DJANGO 6

De débutant à professionnel

Construire un traqueur de bugs multi-entreprises, du modèle de données au déploiement

Apprendre Django par un projet guidé : PostgreSQL, Tailwind CSS, API REST, tests et mise en ligne

Auteur : fyardlest-tech
Éditions : Fyard-Self-Publishing & Fyard-Biblio-Tech
Année de publication : 2026

Sommaire

Ces pages liminaires regroupent les informations d'ouverture du livre :

Mentions légales

Tous droits réservés. Aucune partie de ce livre ne peut être reproduite, stockée dans un système de recherche documentaire, ni transmise sous quelque forme ou par quelque moyen que ce soit, sans l'autorisation écrite préalable de l'auteur ou de l'éditeur, sauf dans le cas de brèves citations insérées dans des articles critiques ou des recensions.

Tout a été mis en œuvre lors de la préparation de ce livre pour garantir l'exactitude des informations présentées. Toutefois, les informations qu'il contient sont vendues sans garantie, expresse ou implicite. Ni l'auteur, ni l'éditeur, ni ses distributeurs ne pourront être tenus responsables des dommages causés, ou présumés avoir été causés directement ou indirectement, par ce livre.

Titre : Django 6, de débutant à professionnel

Auteur : fyardlest-tech

Éditrice : Marie Carmelle

Éditions : Fyard-Self-Publishing & Fyard-Biblio-Tech

Date de publication : 2026

Dédicace

*À ma mère, **Marie Carmelle**,*

Ce livre t'est dédié avec toute ma gratitude et mon amour.

Merci pour tous les sacrifices que tu as consentis afin de m'offrir les meilleures chances dans la vie. Tu m'as tout donné, bien au-delà du matériel : tu m'as transmis des valeurs, le courage de persévérer, le respect des autres, l'humilité et la force de croire en moi, même dans les moments les plus difficiles.

Si je suis devenu la personne que je suis aujourd'hui, c'est grâce à ton amour inconditionnel, à ton soutien indéfectible et aux leçons de vie que tu m'as enseignées chaque jour, souvent par l'exemple plus que par les mots.

J'espère que chacune de ces pages reflète une partie de tout ce que tu m'as permis de construire.

Merci d'avoir toujours cru en moi.

Avec tout mon amour et mon éternelle reconnaissance.

Ton fils, Yardley

À ma fille Séverine,

Tu es la lumière de ma vie, celle qui illumine chacun de mes jours et donne un sens profond à chacun de mes efforts.

Ton sourire, ton innocence et ta joie de vivre sont une source d'inspiration inépuisable. Dans les moments de doute comme dans les plus belles réussites, il me suffit de penser à toi pour retrouver la force d'avancer et l'envie de donner le meilleur de moi-même.

Si j'écris, si je crée et si je relève de nouveaux défis, c'est aussi pour te montrer qu'avec du travail, de la persévérance et de la passion, il est possible de transformer ses rêves en réalité.

J'espère qu'un jour, lorsque tu liras ces lignes, tu comprendras que chacune des heures consacrées à ce livre était aussi une façon de construire un avenir dont tu pourras être fière.

Merci d'être mon bonheur, ma plus belle motivation et la plus grande des bénédictions de ma vie.

Je t'aime infiniment, aujourd'hui, demain et pour toujours. ❤

Ton papa, Yardley

Préface

Savoir programmer ne suffit plus : il faut aussi savoir construire une application complète, qui tienne debout de la première ligne de code jusqu'à sa mise en ligne. C'est précisément ce que ce livre vous apprend, avec le framework devenu une référence du développement web en Python : Django. Vous n'y apprendrez pas Django dans l'abstrait, mais en bâtissant, pas à pas, une véritable application : un traqueur de bugs multi–entreprises, sécurisé, doublé d'une API REST, testé et déployé en ligne.

Sa particularité est sa méthode : on apprend en construisant. Plutôt que d'aligner des concepts, chaque partie vous fait ajouter une pièce à un projet unique qui grandit sous vos yeux. Vous modélisez des données, vous écrivez l'authentification et les opérations de base, vous appliquez une véritable sécurité par rôles, vous soignez l'expérience utilisateur, vous ouvrez le tout par une API, puis vous testez et vous déployez. Le livre assume aussi une forme d'honnêteté : quand une bonne pratique a un coût ou une limite, on le dit, au lieu de la présenter comme magique.

