



Expert
EXPO
EX

DJANGO

Industrialisez vos
développements
Python

Téléchargement
www.editions-eni.fr



Franck FOURNIER

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

Avant-propos

1. Public concerné	19
2. Prérequis	19
3. Objectifs	20
4. Organisation du livre	20

Chapitre 1 Présentation du framework Django

1. Les frameworks en général	21
2. Django	22
2.1 Introduction	22
2.2 Fiche technique	24
2.3 Bref historique	24
2.4 Principes techniques	25
2.5 Les forces et faiblesses de Django	26
2.5.1 Les points forts	26
2.5.2 Les faiblesses	29
3. Pourquoi adopter Django ?	29
3.1 Python « batteries included »	29
3.2 Principes philosophiques de conception	30
3.3 Avantages pour le développeur	32
4. Comparatif avec d'autres frameworks web	33

Chapitre 2

Pour débiter

1. Introduction	37
2. Principes de base, fondamentaux et vue d'ensemble	38
3. Installer Django et démarrer en cinq minutes	40
4. Premiers pas (démarrage pas à pas)	44
4.1 Créer votre premier projet	44
4.2 Créer votre application	52
4.3 Le site d'administration de Django	63
4.4 Les vues Django	72
4.5 Les templates	78
4.6 Utilisation des noms d'URL avec des contextes de nommage (namespaces)	83
4.7 Les formulaires	83
4.8 Les vues génériques	86
4.9 Gestion de tests automatiques	88
4.10 Les fichiers statiques	97

Chapitre 3

Les outils de développement

1. Les outils de développement et les IDE	101
1.1 Eclipse et Pydev	101
1.1.1 Créer un nouveau projet	104
1.1.2 Travailler avec un projet existant	105
1.1.3 Utilisation de Django	106
1.1.4 Run/Debug avec Django	107
1.1.5 Utilisation de l'option autoreload du serveur de test de Django avec PyDev	107
1.2 PyCharm	107
1.2.1 Les atouts de PyCharm	108
1.2.2 Quelques critiques	109
1.2.3 Quelques critiques infondées	109
1.2.4 Pycharm en action en quelques exemples	110
1.3 Autres éditeurs ou IDE pouvant être utilisés avec Python	112

- 2. Les outils de mise au point 113
 - 2.1 Les fichiers de log (traces d'exécution) 114
 - 2.2 Django Debug Toolbar 118
 - 2.2.1 Installation 118
 - 2.2.2 Utilisation 119
 - 2.3 Débugueurs Python 121
 - 2.3.1 pdb 121
 - 2.3.2 pudb (mode console uniquement) 121
 - 2.3.3 trepan 122
 - 2.4 Les outils d'analyse de performance (profiling) 122
 - 2.4.1 django-profiler 2.0.0. 123
 - 2.4.2 hotshot et Django 125
 - 2.4.3 RunSnakeRun 126
 - 2.5 Couverture de code (code coverage) 128
 - 2.5.1 django-coverage 128
 - 2.5.2 django-nose 129
 - 2.5.3 coverage 130

Chapitre 4
Structure d'un projet et des applications

- 1. Structure 131
 - 1.1 Projet 131
 - 1.2 Application 132
- 2. Le fichier settings.py 133

Chapitre 5
L'ORM et les modèles Django

- 1. Principes et généralités 139
- 2. Intérêt de l'ORM 142
- 3. Les modèles 144
 - 3.1 Utiliser les modèles 145
 - 3.2 Les différents types de champs 146
 - 3.3 Arguments à la création des champs 147

3.4	Les relations entre les tables et les modèles	149
3.4.1	Relations de type « plusieurs vers un » (n-1 / many-to-one)	149
3.4.2	Relations de type « plusieurs-vers-plusieurs » (n-m/many-to-many/m2m)	150
3.4.3	Relations de type « un vers un » (1-1/one-to-one)	152
3.4.4	Relations entre modèles d'applications différentes	153
3.5	Comment nommer les attributs, noms de champs du modèle ?	154
3.6	Les options « Meta »	154
3.7	Les autres attributs d'un modèle	155
3.7.1	Définition de méthodes associées à la classe Model	155
3.7.2	Commandes SQL spécifiques	156
3.7.3	Surcharge de méthodes	156
4.	L'héritage entre modèles	157
4.1	Les classes abstraites	158
4.2	L'héritage utilisant plusieurs tables (Multi-table inheritance)	161
4.3	Héritage à l'aide de modèles de procuration/ « proxies » (Proxy models)	162
4.4	Héritage multiple	164
5.	Liste complète des champs et leurs paramètres	165
5.1	Les champs	165
5.2	Options communes à tous les champs	182
6.	Écrire vos propres types de champs	188
7.	Les options globale, la classe Meta	204
8.	Les managers	210
8.1	Comment et pourquoi écrire vos propres managers ?	210
8.2	Ajout de méthodes additionnelles	211
8.3	Modification des données initiales renvoyées par un manager	212
9.	Les QuerySet : effectuer des requêtes	216
9.1	Créer des objets	217
9.2	Modifier une instance	218
9.3	Accéder à une instance/un objet dans la base	220
9.3.1	Accéder à l'ensemble des objets	220
9.3.2	Cas particulier : accéder à un objet spécifique	220
9.3.3	Accéder à des objets spécifiques en effectuant des recherches ou des tris à l'aide de filtres	221

