Engineer

Django

D framework Python pour créer des applications web

Ce que c’est

Django est un framework web écrit en Python. Lorsqu’une requête arrive, la configuration des URL l’oriente vers une fonction ou une classe appelée vue, qui renvoie une réponse, généralement du HTML produit à partir d’un gabarit ou du JSON. Autour de ce noyau, il fournit les éléments nécessaires à la plupart des sites adossés à une base : mappeur objet-relationnel (ORM), migrations de schéma, formulaires, authentification avec permissions et interface d’administration générée automatiquement, que sa documentation présente comme l’une de ses fonctionnalités les plus populaires. Le projet s’écrit avec un D majuscule ; le paquet à installer et à importer est `django`, en minuscules.

Il vient d’une rédaction. À l’automne 2003, Adrian Holovaty et Simon Willison, développeurs chez World Online, le département web d’un journal de Lawrence, au Kansas, sont passés de PHP à Python pour les sites créés au rythme des échéances de la rédaction. En créant des sites comme Lawrence.com, ils ont extrait de leur travail un framework générique, l’ont affiné pendant deux ans et World Online l’a publié en open source à l’été 2005. Il porte le nom de Django Reinhardt, guitariste de jazz manouche, se prononce JANG-oh avec un D muet et est distribué sous licence BSD à 3 clauses.

Avec la numérotation A.B, les versions apportant de nouvelles fonctionnalités arrivaient environ tous les huit mois et seules les versions désignées pour un support à long terme, LTS, recevaient trois ans de support ; la version 5.2 LTS est prise en charge jusqu’en avril 2028. Django 6.2, version LTS prévue pour avril 2027, est la dernière sous ce régime. À partir de Django 2028, les versions apportant de nouvelles fonctionnalités portent le nom de leur année, sortent chaque janvier et bénéficient toutes de la même période de support de trois ans.

La FAQ du projet propose de parler de framework MTV, pour modèle, template (gabarit) et vue : le modèle définit les données, la vue décide quelles données l'utilisateur voit et le gabarit décide de leur présentation. Selon cette lecture, le contrôleur est le framework lui-même, le mécanisme qui envoie chaque requête à la bonne vue. Les modèles sont des classes Python, et leurs requêtes sont des QuerySets à évaluation différée : on peut construire, filtrer et transmettre un QuerySet sans toucher à la base. Elle n'est interrogée qu'à son évaluation, par une boucle, par exemple. Les modifications de modèles deviennent des migrations. `makemigrations` regroupe chaque modification dans un fichier que la documentation compare à un commit de gestion de versions pour le schéma ; `migrate` l'applique. Les objets liés sont chargés à la demande, une requête à la fois, sauf si le QuerySet les demande à l'avance.

```
# Book has a ForeignKey to Author.
# 1 query for the books, then 1 per book for its author.
for book in Book.objects.all():
    print(book.author.name)

# 1 query: an SQL join loads each author with its book.
for book in Book.objects.select_related("author"):
    print(book.author.name)

# 2 queries: the authors, then all their books in one batch.
for author in Author.objects.prefetch_related("book_set"):
    print(author.name, [b.title for b in author.book_set.all()])
```

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que Django ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. La partie invisible pour les utilisateurs : où résident les données, quelles règles s'appliquent et comment les autres systèmes peuvent y accéder.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c'est important

L'intérêt de Django tient aux protections qu'il fournit d'emblée. Ses QuerySets construisent le SQL en séparant les paramètres du texte de requête ; le pilote de base échappe donc les valeurs fournies par l'utilisateur. Ses gabarits échappent les caractères les plus dangereux en HTML. Sa protection CSRF vérifie un secret à chaque POST, et il conserve les mots de passe avec PBKDF2 sur SHA-256, une

méthode d'étirement de clé dont la documentation précise qu'elle est recommandée par le NIST. Une équipe qui écrit du code Django courant bénéficie de ces protections sans les demander.

Ces mêmes choix initiaux dessinent le risque. Chacun peut être annulé par une ligne : requête brute, `mark_safe`, `csrf_exempt` ou `DEBUG = True` laissé sur le serveur en service. Chacune paraît pourtant anodine en revue. L'ORM fait un arbitrage semblable pour les performances : il est facile d'écrire une vue dont le nombre de requêtes croît avec la longueur de la page, alors que le code Python paraît semblable dans les deux cas.

La recommandation est de choisir Django lorsque les données structurent l'application : des enregistrements créés et modifiés par les utilisateurs, des comptes avec des permissions et du personnel ayant besoin d'un écran d'administration sur l'ensemble. Gardez chaque projet sur une version prise en charge, choisissez une LTS tant que durent les versions A.B et notez la fin de support à côté du numéro de version. Exécutez `manage.py check --deploy` sur les paramètres de production dans le pipeline, gardez clé secrète et mot de passe de base hors du dépôt et lisez le SQL de l'ORM pour chaque page importante. Pour un site que le client modifiera surtout sous forme de pages, un système de gestion de contenu ou des pages statiques conviennent mieux qu'un framework.