Vous n'avez pas besoin d'être expert. Quelques bases de Python, un peu de ligne de commande et de la curiosité suffisent. À la fin, vous saurez concevoir des modèles, sécuriser une application, construire une API, écrire des tests et mettre votre projet en ligne, et vous disposerez d'une application concrète à présenter dans votre portfolio.

Pourquoi ce livre ?

Le développement web avec Django est une compétence très recherchée. Pourtant, de nombreux ouvrages se concentrent sur la théorie ou présentent des exemples trop éloignés de la réalité. J'ai souhaité écrire un livre différent : un guide pratique, orienté projet, qui vous accompagne pas à pas dans la construction d'une application professionnelle complète. À travers un projet unique et progressif, vous apprendrez non seulement à utiliser Django, mais aussi à raisonner comme un développeur : où poser la sécurité, comment organiser sa logique métier, pourquoi tester, et comment déployer.

À qui s'adresse ce livre ?

Ce livre s'adresse à toute personne qui veut apprendre le développement web de façon concrète : aux étudiants, aux développeurs débutants ou intermédiaires, aux personnes en reconversion, et à toute personne souhaitant se constituer un projet de portfolio solide pour décrocher un poste.

Il suppose quelques prérequis modestes, mais réels :

- Des bases en Python : lire et écrire un petit script, comprendre les fonctions et les classes.
- Un peu d'aisance avec la ligne de commande : ouvrir un terminal, lancer une commande.
- Avoir déjà entendu parler de Django, même quand tu n’as pas encore créé grand–chose.

Une connaissance de base de HTML, de Git ou des bases de données constitue un atout, mais chaque étape est expliquée afin que les débutants motivés puissent suivre et progresser à leur rythme. Aucune connaissance préalable de PostgreSQL ou de Tailwind n'est requise : tout est expliqué au fil du projet.

Un mot d'encouragement

Construire une application complète peut sembler intimidant au premier abord. Les modèles, les vues, les permissions, les migrations, le déploiement donnent parfois l'impression d'un univers réservé aux experts. Pourtant, chaque professionnel a commencé par son premier modèle, sa première vue et son premier bug à corriger. Prenez le temps d'expérimenter, ne craignez pas les erreurs, et surtout, construisez chaque partie jusqu'au bout. Les difficultés que vous rencontrerez feront partie de votre apprentissage. À la fin de ce parcours, vous ne saurez pas seulement utiliser Django : vous aurez une application concrète, en ligne, que vous pourrez présenter avec fierté dans votre portfolio ou lors d'un entretien. Bon apprentissage, et surtout, prenez plaisir à construire.

Ce que couvre ce livre

Partie 1 – Prise en main

Le Chapitre 1, Bienvenue, présente le projet BugTracker, ce que vous allez construire, et la méthode du livre.

Le Chapitre 2, Préparer son environnement, vous fait installer votre poste de travail : Python, un éditeur de code, Git, et les outils nécessaires.

Les Chapitres 3 et 4 créent le projet Django, le connectent à PostgreSQL, puis installent et configurent Tailwind CSS.

Partie 2 – Modéliser les données

Vous concevez le cœur du projet : l'entreprise et l'utilisateur personnalisé, les rôles avec les groupes de Django, les projets, les tickets, les commentaires et les pièces jointes, puis vous générez des données de démonstration.

Partie 3 – Authentification et CRUD

Vous mettez en place l'inscription, la connexion et la déconnexion, puis vous construisez la création, l'affichage, la modification et l'archivage des projets et des tickets, avec commentaires et pièces jointes.

Partie 4 – Rôles, permissions et sécurité

Le cœur du livre : gérer les rôles, affecter les membres aux projets, assigner les tickets, puis appliquer une matrice d'autorisation complète, vérifiée au serveur.

Partie 5 – Expérience et présentation

Vous soignez l'interface : navigation par rôle, filtres et pagination, page vitrine, tableau de bord avec graphiques, et pages d'erreur personnalisées.

Partie 6 – L'API REST avec DRF

Vous ouvrez l'application vers l'extérieur avec Django REST Framework : serializers, viewsets, authentification par jetons JWT, permissions et documentation.

Partie 7 – Fonctionnalités avancées

Vous ajoutez les invitations par email, l'inscription d'une nouvelle entreprise, et un système de notifications internes.

