



Expert
EXPO

Symfony 4 (LTS)

Développez des sites web PHP
structurés et performants

En téléchargement



code source



+ QUIZ

Version en ligne

OFFERTE !

pendant 1 an

Bilal AMARNI
Etienne LANGLET

eni



Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>
Saisissez la référence ENI de l'ouvrage **EI4SYM** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Chapitre 1 Introduction

1. Avant-propos	23
2. Public visé	24
3. Prérequis	24
4. Objectifs du livre	25
5. Le développement avec les frameworks	26
5.1 Complexité des développements et productivité	26
5.1.1 Productivité et qualité logicielle	26
5.1.2 Intégration et livraison continues	27
5.2 Particularité des développements en PHP	27
5.2.1 Contexte historique de PHP	27
5.2.2 Évolutions	28
5.3 L'apport des frameworks	28
5.3.1 Éviter les problèmes techniques liés à l'organisation du code	29
5.3.2 Définir des responsabilités	29
5.3.3 Ne pas réinventer la roue	30
5.3.4 Utiliser des modèles de conception éprouvés	30
6. Symfony	31
6.1 Historique	31
6.2 Gouvernance et gestion des versions	32
6.2.1 Les versions	32
6.2.2 Le cycle de « release »	33
6.3 Choisir sa version pour un projet	35

2 _____ **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

Chapitre 2

Mise en place d'un projet Symfony

1. L'outillage nécessaire	37
1.1 Introduction	37
1.2 Symfony : Un projet PHP	38
1.2.1 Préconisation d'installation	38
1.2.2 Installation sous Linux	39
1.2.3 Installation sous Windows	41
1.3 L'installeur Symfony	42
1.4 Composer	43
1.4.1 Installer Composer	44
1.5 Les environnements de développement pour Symfony	48
1.5.1 Un IDE pour Symfony !	49
1.5.2 PHPStorm	50
1.5.3 Eclipse IDE for PHP Developers	52
1.5.4 Visual Studio Code	53
1.5.5 Conclusion	55
2. Création d'un projet Symfony	56
2.1 Prérequis	56
2.2 Création via l'installeur Symfony	56
2.3 Création via Composer	57
2.4 Configurer son serveur web	59
2.4.1 Serveur web PHP	59
2.4.2 Apache et Nginx	60
3. Structure de l'application	64
3.1 Arborescence du projet	64
3.2 Règles et conventions d'organisation du projet	65
3.2.1 Le standard PSR-4	65
3.2.2 Conventions de nommage	66

3.3	La configuration d'une application Symfony	68
3.3.1	Les annotations	68
3.3.2	Le format YAML	70
3.3.3	Le format XML	71
3.3.4	Le format PHP	71
3.3.5	Choisir son format de configuration	72

Chapitre 3

Architecture du framework

1.	Le modèle de conception MVC	73
1.1	Définitions et responsabilités	73
1.1.1	La vue	74
1.1.2	Le modèle	74
1.1.3	Le contrôleur	74
1.2	En pratique	75
1.2.1	Le contrôleur frontal	76
1.2.2	Le routage	76
1.2.3	Le contrôleur et le modèle	76
1.2.4	La vue	77
1.2.5	En synthèse	78
2.	Architecture de Symfony	78
2.1	Présentation	78
2.2	Le contrôleur frontal	79
2.3	Le Service Container	80
2.4	Le modèle MVC dans Symfony ?	80
2.5	L'approche par composant	81
3.	Symfony Flex	82
3.1	Présentation	82
3.2	Fonctionnement de Symfony Flex	83
3.3	Les recettes Flex	84
3.3.1	Un exemple concret	84

4 **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

4. Les environnements	85
4.1 Principe et apports	85
4.2 Les fichiers de configuration	86
4.3 Dans le contexte HTTP	87
4.4 Dans le contexte CLI (Command Line Interface)	87
5. Le chargement automatique de classes	87
5.1 Le standard PSR-4	88
5.2 Mécanismes alternatifs	88
5.3 Application aux applications Symfony	89
6. La console	90
6.1 Présentation	90
6.2 Les commandes	90
6.2.1 Lister les commandes disponibles	91
6.2.2 Exécuter une commande	91
6.3 Les options	91
6.4 Les arguments	92
6.5 L'aide sur les commandes	93
6.6 Exécuter rapidement des commandes	94
7. Les outils pour le débogage	95
7.1 Le profiler Symfony	95
7.2 La fonction dump()	97

