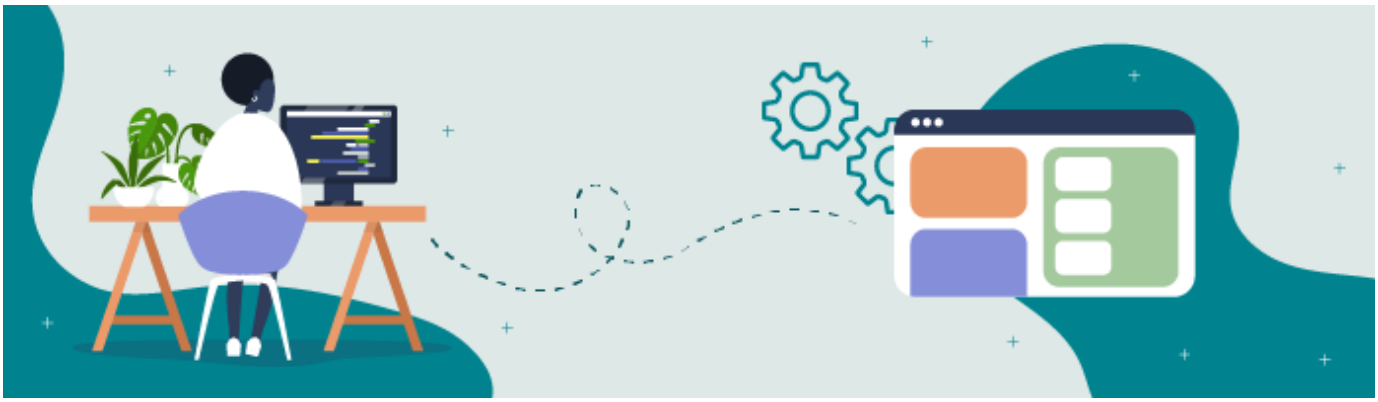
Débutez avec le framework Django

12 heures Moyenne

Mis à jour le 26/10/2022

Configurez un nouveau projet avec l'utilitaire de ligne de commande de Django



Générez du code de base pour un projet Django

Avant de nous plonger dans la construction de l'application Merch Exchange, une poignée de fichiers du projet doit être mise en place.

Il serait fastidieux et source d'erreurs de devoir créer ces fichiers à la main à chaque fois que nous démarrons un nouveau projet Django. Django nous fournit un **utilitaire de ligne de commande** pour automatiser cette tâche (et bien d'autres).

Allez dans votre répertoire `django-web-app` si vous n'y êtes pas déjà, puis tapez la commande suivante dans le terminal :

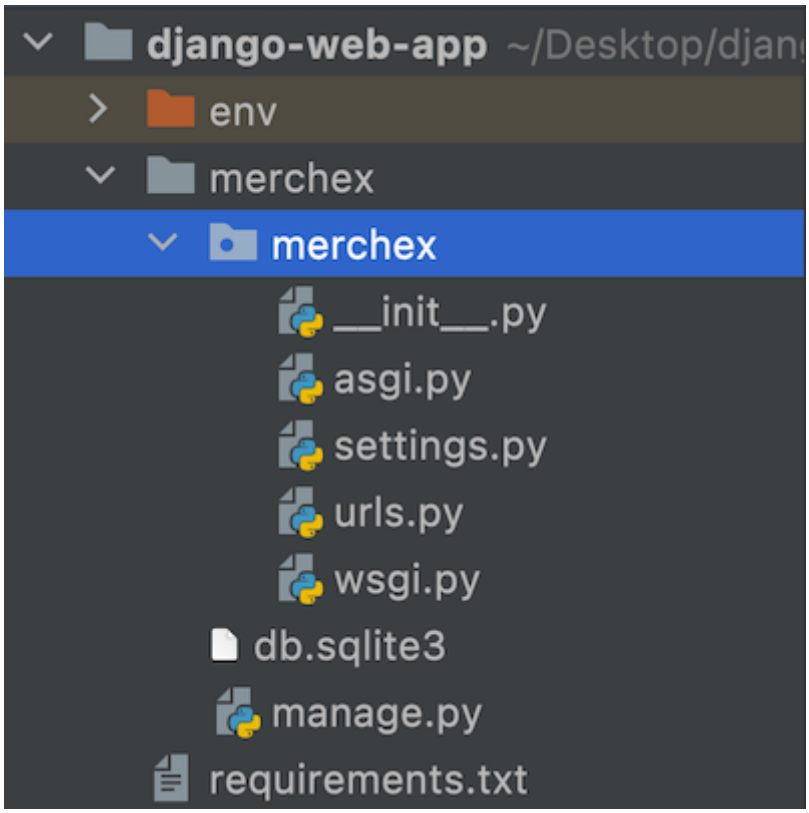
text

```
(env) ~/projects/django-web-app
→ django-admin startproject merchex
```

