translator . . . 370
security.context . . . 370
Et les autres ? . . . 370
Mise en pratique des services . . . 371
Mieux découper un contrôleur . . . 371
Ajout de la couche de service . . . 371

Mise en place dans le UserController 372
Création du service 373
Spécification du fichier de configuration 375
Déclaration du service 375
Utilisation d'un service en tant que contrôleur 378
Solution 1 – Extension du contrôleur de FrameworkBundle 378
Solution 2 – Héritage de ContainerAware 378
Solution 3 – Utilisation des services en tant que contrôleurs 379
Création de filtres Twig 382
Se brancher sur un événement 387
La configuration finale 390

CHAPITRE 18
Les tests automatisés 391

Tests automatisés 391
Test unitaire 391
Test fonctionnel 392
Test fonctionnel ou test unitaire ? 392
Quelques a priori 392
Écrire des tests automatisés est long 392
Écrire des tests automatisés n'est pas intéressant 392
Je n'ai pas besoin d'écrire de tests automatisés, mon code marche 393
Mon code n'est pas testable 393
Conclusion 393
PHPUnit 394
Installation de PHPUnit 394
Configuration de l'environnement de test 395
Mise en pratique 395
Mise en pratique des tests unitaires 396
Tester un filtre Twig 396
Mise en place du test 396
Écriture d'un premier test : le cas nominal 397
Exécution du test 398
Des tests comme spécifications 398
D'autres assertions 399
Second test unitaire : conditions aux limites 399
Un troisième test : le cas d'erreur 400
Bilan de cette série de tests 401
Test unitaire du filtre since 401
Test unitaire d'un contrôleur 401
Le cas nominal 402
Un premier cas d'erreur : pas d'utilisateur authentifié 404
Un second cas d'erreur : l'ajout d'ami échoue 405
Mise en pratique des tests fonctionnels 406
Test fonctionnel du filtre since 406
Préparation du test 406
Écriture du test 407
Test du comportement de la page d'accueil 409
Naviguer et manipuler la page 410
Le client 410

Le DomCrawler 410
Cliquer sur les liens 411
Remplir les formulaires 412
Apprendre à manipuler ces composants 412
Test fonctionnel de la redirection après authentification 412

CHAPITRE 19
Déployer l'application 415
Le cycle de développement d'une application web 415
Les différentes étapes du développement d'une application web 415
Le serveur de développement 416
Le serveur de staging (« mise en scène ») 416
Le serveur de préproduction 416
Le serveur de production 416
Différents cycles de vie 416
Le déploiement 417
Le but du déploiement 417
Les enjeux de l'automatisation 418
Vers l'automatisation 419
Des outils 419
Capistrano ou Capifony 420
Processus de déploiement 423
Spécificités du déploiement d'une application Symfony 424
Préparer le serveur de production 424
Assurer la compatibilité avec Symfony 424
Prévoir le répertoire qui expose le projet 424
Rotation des logs 424
Préparer le déploiement 425
Configurer la clé secrète 425
Remplacer les écrans d'erreur 425
Configuration de production 426
Cache Doctrine 426
Accès à l'environnement de développement depuis un serveur distant 426
Exécuter la Console depuis une interface web 427
Effectuer le déploiement 427
Déploiement du code 428
Déploiement des ressources 428
Mise à jour de la base de données 428
Videz le cache 428
Bilan 429

Conclusion 431
Notions clés 431
Fonctionnement du framework 431
Création de projets 432
Configuration 432
Outils Symfony 432
Contrôleurs 433
Routage 433

Vues 433
Doctrine 434
Logique métier 434
Formulaires 434
Internationalisation 435
Sécurité 435
Ressources externes 435
Tests 435
Déploiement 436
Pour aller plus loin 436

Index 439

1

Introduction

Symfony2 est un framework très populaire. Son point fort est qu'il sait répondre à une grande quantité de besoins différents liés à la conception d'applications web. En corollaire, le panel des possibilités est très étendu et il est difficile de s'y retrouver seul. Ce n'est plus votre cas ! Grâce à ce livre, vous serez guidé dans chacune des étapes clés de la conception d'une application web, de son installation jusqu'à son déploiement. Commençons par faire un tour d'horizon de ce dont nous parlerons tout au long de l'ouvrage et présentons l'application que nous allons concevoir pour illustrer les exemples.

Les objectifs de ce livre

Vous allez bien sûr apprendre à vous servir de Symfony2, mais vous irez plus loin que la seule utilisation du framework : vous découvrirez aussi les techniques de développement web actuellement pratiquées, ce qui vous rendra plus efficace dans la réalisation de vos projets, que vous soyez développeur professionnel ou amateur.

Le but principal de ce livre est de vous faire découvrir les fonctionnalités clés du framework Symfony2, ainsi que les différents composants qui l'accompagnent. Nous partirons de zéro, c'est-à-dire de l'installation d'une application. Petit à petit, nous découvrirons les différents éléments et construirons progressivement une application selon les standards actuels. À la fin de ce livre, nous expliquerons comment déployer l'application pour que des utilisateurs puissent y accéder, ce qui est l'aboutissement d'une application web.