Partie 8 – Qualité, déploiement et portfolio

Vous écrivez des tests automatisés, vous déployez l'application en ligne (Railway, et Azure en complément), et vous préparez votre projet pour votre portfolio.

Les bonus et compléments

Sept bonus prolongent le livre : un aide–mémoire Django, les notifications complètes, la page de profil, l'API complétée, les tests avancés et l'intégration continue, l'envoi d'emails réels, et une préparation à l'entretien de développeur Python / Django. Des compléments de référence détaillent la matrice des permissions et les douze facteurs d'une application prête pour la production.

La Partie 9 et les corrigés

Une neuvième partie, consacrée à l'intelligence artificielle, transforme le BugTracker en une application intelligente dotée d'un agent. Elle vous est envoyée par courriel, avec les corrigés de tous les défis du livre (voir la section Télécharger le code d'exemple).

Pour tirer le meilleur parti de ce livre

Le livre est conçu pour être suivi la main sur le clavier. Tape le code toi-même plutôt que de le copier-coller : c'est en butant sur les erreurs qu'on apprend le plus. Le projet se construit par petites étapes ; à la fin de chaque chapitre, on enregistre son avancement avec Git avant de continuer.

Logiciels nécessaires

- Python 3.12 ou plus récent, pour Django et le projet.
- PostgreSQL, la base de données utilisée tout au long du livre.
- Un éditeur de code (Visual Studio Code recommandé) et un terminal (Git Bash sous Windows).
- Un navigateur web récent, pour utiliser l'application.

Systèmes d'exploitation

Windows 10 ou plus récent, macOS, ou Linux. Les captures et commandes sont données pour un poste Windows, mais tout fonctionne de manière équivalente sur macOS et Linux, avec les commandes fournies pour les deux univers.

Version de Django utilisée : 6.0

Version de Python utilisée : 3.14

Télécharger le code d'exemple

Le code de ce livre est fourni de deux manières complémentaires, qu'il ne faut pas confondre.

1. Le dépôt de code du BugTracker (le projet complet)

Le dépôt indiqué ci-dessous contient le code complet et fonctionnel du BugTracker, déjà corrigé et prêt à l'emploi. C'est la version finie de l'application, utile comme référence ou point de comparaison une fois les parties terminées.

> **Astuce**
> Le dépôt te sert de filet : si tu es bloqué, compare ton code avec le sien. Mais tape le tien d'abord, c'est ainsi qu'on apprend.

2. La Partie 9 et les corrigés (envoyés par email après l'achat)

Après l'achat du livre, la neuvième partie, consacrée à l'intelligence artificielle, ainsi que les corrigés de tous les défis, sont envoyés à l'adresse email utilisée lors de la commande, dès qu'ils sont prêts. C'est le prolongement du livre, qui ajoute au BugTracker un agent intelligent.

> **Note**
> Le courriel arrive à l'adresse liée à ton achat. Pense à vérifier ton dossier de courrier indésirable (spam), et à ajouter l'adresse d'expéditeur à tes contacts. Ce contenu est personnel : ne le partage pas.

Plateforme d'achat du livre : https://livrestek.lemonsqueezy.com/

Dépôt du code, projet complet (GitHub) : https://github.com/fyardlest1/BugTrackerApp-Final.git

Images en couleur (PDF) : Inclus dans le dossier de téléchargement

Démonstration en ligne

Tu peux essayer l'application déployée ici : https://bugtrackerapp.up.railway.app/

Des comptes de démonstration, accessibles en un clic, te permettent d'essayer chaque rôle sans créer de compte.

Conventions utilisées

Plusieurs conventions de mise en forme sont employées tout au long de ce livre.

Code dans le texte : indique un nom de code, une commande, un nom de fichier, un chemin, une variable ou une valeur. Par exemple : *python manage.py migrate* applique les migrations.

Un bloc de code se présente ainsi, avec un liseré coloré sur sa gauche, et précise toujours le fichier concerné :

```
# tickets/models.py
class Ticket(BaseModel):
    title = models.CharField(max_length=200)
```

Des encadrés attirent l'attention selon leur couleur et leur étiquette :

> **Note**
> Une information utile, un rappel ou une précision de contexte.

> **Astuce**
> Un raccourci ou une bonne pratique pour aller plus vite.

