

Table des matières

AVANT-PROPOS	V
Découvrir l'étude de cas développée • V	
En quoi cet ouvrage est-il différent ? • VI	
Organisation de l'ouvrage • VII	
Remerciements • XI	
1. DÉMARRAGE DU PROJET	1
Installer et configurer les bases du projet • 2	
Les prérequis techniques pour démarrer • 2	
Installer les librairies du framework Symfony • 2	
Installation du projet • 5	
Générer la structure de base du projet • 5	
Générer la structure de base de la première application frontend • 6	
Configuration du chemin vers les librairies de Symfony • 7	
Découvrir les environnements émulés par Symfony • 7	
Quels sont les principaux environnements en développement web ? • 8	
Spécificités de l'environnement de développement • 8	
Spécificités de l'environnement de production • 9	
Configurer le serveur web • 10	
Méthode 1 : configuration dangereuse à ne pas reproduire • 10	
Méthode 2 : configuration sûre et recommandée • 11	
Création d'un nouveau serveur virtuel pour Jobeet • 11	
Tester la nouvelle configuration d'Apache • 12	
Contrôler le code source avec Subversion • 14	
Quels sont les avantages d'un gestionnaire de versions ? • 14	
Installer et configurer le dépôt Subversion • 14	
En résumé... • 16	
2. L'ÉTUDE DE CAS	19
À la découverte du projet... • 20	
Découvrir les spécifications fonctionnelles de Jobeet • 22	
Les différents acteurs et applications impliqués • 22	
Utilisation de l'application grand public : le frontend • 22	
Scénario F1 : voir les dernières offres en page d'accueil • 22	
Scénario F2 : voir les offres d'une catégorie • 23	
Scénario F3 : affiner la liste des offres avec des mots-clés • 24	
Scénario F4 : obtenir le détail d'une offre • 24	
Scénario F5 : poster une nouvelle annonce • 25	
Scénario F6 : s'inscrire en tant qu'affilié pour utiliser l'API • 27	
Scénario F7 : l'affilié récupère la liste des dernières offres actives • 27	
Utilisation de l'interface d'administration : le backend • 27	
Scénario B1 : gérer les catégories • 27	
Scénario B2 : gérer les offres d'emploi • 28	
Scénario B3 : gérer les comptes administrateur • 28	
Scénario B4 : configurer le site Internet • 28	
En résumé... • 29	
3. CONCEVOIR LE MODÈLE DE DONNÉES	31
Installer la base de données • 32	
Créer la base de données MySQL • 32	
Configurer la base de données pour le projet Symfony • 32	
Présentation de la couche d'ORM Doctrine • 33	
Qu'est-ce qu'une couche d'abstraction de base de données ? • 34	
Qu'est-ce qu'un ORM ? • 34	
Activer l'ORM Doctrine pour Symfony • 35	
Concevoir le modèle de données • 36	
Découvrir le diagramme UML « entité-relation » • 36	
Mise en place du schéma de définition de la base • 37	
De l'importance du schéma de définition de la base de données... • 37	
Écrire le schéma de définition de la base de données • 37	
Déclaration des attributs des colonnes d'une table en format YAML • 39	
Générer la base de données et les classes du modèle avec Doctrine • 40	
Construire la base de données automatiquement • 40	
Découvrir les classes du modèle de données • 41	
Générer la base de données et le modèle en une seule passe • 42	
Préparer les données initiales de Jobeet • 43	
Découvrir les différents types de données d'un projet Symfony • 43	
Définir des jeux de données initiales pour Jobeet • 44	
Charger les jeux de données de tests en base de données • 46	
Régénérer la base de données et le modèle en une seule passe • 46	
Profiter de toute la puissance de Symfony dans le navigateur • 47	