Nous expliquerons entre autres comment :

- mettre en place une application Symfony2 et la maintenir ;
- utiliser les bundles et améliorer son application à l'aide des bundles conçus par la communauté ;
- comprendre l'architecture MVC (Modèle Vue Contrôleur) et son fonctionnement dans Symfony ;
- faire fonctionner le système de routage, capable de gérer n'importe quel type de requête HTTP ;
- concevoir des modèles de page grâce à Twig, et utiliser toutes les fonctionnalités offertes par le moteur pour bien structurer les vues ;
- créer tout type de modèle de données et d'associations avec Doctrine 2, pour manipuler nos objets métier et faire le lien avec la base de données ;
- structurer la gestion de la base de données grâce aux *repositories* ;
- construire, afficher et traiter des formulaires ;
- valider l'intégrité des données manipulées ;
- comprendre le fonctionnement de la couche de sécurité de Symfony2 ;
- localiser nos applications pour faire des sites multilingues ;
- gérer et manipuler les ressource externes, JavaScript et CSS, aux différentes étapes de leur cycle de vie ;
- utiliser le système de services et découvrir l'intérêt de l'injection de dépendances ;
- bien structurer son application en la découpant en petites unités qui ont chacune une responsabilité unique ;
- écrire et utiliser des tests automatisés, aussi bien unitaires que fonctionnels, afin d'assurer et d'améliorer la qualité de l'application ;
- déployer son application, pour la mettre en ligne mais aussi pour gérer efficacement notre processus de développement et de déploiement.

Le développement logiciel se comprend par la pratique. C'est pourquoi nous essaierons de fournir des exemples concrets le plus souvent possible. Nous réaliserons une application au fur et à mesure des chapitres et vous trouverez l'essentiel du code source dans ces pages.

L'accent sera mis sur les bonnes pratiques : parmi les différentes possibilités d'aboutir à un résultat, nous mettrons en place celle qui donnera le code le plus pérenne et le plus facile à maintenir. Ce point est parfois subjectif et sujet à critique. Ce qui est vrai à un moment de la vie du projet ou du framework ne le sera pas toujours, mais nous nous efforcerons de vous présenter du code dans cette direction.

Lorsque ce sera pertinent, des exercices pratiques vous permettront d'aller plus loin dans les connaissances acquises.

En fait, l'idée essentielle de ce livre est d'arriver à vous rendre autonome avec Symfony2. Nous vous apprendrons à utiliser les différentes possibilités offertes par les composants essentiels du framework, tout en vous transmettant la philosophie cachée derrière certains choix d'architecture pour faire de vous un meilleur développeur.

La complexité du framework est telle que ce livre ne peut en couvrir tous les aspects. Il a donc été nécessaire de faire des choix et d'omettre certains composants ou fonctionnalités. Vous aurez cependant les connaissances nécessaires pour découvrir et comprendre seul ce qui n'a pas été abordé et juger s'il est pertinent de l'intégrer dans vos projets.

L'application fil rouge

Nous allons développer une application tout au long de ce livre. Chaque chapitre introduira de nouvelles notions, que nous mettrons en pratique en ajoutant de nouvelles fonctionnalités dans notre application. Nous aborderons ainsi les concepts essentiels d'une application Symfony2, tout en voyant en pratique comment ils s'articulent dans une application web actuelle.

Un réseau social

Notre application ressemblera à Facebook ou Twitter : nous allons concevoir un petit réseau social. Ce n'est volontairement pas un projet original : tout le monde connaît le principe d'un réseau social, ce qui évitera de décrire longuement chaque fonctionnalité. Nous nous concentrerons sur la façon dont le framework peut nous aider. Cela montrera également que Symfony2 permet de rapidement concevoir une application dont la logique métier est complexe.

Les fonctionnalités de l'application

Les utilisateurs

Parmi les fonctionnalités de base, nous mettrons en place une gestion d'utilisateurs, qui permettra a minima :

- la création de compte utilisateur ;
- l'authentification ;
- la déconnexion.

La timeline

La fonctionnalité de base de notre réseau social, c'est la *timeline* d'un utilisateur.

Une timeline est un écran qui contient des *statuts*, un statut étant un court message que l'utilisateur souhaite partager.

Le propriétaire de la timeline peut donc y publier de nouveaux statuts.

Les amis

Un réseau social est une application dans laquelle les utilisateurs ont des liens. Dans notre application, nous pourrons gérer des « amis ». Un ami est un utilisateur dont on peut suivre en particulier le fil. Nous ajouterons les fonctionnalités suivantes :

- ajouter et supprimer des amis ;
- voir la liste des amis d'un utilisateur ;
- voir la timeline des amis d'un utilisateur (contient les statuts des amis de l'utilisateur, du plus récent au plus ancien).