On dirait que rien ne se passe... mais regardez maintenant le contenu du répertoire :

```text
(env) ~/projects/django-web-app
→ ls
merchex     requirements.txt
```

Nous avons un nouveau répertoire : `merchex` . Utilisons notre éditeur de code pour regarder ce qu'il contient :



Le répertoire merchex dans l'éditeur de code.

Nous pouvons voir que de nouveaux fichiers ont été créés.

> Qu'avons-nous fait ?

Nous venons de générer automatiquement notre code de base Django.

Le **code de base** est un code gabarit ou code de démarrage : c'est le code de base dont nous avons besoin pour un projet Django fonctionnel, mais vide.

Nous avons généré notre code de base en lançant la commande `django-admin` , suivie de la sous-commande `startproject` et en passant l'argument `merchex` comme nom à utiliser pour le projet.

C'est ce nom qui est utilisé pour le répertoire de premier niveau. Entrons dans ce répertoire et listons son contenu :

text

```
(env) ~/projects/django-web-app
→ cd merchex
(env) ~/projects/django-web-app/merchex
→ ls
merchex     manage.py
```

À l'intérieur, nous avons un autre répertoire appelé `merchex` , dont nous parlerons plus tard dans ce chapitre. Nous avons également un script Python appelé `manage.py` .

Maintenant, lorsque nous utiliserons l'utilitaire de ligne de commande de Django, nous l'appellerons via `manage.py` au lieu de `django-admin` comme nous le faisions auparavant. En fait, `manage.py` est conçu pour fonctionner spécifiquement avec notre projet, alors que `django-admin` est une version plus générique de l'utilitaire.

Avec le code de base en place, nous avons tout ce dont nous avons besoin pour lancer notre site pour la première fois.

Exécutez le serveur de développement

Appelons à nouveau l'utilitaire de ligne de commande, cette fois via « manage.py », et lançons la sous-commande `runserver` .

> Le fichier `manage.py` est un script Python ; nous allons donc ajouter le préfixe `python` à chaque fois que nous l'exécuterons, par exemple : `python manage.py runserver` .