> ⚠ **Attention**
> Un point délicat à ne pas manquer, sous peine d'erreur.

> **Comment fonctionne Django**
> Un éclairage sur le fonctionnement interne de Django, pour comprendre la mécanique.

> **Bonne pratique pro**
> La façon dont on procéderait sur un vrai projet professionnel.

> **Ce qu'un recruteur regarde**
> Le détail qui fait la différence en entretien ou aux yeux d'un relecteur.

> **En entreprise**
> Comment cette fonctionnalité est réellement utilisée en équipe.

On trouve aussi, dans chaque chapitre, les sections récurrentes Pourquoi cette étape, Ce que nous avons construit, Enregistrer ton avancement, et Ton défi.

Rester en contact

Les retours des lecteurs sont toujours les bienvenus. Voici les moyens de me contacter et de signaler d'éventuelles erreurs.

Retours généraux (email) : tekprototypeqc@gmail.com

Signaler une erreur (errata) : tekprototypeqc@gmail.com

Signaler une copie illégale : tekprototypeqc@gmail.com

Proposer une contribution : tekprototypeqc@gmail.com

Table des matières

Django 6, de débutant à professionnel

CHAPITRE 1

Bienvenue : le projet BugTracker

Bienvenue. Avant d'écrire la moindre ligne de code, prenons cinq minutes ensemble pour comprendre ce que nous allons construire, et surtout pourquoi. Imagine que je suis assis à côté de toi. Nous n'allons pas lire une documentation. Nous allons bâtir, écran après écran, une vraie application que tu pourras montrer fièrement à un recruteur.

Ce livre a un seul objectif : t'amener d'un projet Django vide jusqu'à une application professionnelle, sécurisée et déployée en ligne. Pas un projet jouet. Un vrai traqueur de bugs multi-entreprises, avec des rôles, une API, des graphiques et un déploiement. Le genre de projet qui fait la différence sur un CV de développeur junior et intermédiaire.

Pourquoi ce chapitre existe

Beaucoup de tutoriels te jettent directement dans le code. Résultat : tu recopies des lignes sans savoir où tu vas. Ici, on commence par la carte du voyage. À la fin de ce chapitre, tu sauras exactement ce que tu construis, avec quels outils, et comment travailler efficacement tout au long du livre.

> **Note**
> Ce livre s'appuie sur un vrai cahier des charges de projet portfolio. Chaque fonctionnalité que nous allons coder correspond à une exigence concrète que l'on retrouve dans les outils professionnels comme **Jira** ou **Azure DevOps**.

Ce que tu vas construire

Un **BugTracker**, c'est un système de suivi d'incidents. En clair : un endroit où une équipe déclare des problèmes (des bugs), les organise par projet, les assigne à des développeurs, en discute et suit leur résolution. C'est l'un des outils les plus utilisés au quotidien dans une équipe logicielle.

Voici, en une phrase, l'application finale : plusieurs entreprises peuvent utiliser la même application sans jamais voir les données des autres, chaque entreprise gère ses projets et ses tickets, et chaque utilisateur joue un rôle qui détermine ce qu'il a le droit de faire.