Les commentaires

Parler tout seul dans son coin est amusant mais un peu limité. Afin d'ajouter une vraie dimension sociale, nous donnerons aux utilisateurs la possibilité de commenter un statut et de voir les commentaires sur un statut.

Les différents écrans

La page d'accueil

Il y aura des pages entièrement statiques ; c'est le cas par exemple de la page d'accueil. Ce n'est pas forcément la page la plus intéressante, mais il faut bien commencer par quelque chose ! Elle contient des liens vers les pages de connexion et de création de compte.

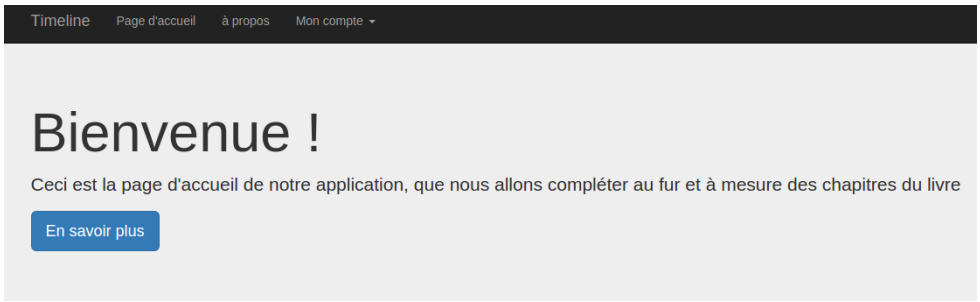


Figure 1–1 Notre future page d'accueil.

L'écran de connexion

Nous devons gérer des utilisateurs. Il faut donc leur permettre de créer un compte et de se connecter. Nous allons leur fournir les différents écrans généralement disponibles pour cela.

La timeline d'un utilisateur quelconque

C'est la fonctionnalité clé de notre application. On y voit les statuts que l'utilisateur a postés, du plus récent au plus ancien. En dessous de chaque statut, on peut voir la liste des commentaires associés à leur auteur.

On voit également la liste des amis de l'utilisateur.

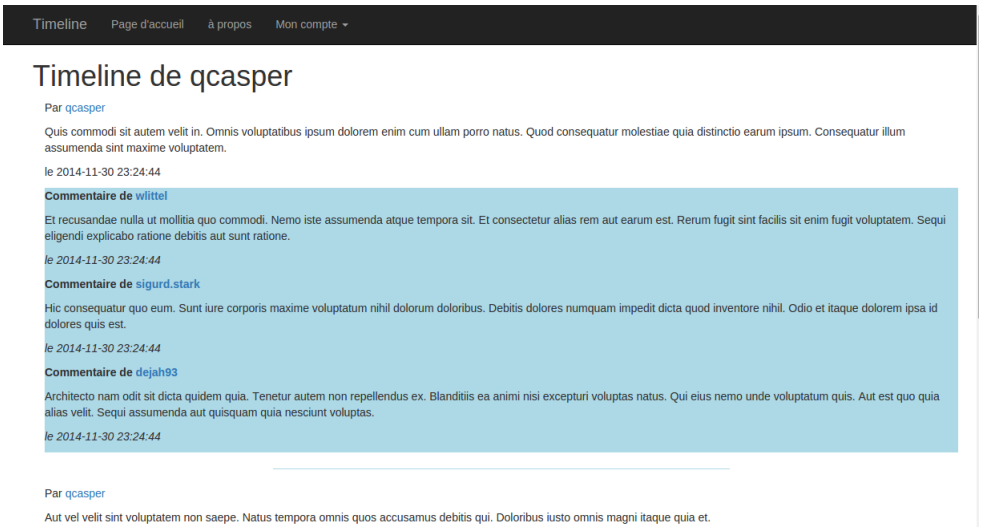


Figure 1–2 La timeline d'un utilisateur, vue depuis un utilisateur déconnecté.

Sur cette capture d'écran, on peut voir deux statuts publiés par l'utilisateur, ainsi qu'une série de commentaires sur le premier statut.

La timeline de l'utilisateur connecté

Lorsque l'utilisateur connecté se rend sur sa propre timeline, il a accès aux fonctionnalités précédentes et peut en plus publier de nouveaux statuts. Il a également la possibilité d'ajouter ou de supprimer un utilisateur de sa liste d'amis via un bouton.

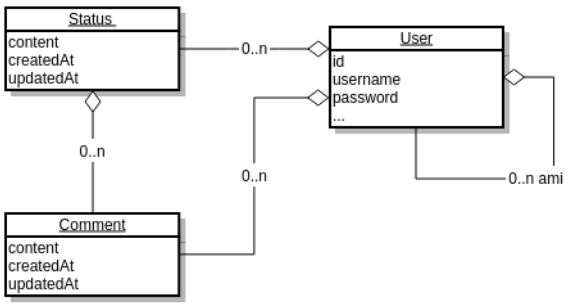
La timeline des amis d'un utilisateur

Cela fait de nombreuses timelines ! Comme notre application est un réseau social, il faut un écran qui centralise les statuts publiés par les amis d'un utilisateur. Il se présente de la même manière que la timeline d'un utilisateur quelconque : la différence est que les statuts émanent de plusieurs personnes différentes.