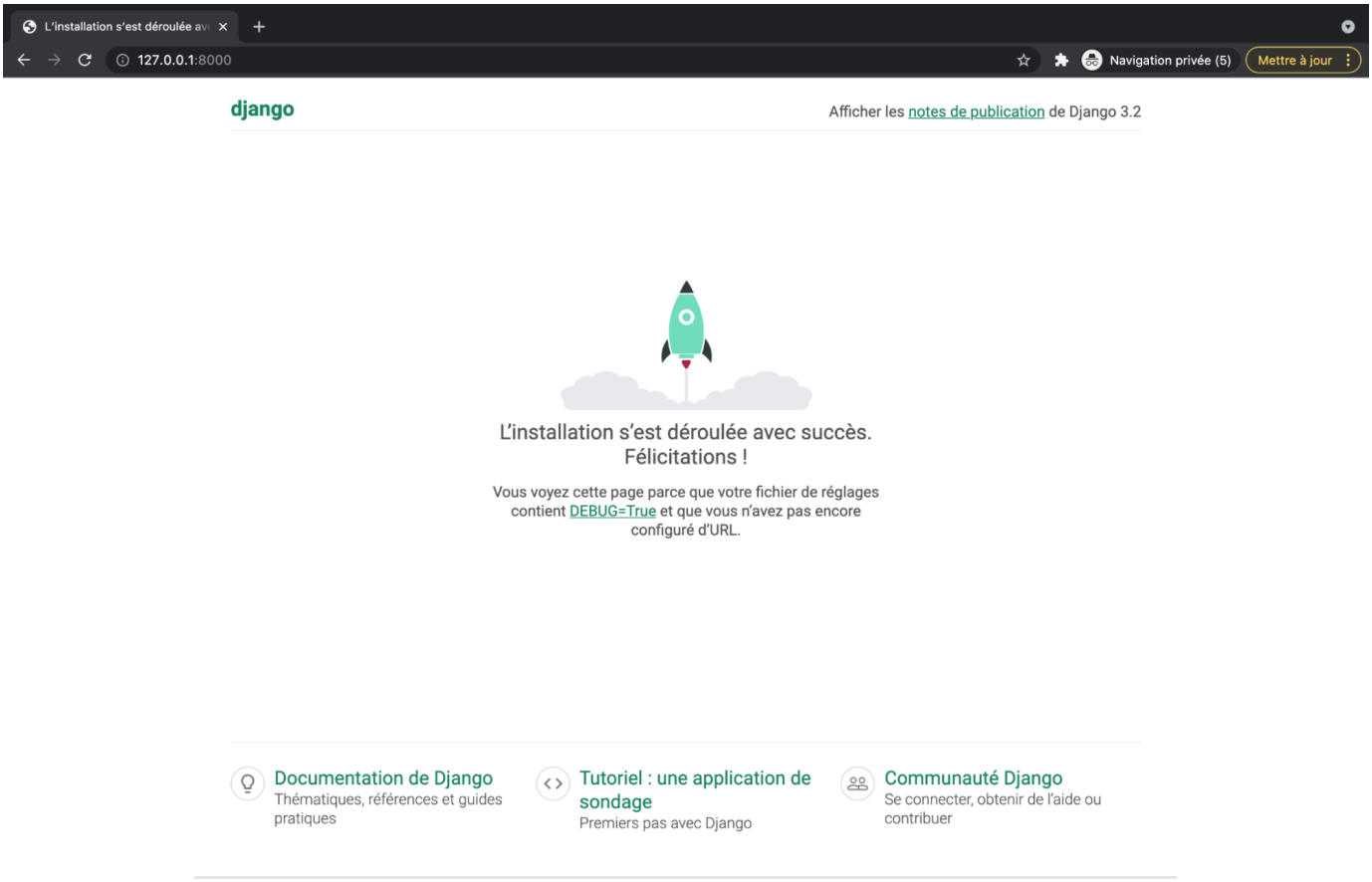
text

```
(env) ~/projects/django-web-app/merchex
→ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the
migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 07, 2021 - 17:58:59
Django version 3.1.6, using settings 'merchex.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

L'utilitaire nous indique que le serveur de développement a démarré à l'adresse http://127.0.0.1:8000/. Jetons un coup d'œil à cette adresse dans le navigateur.



L'adresse http://127.0.0.1:8000/ dans un navigateur.

Nous avons un site Django qui fonctionne !

> Pour consulter notre site, au lieu de visiter un nom de domaine (comme openclassrooms.com), nous utilisons l'adresse **http://127.0.0.1:8000/**. C'est ainsi que nous indiquons au navigateur qu'il doit communiquer avec une application hébergée *sur cet ordinateur* et non sur un serveur situé quelque part sur Internet.

Prenez un moment pour admirer la page par défaut de Django, puis jetez un autre coup d'œil à votre terminal :

text

```
...
[07/Feb/2021 18:12:43] "GET / HTTP/1.1" 200 16351
[07/Feb/2021 18:12:43] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
...
```

Pendant que vous parcourez les pages de votre application web, vous verrez des messages de journal apparaître dans le terminal. Ils peuvent s'avérer utiles pour le débogage, comme nous le verrons tout au long du cours.

Créez la base de données du projet

Revenons à un message qui est apparu lorsque nous avons exécuté la sous-commande `runserver` :

text

```
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
```

> Que sont les migrations ?

Nous apprendrons beaucoup de choses sur les migrations tout au long de ce cours, mais pour l'instant, nous devons simplement comprendre que les **migrations** représentent un moyen de configurer la base de données de notre application.

Utilisons maintenant l'utilitaire de ligne de commande pour créer notre base de données.

Dans le terminal, tapez Ctrl-C pour arrêter le serveur et revenir à une invite :

text

```
^C
(env) ~/projects/django-web-app/merchex
➜
```

Exécutez la sous-commande `migrate` :

text

```
(env) ~/projects/django-web-app/merchex
➜ python manage.py migrate
Operations to perform:
Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
Applying ...
```

Maintenant, listez le contenu du répertoire et vous verrez qu'un fichier de base de données a été créé, appelé `db.sqlite3` :

text