9.4	Filtrer les QuerySet, principe de la syntaxe (Field lookups)	224
9.4.1	Les expressions « F »	228
9.4.2	La gestion du cache des QuerySet	229
9.5	Requêtes complexes utilisant les « objets Q »	230
9.6	Détail de l'API des lookup	231
9.7	Attributs des QuerySet	234
9.8	Les méthodes des QuerySet retournant des QuerySet	235
9.9	Les méthodes des QuerySet qui ne retournent pas de QuerySet	242
9.10	La destruction des objets	245
9.11	Copier un objet	245
9.12	Mettre à jour plusieurs objets à la fois avec update	246
9.13	L'agrégation	247
9.14	Requêtes SQL via l'ORM (raw SQL queries)	251
9.14.1	Exécuter SQL en passant par un modèle	251
9.14.2	Ajouter des attributs additionnels (annotations)	252
9.14.3	Paramétrer la chaîne de caractères de la requête SQL	253
9.14.4	Exécuter SQL directement sans passer par un modèle	253
9.14.5	Utilisation avec plusieurs bases de données	254
9.14.6	Améliorer la présentation des résultats	255
9.15	Gestion des transactions	255
10.	ORM divers	259
10.1	Gérer plusieurs bases de données	259
10.1.1	Première étape : définir plusieurs bases de données	259
10.1.2	Deuxième étape : routage des bases de données	260
10.2	Support de bases de données non créées par Django	266
10.3	Les données initiales, les « fixtures »	268
10.3.1	Utilisation des fixtures	268
10.3.2	Charger les fixtures automatiquement	269
10.3.3	Ajouter des répertoires de fixtures	269
10.3.4	Utilisation de SQL	269
10.4	Optimiser les accès à la base de données	270
11.	Faire évoluer les modèles au cours du développement	274
11.1	Django migrate	275
11.1.1	Principe d'utilisation	275
11.1.2	Dépendances entre les applications	277
11.1.3	Les fichiers du système de migration	277
11.1.4	Traitements sur les données	277

11.1.5 Réduire le nombre de migrations en les regroupant	279
11.1.6 Mise à niveau à partir de South	280
11.2 Autres systèmes de gestion des migrations :	
South, Django-evolution	280
12. Django et SQLAlchemy	282
12.1 Intégration manuelle	283
12.2 SQLAlchemy et Django avec Django-sabridge	285
12.3 SQLAlchemy et Django avec Aldjemy	286

Chapitre 6

Les requêtes HTTP, les URL et les vues

1. Introduction	287
2. La configuration des URL (URL dispatcher)	288
2.1 Introduction	288
2.2 Rappels sur les expressions régulières	289
2.3 Un exemple de configuration d'URL	290
2.4 Gestion des pages d'erreur	292
2.5 Comment importer les vues ?	293
2.6 Le préfixe de vue dans la fonction patterns	294
2.7 Inclusion hiérarchique de configuration d'URL	295
2.8 Capture des expressions régulières lors de l'utilisation de la directive include	295
2.9 Ajout de paramètres nommés à l'appel des vues dans la configuration des URL	296
2.10 Résolution inverse des URL	297
2.11 Nommer les URL	299
2.12 Les espaces de noms des URL : « URL namespaces »	300
2.13 Inversion des URL avec des espaces de noms	300
2.13.1 Comment positionner current_app ?	303
2.13.2 Syntaxe pour utiliser include avec les espaces de noms	303
3. Les vues	304
3.1 Les vues basées sur les fonctions	304
3.1.1 L'objet HttpRequest	305
3.1.2 Les objets QueryDict	309
3.1.3 Les objets HttpResponse	313

3.2	Retourner des erreurs depuis les vues	316
3.3	Les décorateurs spécifiques aux vues	319
3.3.1	Gérer les types de requêtes HTTP	319
3.3.2	Traitement conditionnel des vues	319
3.3.3	Gestion des droits et autorisations	320
3.4	Les fonctions « raccourcis » de Django (Django shortcut functions)	322
3.4.1	render	322
3.4.2	render_to_response	324
3.4.3	redirect	325
3.4.4	get_object_or_404	326
3.4.5	get_list_or_404	327
3.5	Les vues basées sur les classes (class-based view)	327
3.5.1	Introduction	328
3.5.2	Que sont les vues basées sur des classes ?	329
3.5.3	L'utilisation de mixins	331
3.5.4	Traitement des formulaires dans les vues basées sur des classes	332
3.5.5	Les décorateurs et les vues basées sur des classes	334
3.5.6	Les classes génériques de base livrées avec Django	335
3.5.7	Conclusion	349