Générer le premier module fonctionnel « job » • 47	
Composition de base d'un module généré par Symfony • 47	
Découvrir les actions du module « job » • 48	
Comprendre l'importance de la méthode magique __toString() • 49	
Ajouter et éditer les offres d'emploi • 50	
En résumé... • 50	
4. LE CONTRÔLEUR ET LA VUE 53	
L'architecture MVC et son implémentation dans Symfony • 54	
Habiller le contenu de chaque page avec un même gabarit • 55	
Décorer une page avec un en-tête et un pied de page • 55	
Décorer le contenu d'une page avec un décorateur • 56	
Intégrer la charte graphique de Jobeet • 58	
Récupérer les images et les feuilles de style • 58	
Configurer la vue à partir d'un fichier de configuration • 59	
Configurer la vue à l'aide des helpers de Symfony • 61	
Générer la page d'accueil des offres d'emploi • 62	
Écrire le contrôleur de la page : l'action index • 62	
Créer la vue associée à l'action : le template • 63	
Personnaliser les informations affichées pour chaque offre • 64	
Générer la page de détail d'une offre • 66	
Créer le template du détail de l'offre • 66	
Mettre à jour l'action show • 67	
Utiliser les emplacements pour modifier dynamiquement le titre des pages • 68	
Définition d'un emplacement pour le titre • 68	
Fixer la valeur d'un slot dans un template • 68	
Fixer la valeur d'un slot complexe dans un template • 69	
Déclarer une valeur par défaut pour le slot • 69	
Rediriger vers une page d'erreur 404 si l'offre n'existe pas • 70	
Comprendre l'interaction client/serveur • 71	
Récupérer le détail de la requête envoyée au serveur • 71	
Récupérer le détail de la réponse envoyée au client • 72	
En résumé... • 73	
5. LE ROUTAGE..... 75	
À la découverte du framework de routage de Symfony • 76	
Rappels sur la notion d'URL • 76	
Qu'est-ce qu'une URL ? • 76	
Introduction générale au framework interne de routage • 77	
Configuration du routage : le fichier routing.yml • 77	
Découverte de la configuration par défaut du routage • 77	
Comprendre le fonctionnement des URL par défaut de Symfony • 79	
Personnaliser les routes de l'application • 80	
Configurer la route de la page d'accueil • 80	
Configurer la route d'accès au détail d'une offre • 80	
Forcer la validation des paramètres des URLs internes de l'application • 82	
Limiter une requête à certaines méthodes HTTP • 82	
Optimiser la création de routes grâce à la classe de route d'objets Doctrine • 83	
Transformer la route d'une offre en route Doctrine • 83	
Améliorer le format des URL des offres d'emploi • 84	
Retrouver l'objet grâce à sa route depuis une action • 86	
Faire appel au routage depuis les actions et les templates • 87	
Le routage dans les templates • 87	
Le routage dans les actions • 88	
Découvrir la classe de collection de routes sfDoctrineRouteCollection • 88	
Déclarer une nouvelle collection de routes Doctrine • 88	
Émuler les méthodes PUT et DELETE • 90	
Outils et bonnes pratiques liés au routage • 91	
Faciliter le débogage en listant les routes de l'application • 91	
Supprimer les routes par défaut • 93	
En résumé... • 93	
6. OPTIMISATION DU MODÈLE ET REFACTORING 95	
Présentation de l'objet Doctrine_Query • 96	
Débuguer le code SQL généré par Doctrine • 97	
Découvrir les fichiers de log • 97	
Avoir recours à la barre de débogage • 97	
Intervenir sur les propriétés d'un objet avant sa sérialisation en base de données • 98	
Redéfinir la méthode save() d'un objet Doctrine • 98	
Récupérer la liste des offres d'emploi actives • 99	
Mettre à jour les données de test pour s'assurer de la validité des offres affichées • 99	
Gérer les paramètres personnalisés d'une application dans Symfony • 100	
Découvrir le fichier de configuration app.yml • 100	
Récupérer une valeur de configuration depuis le modèle • 101	
Remanier le code en continu pour respecter la logique MVC • 101	
Exemple de déplacement du contrôleur vers le modèle • 102	
Avantages du remaniement de code • 102	
Ordonner les offres suivant leur date d'expiration • 103	
Classer les offres d'emploi selon leur catégorie • 103	
Limiter le nombre de résultats affichés • 105	
Modifier les données de test dynamiquement par l'ajout de code PHP • 107	
Empêcher la consultation d'une offre expirée • 108	
Créer une page dédiée à la catégorie • 110	
En résumé... • 110	