```
(env) ~/projects/django-web-app/merchex
➜ ls
merchex     db.sqlite3  manage.py
```

Il s'agit de la base de données qui contiendra toutes les données de notre application, c'est donc une bonne chose de l'avoir créée dès maintenant.

> Est-ce que vous voyez l'erreur `./manage.py not found` ? Reportez-vous au cours **Débuggez un projet avec Python** pour des conseils de débogage.

> Si vous utilisez Git pour suivre votre code source, je vous recommande d'ajouter `db.sqlite3` à votre fichier .gitignore, afin qu'il ne soit pas suivi dans votre repository. Il s'agit d'une bonne

pratique pour les vrais projets, mais cela peut également vous éviter des tracas dans un projet d'apprentissage comme celui-ci.

Générez le code de base pour une application Django

Ensuite, nous allons créer ce que Django appelle une « application ».

Cela peut sembler étrange : après tout, tout ce cours porte sur la création d'une « application » : une « application web ». Voyons donc ce qu'est une application dans le contexte de Django.

Dans Django, une **application** est une sous-section de votre projet entier. Django nous encourage à compartimenter notre projet entier Django en applications, pour deux raisons principales :

- cela permet de garder notre projet organisé et gérable au fur et à mesure qu'il se développe ;
- cela signifie qu'une application peut éventuellement être réutilisée dans plusieurs projets.

Comme il s'agit de notre tout premier projet Django, il sera suffisamment petit pour que notre code puisse s'intégrer sans peine dans une seule application.

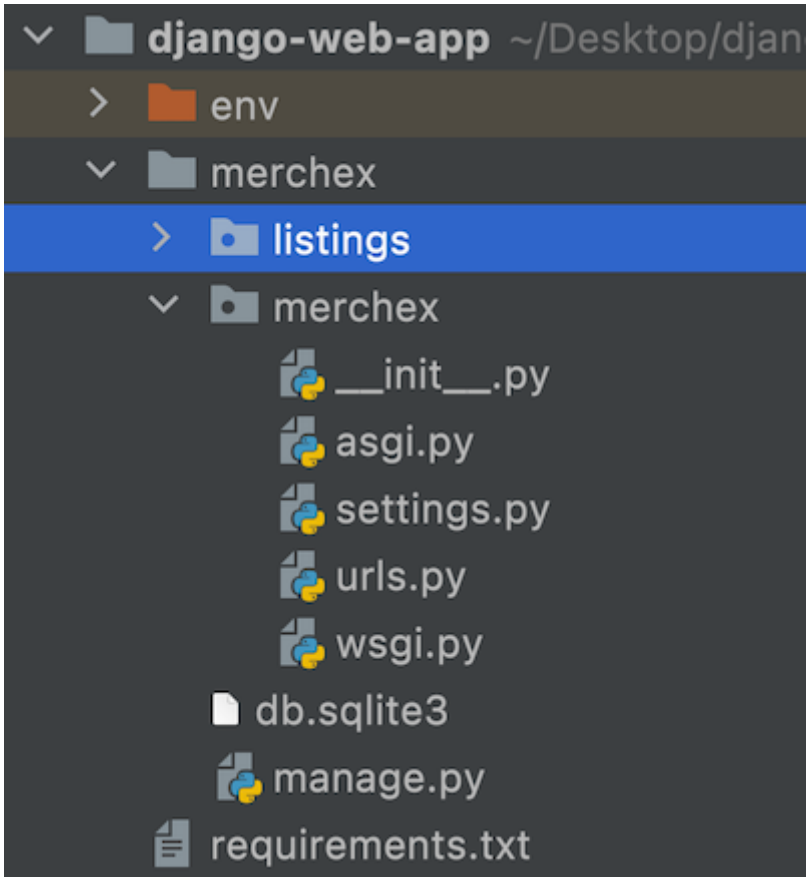
Chaque application doit avoir un nom approprié qui représente le concept dont l'application est responsable. Le premier concept que nous voulons développer dans Merch Exchange est la liste des marchandises. Nous allons donc nommer notre première application « listings ».

Créons maintenant cette application, en utilisant la sous-commande `startapp` dans l'utilitaire de ligne de commande :

text