Chapitre 7

Les templates

1.	Principes et généralités	351
1.1	Intérêt des templates	352
1.1.1	La séparation des préoccupations ou séparation des problèmes	353
1.1.2	La flexibilité : facilité à modifier la présentation, l'aspect ou le design du site	353
1.1.3	La facilité de « localisation » (adaptation à une langue) de l'interface du site	354
1.1.4	Webdesign et codage en parallèle	354
1.1.5	La normalisation et la standardisation des contenus et des affichages	354
1.1.6	Réutilisation et généricité	354
1.2	Django par rapport aux autres systèmes	355

2.	Le principe et la syntaxe du langage de templates de Django	356
2.1	Les variables	357
2.2	Les filtres (filters)	359
2.3	Les tags	361
2.4	Les commentaires	363
2.5	Appeler des méthodes depuis les templates	363
3.	L'héritage dans les templates	364
4.	Échappement automatique du HTML	368
4.1	Désactivation de l'échappement automatique	369
4.2	Comportement conjoint du tag autoescape et du filtre escape	371
4.3	Les paramètres de type chaîne de caractères et l'échappement automatique	372
5.	Les bibliothèques de tags et de filtres spécifiques	372
6.	Les tags et les templates de base	373
6.1	Les tags de base	373
6.2	Les filtres de Django	394
6.3	Filtres et tags pour la localisation et l'internationalisation	415
6.3.1	Bibliothèque i18n.	415
6.3.2	Bibliothèque l10n.	415
6.3.3	Bibliothèque tz	415
6.4	Autres bibliothèques de tags et de filtres.	415
6.4.1	django.contrib.humanize.	416
6.4.2	django.contrib.webdesign	416
6.4.3	static	416
6.5	Les "context processors" de Django	417
6.6	Créer des tags et des filtres spécifiques	422

Chapitre 8

Le middleware Django

1.	Présentation	439
2.	Installer et activer un middleware	440
3.	Principes de fonctionnement	440
3.1	Ordre précis d'exécution des méthodes des middlewares	441
3.2	Comment Django va-t-il procéder ?	442

- 3.3 Détail des méthodes 444
- 3.4 Cas particulier du streaming 445
- 4. Liste et description des middlewares fournis avec Django 447
- 5. Ordre d'installation des middlewares Django 450

Chapitre 9
L'administration de Django

- 1. Introduction 451
- 2. Activer et démarrer le site d'administration 452
- 3. La configuration de votre site d'administration 456
 - 3.1 Les objets ModelAdmin/la classe ModelAdmin 457
 - 3.1.1 Options concernant la page « liste des objets d'un modèle » . . 457
 - 3.1.2 Options concernant la page « création / modification » 465
 - 3.1.3 Options concernant l'écran de création et modification 467
 - 3.1.4 Options permettant de modifier les templates
utilisées par l'admin. 472
 - 3.1.5 Les méthodes de la classe ModelAdmin 473
 - 3.1.6 Ajouter du CSS et du JavaScript via ModelAdmin « asset » . . 478
 - 3.2 Ajouter des codes de validation spécifiques 479
 - 3.3 Utiliser les modèles « inline » 479
 - 3.3.1 Les options (attributs, méthodes) ajoutées 480
 - 3.3.2 Méthodes concernant les inlines dans ModelAdmin 482
 - 3.3.3 Traitement des modèles à deux clefs étrangères 482
 - 3.3.4 Traitement des modèles avec les relations
n-m (many-to-many) 483
 - 3.4 Personnaliser 485
 - 3.4.1 Modifier les templates 485
 - 3.4.2 Gérer plusieurs sites d'administration 486
- 4. Les actions d'administration 488
 - 4.1 Écrire vos propres actions 488
 - 4.2 Intégrer les actions comme des méthodes de ModelAdmin 491
 - 4.3 Les actions qui ont besoin d'une page intermédiaire 491

- 5. Le générateur automatique de documentation 493
 - 5.1 Installation 493
 - 5.2 Balises spécifiques 494
 - 5.3 Structure type de la documentation d'un modèle 494
 - 5.4 Structure de la documentation d'une vue 494
 - 5.5 Structure de la documentation des templates. 496
- 6. Personnaliser les champs et les widgets de l'admin. 496