Le modèle de données

Les entités en jeu sont synthétisées dans le schéma suivant.

Figure 1–3
Un premier aperçu de notre modèle de données.



Un utilisateur peut poster zéro, un ou plusieurs statut(s). Chaque statut peut recevoir zéro, un ou plusieurs commentaire(s) et chaque commentaire est forcément associé à un unique utilisateur-auteur. Enfin, un utilisateur peut avoir zéro, un ou plusieurs ami(s).

C'est tout ce que nous mettrons dans le modèle et dans les fonctionnalités. Nous n'ajouterons pas de possibilité d'aimer un statut ou de le partager. Le but n'est pas de développer une véritable application de réseau social, mais de mettre en pratique les différents concepts que nous allons découvrir. Cependant, si cela vous intéresse, vous aurez à la fin de ce livre tous les outils pour compléter notre application par vous-même.

Nous verrons plus en détail, dans les chapitres qui vont suivre, ce que contient chaque objet et quels sont les équivalences avec la base de données.

À propos des exemples de code de ce livre

Ce livre contient tout le code des différentes fonctionnalités évoquées plus haut. L'application a été réalisée avec Symfony2.6. Cependant, même si nous présentons comment réaliser une application complexe de qualité professionnelle, l'application finale n'est pas complète, faute de place. Notamment, nous écrirons peu de règles CSS et nous ne fournirons pas toutes les traductions de tous les écrans. Afin d'obtenir une application fonctionnelle de qualité professionnelle, il vous faudra compléter l'application présentée dans ce livre.

À la fin des différents chapitres, nous reviendrons sur ce qui reste à faire et sur les possibilités d'améliorer l'existant, que ce soit en ajoutant de nouvelles fonctionnalités ou en améliorant celles déjà présentes. Cela vous aidera à finaliser l'application pour, pourquoi pas, lancer votre propre réseau social !

Téléchargez le code source de tous les exemples du livre sur le site www.editions-eyrolles.com.

2

Le choix du framework

Avant de vous lancer dans la lecture de ce livre, vous êtes peut-être dubitatif sur les atouts d'un framework, et à plus forte raison sur ceux de Symfony2. Après tout, si vous êtes bon dans ce que vous faites, pourquoi vous embêter à utiliser du code écrit par d'autres, avec tous les problèmes que cela va engendrer ? En plus, il y a de nombreux frameworks. A-t-on vraiment besoin d'en utiliser un ? Comme à chaque fois qu'il faut faire un choix, la solution retenue se fera probablement au détriment des autres. Pourquoi choisir Symfony2 plutôt qu'un autre framework ? Dans ce premier chapitre, nous allons répondre à ces questions.

Pourquoi utiliser un framework

Si vous êtes amené à utiliser un framework de développement, que ce soit pour le Web ou non, voici quelques-uns des éléments qu'il va vous apporter.

Nous parlerons ici le plus souvent des frameworks *open source* car c'est le cas de la plupart des frameworks web, en particulier de Symfony2. Leurs atouts sont valables également pour des frameworks commerciaux, avec quelques nuances car la communauté et le fonctionnement sont un peu différents.

Ne pas réinventer la roue

Une des principales forces d'un framework est de vous fournir une base de code pour éviter de réécrire encore et encore des fonctionnalités bas niveau ou récurrentes, qui ont déjà été codées par d'autres.

La gestion du routage en est un bon exemple. Il s'agit de faire le lien entre une route et un contrôleur. Il est apparemment facile d'écrire son code de gestion du routage. Dans un premier temps, on veut juste associer une URL à un contrôleur. Puis on veut gérer différents types de requêtes (`GET`, `POST`). Puis on veut des URL dynamiques, qui gèrent des paramètres passés en arguments. Et ainsi de suite, la liste des fonctionnalités que l'on peut imaginer peut continuer longtemps. Rapidement, la complexité augmente beaucoup et ce qui est une tâche bas niveau devient un problème complexe qu'il faut développer (ajouter des fonctionnalités) et maintenir (éviter les régressions). Cela prend généralement beaucoup de temps et vous éloigne de ce qui vous intéresse, la logique métier, ce pour quoi vous faites le développement au départ...

Les clients veulent du code métier

Que vous travailliez dans une PME, une SSII, une agence web, une start-up ou encore une grande entreprise du CAC 40, le problème est toujours le même. Le but de l'équipe de développement est de réaliser des outils ou des applications qui vont répondre à un besoin métier, pas simplement d'écrire du code ou de réinventer des choses qui ont déjà été faites de nombreuses fois.

Un framework résout ce problème en s'occupant des fonctionnalités bas niveau pour vous, vous laissant vous concentrer sur les aspects métier.