Chapitre 4

Routage et contrôleur

1. Fonctionnement du routage dans Symfony	99
1.1 Définition	99
1.2 Le répertoire public et le contrôleur frontal	100
1.3 Une requête, une action	100
2. Définition des routes	101
2.1 Les différents formats de définition	101
2.2 Les options sur la définition des routes	102

3.	Configurer le path.	103
3.1	Illustration par l'exemple : /hello/world.	103
3.2	La notation du contrôleur	106
3.3	Importer des routes depuis d'autres fichiers.	107
3.4	Comprendre l'ordre de chargement des routes.	108
3.5	Préfixer les routes	109
3.6	Les paramètres de substitution des routes	110
3.7	Les restrictions sur les paramètres.	112
3.8	Obtenir des informations sur le routage.	114
4.	Routage par nom de domaine	115
4.1	Prérequis	115
4.2	Exemple de mise en œuvre.	116
4.3	Explications.	118
5.	Le contrôleur.	119
5.1	Modèle de programmation et règles	119
5.2	Travailler avec les services	119
5.3	Utiliser les paramètres de substitution	120
5.3.1	Paramètres de substitution des routes	120
5.3.2	Exemples	120
5.4	Travailler avec les URL.	122
5.5	Effectuer une redirection	123
5.6	La délégation de requête.	124
5.7	La gestion des erreurs et des pages d'erreurs dans les contrôleurs	124
5.7.1	Le contrôleur	125
5.7.2	La vue	127

6 **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

Chapitre 5

L'injection de dépendances

1. Le modèle de conception IoC : Inversion Of Control	131
1.1 Apports dans une architecture applicative	131
1.2 IoC et injection de dépendances	133
2. L'injection de dépendances	133
2.1 Principes de base	133
2.2 Les différentes techniques d'injection de dépendances	133
2.2.1 L'injection de dépendances par le constructeur	133
2.2.2 L'injection de dépendances par setter (mutateur)	134
2.2.3 L'injection de dépendances par propriété	135
2.3 Les avantages	136
3. Le Service Container	136
3.1 Les services	137
3.2 Explications au travers d'un service X	137
4. Créer un service et configurer ses injections	138
4.1 Créer un service	138
4.2 Les différents types d'injections dans un service Symfony	139
4.2.1 Injection par constructeur	139
4.2.2 Injection par méthode	139
4.2.3 Injection par propriété	140
4.3 Injection automatique avec l'autowiring	141
4.4 Les services « lazy »	142
5. Le chargement automatique de services	143
5.1 La configuration	143
5.2 Exemple d'utilisation	145
6. Créer des services réutilisables et distribuables	147
6.1 Le concept de bundle	147
6.1.1 Créer un bundle	147
6.1.2 Arborescence du bundle	148
6.2 La définition de services dans un bundle	148

6.3	La configuration	149
6.3.1	Définir une arborescence.	151
6.3.2	Les différentes étapes du traitement de la configuration.	152
6.3.3	Récupérer la configuration validée	156
6.4	Les « Compiler Passes »	159
6.4.1	Concept	159
6.4.2	Les tags	162
6.4.3	Le Compiler Pass	162

Chapitre 6

Les templates avec Twig

1.	Présentation et concepts	165
1.1	Le concept de Templating	165
1.2	Templating et modèle MVC	165
2.	Twig	166
2.1	Présentation	166
2.2	Pourquoi un nouveau langage ?	167
2.3	Mise en pratique	168
2.4	Remarques sur l'utilisation	169
2.5	La notation des templates	170
2.6	Extension du système de templates	171
2.7	L'annotation @Template	171
3.	Les gabarits de pages (layouts) et les blocks	173
3.1	La composition de pages	173
3.2	Définition des gabarits	173
3.3	Les blocks	175
4.	Le langage Twig	178
4.1	Les différents types d'instructions	178
4.2	Manipulation des variables	179
4.2.1	Utilisation de variables dans les templates	179
4.2.2	Utilisation des variables de type tableau ou objet	181

8 — **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

4.3	Structures de contrôle et tags	181
4.3.1	Les conditions	182
4.3.2	Les boucles	182
4.4	Les balises Twig (tags)	185
4.4.1	Créer et modifier des variables	185
4.4.2	Twig et l'échappement	186
4.5	Inclure des templates	189
5.	Les filtres et les fonctions	190
5.1	Présentation des filtres	190
5.1.1	Utilisation et syntaxe	190
5.2	Les principaux filtres Twig	191
5.2.1	Chaînes de caractères	191
5.2.2	Échappement	192
5.2.3	L'encodage	193
5.3	Les fonctions	195
5.3.1	Twig et le routage	195
5.3.2	Débogage avec la fonction dump	197
6.	La gestion des ressources statiques (images, feuilles de style, scripts JS...)	197
6.1	Les ressources statiques dans une application Symfony	197
6.2	Le cas des ressources statiques externes	198
6.3	Référencer les ressources publiques depuis un template	199
6.4	Cas pratique avec le framework CSS Bootstrap 4	200