Chapitre 10

Les formulaires Django

- 1. Introduction 499
- 2. Gestion manuelle 499
- 3. Utilisation des formulaires Django. 501
 - 3.1 La classe Form de Django 503
 - 3.2 Notions complémentaires 504
 - 3.3 Contrôler la façon dont le formulaire est rendu dans un template . . 505
 - 3.4 Contrôler finement la façon dont le formulaire est rendu dans un template 506
- 4. Les types de champs d'un formulaire 510
 - 4.1 Les arguments et les attributs de Field 510
 - 4.2 Les sous-classes prédéfinies dérivées de Field 513
 - 4.2.1 Les sous-classes représentant des nombres. 513
 - 4.2.2 Les sous-classes représentant des chaînes de caractères 514
 - 4.2.3 Les sous-classes représentant des choix 515
 - 4.2.4 Les sous-classes représentant des dates et/ou des heures 516
 - 4.2.5 Les sous-classes diverses. 518
 - 4.2.6 Les sous-classes gérant les relations 519
 - 4.3 Créer vos propres types de champs 520
- 5. Créer un formulaire à partir d'un modèle 520
 - 5.1 Équivalence entre les types de champs de formulaires et ceux du modèle 521
 - 5.2 La classe ModelForm 524
 - 5.2.1 Le contrôle et la validation dans une instance de ModelForm 524
 - 5.2.2 Enregistrer les données : la méthode save() de ModelForm . . 524

- 5.3 Choisir les champs du modèle utilisés dans le formulaire 526
- 5.4 Personnaliser les formulaires 527
 - 5.4.1 Spécifier des paramètres dans la métaclasse incluse 527
 - 5.4.2 Redéfinir finement chaque champ 528
 - 5.4.3 La localisation/internationalisation
des champs du formulaire 530
- 6. CSS et JavaScript dans des formulaires : la classe Media 531
 - 6.1 Définir des ressources associées à un Widget 532
 - 6.1.1 Comment accéder aux ressources : le « path » 534
 - 6.1.2 Utiliser les ressources dans les templates 534
 - 6.1.3 Les ressources de l'Admin Django 534
 - 6.2 Définir des ressources associées à un formulaire 537
- 7. Les ensembles de formulaires (formsets) 538
 - 7.1 Initialisation du formset 539
 - 7.1.1 Gérer le nombre maximal de formulaires affichés à l'écran . . 540
 - 7.1.2 Gérer la validation des formsets 540
 - 7.1.3 Contrôler et valider le nombre de formulaires
dans un formset 541
 - 7.2 Le gestionnaire de formulaires ManagementForm 542
 - 7.2.1 Ordonner les formulaires et gérer la destruction 542
 - 7.2.2 Utiliser les formsets dans les templates 544
 - 7.2.3 Les formsets basés sur un modèle 545
 - 7.2.4 Spécifier le QuerySet avec lequel travaille le formset 545
 - 7.2.5 Utiliser un formulaire spécifique avec un formset
basé sur un modèle 546
 - 7.2.6 La méthode save() 546
 - 7.2.7 Utiliser un formset basé sur un modèle dans une vue 547
 - 7.2.8 Utiliser un formset basé sur un modèle dans un template . . . 548
 - 7.2.9 Les formsets de type « inline » 549
- 8. Les vues génériques basées sur des classes et les formulaires 550
 - 8.1 Utilisation des vues génériques avec un formulaire simple (Form) . . 551
 - 8.2 Utilisation des vues génériques avec un formulaire
utilisant un modèle (ModelForm) 552
- 9. La gestion des fichiers 555

10. Formulaires de type software wizard (assistant logiciel)	556
10.1 Principe de fonctionnement	557
10.2 Utilisation en cinq étapes	557
10.3 Gérer des étapes conditionnelles	561
10.4 Comment donner des noms aux étapes ?	562
10.5 Utiliser un template différent pour chaque formulaire	563
10.6 Initialiser le contenu des formulaires	563
10.7 Gérer les fichiers	564
10.8 Quelques méthodes de la classe WizardView	565
11. La localisation des formulaires	568
11.1 La traduction	568
11.2 La localisation des formats	569

Chapitre 11

La traduction et la localisation

1. Principes et généralités	571
2. Traduction	572
2.1 Principe général	572
2.1.1 Les chaînes formatées	574
2.1.2 Gestion du pluriel	575
2.1.3 Gestion des homonymes, marqueurs contextuels	576
2.1.4 Donner des informations aux traducteurs	577
2.2 Traduire dans les templates	580
2.2.1 Traduire des chaînes en paramètre dans des tags ou des filtres	582
2.2.2 Donner des informations aux traducteurs	582
2.2.3 Sélectionner la langue dans le template	582
2.2.4 Liste des autres tags liés à la traduction (i18n)	583
2.2.5 Les filtres liés à la traduction (i18n)	584
2.3 Les traductions en JavaScript	584
2.3.1 Utilisation dans le code JavaScript	585
2.3.2 Gestion du pluriel	585
2.3.3 Performance	586
2.4 Traduction des schémas d'URL	587