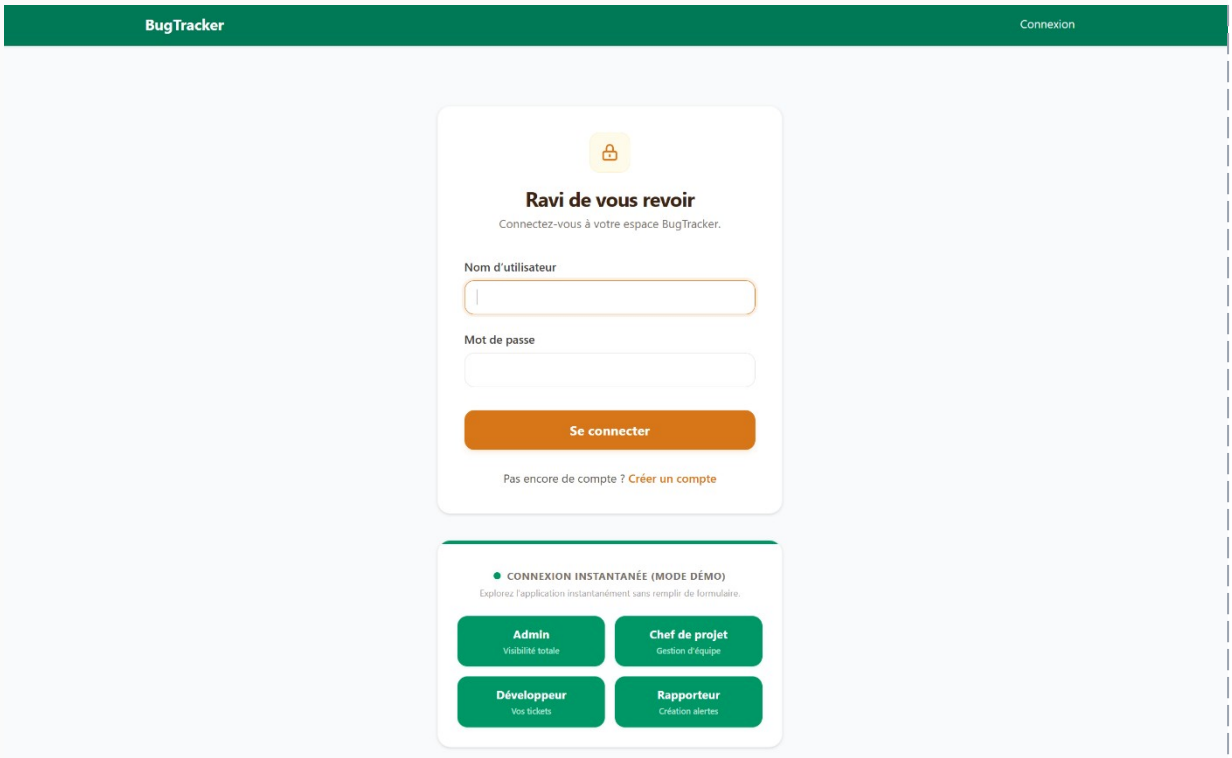
Concrètement, ton application saura :

- Séparer totalement les données de chaque entreprise. Un utilisateur de l'entreprise **A** ne verra jamais un ticket de l'entreprise **B**.
- Gérer une authentification complète : inscription, connexion, déconnexion.
- Attribuer des rôles : Administrateur, Chef de projet, Développeur, Rapporteur.
- Créer, modifier, archiver et restaurer des projets et des tickets.

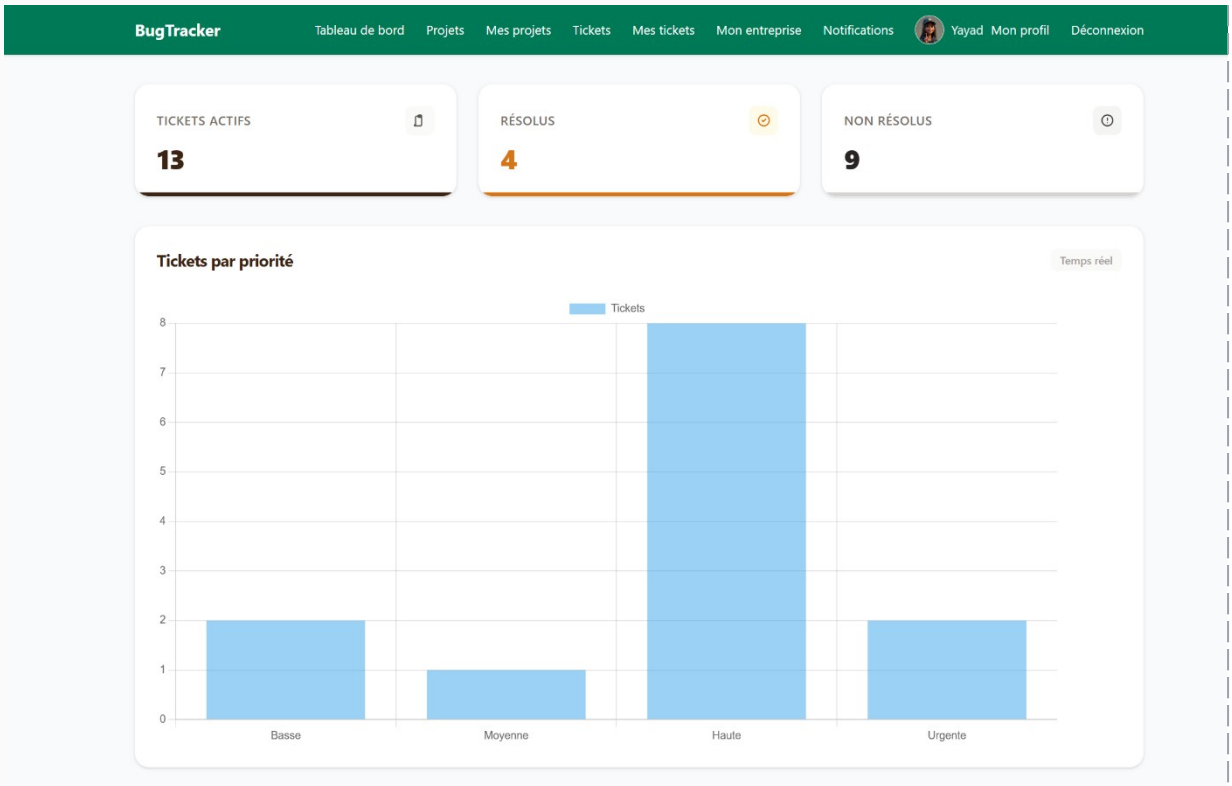
- Commenter les tickets et y joindre des fichiers.
- Afficher un tableau de bord avec des graphiques.
- Exposer une API REST prête pour une future application mobile.
- Se déployer en ligne, avec un lien que tu mettras dans ton portfolio.