La qualité de la base de code

Dans un framework, le code est généralement de bonne qualité. Il y a des tests automatisés. La structure du projet a été pensée en amont puis a évolué au fil du temps. Lorsque le code est *open source*, des gens l'ont lu et corrigé. Des utilisateurs ont remonté des bogues, qui ont été corrigés par des développeurs, qui ont au passage écrit un test automatisé pour éviter l'apparition de régression. Lorsqu'il y a des failles de sécurité, des gens qui ont des compétences dans le domaine peuvent se pencher sur le problème et lui trouver rapidement une solution.

Si vous faites le choix de concevoir un framework en interne, vous vous privez de tous ces atouts : même avec une très bonne équipe, il est difficile de faire mieux que des bibliothèques maintenues par plusieurs dizaines (voire centaines, ou milliers) de développeurs au cours de plusieurs années.

L'art des bonnes pratiques

Le génie logiciel s'articule autour de ce qu'on appelle les bonnes pratiques. Il y a plusieurs manières de faire les choses, des bonnes et des moins bonnes. Au fil des années et avec l'expérience, a émergé une connaissance de ce qui est une bonne manière de développer une application et de ce qui ne fonctionne pas. Utiliser un framework, surtout s'il est récent, permet d'être au goût du jour en ce qui concerne les bonnes pratiques.

Les outils du framework vous permettent d'écrire du code meilleur non seulement pour vous, mais pour les autres personnes amenées à travailler avec vous.

Travailler avec du code de qualité est essentiel pour que vous ne vous arrachiez pas les cheveux à longueur de journée. Nous énoncerons certaines bonnes pratiques au cours des différents chapitres, que ce soit pour expliquer comment fonctionne Symfony ou comment bien structurer son code. Certaines d'entre elles font penser à du bon sens, mais il y a un univers entre percevoir un conseil comme évident et avoir le recul nécessaire pour savoir quand il est bon de le mettre en pratique.

Comme le sous-entend la phrase précédente, il faut parfois choisir d'appliquer ou non des bonnes pratiques. Ce sont en effet des conseils issus de l'industrie, et non des règles universelles qu'il faut appliquer partout. La réalité du terrain est complexe et il est parfois contre-productif de chercher à appliquer tous les conseils préconisés. Seuls l'expérience, les essais, les erreurs et le dialogue avec le reste de l'équipe vous aideront à déterminer les compromis qu'il est pertinent de faire et les bonnes pratiques qu'il faut mettre en place.

Des tests automatisés

Lorsque le framework le permet, il contient des tests automatisés, c'est-à-dire des fragments de code qui vont tester qu'un autre fragment de code fonctionne comme il est censé le faire.

Lorsque le framework évolue et qu'il y a des mises à jour, on fait tourner la batterie de tests automatisés. Si l'un d'eux ne passe plus, cela veut dire que quelque chose a été cassé.

Attention, si un framework ne contient pas de tests unitaires ou fonctionnels, c'est une mauvaise chose qui devrait vous mettre la puce à l'oreille. Si le framework évolue et que des modifications cassent des fonctionnalités que vous utilisez, il est possible que vous ne le découvriez que de manière empirique, en utilisant l'application.

Des mises à jour et de la maintenance

Derrière un framework, il y a une équipe ou une communauté de gens qui s'occupent de remonter les dysfonctionnements, de les corriger et de faire évoluer le framework pour qu'il soit encore meilleur. Lorsque la communauté est grande, cela veut dire que beaucoup de gens utilisent au quotidien les mêmes fragments de code que vous. Lorsqu'il y a un bogue, il est vite détecté et remonté à l'équipe de développement. Il y a peu de risques qu'il reste des bogues majeurs.

Si vous concevez votre propre framework ou vos propres outils, vous connaîtrez les limites que vous aurez découvertes, mais pas forcément tous les bogues, pas même tous les plus gros.

Harmoniser la structure du code

Écrire du code seul est facile, mais qu'en est il de :

- écrire du code de qualité (quoi que cela signifie) ?
- écrire du code à plusieurs ?
- maintenir du code sur plusieurs mois, voire plusieurs années ?
- travailler sur du code écrit par d'autres ?

En fait, travailler en équipe est très compliqué car cela pose de nombreux problèmes de communication et d'expérience. Tout le monde n'a pas participé au projet depuis le début, tout le monde ne connaît pas les raisons de tous les choix techniques, tout le monde n'a pas les mêmes affinités avec certains aspects du projet. Bref, lorsqu'une base de code devient conséquente, il devient indispensable de mettre en place des solutions pour faciliter la communication. Un moyen pour cela est d'avoir une structure de code uniforme et cohérente. Un framework va chercher à mettre ceci en avant, grâce à des conventions et des normes.

Des outils

Le but d'un framework est de développer plus rapidement, d'une part en fournissant des bibliothèques qui évitent de réécrire des fonctionnalités récurrentes, et d'autre part en fournissant des outils pour aller plus vite.