2.5	Créer les fichiers de langues	588
2.5.1	Quelques options de makemessages	589
2.5.2	Format des fichiers .po	590
2.5.3	Compiler les messages	590
2.6	Gettext	591
2.7	La vue Django de choix de langage	591
2.8	Affecter le langage dans le programme	592
2.9	Configurer votre projet pour qu'il soit multilingue	593
2.10	Les fichiers de traduction utilisés par Django	594
3.	La localisation	595
3.1	Activer et désactiver la localisation dans les templates	595
3.1.1	Le tag	595
3.1.2	Les filtres	596
3.1.3	La localisation dans les formulaires	596
3.2	Créer des formats spécifiques	596
4.	Les timezones	598
4.1	Le principe	599
4.2	Affecter le fuseau horaire à une session	600
4.3	L'utilisation des timezones dans les formulaires	602
4.4	L'utilisation des timezones dans les templates	602
4.4.1	Les tags	602
4.4.2	Les filtres	603
4.5	Quelques points particuliers	604
4.5.1	MySQL	604
4.5.2	Conversion d'un objet datetime ordinaire connu	604

Chapitre 12

Approfondissements

1.	Générer d'autres formats que du HTML	605
1.1	Générer un fichier PDF	605
1.2	Générer un fichier CSV	609
2.	Personnaliser le modèle « USER MODEL » de Django	609
2.1	Étendre le modèle User	610
2.2	Remplacer le modèle User de Django	612
2.3	Écrire un modèle User personnalisé	613

3.	Les signaux	616
3.1	Recevoir un signal	617
3.2	Recevoir un signal émis par un émetteur spécifique.	618
3.3	Ne pas recevoir le même signal plusieurs fois	618
3.4	Définir des signaux	619
3.5	Envoyer des signaux	619
3.6	Ne plus recevoir un signal.	620
4.	La sécurité.	620
4.1	Protection contre le clickjacking	620
4.2	Protection contre les attaques XSS	622
4.3	Protection contre les attaques SRF	622
4.4	Les signatures cryptographiques	627
5.	La création de commandes en mode console	630
6.	Les exceptions	633
6.1	Exceptions de base du noyau Django	633
6.2	Exceptions relatives à la résolution des URL.	634
6.3	Exceptions relatives aux bases de données	635
6.4	Exception relative aux transactions.	635
6.5	Exception HTTP	635

Chapitre 13

Les applications Django

1.	Introduction	637
2.	Les applications Django incluses dans le package contrib.	637
2.1	L'administration de Django : « the Django admin site »	638
2.2	django.contrib.auth	638
2.3	Le framework contenttypes	642
2.4	Le middleware CSRF	651
2.5	L'application flatpages	651
2.5.1	Configuration utilisant les URL	652
2.5.2	Configuration utilisant le middleware	653
2.5.3	Gérer les flatpages	653
2.5.4	Les templates des flatpages	653
2.5.5	Le tag get_flatpages	654

2.5.6	Limiter la visibilité d'une flatpage aux utilisateurs enregistrés	654
2.5.7	Filtrer la liste des pages	654
2.6	Le package <code>django.contrib.formtools</code>	655
2.7	L'application <code>GeoDjango</code>	655
2.8	Le jeu de filtres <code>django.contrib.humanize</code>	655
2.9	Le framework messages	657
2.9.1	Utiliser les messages dans les vues	658
2.9.2	Utiliser les messages dans les templates	658
2.9.3	Gérer le niveau minimal des messages affichés	659
2.9.4	Ajouter des étiquettes (tags) aux messages	659
2.9.5	Utiliser les messages avec les vues basées sur des classes (Class-Based Views)	659
2.9.6	Expiration des messages	660
2.10	L'application <code>redirects</code>	660
2.11	Le framework <code>sitemaps</code>	661
2.11.1	La classe <code>Sitemap</code>	662
2.11.2	Gérer les autres vues, les vues isolées	663
2.11.3	Créer un index de plusieurs fichiers <code>sitemap</code>	664
2.11.4	Utilisation de templates avec les <code>sitemaps</code>	665
2.12	Le framework <code>sites</code>	666
2.13	Le framework <code>syndication feed</code>	666
2.13.1	Gérer plusieurs flux RSS avec la même classe	668
2.13.2	Gestion des langues	671
2.14	Outils pour les webdesigners : <code>django.contrib.webdesign</code>	671
2.15	L'application <code>staticfiles</code>	672
2.15.1	Configuration rapide	672
2.15.2	Variables de configuration	673
2.15.3	Commandes consoles	673
2.15.4	Les classes de stockage (storages)	674
2.15.5	Les tags de template de l'application <code>staticfiles</code>	675
2.15.6	Servir les fichiers lors du développement avec la vue « <code>Static file view</code> »	676
2.15.7	Servir les fichiers des utilisateurs lors du développement	676
2.15.8	Déploiement	677