7. CONCEVOIR ET PAGINER LA LISTE D'OFFRES

- D'UNE CATÉGORIE 113**
- Mise en place d'une route dédiée à la page de la catégorie • 114
 - Déclarer la route category dans le fichier routing.yml • 114
 - Implémenter l'accessor getSlug() dans la classe JobeetJob • 114
- Personnaliser les conditions d'affichage du lien de la page de catégorie • 115
 - Intégrer un lien pour chaque catégorie ayant plus de dix offres valides • 115
 - Implémenter la méthode countActiveJobs() de la classe JobeetCategory • 116
 - Implémenter la méthode countActiveJobs() de la classe JobeetCategoryTable • 116
- Mise en place du module dédié aux catégories • 118
 - Générer automatiquement le squelette du module • 118
 - Ajouter un champ supplémentaire pour accueillir le slug de la catégorie • 119
 - Création de la vue de détail de la catégorie • 119
 - Mise en place de l'action executeShow() • 119
 - Intégration du template showSuccess.php associé • 120
 - Isoler le HTML redondant dans les templates partiels • 121
 - Découvrir le principe de templates partiels • 121
 - Création d'un template partiel _list.php pour les modules job et category • 122
 - Faire appel au partiel dans un template • 122
 - Utiliser le partiel _list.php dans les templates indexSuccess.php et showSuccess.php • 123
- Paginer une liste d'objets Doctrine • 123**
 - Que sont les listes paginées et à quoi servent-elles ? • 123
 - Préparer la pagination à l'aide de sfDoctrinePager • 124
 - Initialiser la classe de modèle et le nombre maximum d'objets par page • 124
 - Spécifier l'objet Doctrine_Query de sélection des résultats • 125
 - Configurer le numéro de la page courante de résultats • 125
 - Initialiser le composant de pagination • 125
 - Simplifier les méthodes de sélection des résultats • 126
 - Implémenter la méthode getActiveJobsQuery de l'objet JobeetCategory • 126
 - Remanier les méthodes existantes de JobeetCategory • 126
 - Intégrer les éléments de pagination dans le template showSuccess.php • 127
 - Passer la collection d'objets Doctrine au template partiel • 127
 - Afficher les liens de navigation entre les pages • 128
 - Afficher le nombre total d'offres publiées et de pages • 129
 - Description des méthodes de l'objet sfDoctrinePager utilisées dans le template • 129
 - Code final du template showSuccess.php • 130

En résumé... • 131

8. LES TESTS UNITAIRES 133

- Présentation des types de tests dans Symfony • 134
- De la nécessité de passer par des tests unitaires • 134
- Présentation du framework de test lime • 135
 - Initialisation d'un fichier de tests unitaires • 135
 - Découverte des outils de tests de lime • 135
- Exécuter une suite de tests unitaires • 136
- Tester unitairement la méthode slugify() • 137
 - Déterminer les tests à écrire • 137
 - Écrire les premiers tests unitaires de la méthode • 138
 - Commenter explicitement les tests unitaires • 138
- Implémenter de nouveaux tests unitaires au fil du développement • 140
 - Ajouter des tests pour les nouvelles fonctionnalités • 140
 - Ajouter des tests suite à la découverte d'un bug • 141
 - Implémenter une meilleure méthode slugify • 142
- Implémentation des tests unitaires dans le framework ORM Doctrine • 144
 - Configuration de la base de données • 144
 - Mise en place d'un jeu de données de test • 145
 - Vérifier l'intégrité du modèle par des tests unitaires • 145
 - Initialiser les scripts de tests unitaires de modèles Doctrine • 145
 - Tester la méthode getCompanySlug() de l'objet JobeetJob • 146
 - Tester la méthode save() de l'objet JobeetJob • 146
 - Implémentation des tests unitaires dans d'autres classes Doctrine • 147
- Lancer l'ensemble des tests unitaires du projet • 148
- En résumé... • 148

9. LES TESTS FONCTIONNELS 151

- Découvrir l'implémentation des tests fonctionnels • 152
 - En quoi consistent les tests fonctionnels ? • 152
 - Implémentation des tests fonctionnels • 153
- Manipuler les composants de tests fonctionnels • 153
 - Simuler le navigateur grâce à l'objet sfBrowser • 153
 - Tester la navigation en simulant le comportement d'un véritable navigateur • 153
 - Modifier le comportement du simulateur de navigateur • 154
 - Préparer et exécuter des tests fonctionnels • 155
 - Comprendre la structure des fichiers de tests • 155
 - Découvrir le testeur sfTesterRequest • 157
 - Découvrir le testeur sfTesterResponse • 157
 - Exécuter les scénarios de tests fonctionnels • 158
- Charger des jeux de données de tests • 158

Écrire des tests fonctionnels pour le module d'offres • 159

- Les offres d'emploi expirées ne sont pas affichées • 160
- Seulement N offres sont listées par catégorie • 160
- Un lien vers la page d'une catégorie est présent lorsqu'il y a trop d'offres • 161
- Les offres d'emploi sont triées par date • 162
- Chacune des offres de la page d'accueil est cliquable • 163

Autres exemples de scénarios de tests pour les pages des modules job et category • 164

- Déboguer les tests fonctionnels • 167
- Exécuter successivement des tests fonctionnels • 167
- Exécuter les tests unitaires et fonctionnels • 168
- En résumé... • 168