Chapitre 7

Accéder aux bases de données avec Doctrine

1.	Présentation et concepts	203
1.1	Les principes de l'ORM	203
1.2	Architecture de Doctrine	204
1.2.1	DBAL	204
1.2.2	Entité	205
1.2.3	ORM	205

1.3	La notion d'entité	205
2.	Installer et configurer Doctrine	206
2.1	Mise en place de Doctrine	206
2.2	Relation avec PDO	207
2.3	Configuration	208
3.	Définition des entités et de leur mapping	209
3.1	Règles de conception des entités	209
3.2	Les syntaxes pour le mapping des entités	211
3.3	Le mapping d'entités simples	214
3.3.1	Définir une entité avec <code>@ORM\Entity</code>	214
3.3.2	Gérer les colonnes de la table avec <code>@ORM\Column</code>	216
3.3.3	<code>@ORM\Table</code>	219
3.3.4	Les clés primaires	221
3.3.5	Configurer les index	222
3.4	Le mapping des relations (clés étrangères)	223
3.4.1	<code>@ORM\OneToOne</code>	224
3.4.2	<code>@ORM\ManyToOne</code>	226
3.4.3	<code>@ORM\ManyToMany</code>	227
3.4.4	Relations bidirectionnelles	229
3.5	Les outils de la console	234
3.5.1	Repérer les erreurs de mapping	234
3.5.2	Générer le schéma des données à partir des entités	235
3.5.3	Générer les entités à partir du schéma des données	235
4.	Manipulation des entités avec l'EntityManager	237
4.1	Le rôle de l'EntityManager	237
4.2	Insertion de données	237
4.3	Modification de données	239
4.4	Suppression de données	240
4.5	Autres opérations de l'EntityManager	241
4.5.1	<code>refresh()</code>	241
4.5.2	<code>detach()</code>	241
4.6	Les opérations en cascade	242

10 ————— **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

5.	Récupérer des entités	245
5.1	Le repository	245
5.1.1	Un rôle de centralisateur	245
5.1.2	Les méthodes de base du repository	246
5.1.3	Les méthodes personnalisées du repository	248
5.2	Le DQL	249
5.2.1	SELECT	251
5.2.2	FROM	251
5.2.3	JOIN et LEFT JOIN	252
5.2.4	WHERE	255
5.2.5	ORDER BY	259
5.2.6	Les limites	260
5.2.7	Les limites et la pagination	261
5.3	Le QueryBuilder	263
6.	Fonctionnalités avancées	265
6.1	Les extensions Doctrine	265
6.1.1	Installation	266
6.1.2	Utilisation d'un slug sur une entité	266

Chapitre 8

La gestion des événements applicatifs

1.	Concepts et écoute d'événement applicatifs	269
1.1	Le propagation des événements	270
1.2	L'écoute des événements	272
2.	Les événements du Kernel	276
2.1	Les différents type d'événements	276
2.2	Applications	277
3.	Les événements de la console	279
3.1	Prérequis	279
3.2	Les événements	279

Chapitre 9

Les formulaires

1. Un composant MVC	283
1.1 Le modèle	283
1.2 Le contrôleur	285
1.3 La vue	286
2. Fonctionnement du composant	287
2.1 L'objet « Form »	287
2.1.1 Soumission	287
2.1.2 Validation	288
2.1.3 Vue	288
2.2 Les types	288
2.3 Les options	289
2.4 Les objets « Form » et « FormBuilder »	290
2.4.1 Le FormBuilder	290
2.4.2 Structure de l'objet Form	290
2.5 Association avec l'objet de la couche Modèle	292
2.6 Formulaires sans objet	293
2.7 La représentation des valeurs	294
2.7.1 Transformation des données	294
2.7.2 Illustration avec le type date	295
3. Les types de champs de formulaire	296
3.1 L'héritage	296
3.2 FormType	297
3.2.1 label	297
3.2.2 label attr	297
3.2.3 data	297
3.2.4 required	298
3.2.5 disabled	298
3.2.6 mapped	298
3.2.7 property_path	298
3.2.8 attr	299