Un framework vous fournit des outils pour générer le code squelette d'applications. Vous avez également des outils pour faciliter le débogage.

Bref, un framework est souvent accompagné d'une panoplie d'éléments qui vous permettent d'aller plus vite dans le développement.

La documentation

Un framework sans documentation est peu utilisé. Donc, une documentation officielle étoffée est généralement disponible avec chaque outil pour en faciliter la prise en main.

Une communauté

Nous en avons déjà parlé, un framework est utilisé par une communauté. Nous avons déjà montré que cela présentait des avantages au niveau de la qualité du code, du volume de bogues, de la maintenance, de la documentation et des problèmes de sécurité. En soi, il s'agit déjà là d'atouts non négligeables qui devraient vous convaincre, ou du moins faire pencher la balance en faveur de l'utilisation d'un framework.

Cependant, la force d'avoir une communauté d'utilisateurs, c'est aussi l'énergie humaine qui s'en dégage. La sensation d'appartenir à un groupe est quelque chose de fort, qui pousse tout le monde à aller de l'avant.

Les passionnés s'investissent de plusieurs manières : ajout de fonctionnalités dans le code, aide aux nouveaux utilisateurs en répondant à leurs questions, mise à jour de la documentation, écriture d'articles, animation de conférences et d'événements locaux…

Malgré cette diversité, tous sont guidés par l'idée d'améliorer les choses, de progresser, de perfectionner ses outils et d'aider les autres à mieux développer, mieux utiliser les outils. C'est un sentiment très fort et motivant.

Développer plus vite et mieux

Si vous ne deviez retenir qu'une seule idée parmi toutes les précédentes, retenez celle-ci, qui en est la synthèse : un framework vous fournit le moyen, à vous et votre équipe, de développer plus rapidement des applications de meilleure qualité.

Avec des outils pour automatiser ou simplifier les tâches, vous écrirez plus de code et plus rapidement. Vous gagnerez en rapidité de développement et en confiance car les bibliothèques fournies sont utilisées par un grand nombre de personnes. Une fois passée l'étape de l'apprentissage, le framework vous permettra de vous concentrer sur vos besoins métier pour réaliser des applications de qualité qui répondent aux besoins de vos clients, sans avoir à résoudre des problématiques bas niveau déjà résolues par d'autres, mieux, avant vous. Par l'utilisation de conventions et en créant des normes dans le code, le framework va devenir un outil de communication dans l'équipe : une fois ces conventions comprises et assimilées, tout le monde saura où trouver telle ou telle information dans le projet.

Pourquoi choisir Symfony2

Choisir un framework est une tâche compliquée. Nous allons essayer de revenir sur certains des points auxquels il est particulièrement important de penser.

Il n'y a pas de mauvais choix de framework

Bien que ce livre se concentre sur Symfony2, le choix d'un outil plutôt qu'un autre est complexe. Le sujet est source de débats infinis. Avant toute chose, il est important d'être convaincu qu'il n'y a pas de mauvais choix de framework si la solution que vous retenez convient à votre équipe.

Lorsqu'on choisit un tel outil, c'est en grande partie pour ses fonctionnalités. Il y a un aspect pratique, on veut développer plus rapidement du meilleur code.

Pourtant, un aspect essentiel à prendre en considération est l'état d'esprit du framework : où il en est, vers où il se dirige, comment fonctionne sa communauté. On peut généralement quantifier chacun des deux aspects, mais il y a une part d'affectif très forte chez les développeurs dans le choix de leurs outils. Cet aspect ne doit pas être négligé.

Symfony2 est un très bon framework

Il aurait été difficile pour moi d'écrire un livre sur le sujet si je n'en étais pas convaincu : Symfony2 est un excellent framework. Nous avons parlé précédemment des avantages qu'un framework procure, mais nous allons maintenant revenir dessus pour présenter plus spécifiquement comment ils peuvent se concrétiser ; nous détaillerons également quelques éléments de réflexion supplémentaires.

Une bonne réputation

Symfony2 a une excellente réputation, en particulier en France. Conçu originellement par SensioLabs, une société française qui maintient Symfony depuis 2005, il est aujourd'hui utilisé dans de nombreuses grosses applications web à fort trafic.

- http://sensiolabs.com/

On peut notamment citer la plate-forme vidéo DailyMotion et le site de covoiturage BlablaCar, mais il y en a de nombreux autres.

- http://www.dailymotion.com/
- http://www.covoiturage.fr/

D'autres ont fait le choix d'utiliser des composants du framework plutôt que son intégralité (vous saurez bientôt ce que cela veut dire). C'est le cas de l'outil de forum open source phpBB. C'est également le choix qu'a fait l'outil de gestion de contenu Drupal lors de la refonte pour son passage en version 8. Tous les deux sont parmi les plus utilisés dans leurs domaines respectifs.