10. ACCÉLÉRER LA GESTION DES FORMULAIRES 171**À la découverte des formulaires avec Symfony • 172**

- Les formulaires de base • 172
- Les formulaires générés par les tâches Doctrine • 174
- Personnaliser le formulaire d'ajout ou de modification d'une offre • 174
- Supprimer les champs inutiles du formulaire généré • 175
- Redéfinir plus précisément la configuration d'un champ • 175
 - Utiliser le validateur sfValidatorEmail • 176
 - Remplacer le champ permettant le choix du type d'offre par une liste déroulante • 176
 - Personnaliser le widget permettant l'envoi du logo associé à une offre • 178
 - Modifier plusieurs labels en une seule passe • 180
 - Ajouter une aide contextuelle sur un champ • 180
 - Présentation de la classe finale de configuration du formulaire d'ajout d'une offre • 180

Manipuler les formulaires directement dans les templates • 182

- Générer le rendu d'un formulaire • 182
- Personnaliser le rendu des formulaires • 183
 - Découvrir les méthodes de l'objet sfForm • 183
 - Comprendre et implémenter les méthodes de l'objet sfFormField • 184

Manipuler les formulaires dans les actions • 184

- Découvrir les méthodes autogénérées du module job utilisant les formulaires • 185
- Traiter les formulaires dans les actions • 186
 - Simplifier le traitement du formulaire dans le module job • 186
 - Comprendre le cycle de vie du formulaire • 187
 - Définir les valeurs par défaut d'un formulaire généré par Doctrine • 187
- Protéger le formulaire des offres par l'implémentation d'un jeton • 188

Générer le jeton automatiquement à la création • 188

Redéfinir la route d'édition de l'offre grâce au jeton • 189

Construire la page de prévisualisation • 190**Activer et publier une offre • 192**

- Préparer la route vers l'action de publication • 192
- Implémenter la méthode executePublish() • 193
- Implémenter la méthode publish() de l'objet JobeeJob • 193
- Empêcher la publication et l'accès aux offres non actives • 194
- En résumé... • 195

11. TESTER LES FORMULAIRES 197**Utiliser le framework de formulaires de manière autonome • 198****Écrire des tests fonctionnels pour les classes de formulaire • 199**

- Tester l'envoi du formulaire de création d'offre • 199
 - Renommer le nom des champs du formulaire • 200
 - Soumettre le formulaire à l'aide de la méthode click() • 200
- Découvrir le testeur sfTesterForm • 201
 - Tester si le formulaire est erroné • 201
 - Les méthodes de l'objet sfTesterForm • 201
 - Déboguer un formulaire • 202
- Tester les redirections HTTP • 202
- Tester les objets générés par Doctrine • 202
 - Activer le testeur sfTesterDoctrine • 203
 - Tester l'existence d'un objet Doctrine dans la base de données • 203
- Tester les erreurs des champs du formulaire • 203
 - La méthode isError() pour le contrôle des champs • 204
 - Tester la barre d'administration d'une offre • 205
- Forcer la méthode HTTP d'un lien • 206
 - Forcer l'utilisation de la méthode HTTP PUT • 206
 - Forcer l'utilisation de la méthode HTTP DELETE • 206

Écrire des tests fonctionnels afin de découvrir des bogues • 207

- Simuler l'autopublication d'une offre • 207
- Contrôler la redirection vers une page d'erreur 404 • 208
- Empêcher l'accès au formulaire d'édition lorsque l'offre est publiée • 209

Tester la prolongation d'une offre • 210

- Comprendre le problème des offres expirées à réactiver • 210
- Une route dédiée pour prolonger la durée d'une offre • 210
- Implémenter la méthode executeExtend() aux actions du module job • 211
- Implémenter la méthode extend() dans JobeeJob • 212
- Tester la prolongation de la durée de vie d'une offre • 212

Sécuriser les formulaires • 214

- Sérialisation d'un formulaire Doctrine • 214
- Sécurité native du framework de formulaire • 214

Se protéger contre les attaques CSRF et XSS • 216

Les tâches automatiques de maintenance • 216Créer la nouvelle tâche de maintenance `jobeet:cleanup` • 217Implémenter la méthode `cleanup()` de la classe`JobeetJobTable` • 218

En résumé... • 219

12. LE GÉNÉRATEUR D'INTERFACE D'ADMINISTRATION..... 221**Création de l'application « backend » • 222**

Générer le squelette de l'application • 222

Recharger les jeux de données initiales • 222

Générer les modules d'administration • 223

Générer les modules `category` et `job` • 223**Personnaliser l'interface utilisateur****et l'ergonomie des modules du backoffice • 224**