À quoi ressemblera le résultat final

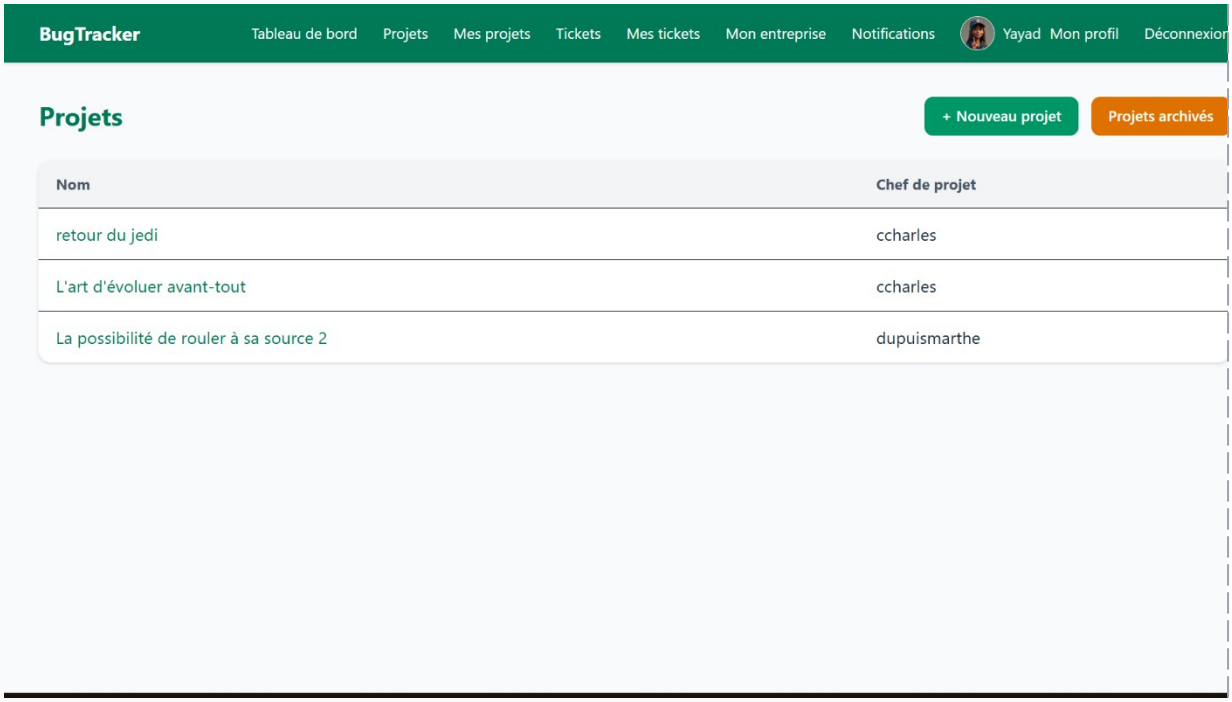
Garde ces images en tête pendant tout le livre. C'est la destination. Tu insèreras ici tes propres captures une fois l'application terminée.