```
(env) ~/projects/django-web-app/merchex
→ python manage.py startapp listings
```

Comme précédemment avec la sous-commande `startproject`, il semble que rien ne se soit passé, jusqu'à ce que nous regardions la structure de notre projet :



La structure du projet.

Nous avons un nouveau répertoire appelé « listings », contenant plusieurs fichiers de code de base. C'est notre répertoire d'applications, et il contiendra tout le code relatif aux groupes.

> Notez que « listings », votre *répertoire d'applications*, se trouve à côté de « merchex », votre *répertoire de projet*. Remarquez aussi que le répertoire du projet a le même nom (`merchex`) que le répertoire au-dessus de lui par défaut. C'est la convention dans la structure d'un projet Django.

Chaque répertoire d'applications est spécifique à une application. Mais votre répertoire de projet contient des fichiers de configuration pour l'ensemble du projet : c'est un peu la « tour de contrôle » de votre projet Django.

La dernière étape de l'ajout de notre application « listings » à notre projet « merchex » consiste à « installer » l'application dans le projet.

Lorsque nous avons généré le code de base de notre projet, l'un des fichiers créés s'appelait settings.py. Ouvrez maintenant ce fichier et trouvez une liste Python appelée `INSTALLED_APPS` . En *bas* de cette liste, ajoutez la chaîne de caractères `'listings'` :

python

```
# ~/projects/django-web-app/merchex/merchex/settings.py

INSTALLED_APPS = [
```

```
4 'django.contrib.admin',
5 ...
6 'django.contrib.staticfiles',
7
8 'listings',
9 ]
```

> Super, j'ai installé mon application dans mon projet Django ! Mais que sont toutes ces autres applications de la liste ?

Par défaut, le code de base de Django comprend l'installation d'un certain nombre d'applications utiles, que la plupart des projets utiliseront probablement à un moment ou à un autre. Celles-ci incluent `django.contrib.admin` que nous utiliserons plus tard dans le cours.

Ainsi, même si le code que nous écrivons durant ce cours tient dans une seule application, nous serons en mesure de constater la puissance de l'intégration de plusieurs applications réutilisables dans notre projet.

> Contrairement à notre application « listings », ces applications supplémentaires ne se trouvent pas dans notre code source, nous ne voyons pas de répertoire dans notre code appelé `django.contrib.admin` , par exemple. Ces applications sont, en fait, importées depuis le package de Django que nous avons installé au début de ce chapitre avec `pip` .

> La meilleure pratique consiste à ajouter notre application en bas de la liste afin qu'elle soit la dernière à se charger.

Examinez la structure du projet

Nous avons généré beaucoup de fichiers dans ce chapitre, alors prenons le temps de revoir la structure de notre projet :

- django-web-app/ - *le répertoire racine de notre repository*
  - .gitignore
  - requirements.txt - *une liste des packages requis*
  - merchex/ - *le répertoire contenant notre projet Django, l'application, la base de données et l'utilitaire de ligne de commande*
    - manage.py - *le script utilitaire de ligne de commande de Django*
    - db.sqlite3 - *le fichier de la base de données de Django*
    - merchex/ - *le répertoire du projet, généré par « django-admin startproject merchex » : la « tour de contrôle » de notre projet*
      - settings.py - *la configuration de l'ensemble du projet*
      - *...et d'autres fichiers relatifs au projet.*

        - listings/ - *le répertoire de l'application généré par « python manage.py startapp listings »*
            - *...les fichiers spécifiques aux applications que nous explorerons tout au long du cours*

Impressionnant ! Nous avons couvert beaucoup de choses dans ce chapitre ! Nous avons effectué beaucoup de travail dans le terminal ici, et parfois il est utile de regarder le terminal en action : alors regardez ce screencast pour vérifier que vous avez compris.

02:25

En résumé

- Lorsque nous démarrons un nouveau projet Django, nous installons la dernière version de Django avec `pip install django` .
- Nous générons notre code de de base, initialisons notre base de données et démarrons le serveur de développement à l'aide de l'utilitaire de ligne de commande de Django.
- Nous vérifions que tout fonctionne comme il se doit en naviguant vers le front-end du site à l'adresse http://127.0.0.1:8000/.

Maintenant que vous avez créé un projet et une application de base, et que vous avez vérifié que votre environnement local est prêt pour le développement, vous êtes prêt à créer la première page web de votre application.

J'ai terminé ce chapitre et je passe au suivant

- Formations jusqu'à 100 % financées
- Date de début flexible
- Projets professionnalisants
- Mentorat individuel

Trouvez la formation et le financement faits pour vous

Être orienté

Comparer nos types de formations

Installez Django avec pip

Servez du contenu à l'aide d'une vue

Les professeurs

Patrick Wampé

Développeur full stack et Data Scientist. Formateur dans plusieurs écoles d'informatique, il a également écrit un livre sur l'IA.

Patrick Heneghan

Software engineer in the UK, coding mostly in Python on backend systems.

Rafiq Hilali

British Software Engineer and Django expert with Lambert Labs. Currently based in BC, Canada.

EN PLUS

Français

Télécharger dans l'App Store