3.2.9	trim.....	299
3.2.10	error_bubbling.....	299
3.3	TextType.....	300
3.4	PasswordType.....	300
3.5	RepeatedType.....	301
3.5.1	type.....	301
3.5.2	first_options et second_options.....	301
3.5.3	options.....	301
3.5.4	first_name.....	302
3.5.5	second_name.....	302
3.5.6	invalid_message.....	302
3.6	ChoiceType.....	302
3.6.1	choices.....	302
3.6.2	expanded et multiple.....	303
3.6.3	placeholder.....	303
3.6.4	preferred_choices.....	304
3.6.5	Types similaires.....	304
3.7	EntityType.....	304
3.7.1	class.....	305
3.7.2	choice_label.....	305
3.7.3	query_builder.....	305
3.7.4	group_by.....	306
3.7.5	em.....	306
3.8	DateType.....	306
3.8.1	widget.....	306
3.8.2	format.....	307
3.8.3	model_timezone.....	307
3.8.4	view_timezone.....	307
3.8.5	years.....	308
3.8.6	months.....	308
3.8.7	days.....	308
3.8.8	placeholder.....	308
3.8.9	Types similaires.....	308

3.9	FileType.	309
3.9.1	multiple	309
3.9.2	Récupérer les fichiers.	309
3.9.3	Traiter les fichiers	310
3.10	CheckboxType	311
3.11	SubmitType, ResetType et ButtonType.	312
4.	Créer des formulaires réutilisables.	313
4.1	Définir un formulaire avec la classe AbstractType.	314
4.2	Utiliser un formulaire défini dans une classe	320
4.2.1	Définition manuelle	320
4.2.2	Avec l'injection de dépendances	321
5.	Validation des données.	322
5.1	Objectifs	322
5.2	La définition des contraintes de validation.	322
5.2.1	Ajout des contraintes lors de la configuration d'un formulaire.	323
5.2.2	Ajout des contraintes sur l'objet associé au formulaire	325
5.2.3	Les différents formats de configuration.	330
5.2.4	Les options	333
5.3	Les contraintes et leurs options.	335
5.3.1	NotBlank et NotNull.	336
5.3.2	IsNull et Blank.	336
5.3.3	IsTrue, IsFalse	336
5.3.4	Type	337
5.3.5	Email, Url et Ip	337
5.3.6	Regex	339
5.3.7	Length, Count	339
5.3.8	Range	341
5.3.9	Comparaisons	341
5.3.10	Dates.	342
5.3.11	File	342
5.3.12	Image	343

14 — **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

5.3.13	Choice	343
5.3.14	UniqueEntity	344
5.3.15	Données financières	345
5.3.16	Callback	346
5.3.17	All	347
5.3.18	Valid	347
5.4	Groupes de validation	348
5.5	Validation d'un objet hors du contexte d'un formulaire	350
6.	Personnaliser le rendu - thèmes de formulaires	351
6.1	Afficher le formulaire manuellement	352
6.1.1	form_start()	352
6.1.2	form_end()	353
6.1.3	form_widget()	353
6.1.4	form_errors()	354
6.1.5	form_label()	354
6.1.6	form_row()	354
6.1.7	form_rest()	354
6.1.8	Arborescence des parties de formulaires	355
6.2	Créer des thèmes	356
6.2.1	Formulaire d'exemple	356
6.2.2	Créer et associer un thème de formulaires	357
6.2.3	Comprendre le nom des blocks	361

Chapitre 10

La sécurité dans une application Symfony

1.	Présentation et concepts de sécurité	363
1.1	Les challenges de la sécurité des applications web	363
1.2	La sécurité dans Symfony	364
2.	Authentification	365
2.1	Pare-feu	365
2.1.1	Pare-feu pour ressources statiques/développement	366
2.2	Authentification HTTP	367

2.3	Authentification par formulaire de connexion	367
2.4	Connexion automatique des utilisateurs	370
2.5	Déconnexion des utilisateurs	371
3.	Utilisateurs et rôles	372
3.1	L'utilisateur	372
3.1.1	Récupérer l'utilisateur courant	373
3.2	Les fournisseurs d'utilisateurs	374
3.2.1	En mémoire	374
3.2.2	Fournisseur d'utilisateurs de bases de données	375
3.2.3	Fournisseur d'utilisateurs personnalisé	377
3.2.4	Notes additionnelles	381
3.3	Cryptage des mots de passe	382
3.3.1	Encodeurs	382
3.3.2	Le salage	383
3.3.3	Crypter un mot de passe	385
3.4	Les rôles	386
4.	Autorisations	389
4.1	Les rôles, au cœur du processus	389
4.2	Vérifier le rôle de l'utilisateur	390
4.3	Sécuriser une action	391
4.4	Sécuriser une section de l'application	392
4.5	Sécuriser selon d'autres critères	393
4.6	Pour aller plus loin	395