3.	La gestion des caches	677
3.1	Le cache côté client ou cache du navigateur	677
3.2	Le décorateur condition	679
3.3	Raccourcis	680
3.4	Utilisation de condition avec des méthodes autres que GET	681
3.5	Le cache côté serveur.	681
3.6	Activer les mécanismes du cache	682
3.7	Les paramètres de configuration des caches	686
3.8	Cacher l'intégralité du site	687
3.9	Cacher vue par vue	687
3.10	Gestion des caches en aval	689
3.11	Autres headers de contrôle du cache	690
4.	Les applications et projets Django de la communauté	692
4.1	Où trouver des applications Django ?	692
4.2	Les tags et l'application django-taggit	693
4.2.1	La classe TaggableManager	694
4.2.2	Requêtes et API	695
5.	Le déploiement et l'hébergement	696
5.1	Quel Python utiliser, pour quelle compatibilité ?	696
5.1.1	Jython	696
5.1.2	Python 3	697
5.1.3	Les autres Python, Cython, PyPy, etc.	699
5.2	Préparer le déploiement en production	700
5.2.1	Où installer le code de Django ?	700
5.2.2	Éteindre le débogueur	701
5.2.3	Écrire et tester les templates des pages d'erreurs	701
5.2.4	Mettre en place un suivi et une gestion des erreurs	702
5.2.5	Optimiser les réglages pour la production	703
5.2.6	Croissance et escalade de la solution d'hébergement	705
5.2.7	Checklist de lancement en production	709
6.	Hébergement : le déploiement en production	712
6.1	Rappel sur les méthodes d'hébergement	712
6.2	Déploiement avec Apache et le module mod_python	714
6.2.1	Configuration de base	714
6.2.2	Faire tourner plusieurs sites Django avec Apache	715
6.2.3	Remarques concernant les tests et le débogage	715

- 6.3 Déployer Django sur un serveur dédié 716
- 6.4 Configuration Django sur un serveur avec WSGI 717
 - 6.4.1 Configuration des réglages de WSGI (monprojet/wsgi.py) .. 718
 - 6.4.2 Utilisation d'un middleware (intermédiaire) WSGI 718
- 6.5 Déployer Django avec WSGI sur un serveur Apache
 - avec mod_wsgi 719
 - 6.5.1 Configuration de départ 719
 - 6.5.2 Les fichiers statiques et media 720
- 6.6 Utiliser Django avec FastCGI, SCGI ou AJP 721
- 6.7 Google App Engine 724
 - 6.7.1 La base de données GAE 724
 - 6.7.2 Utiliser la base de données Cloud SQL..... 725
 - 6.7.3 Configurer la base pour la production 726
 - 6.7.4 Les commandes en mode console 727
 - 6.7.5 Adapter votre projet Django à GAE 728
 - 6.7.6 Utiliser Django avec la base de données
de Google App Engine 730
- 6.8 Hébergement ne proposant qu'une interface CGI basique 732

- Index..... 737

Chapitre 7

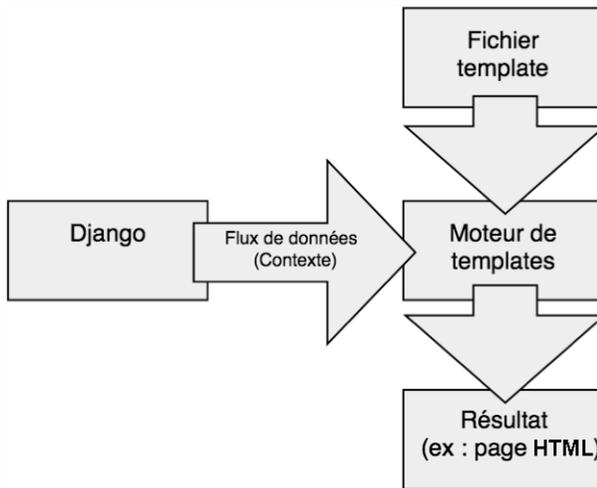
Les templates

1. Principes et généralités

Nous utilisons le mot anglais de *template*, car il est communément utilisé, répandu, et compris dans ce contexte par les professionnels. De plus, en cas de recherche sur le Web, il est plus facile de trouver la bonne information avec *template* qu'avec une traduction française. Pour mémoire, *template* peut être traduit ainsi : modèle/patron/gabarit.

Le principe général de fonctionnement d'un moteur de templates est le suivant : un fichier décrivant la structure de ce que l'on veut produire ou afficher va servir d'ossature ou de modèle. Le moteur (ou processeur) de templates va venir « habiller » ce squelette en fonction des données produites par le programme, le résultat sera une page dynamiquement générée. Concrètement, pour Django, c'est un fichier texte qui est rempli par le moteur de templates avec des variables générées par le framework afin de produire un document au format requis, le plus souvent HTML.

Schémas de principe :



Le principe mis en œuvre dans Django pour gérer des templates est donc, pour ceux qui connaissent déjà le principe, similaire à celui que l'on peut rencontrer dans d'autres langages/systèmes de templates, tels que Jinja2, HAML, OpenDoc, Mako, boiler plate, etc. Il y a cependant des spécificités propres aux templates Django, en particulier une excellente intégration avec l'ensemble du framework.

1.1 Intérêt des templates

Les principaux avantages apportés par ce type d'outil sont :

- La séparation des préoccupations (respect du principe SoC), en particulier pour assurer la séparation entre le webdesign et la programmation.
- Une grande flexibilité d'évolution de l'aspect du site, de sa présentation.
- La facilité de « localisation » (adaptation à une langue) de l'interface du site.
- La possibilité de faire travailler séparément et en même temps des personnes différentes sur la conception graphique et sur le code.
- La normalisation et la standardisation des contenus et des affichages.
- La réutilisation et la généricité du code produit.