Découvrir les fonctions des modules d'administration • 224

Améliorer le layout du backoffice • 225

Comprendre le cache de Symfony • 227

Introduction au fichier de configuration `generator.yml` • 228**Configurer les modules autogénérés par Symfony • 229**Organisation du fichier de configuration `generator.yml` • 229

Configurer les titres des pages des modules auto générés • 229

Changer le titre des pages du module `category` • 229Configurer les titres des pages du module `job` • 230

Modifier le nom des champs d'une offre d'emploi • 231

Redéfinir globalement les propriétés des champs
du module • 231Surcharger localement les propriétés des champs
du module • 231

Comprendre le principe de configuration en cascade • 232

Configurer la liste des objets • 232

Définir la liste des colonnes à afficher • 232

Colonnes à afficher dans la liste des catégories • 232

Liste des colonnes à afficher dans la liste des offres • 233

Configurer le layout du tableau de la vue liste • 233

Déclarer des colonnes « virtuelles » • 234

Définir le tri par défaut de la liste d'objets • 235

Réduire le nombre de résultats par page • 235

Configurer les actions de lot d'objets • 236

Désactiver les actions par lot dans le module `category` • 236Ajouter de nouvelles actions par lot dans le module `job` • 237

Configurer les actions unitaires pour chaque objet • 239

Supprimer les actions d'objets des catégories • 239

Ajouter d'autres actions pour chaque offre d'emploi • 240

Configurer les actions globales de la vue liste • 240

Optimiser les requêtes SQL de récupération
des enregistrements • 243**Configurer les formulaires des vues de saisie de données • 245**Configurer la liste des champs à afficher dans les formulaires
des offres • 245

Ajouter des champs virtuels au formulaire • 247

Redéfinir la classe de configuration du formulaire • 247

Implémenter une nouvelle classe de formulaire par défaut • 247

Implémenter un meilleur mécanisme de gestion
des photos des offres • 249**Configurer les filtres de recherche de la vue liste • 251**Supprimer les filtres du module de `category` • 251Configurer la liste des filtres du module `job` • 251**Personnaliser les actions d'un module autogénéré • 252****Personnaliser les templates d'un module autogénéré • 253****La configuration finale du module • 255**Configuration finale du module `job` • 255Configuration finale du module `category` • 256

En résumé... • 257

13. AUTHENTIFICATION ET DROITS AVEC L'OBJET sfUSER ... 259**Découvrir les fonctionnalités de base de l'objet sfUser • 260**

Comprendre les messages « flash » de feedback • 261

À quoi servent ces messages dans Symfony ? • 261

Écrire des messages flash depuis une action • 261

Lire des messages flash dans un template • 262

Stocker des informations dans la session courante
de l'utilisateur • 262

Lire et écrire dans la session de l'utilisateur courant • 263

Implémenter l'historique de navigation de l'utilisateur • 263

Refactoriser le code de l'historique de navigation
dans le modèle • 264Implémenter l'historique de navigation dans la classe
`myUser` • 264Simplifier l'action `executeShow()` de la couche contrôleur • 265

Afficher l'historique des offres d'emploi consultées • 265

Implémenter un moyen de réinitialiser l'historique
des offres consultées • 267**Comprendre les mécanismes de sécurisation des applications • 268**

Activer l'authentification de l'utilisateur sur une application • 268

Découvrir le fichier de configuration `security.yml` • 268

Analyse des logs générés par Symfony • 269

Personnaliser la page de login par défaut • 269

Authentifier et tester le statut de l'utilisateur • 270

Restreindre les actions d'une application à l'utilisateur • 270

Activer le contrôle des droits d'accès sur l'application • 271

Établir des règles de droits d'accès complexes • 271

Gérer les droits d'accès via l'objet `sfBasicSecurityUser` • 272**Mise en place de la sécurité de l'application backend de Jobeet • 273**Installation du plug-in `sfDoctrineGuardPlugin` • 273