- https://www.phpbb.com/
- https://www.drupal.org/

Les avantages d'un framework mais aussi des composants indépendants

Symfony2 est un framework dit *full-stack*, c'est-à-dire qu'il rassemble autour de lui de nombreux composants pour qu'on puisse développer des applications métier sans avoir à se préoccuper de toutes les problématiques bas niveau de routage, de gestion de cache ou d'architecture MVC.

En cela, il a donc tous les atouts d'un framework. Lorsque les applications sont développées en équipe, le framework devient un moyen de communication qui permet de créer des normes décrivant la manière dont doivent être réalisées les choses, ce qui aide chacun à mieux s'y retrouver lorsque le projet grossit.

Symfony2 est également construit autour de composants indépendants. Tous les composants utilisés dans le framework (routage, gestion des vues, ORM…) peuvent l'être indépendamment. Il est possible d'utiliser Twig pour afficher des vues en dehors du contexte de Symfony2, mais également le composant de console pour développer des outils en ligne de commande.

En fait, si l'on ne souhaite pas utiliser tout ce que propose Symfony2, il est même possible de recréer son propre framework à partir de ces composants. SensioLabs s'occupe d'ailleurs de Silex, un microframework qui utilise quelques composants essentiels de Symfony et sert à développer rapidement des applications légères, pour faire des preuves de concept par exemple.

- http://silex.sensiolabs.org/

Il est tout à fait possible de réaliser la même chose soi-même. En 2012, quelques mois après la sortie de la première version stable de Symfony2, Fabien Potencier, son concepteur, écrivait déjà une série d'articles expliquant comment faire fonctionner ensemble les composants de base du framework pour monter son propre outil.

- http://fabien.potencier.org/article/50/create-your-own-framework-on-top-of-the-symfony2-components-part-1

Fiable et stable

Le framework est utilisé par un nombre conséquent et grandissant de personnes, le site officiel avançant « plus de 150 000 développeurs ». Lorsqu'il y a un bogue, il est remonté et corrigé rapidement.

De plus, la communauté est rigoureuse et stricte sur la qualité du code. Il y a de nombreux tests unitaires et fonctionnels, ce qui garantit la solidité de l'ensemble.

De l'innovation

Symfony2 arrive après Symfony premier du nom. La version stable, sortie en 2011, a apporté un certain nombre d'innovations, jusqu'alors peu répandues dans l'univers PHP. On peut citer notamment l'injection de dépendances, absente des autres frameworks PHP majeurs au même moment.

Symfony2 et sa communauté ont également entraîné, de manière directe ou non, des apports majeurs dans la communauté PHP. On peut notamment citer l'outil Composer de gestion de dépendances PHP, omniprésent aujourd'hui.

- https://getcomposer.org/

Symfony2 met l'accent sur la sécurité

La sécurité est une problématique particulièrement complexe dans le domaine du Web. Tout le monde cherche à se prémunir contre les failles de sécurité, car il est crucial pour l'entreprise de protéger les données qu'elle possède, ainsi que ses utilisateurs.

Il est pourtant courant d'écrire du code vulnérable à de nombreuses attaques, car il est très difficile, impossible même, d'être informé de toutes les failles de sécurité. Même lorsque l'on connaît les failles de sécurité, il n'est pas garanti que l'on s'en protège efficacement. C'est le cas par exemple des injections SQL, que tout le monde connaît mais que l'on trouve pourtant encore sur de nombreux sites, y compris des très gros. L'absence de rigueur, ou une mauvaise compréhension de ce qui se passe à un endroit du code peut entraîner l'apparition d'erreurs si l'on n'y prête pas particulièrement attention.

Parmi les failles les plus connues, il y a l'injection SQL, les failles XSS *(Cross Site Scripting)* et CSRF *(Cross Site Request Forgery)*. Afin de se protéger contre elles, Symfony2 propose plusieurs mécanismes.

L'utilisation de Doctrine permet de s'assurer que les paramètres utilisés dans les requêtes SQL sont bien échappés, ce qui protège contre les failles par injection SQL.

Le moteur de vue Twig, fourni avec Symfony2, échappe lui aussi par défaut le texte affiché pour éviter l'injection de code dans les pages, pouvant ainsi mener à des failles XSS qui exécutent du code tiers indésirable.

Enfin, une bibliothèque permet de se protéger contre les failles CSRF, en gérant de manière autonome les clés qui servent à valider que des données transmises à une page sont bien authentiques. Intégrée de manière transparente dans les formulaires Symfony2, elle assure que les formulaires n'ont pas été usurpés par des utilisateurs peu scrupuleux.

Les fonctionnalités proposées par Symfony2

Un framework, c'est souvent un ensemble de bibliothèques qui permettent de gérer de nombreuses tâches bas niveau. Voici une liste non exhaustive des fonctionnalités proposées par le framework ou par les bibliothèques livrées avec la distribution standard :

- structure MVC ;
- moteur de routage ;
- moteur de vue ;
- gestion de cache ;
- internationalisation ;
- plusieurs environnements (dev, prod, test) ;
- code testé et facilement testable ;
- entièrement configurable ;
- ORM *(Object Relation Mapper)* ;
- injection de dépendances ;
- outils pour le développeur (logs, barre de débogage, console) ;
- protection native contre les injections SQL, les failles XSS et CSRF ;
- système de plug-in (bundle).