Chapitre 11

Tester son application Symfony

1.	Introduction au test logiciel	397
1.1	Les tests : un indispensable pour la qualité logicielle	397
1.2	Les différentes catégories de tests	398
1.2.1	Les tests unitaires	398
1.2.2	Les tests d'intégration	398
1.2.3	Les tests fonctionnels	399

16 _____ **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

1.3	Analogie.	399
1.4	L'approche des tests en PHP	400
2.	Les tests unitaires avec PHPUnit.	401
2.1	Mise en place des tests	401
2.2	Règle d'écriture des tests	401
2.3	Exécuter les tests	403
2.3.1	Exécuter une partie des tests	403
3.	Les tests fonctionnels	404
3.1	Différence par rapport aux tests unitaires et d'intégration	404
3.2	Tester une action	404
3.3	Les objets pour l'écriture des test	405
3.3.1	L'objet Client	405
3.3.2	L'objet Crawler	408
3.4	Soumettre un formulaire	409
3.5	Pour aller plus loin	410

Chapitre 12

Journalisation et surveillance avec Symfony

1.	Générer des journaux avec Monolog.	413
1.1	La journalisation	413
1.2	La librairie Monolog	414
1.3	Le service logger	415
1.4	Le fichier journal.	416
1.4.1	Identifier la cause d'un bogue	416
1.4.2	Le problème	416
1.5	Les gestionnaires (handlers).	417
1.5.1	Définir plusieurs gestionnaires	417
1.5.2	Envoyer des logs par e-mail.	418
1.5.3	Utiliser un tampon (buffer)	419
1.5.4	Ajouter des informations complémentaires	419

1.6	Les canaux (channels)	421
1.6.1	Ajouter ses propres canaux	421
1.6.2	Envoyer un enregistrement sur un canal donné	422
1.6.3	Configurer les gestionnaires par canaux	422
1.6.4	Gestion des erreurs 404	423
2.	Le monitoring avec Prometheus et Grafana	425
2.1	Un allié proactif au logging	425
2.2	Préparation d'une application Symfony pour Prometheus	426
2.3	Instrumenter les mesures	431
2.4	Pour aller plus loin	432

Chapitre 13 Amélioration des performances

1.	La mise en cache de pages	433
1.1	Autour du protocole HTTP	433
1.2	Un serveur proxy inverse (ou « reverse proxy »)	434
1.2.1	HttpCache	436
1.2.2	Nginx	437
1.2.3	Varnish	438
1.3	Les en-têtes	443
1.4	Les réponses publiques et privées	443
1.5	L'expiration	444
1.5.1	L'en-tête Expires	444
1.5.2	Les directives max-age et s-max-age	445
1.5.3	L'annotation @Cache	445
1.6	La validation	447
1.6.1	Par date avec Last-Modified	447
1.6.2	Par empreinte avec l'en-tête ETag	449
1.7	Les ESI	450
1.7.1	Activation	450
1.7.2	Générer une balise ESI	451

18 _____ **Symfony 4 (LTS)**

Développez des sites web PHP structurés et performants

2.	L'autochargement des classes	452
2.1	Générer un classmap	453
3.	Le cache avec Doctrine	453
3.1	Les différents types de cache	453
3.2	Configuration	454
4.	Le cache d'annotations	455
5.	Les sessions	456
6.	Autres optimisations	457
6.1	Choix de sa SAPI PHP	457
6.1.1	Qu'est-ce qu'une SAPI ?	457
6.1.2	Module du serveur	458
6.1.3	CGI	458
6.1.4	FastCGI	459
6.1.5	Conclusion	459
6.2	Mise en cache d'OPCodes	460
6.2.1	Les OPCodes	460
6.2.2	Une étape lourde	461
6.2.3	La mise en cache	461
6.3	La compression des réponses	461
6.3.1	Compression gzip	461
6.3.2	Précompression	462
6.4	Optimisation des images	463
6.4.1	Validation	463
6.4.2	Expiration	463
6.4.3	Autres techniques	463
7.	Test des performances d'un site web	464
7.1	Côté serveur	464
7.1.1	Apache Bench	464
7.1.2	Xhprof	465
7.2	Côté client	465