- Mise en place des sécurités de l'application backend • 274
 - Générer les classes de modèle et les tables SQL • 274
 - Implémenter de nouvelles méthodes à l'objet User via la classe sfGuardSecurityUser • 274
 - Activer le module sfGuardAuth et changer l'action de login par défaut • 275
 - Créer un utilisateur administrateur • 276
 - Cacher le menu de navigation lorsque l'utilisateur n'est pas authentifié • 276
 - Ajouter un nouveau module de gestion des utilisateurs • 277
- Implémenter de nouveaux tests fonctionnels pour l'application frontend • 278
- En résumé... • 279
- 14. LES FLUX DE SYNDICATION ATOM 281**
 - Découvrir le support natif des formats de sortie • 282
 - Définir le format de sortie d'une page • 282
 - Gérer les formats de sortie au niveau du routage • 283
 - Présentation générale du format ATOM • 283
 - Les informations globales du flux • 284
 - Les entrées du flux • 284
 - Le flux ATOM minimal valide • 284
 - Générer des flux de syndication ATOM • 285
 - Flux ATOM des dernières offres d'emploi • 285
 - Déclarer un nouveau format de sortie • 285
 - Rappel des conventions de nommage des templates • 286
 - Ajouter le lien vers le flux des offres dans le layout • 287
 - Générer les informations globales du flux • 288
 - Générer les entrées du flux ATOM • 289
 - Flux ATOM des dernières offres d'une catégorie • 290
 - Mise à jour de la route dédiée de la catégorie • 291
 - Mise à jour des liens des flux de la catégorie • 291
 - Factoriser le code de génération des entrées du flux • 292
 - Simplifier le template indexSuccess.atom.php • 293
 - Générer le template du flux des offres d'une catégorie • 293
 - En résumé... • 295
- 15. CONSTRUIRE DES SERVICES WEB 297**
 - Concevoir le service web des offres d'emploi • 298
 - Préparer des jeux de données initiales des affiliés • 298
 - Construire le service web des offres d'emploi • 300
 - Déclaration de la route dédiée du service web • 300
 - Implémenter la méthode getForToken() de l'objet JobeetJobTable • 301
 - Implémenter la méthode getActiveJobs() de l'objet JobeetAffiliate • 301
 - Développer le contrôleur du service web • 302
 - Implémenter l'action executeList() du module api • 302
 - Implémenter la méthode asArray() de JobeetJob • 303
 - Construction des templates XML, JSON et YAML • 304
 - Le format XML • 304
 - Le format JSON • 305
 - Le format YAML • 306
 - Écrire des tests fonctionnels pour valider le service web • 309
 - Formulaire de création d'un compte d'affiliation • 310
 - Déclarer la route dédiée du formulaire d'inscription • 310
 - Générer un module d'amorçage • 311
 - Construction des templates • 311
 - Implémenter les actions du module affiliate • 312
 - Tester fonctionnellement le formulaire • 314
 - Développer l'interface d'administration des affiliés • 315
 - Générer le module d'administration affiliate • 315
 - Paramétrer le module affiliate • 316
 - Implémenter les nouvelles fonctionnalités d'administration • 317
 - Envoyer des e-mails avec Zend_Mail • 320
 - Installer et configurer le framework Zend • 320
 - Implémenter l'envoi d'un e-mail à l'activation du compte de l'affilié • 321
 - En résumé... • 323
- 16. DÉPLOYER UN MOTEUR DE RECHERCHE 325**
 - Découverte de la librairie Zend_Search_Lucene • 326
 - Rappels historiques au sujet de Symfony • 326
 - Présentation de Zend Lucene • 326
 - Indexer le contenu de Jobeet • 327
 - Créer et récupérer le fichier de l'index • 327
 - Mettre à jour l'index à la sérialisation d'une offre • 328
 - Sécuriser la sérialisation d'une offre à l'aide d'une transaction Doctrine • 330
 - Effacer l'index lors de la suppression d'une offre • 331
 - Manipuler l'index des offres d'emploi • 331
 - Régénérer tout l'index des offres d'emploi • 331
 - Implémenter la recherche d'informations pour Jobeet • 331
 - Tester la méthode getForLuceneQuery() de JobeetJob • 334
 - Nettoyer régulièrement l'index des offres périmées • 335
 - En résumé... • 337
- 17. DYNAMISER L'INTERFACE UTILISATEUR AVEC AJAX 339**
 - Choisir un framework JavaScript • 340
 - Découvrir la librairie jQuery • 340
 - Télécharger et installer jQuery • 341
 - Récupérer la dernière version stable du projet • 341
 - Charger la librairie jQuery sur chaque page du site • 341
 - Découvrir les comportements JavaScript avec jQuery • 342
 - Intercepter la valeur saisie par l'utilisateur dans le moteur de recherche • 343