Des ressources et de l'assistance

Une communauté en ébullition

Nous avons déjà évoqué cet élément essentiel de tout framework : la communauté. Celle qui gravite autour de Symfony2 est particulièrement riche.

Le projet Symfony2 est l'un des plus suivis sur Github. Selon le site officiel, il y aurait plus de 1 000 contributeurs et plus de 150 000 utilisateurs de Symfony2.

▸ https://github.com/symfony/symfony

La conséquence d'un tel volume de contributeurs est que le framework évolue rapidement. Il y a deux versions mineures par an, à six mois d'intervalle, en mai et novembre.

▸ http://symfony.com/fr/doc/current/contributing/community/releases.html

En France, un regroupement de développeurs, l'AFSY (Association francophone des utilisateurs de Symfony) se charge d'organiser des événements pour rassembler les développeurs, leur permettre d'échanger et de partager des bonnes pratiques.

Parmi ces événements, il y a les SfPots, des meetings généralement mensuels qui ont lieu dans plusieurs villes de France, lors desquels plusieurs intervenants font des présentations sur un sujet qui leur tient à cœur et proche de Symfony. Après ces quelques conférences, il est d'usage de partager un verre avec tout le monde.

▸ http://www.meetup.com/afsy-sfpot/

Organisée par SensioLabs, une conférence a lieu plusieurs fois par an dans plusieurs pays d'Europe : le SymfonyLive. Durant plusieurs jours, elle traite des dernières nouveautés et est l'occasion de participer à des ateliers.

▸ http://live.symfony.com/

Documentation

La documentation de Symfony2 est particulièrement fournie et répond à de nombreuses questions. Elle évolue avec les différentes versions ; assurez-vous de bien utiliser la version de la documentation qui correspond à celle de votre framework ! Si vous n'utilisez pas la dernière version, cela vous explique comment utiliser malgré tout tel ou tel composant.

Comme la communauté est internationale, il arrive que la documentation en anglais soit plus fournie, notamment sur des détails pointus, mais la documentation française contient beaucoup d'articles.

Voici quelques ressources présentes sur le site officiel qui vous seront utiles.

- Le « livre » présente comment utiliser chacun des composants dans le contexte d'un projet d'application web, expose les différentes possibilités offertes et explique pourquoi les composants fonctionnent comme ils le font.

▸ http://symfony.com/fr/doc/current/book/index.html

- Le livre de recettes fournit des exemples de cas d'usage récurrents pour divers composants et explique comment les mettre en œuvre.

▸ http://symfony.com/fr/doc/current/cookbook/index.html

- Enfin, alors que les précédentes documentations avaient une vision assez macro du framework, il existe aussi une documentation pour chaque composant.

▸ http://symfony.com/fr/doc/current/components/index.html

Cette documentation est open source comme le reste du projet et il est tout à fait possible d'y contribuer, que ce soit en anglais, en français, ou dans les autres langues disponibles. Si vous voyez une faute, une imprécision ou un oubli dans la documentation, peut-être est-ce l'occasion de devenir contributeur au projet Symfony ?

▸ https://github.com/symfony/symfony-docs/
▸ https://github.com/symfony-fr/symfony-docs-fr

De l'aide

Symfony2 est un framework populaire et de nombreuses ressources sont disponibles pour apprendre à l'exploiter au mieux. Vous lisez l'une d'entre elles !

Internet regorge de ressources pour vous aider à utiliser Symfony2. Une assistance non officielle est prodiguée par la communauté, qui produit de nombreux articles et tutoriels. Faites attention cependant, privilégiez les ressources fiables et à jour avec la version que vous utilisez de Symfony2. Dans le doute, un coup d'œil à la documentation officielle devrait vous aider à clarifier une incompréhension ou une imprécision.

▸ http://symfony.com/doc/current/index.html

De nombreuses présentations issues de conférences, par exemple lors du forum PHP ou des divers SymfonyLive, fournissent des ressources précieuses : retours d'expérience, vulgarisation, mise en pratique de composants…

L'aide sur Symfony s'effectue donc de plusieurs manières. D'une part, la communauté maintient et pratique l'entraide de diverses façons (articles, corrections de bogues, réponses aux questions, animation d'événements et de conférences). D'autre part, parce que ce n'est pas toujours suffisant pour les besoins d'une entreprise, SensioLabs propose également des formations et de l'accompagnement, à l'aide de ses développeurs qui maîtrisent le framework. Ils conçoivent également des outils, comme Insight qui permet de contrôler la qualité du code d'un projet grâce à divers indicateurs et recommandations.

▸ https://insight.sensiolabs.com/