1.1.1 La séparation des préoccupations ou séparation des problèmes

Un principe important de la programmation moderne est le SoC (*Separation of Concerns*), ou la séparation des préoccupations. Ici le but est de séparer les aspects de design web, bien mieux maîtrisés par une partie de l'équipe (webdesigners, programmeurs JavaScript, etc.), des aspects de pure programmation du framework, mieux maîtrisés par les programmeurs. En bref, de séparer le développement de la mise en page. Un objectif chez tous les développeurs web est de développer des applications souples et faciles à maintenir. Pour cela il est fondamental de séparer la logique métier de la logique de présentation. Les systèmes de templates web permettent de maintenir cette séparation.

De ce point de vue, les templates Django répondent au besoin, car contrairement à ce que l'on peut voir dans d'autres systèmes, les templates Django ne sont absolument pas du code Python inséré dans du HTML, mais bien un langage permettant aux équipes de prendre totalement en charge cette partie du travail et de se consacrer au design.

- Pour les concepteurs du site : lorsque chaque page web est générée par un moteur de templates, elle peut être conçue de manière modulaire et structurée avec des composants pouvant être modifiés indépendamment les uns des autres. Ces composants peuvent être par exemple : un en-tête, un pied de page, une barre de navigation globale, une barre de navigation locale, un fil d'Ariane (breadcrumb) et du contenu tel que des articles, images, vidéos, etc.
- Pour les programmeurs : le langage de template offre des capacités logiques plus restreintes qu'un langage de développement classique, car ces capacités doivent servir uniquement à réaliser des adaptations de présentation et à prendre des décisions simples. Les templates ne sont absolument pas faits pour mettre en œuvre des algorithmes complexes.

De cette séparation entre la présentation et le code découlent également la plupart des autres avantages ci-dessous.

1.1.2 La flexibilité : facilité à modifier la présentation, l'aspect ou le design du site

L'une des raisons derrière la séparation est la nécessité d'une évolutivité maximale dans le code et les ressources consacrées à la partie de présentation. En effet, sous la pression des nouvelles demandes des utilisateurs, de l'évolution des préférences ou des goûts des utilisateurs, de la mode, et du désir de présenter de manière nouvelle des contenus préexistants, il est parfois nécessaire de modifier radicalement l'aspect public du site web tout en perturbant le moins possible l'infrastructure technique sous-jacente.

Un exemple d'actualité assez pertinent étant la nécessité d'adapter les contenus existants des sites Internet au surf sur mobile ou tablette. Grâce aux templates, cette adaptation, bien que conséquente, est assez simple à réaliser sans toucher au code de l'application.

Dans les templates Django, les modifications ne dépendent pas du contenu. Cela signifie que les concepteurs de sites web peuvent mettre à jour la présentation sans avoir à s'occuper des considérations techniques.

1.1.3 La facilité de « localisation » (adaptation à une langue) de l'interface du site

Les menus et d'autres éléments de l'affichage sont facilement uniformisés pour les utilisateurs du site dans une langue donnée, en créant les templates appropriés.

1.1.4 Webdesign et codage en parallèle

Comme le code et l'affichage sont bien séparés, il est aisé de comprendre qu'au-delà de la coordination nécessaire, deux équipes ou deux personnes différentes pourront travailler en même temps sur le codage et la présentation du site.

1.1.5 La normalisation et la standardisation des contenus et des affichages

Comme souvent, certains affichages sont similaires, par exemple : l'affichage d'une liste de produits, d'un tableau de statistiques, etc. Nous allons pouvoir utiliser la même template pour tous les éléments similaires, et ainsi rendre l'aspect du site plus cohérent et plus uniforme. L'utilisateur va comprendre la logique qui est à l'œuvre et mieux se repérer dans la navigation. De plus, certains langages de templates, dont Django, implémentent une fonctionnalité intéressante, qui est l'héritage entre templates. Ainsi, comme on va le voir dans ce chapitre au paragraphe consacré à l'héritage, on pourra définir des modèles structurés d'affichage, par exemple pour le site : par catégorie, par sous-catégorie et ainsi de suite. Ainsi on peut assurer une grande cohérence graphique de l'aspect du site tout en minimisant le travail nécessaire pour réaliser une telle opération.

1.1.6 Réutilisation et généricité

Une agence web va pouvoir capitaliser certaines parties répétitives de conception de page web. Les utilisateurs non spécialistes, qui n'ont pas la possibilité ou la capacité d'embaucher des développeurs pour concevoir un système cousu main, pourront utiliser tes templates disponibles sur Internet et les adapter, par exemple « Twitter bootstrap ».