- Exécuter un appel Ajax pour interroger le serveur web • 344
- Cacher dynamiquement le bouton d'envoi du formulaire • 345
- Informé l'utilisateur de l'exécution de la requête Ajax • 345**
 - Faire patienter l'utilisateur avec un « loader » • 345
 - Déplacer le code JavaScript dans un fichier externe • 346
- Manipuler les requêtes Ajax dans les actions • 347**
 - Déterminer que l'action provient d'un appel Ajax • 347
 - Message spécifique pour une recherche sans résultat • 348
- Simuler une requête Ajax avec les tests fonctionnels • 349**
 - En résumé... • 349

18. INTERNATIONALISATION ET LOCALISATION 351

- Que sont l'internationalisation et la localisation ? • 352**
- L'utilisateur au cœur de l'internationalisation • 353**
 - Paramétrer la culture de l'utilisateur • 353
 - Définir et récupérer la culture de l'utilisateur • 353
 - Modifier la culture par défaut de Symfony • 353
 - Déterminer les langues favorites de l'utilisateur • 354
 - Utiliser la culture dans les URLs • 355
 - Transformer le format des URLs de Jobeet • 355
 - Attribuer dynamiquement la culture de l'utilisateur d'après la configuration de son navigateur • 356
 - Tester la culture avec des tests fonctionnels • 359
 - Mettre à jour les tests fonctionnels qui échouent • 359
 - Tester les nouvelles implémentations liées à la culture • 359
 - Changer de langue manuellement • 360
 - Installer le plug-in sfFormExtraPlugin • 361
 - Intégration non conforme du formulaire de changement de langue • 361
 - Intégration du formulaire de changement de langue avec un composant Symfony • 362
- Découvrir les outils d'internationalisation de Symfony • 365**
 - Paramétrer le support des langues, jeux de caractères et encodages • 365
 - Traduire les contenus statiques des templates • 367
 - Utiliser le helper de traduction `__()` • 367
 - Extraire les contenus internationalisés vers un catalogue XLIFF • 369
 - Traduire des contenus dynamiques • 370
 - Le cas des chaînes dynamiques simples • 371
 - Traduire des contenus pluriels à partir du helper `format_number_choice()` • 372
 - Traduire les contenus propres aux formulaires • 373
- Activer la traduction des objets Doctrine • 373**
 - Internationaliser le modèle `JobeetCategory` de la base • 374
 - Mettre à jour les données initiales de test • 374

- Surcharger la méthode `findOneBySlug()` du modèle `JobeetCategoryTable` • 375
- Méthodes raccourcies du comportement `I18N` • 376
- Mettre à jour le modèle et la route de la catégorie • 376
 - Implémenter la méthode `findOneBySlugAndCulture()` du modèle `JobeetCategoryTable` • 377
 - Mise à jour de la route `category` de l'application frontend • 377
- Champs internationalisés dans un formulaire `Doctrine` • 378
 - Internationaliser le formulaire d'édition d'une catégorie dans le backoffice • 378
 - Utiliser la méthode `embedI18n()` de l'objet `sfFormDoctrine` • 378
 - Internationalisation de l'interface du générateur d'administration • 379
 - Forcer l'utilisation d'un autre catalogue de traductions • 380
 - Tester l'application pour valider le processus de migration de l'I18N • 380
- Découvrir les outils de localisation de Symfony • 381**
 - Régionaliser les formats de données dans les templates • 381
 - Les helpers du groupe `Date` • 381
 - Les helpers du groupe `Number` • 381
 - Les helpers du groupe `I18N` • 382
 - Régionaliser les formats de données dans les formulaires • 382
- En résumé... • 383

19. LES PLUG-INS 385

- Qu'est-ce qu'un plug-in dans Symfony ? • 386**
 - Les plug-ins `Symfony` • 386
 - Les plug-ins privés • 386
 - Les plug-ins publics • 387
 - Une autre manière d'organiser le code du projet • 387
 - Découvrir la structure de fichiers d'un plug-in `Symfony` • 387
- Créer le plug-in `sfJobeetPlugin` • 388**
 - Migrer les fichiers du modèle vers le plug-in • 389
 - Déplacer le schéma de description de la base • 389
 - Déplacer les classes du modèle, de formulaires et de filtres • 389
 - Transformer les classes concrètes en classes abstraites • 389
 - Reconstruire le modèle de données • 390
 - Supprimer les classes de base des formulaires `Doctrine` • 392
 - Déplacer la classe `Jobeet` vers le plug-in • 392
 - Migrer les contrôleurs et les vues • 393
 - Déplacer les modules vers le plug-in • 393
 - Renommer les noms des classes d'actions et de composants • 393
 - Mettre à jour les actions et les templates • 394
 - Mettre à jour le fichier de configuration du routage • 395
 - Activer les modules de l'application frontend • 397