Chapitre 14

Internationalisation des applications Symfony

1. Introduction	467
1.1 Culture, internationalisation et régionalisation.....	467
1.1.1 La culture (Locale)	468
1.1.2 Internationalisation.....	468
1.1.3 Régionalisation	468
1.2 L'internationalisation dans Symfony	469
2. Détecter la culture d'un utilisateur	469
2.1 Les techniques.....	469
2.1.1 Négociation de contenu	469
2.1.2 Par l'URL.....	470
2.2 En pratique	470
3. Activation des traductions.....	471
3.1 Le composant translator	471
3.2 Configuration du framework.....	471
4. Les routes et les traductions.....	472
5. Les fichiers de traductions	473
5.1 Organisation et règles de nommage	473
5.1.1 Règle de nommage.....	473
5.2 Outillage pour la création des fichiers de traduction	474
5.2.1 Afficher la liste des traductions manquantes	474
5.2.2 Générer un fichier de traduction	475
6. Traduction d'un message	475
6.1 Le service translator	475
6.2 Les paramètres de substitution (placeholders)	476
6.3 Utilisation dans les templates Twig	477

Chapitre 15**Développer une API REST avec Symfony**

1. Introduction à REST et concepts fondamentaux.	479
1.1 Les concepts de REST.	479
1.1.1 Les ressources.	480
1.1.2 Le changement d'état d'une ressource.	480
1.2 Architecture et protocole HTTP.	480
1.3 Les Single-Page Applications.	481
2. La gestion du format JSON.	482
2.1 Présentation du format JSON.	482
2.1.1 Représentation des données en JSON.	483
2.1.2 Types de données.	483
2.1.3 Structures.	484
2.2 Le support de JSON en PHP.	485
2.3 Et dans Symfony ?	486
3. Mise en place d'une API REST.	486
3.1 Le service serializer.	487
3.1.1 Sérialiser des données.	487
3.1.2 Désérialiser des données.	488
3.2 Adaptation des contrôleurs.	488
4. Les objets de requête et de réponse.	489
4.1 Le contenu et les en-têtes de requête.	490
4.2 Manipulation de la réponse avec Response et JsonResponse.	491
4.2.1 Renvoyer une réponse au format JSON.	492
4.3 Les codes de réponse HTTP dans une API REST.	493
4.3.1 Problématique de l'état de la réponse.	493
4.3.2 Expression de la réponse avec HTTP.	493
4.3.3 Mise en œuvre.	494

5. Tester une API REST	495
5.1 Les limites du navigateur web	495
5.2 Les outils	496
5.2.1 Postman	496
5.2.2 SOAP UI	498

Annexes

1. Créer une commande pour la console	499
1.1 La configuration d'une commande	499
1.2 Les objets input et output	500
1.3 Le Service Container	502
1.4 Commande d'exemple	503
2. Envoyer des e-mails grâce à SwiftMailer	504
2.1 Le protocole SMTP	505
2.2 Le transport	505
2.2.1 Le transport smtp	506
2.2.2 Le transport sendmail	506
2.3 Envoi d'un e-mail	507
3. Travailler avec les sessions	508
3.1 Introduction	508
3.2 Intégration des sessions dans Symfony	509
3.3 Configuration du gestionnaire de sauvegarde	509
3.3.1 Avec PHP	509
3.3.2 Avec Symfony	510
3.4 Les messages « flash »	511
4. Déployer une application symfony	513
4.1 Le déploiement	513
4.2 Faut-il déployer par FTP ?	513
4.3 Les différentes étapes	514

4.4	Capistrano et Capifony	515
4.4.1	Installation.....	515
4.4.2	Configuration	516
4.4.3	Déploiement	518
4.5	Fonctionnalités avancées	518
Index		519

Chapitre 6

Les templates avec Twig

1. Présentation et concepts

1.1 Le concept de Templating

La notion de template (ou modèle) fait référence au principe d'élaboration de modèles statiques de pages HTML dans lesquels viendront s'insérer des données dynamiques résultant de l'exécution du code PHP.

Le templating est l'approche générale de découpage entre ces données statiques et dynamiques, toujours dans l'optique de séparer les responsabilités : les traitements applicatifs d'un côté, la présentation des données issues de ces traitements de l'autre.

1.2 Templating et modèle MVC

Dans la mise en œuvre du modèle MVC (cf. Architecture du framework - Le modèle de conception MVC), les templates endossent la responsabilité de l'affichage des données, c'est-à-dire : la vue. Ou plus précisément « les vues » dans la mesure où chaque action implémentée dans les contrôleurs a la responsabilité de prévoir un affichage spécifique en invoquant le template associé.