1.2 Django par rapport aux autres systèmes

Si l'on souhaite comparer en matière de fonctionnalités les templates de Django aux autres langages de templates, on remarque que le langage de templates fourni avec Django n'a rien à envier aux autres langages (voir http://en.wikipedia.org/wiki/Comparison_of_web_template_engines en anglais). En effet, les templates Django gèrent un langage structuré de programmation, avec des variables, des fonctions, l'inclusion de templates, des instructions de contrôle (boucles, tests...), l'évaluation et l'affectation, la gestion des erreurs et des exceptions ainsi que l'héritage. Enfin, les possibilités des templates Django sont largement extensibles par l'utilisateur.

Un reproche que l'on pourrait faire à Django est que les templates Django, bien que pouvant être directement édités dans un logiciel de création web (par exemple Komposer ou Dreamweaver), ne représentent pas toujours directement ce qui va être affiché au final (absence de notion de templates naturels). Cependant, on peut avec un peu de pratique contourner ce problème.

Un point très positif, les templates Django sont bien adaptés au framework Django dont ils collent bien à la logique et sont proches de la logique Python. Cela permet donc d'avoir à la fois une approche SoC et une absence de couture (seamless) entre les deux parties du développement.

Un autre point très positif est qu'on peut également remarquer que, bien qu'étant doté d'une syntaxe cohérente, comme on va le voir juste après, le langage de templates Django peut être utilisé avec n'importe quel format de fichier texte. En effet, les templates Django ne nécessitent pas de structuration particulière du fichier dans lequel ils sont utilisés. C'est un point fort par rapport aux langages de templates qui s'appuient par exemple sur une syntaxe spécifique, telle qu'un langage de balises comme XML ou HTML (par exemple Zope TAL). Ainsi, le langage de templates Django est utilisable non seulement pour produire du XML ou du HTML, mais également pour produire des fichiers JavaScript ou CSS et bien évidemment du texte brut. En fait les templates Django permettent de produire pratiquement tout type de fichier dont le format est basé sur du texte, par exemple du CSV (*Comma-Separated Values*).

Enfin, pour les développeurs qui seraient réfractaires à ce système ou qui auraient déjà une pratique et une habitude d'un autre langage de templates, Django permet dans une certaine mesure de compléter le système intégré de templates Django par un autre, comme cela sera expliqué plus tard. Les systèmes alternatifs que l'on peut installer sont Jinja2, HAML, OpenDoc et Mako. Cette possibilité permet donc aux développeurs de garder leurs habitudes de travail et de capitaliser sur des bibliothèques de templates existantes, que l'on va pouvoir réutiliser avec quelques adaptations à la marge.

2. Le principe et la syntaxe du langage de templates de Django

Un fichier de template Django est, comme vu précédemment, tout simplement un fichier texte dans lequel sont installées des balises spécifiques, permettant de générer à peu près n'importe quel format basé sur du texte (HTML, XML, CSV, etc.).

Un fichier de template Django contient des commentaires, des **variables** (variables qui seront remplacées à la volée par la valeur de la variable lors de l'évaluation du template) et des **tags** (instructions, mots-clefs ou étiquettes) qui contrôlent la logique du flux d'affichage du fichier template.

Les instructions, variables et commentaires sont délimités par trois balises : `{{ .. }}` pour les variables, `{% ... %}` pour les tags ou instructions et enfin `{# ... #}` pour les commentaires sur une même ligne.

Par la suite nous continuerons à utiliser le mot **tag** pour rester en cohérence avec la documentation Django. Rappelons qu'il s'agit d'une instruction, d'un mot-clef. Les **tags** proposés par le langage de templates Django fonctionnent de manière similaire aux instructions d'un langage de programmation. Ainsi, le tag **if** permet de traiter des expressions booléennes, le tag **for** permet de construire des boucles. Attention, ces différents tags ne sont pas exécutés de manière totalement similaire aux expressions Python correspondantes. De base, seuls les tags et les filtres décrits ci-après sont supportés. L'utilisateur pourra en ajouter, soit en installant des add-ons ou des applications complémentaires, soit en développant ses propres tags et filtres.

Du point de vue technique, les variables à destination du template sont stockées dans un dictionnaire Python appelé `context` (contexte), ce dictionnaire est passé en paramètre des fonctions de rendu des templates par le programmeur.

- Les clefs du dictionnaire correspondent aux noms des variables qui seront accessibles lors de l'exécution du rendu du template.
- La valeur associée à la clef est l'objet Python correspondant. Le contexte global de l'évaluation d'un template est composé du dictionnaire passé en paramètre par l'utilisateur lors de l'appel au moteur de rendu.

Ce contexte utilisateur est complété par des variables globales si et seulement si on fait appel à `RequestContext(request, context)`, ce qui est le cas par défaut pour la fonction `render`. Chaque « template context processor » configuré dans `settings.py` ajoute alors ses propres variables « globales ».

Exemple :

```
mon_contexte = {
    'liste_des_salles' :Salles.objects.all(), 'ma_variable ' :2}
    result = render('mon_template.html', mon_contexte,)
```