À quoi cela ressemble en pratique

Entre 2015 et 2022, le travail de développement web comprenait des applications Django autour d'une plateforme IPTV : les parties d'un service fonctionnant jour et nuit sur lesquelles les gens cliquaient réellement. Nouvelles fonctionnalités, corrections de bugs et amélioration continue des performances et de la stabilité se faisaient à l'échelle de l'équipe. Sur un tel service, dit la page, la stabilité est la contrainte que doivent respecter les fonctionnalités. Les applications continuaient de s'enrichir tout en restant fiables pour les équipes internes et les clients.

Plus tôt, de 2003 à 2014, le travail de développement web full stack a produit cent applications web pour des clients aux besoins très différents. Pour un besoin sur mesure, les outils étaient HTML/HTML5, CSS/SCSS et Django. Pour un besoin plus standard, que le client pouvait aussi gérer lui-même, WordPress ou Joomla était la réponse plus rapide et plus raisonnable. La page présente la

compétence comme la capacité de distinguer ces cas, y compris de dissuader un client d'une construction sur mesure inutile ou de l'y encourager si elle était nécessaire. C'est la même distinction que fait la recommandation.

La page consacrée à la pile technologique conserve ces deux aspects. Sa couche web privilégie le statique, avec Django et PHP lorsqu'une application doit être servie. Sa liste de langages traite SQL comme un langage à part entière, plutôt que comme ce qu'un ORM génère, car c'est là que se trouve l'essentiel du travail de performance pour lequel l'entreprise est sollicitée. Pour une application Django, l'ORM écrit donc les requêtes, mais quelqu'un les lit toujours.

Où les problèmes surviennent

- **Une boucle exécute une requête par ligne.** Dans le mode de récupération par défaut, lire une clé étrangère que le QuerySet n'a pas chargée exécute une requête pour cet objet. Cent livres avec leurs auteurs coûtent donc 101 requêtes. Les gabarits rendent cela plus difficile à voir : ils appellent des méthodes comme `all` sans parenthèses. Utilisez `select_related()` pour les clés étrangères et les champs un-à-un, et `prefetch_related()` pour les relations plusieurs-à-plusieurs et les clés étrangères inverses. Depuis Django 6.1, le mode `FETCH_PEERS` regroupe ces lectures dans une requête supplémentaire ; `FETCH_RAISE` transforme une requête imprévue en exception pour le code qui doit rester rapide.
- **Le mode débogage arrive en production.** La liste de contrôle de déploiement demande de ne jamais activer `DEBUG` en production : les pages d'erreur divulguent des extraits de code source, des variables locales, des paramètres et les bibliothèques utilisées. Avec `DEBUG = False`, Django ne sert aucune requête tant que `ALLOWED_HOSTS` n'a pas de valeur adaptée ; un joker impose de valider vous-même l'en-tête Host. Exécutez `manage.py check --deploy` sur le fichier de paramètres de production pour automatiser une partie de cette liste.
- **La clé secrète reste dans le dépôt.** `SECRET_KEY` protège les sessions pour la plupart des moteurs de session, tous les jetons de réinitialisation de mot de passe et toute signature cryptographique qui ne désigne pas d'autre clé. La référence des paramètres prévient qu'une clé connue peut mener à une élévation de privilèges et à l'exécution de code à distance. Chargez-la depuis l'environnement ou un fichier, gardez-la hors de la gestion de versions, effectuez sa rotation avec `SECRET_KEY_FALLBACKS` et retirez rapidement les anciennes clés de cette liste.

- **Une protection par défaut est désactivée pour une vue.** Les QuerySets sont paramétrés, mais les requêtes brutes, `extra()` et `RawSQL` laissent l'échappement au développeur. Les gabarits échappent le HTML, mais `mark_safe`, le filtre `safe` et la désactivation de l'échappement automatique lui rendent aussi cette tâche. `csrf_exempt` supprime entièrement le contrôle d'une vue. Chaque cas mérite une raison explicite en revue, et le HTML stocké par les utilisateurs devrait être assaini en entrée comme en sortie.
- **Une migration s'arrête à mi-chemin.** Sous PostgreSQL et SQLite, toutes les opérations d'une migration s'exécutent dans une transaction unique par défaut. MySQL n'entoure pas les modifications de schéma de transactions ; un échec partiel laisse donc des modifications à défaire à la main. Lisez les changements détectés par `makemigrations`, vérifiez le SQL avec `sqlmigrate` avant la production et commitez chaque migration avec la modification de modèle correspondante.

Pour aller plus loin

- Documentation Django — la documentation de la version stable actuelle, du tutoriel à la référence.
- Django : liste de contrôle de déploiement — les paramètres à changer avant la mise en service, dont une partie est automatisée par `check --deploy`.
- Django : optimisation des accès à la base — évaluation des QuerySets, chargement des objets liés et habitudes qui stabilisent le nombre de requêtes.
- Django : migrations — le processus de migration, `sqlmigrate` et le traitement des changements de schéma par chaque moteur.
- Sécurité dans Django — les protections fournies par défaut par le framework et leurs limites.
- Télécharger Django — les séries de versions prises en charge, leurs dates de support principal et étendu.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que Django ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/django/