Chaque template est donc responsable d'une génération de contenu HTML à destination du navigateur web. Ce contenu HTML est produit en combinant le code statique, utilisé pour la mise en forme générale des pages, avec le code dynamique permettant d'intégrer les données, bien que ce dernier soit simplement limité à l'expression de variables contenant ces données. En effet, il est hors de question de trouver du code PHP exécutant une requête SQL dans une vue !

2. Twig

2.1 Présentation

Twig est un moteur de templates, une librairie permettant de gérer la couche « présentation », pour les applications utilisant le modèle de conception MVC.

Globalement, le principe est d'extraire tout ce qui est relatif à la vue dans des fichiers dédiés, appelés « *templates* », qui représentent pour la plupart du HTML.

Nous insistons ici sur le terme « représenter ». En effet, les templates ne contiennent pas forcément le code HTML tel qu'il sera retourné par l'application, ils contiennent bien souvent du code permettant de générer ce code HTML. Ce code utilise un langage spécifique au moteur de template et ce langage est communément appelé, par extension, du « Twig ».

Twig est un projet qui voit le jour en 2008. Son auteur Armin Ronacher s'inspire très largement du framework Jinja, un moteur de template pour Python. Aujourd'hui, le projet Twig est maintenu par les équipes de développement de Symfony dont il est le moteur de template par défaut. Le site officiel de Twig est accessible à l'adresse <https://twig.symfony.com>.

2.2 Pourquoi un nouveau langage ?

Une première question, tout à fait légitime, peut vous venir à l'esprit suite à la description que nous venons d'effectuer : pourquoi utiliser Twig et non PHP ?

On pourrait très bien imaginer une utilisation correcte du modèle de conception MVC, avec extraction de la couche Vue au sein de fichiers dédiés (*templates*) et utilisant le langage PHP.

Sachez que Symfony permet cela (même si c'est Twig qui est configuré par défaut). Néanmoins, nous n'aborderons pas cette solution au cours de ce chapitre ; nous nous concentrerons uniquement sur Twig, et ce pour plusieurs raisons :

- Twig est rapide. Bien qu'il utilise son propre langage, et par conséquent son propre moteur de parsing (car, au final, c'est du code PHP qui doit être généré). Le code PHP généré par chaque template est mis en cache. Ce processus n'a donc pas lieu à chaque requête. L'impact sur les performances, par rapport à du PHP pur, est donc *minime*.
- Twig est sécurisé. Les données affichées sont immunisées par défaut contre les éventuelles attaques malveillantes, comme le cross-site scripting (XSS) par exemple.
- Twig est adapté aux templates. Ce langage a été spécialement conçu dans cette optique et nous verrons tout au long de ce chapitre qu'il nous offre une multitude de raccourcis (qui sont impossibles en PHP) nous permettant de garder nos templates lisibles et simples. Si vous avez déjà eu l'occasion d'utiliser le framework Django (pour le langage Python), vous ne serez pas dépaysé : la syntaxe de Twig est très semblable à celle utilisée dans les templates sous Django.
- Twig est largement utilisé car il est le moteur de template par défaut de Symfony. La plupart des projets l'utilisent et beaucoup de bundles open source supportent uniquement Twig, ce qui est également le cas des tutoriels ou articles qu'on peut trouver sur Internet. Ils sont très souvent orientés sur ce moteur de template et non PHP.

2.3 Mise en pratique

Voyons comment créer concrètement cette couche Vue au sein de notre application. Nous connaissons pour l'instant le processus entre une requête et une réponse (cf. Architecture du framework et Routage et contrôleur). Nous savons que le contrôleur est chargé de retourner une réponse HTTP, sous la forme d'un objet de type **Symfony\Component\HttpFoundation\Response** :

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController
{
    /**
     * @Route("/")
     */
    public function hello()
    {
        return new Response('Hello world!');
    }
}
```

Ici, l'application retourne une réponse ayant comme contenu **Hello world!** (la vue), qui est directement générée depuis le contrôleur. Dans ce cas précis, cela n'est pas particulièrement choquant, mais imaginez une page web, avec ses nombreuses balises... le code du contrôleur deviendrait rapidement illisible.

Déléguons la génération du contenu de la réponse à Twig.

Contrôleur

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController
{
```

```
/**
 * @Route("/")
 */
public function hello()
{
    return $this->render(
        'default/hello.html.twig'
    );
}
```

Template

```
{# templates/default/hello.html.twig #}
Hello world!
```

Remarque

{# templates/default/hello.html.twig #} n'est pas obligatoire car pour Twig, c'est un commentaire. Ici, nous l'utilisons uniquement dans le but d'aider le lecteur, en indiquant le chemin vers le template.