Migrer les tâches automatiques de Jobeet	• 398
Migrer les fichiers d'internationalisation de l'application	• 398
Migrer le fichier de configuration du routage	• 399
Migrer les ressources Web	• 399
Migrer les fichiers relatifs à l'utilisateur	• 399
Configuration du plug-in	• 399
Développement de la classe JobeetUser	• 400
Comparaison des structures des projets et des plug-ins	• 402
Utiliser les plug-ins de Symfony	• 403
Naviguer dans l'interface dédiée aux plug-ins	• 403
Les différentes manières d'installer des plug-ins	• 404
Contribuer aux plug-ins de Symfony	• 405
Packager son propre plug-in	• 405
Construire le fichier README	• 405
Ajouter le fichier LICENSE	• 405
Écrire le fichier package.xml	• 405
Héberger un plug-in public dans le dépôt officiel de Symfony	• 408
En résumé...	• 409
20. LA GESTION DU CACHE 411
Pourquoi optimiser le temps de chargement des pages ?	• 412
Créer un nouvel environnement pour tester le cache	• 413
Comprendre la configuration par défaut du cache	• 413
Ajouter un nouvel environnement cache au projet	• 414
Configuration générale de l'environnement cache	• 414
Créer le contrôleur frontal du nouvel environnement	• 414
Configurer le nouvel environnement	• 415
Manipuler le cache de l'application	• 415
Configuration globale du cache de l'application	• 416
Activer le cache ponctuellement page par page	• 416
Activation du cache de la page d'accueil de Jobeet	• 416
Principe de fonctionnement du cache de Symfony	• 417
Activer le cache de la page de création d'une nouvelle offre	• 418
Nettoyer le cache de fichiers	• 418
Activer le cache uniquement pour le résultat d'une action	• 419
Exclure la mise en cache du layout	• 419
Fonctionnement de la mise en cache sans layout	• 420
Activer le cache des templates partiels et des composants	• 421
Configuration du cache	• 421
Principe de fonctionnement de la mise en cache	• 422
Activer le cache des formulaires	• 423
Comprendre la problématique de la mise en cache des formulaires	• 423
Désactiver la création du jeton unique	• 424
Retirer le cache automatiquement	• 425
Configurer la durée de vie du cache de la page d'accueil	• 425
Forcer la régénération du cache depuis une action	• 425
Tester le cache à partir des tests fonctionnels	• 427
Activer le cache pour l'environnement de test	• 427
Tester la mise en cache du formulaire de création d'une offre d'emploi	• 427
En résumé...	• 428
21. LE DÉPLOIEMENT EN PRODUCTION 431
Préparer le serveur de production	• 432
Vérifier la configuration du serveur web	• 432
Installer l'accélérateur PHP APC	• 433
Installer les bibliothèques du framework Symfony	• 433
Embarquer le framework Symfony	• 433
Garder Symfony à jour en temps réel	• 434
Personnaliser la configuration de Symfony	• 436
Configurer l'accès à la base de données	• 436
Générer les liens symboliques pour les ressources web	• 436
Personnaliser les pages d'erreur par défaut	• 436
Remplacer les pages d'erreur interne par défaut	• 436
Personnaliser les pages d'erreur 404 par défaut	• 437
Personnaliser la structure de fichiers par défaut	• 437
Modifier le répertoire par défaut de la racine web	• 437
Modifier les répertoires du cache et des logs	• 438
À la découverte des factories	• 438
Initialisation des objets du noyau grâce à factories.yml	• 439
Modification du nom du cookie de session	• 439
Remplacer le moteur de stockage des sessions par une base de données	• 440
Définir la durée de vie maximale d'une session	• 440
Définir les objets d'enregistrement d'erreur	• 441
Déployer le projet sur le serveur de production	• 442
Que faut-il déployer en production ?	• 442
Mettre en place des stratégies de déploiement	• 442
Déploiement à l'aide d'une connexion SSH et rsync	• 442
Configurer rsync pour exclure certains fichiers du déploiement	• 443
Nettoyer le cache de configuration du serveur de production	• 444
En résumé...	• 445
A. LE FORMAT YAML 447
Les données scalaires	• 448
Les chaînes de caractères	• 448
Les nombres	• 449
Les entiers	• 449
Les nombres octaux	• 449
Les nombres hexadécimaux	• 450
Les nombres décimaux	• 450
Les nombres exponentiels	• 450