2.4 Remarques sur l'utilisation

Nous avons vu au cours des chapitres précédents (Architecture du framework et L'injection de dépendances) que, pour Symfony, tous les outils/fonctionnalités étaient regroupés dans des « services ». Twig n'échappe pas à cette règle.

Effectivement, nous invoquons la méthode **render** de la classe **Symfony\Bundle\FrameworkBundle\Controller\AbstractController**. En allant voir son contenu, on remarque qu'elle fait bel et bien appel à un service (dont l'identifiant est « templating »).

Cela rejoint nos explications précédentes (cf. Architecture du framework - Architecture de Symfony, se référer notamment au schéma descriptif) ; la **vue** est tout simplement un service, sur lequel nous pouvons nous appuyer pour générer des réponses, au même titre que n'importe quel autre service. Ce n'est pas un service spécial ou particulier.

2.5 La notation des templates

De la même manière que le composant Router utilise une notation particulière pour référencer une action, chaque action utilisant un template doit le référencer selon une certaine convention.

Les templates sont enregistrés dans le répertoire **templates/** de l'application. Ce répertoire contient généralement un sous-répertoire pour chaque contrôleur contenant les templates spécifiques à ce contrôleur. L'emplacement de ce répertoire est défini dans la configuration de Twig qui se trouve dans le fichier **config/packages/twig.yaml** ; en voici le contenu par défaut :

```
twig:
  default_path: '%kernel.project_dir%/templates'
  debug: '%kernel.debug%'
  strict_variables: '%kernel.debug%'
  exception_controller: null
```

Nous pouvons notamment y constater la définition de la variable **default_path** qui pointe vers ce répertoire **templates/**.

■ Remarque

%kernel.project_dir% est une variable représentant le répertoire racine du projet. Elle est utilisée dans de nombreux fichiers de configuration Symfony.

La notation pour les templates est la suivante : **chemin/vers/template** ou le chemin est exprimé de manière relative à partir du répertoire **templates/**. Se présentent alors deux possibilités de localisation des templates :

- Les templates directement localisés dans le répertoire **templates/**, qui seront des templates généraux pour l'application ou bien des gabarits de page (cette notion est évoquée un peu plus loin dans ce chapitre). En utilisant par exemple **base.html.twig** dans un contrôleur, on référence le fichier **templates/base.html.twig**.
- Les templates associés à une action de contrôleur qui sont, eux, rangés dans un dossier portant le nom du contrôleur. Ainsi, conventionnellement, l'action **afficher()** du contrôleur **AdminController** sera associée au template présent dans **templates/admin/afficher.html.twig**, son appel dans le contrôleur se faisant avec le nom **admin/afficher.html.twig**.

2.6 Extension du système de templates

En plus du répertoire conventionnel **templates/**, il est possible d'ajouter d'autres dossiers contenant des templates afin de mieux organiser son code. Pour cela il faut intervenir dans la configuration de Twig stockée dans le fichier **config/packages/twig.yaml**. La propriété **paths** de ce fichier permet de référencer plusieurs répertoires supplémentaires et de les associer à un espace de noms afin d'éviter tout conflit de nommage. Prenons l'exemple de la configuration suivante ajoutée au fichier **config/packages/twig.yaml** :

```
twig:
    # ...
    paths:
        'admin/templates': 'admin'
        'backend/templates': 'back'
```

Les répertoires **admin/templates** et **backend/templates**, tous deux situés à la racine du projet, sont ajoutés au système de localisation des templates par Symfony et possèdent respectivement les espaces de noms **admin** et **back**. Les fichiers de ces répertoires seront identifiés par la syntaxe **@EspaceDe-Nom/nom_du_fichier.html.twig**.

Avec cet ajout, pour référencer le fichier **index.html.twig** situé dans **admin/templates**, on utilisera **@admin/index.html.twig**.

2.7 L'annotation @Template

Le bundle SensioFrameworkExtraBundle intégré par défaut, offre un raccourci intéressant pour l'affichage de vos templates. Plutôt que d'invoquer la méthode **render()** du contrôleur comme nous venons de le voir, vous pouvez tout simplement apposer une annotation sur votre action :

```
namespace App\Controller;

use
    Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Template;
```