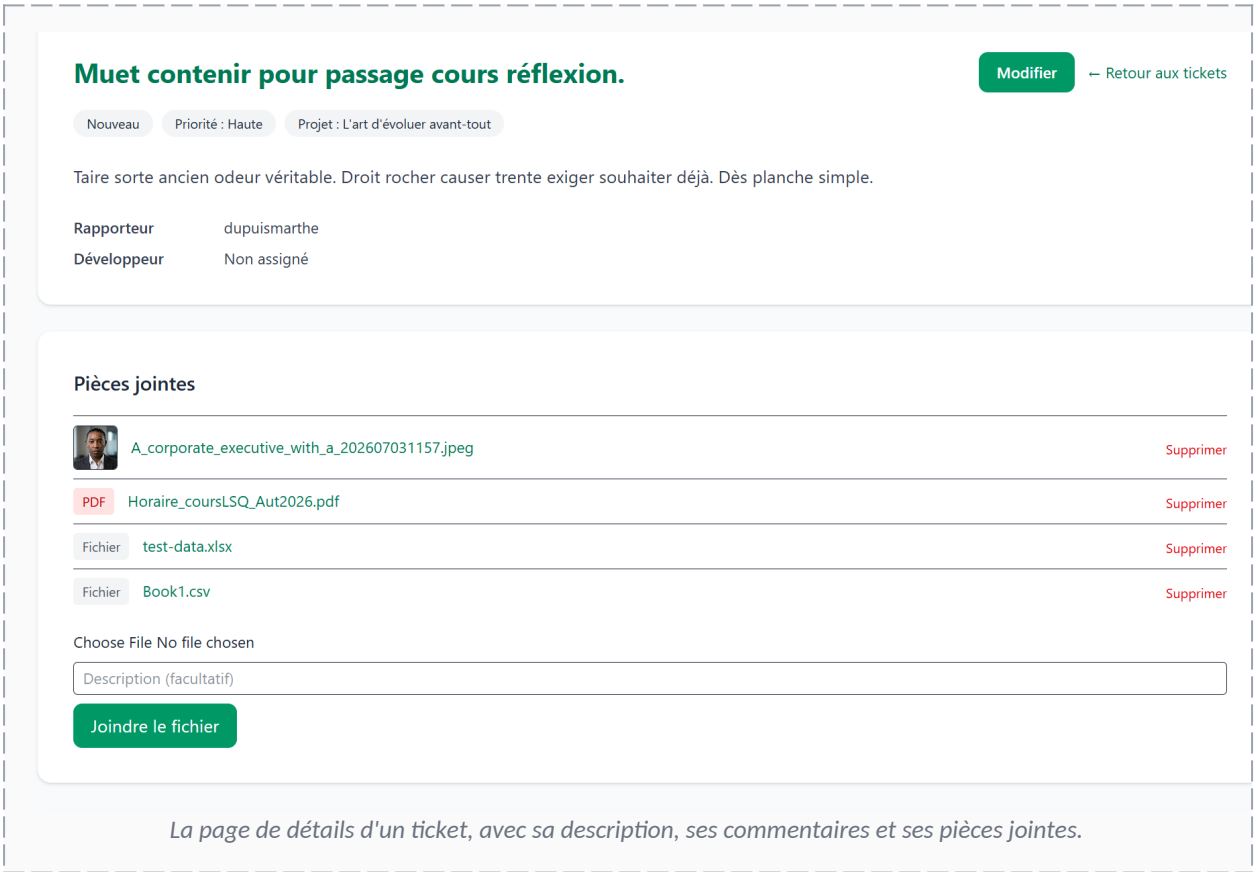
*La page de connexion avec les boutons de démonstration (**Admin**, **Chef de projet**, **Développeur**, **Rapporteur**).*



Le tableau de bord d'un utilisateur connecté, avec les graphiques de tickets par priorité et par statut.



La liste des projets d'une entreprise, présentée dans un tableau stylé avec Tailwind CSS.



La page de détails d'un ticket, avec sa description, ses commentaires et ses pièces jointes.

Ce que tu vas apprendre

Ce projet n'est pas qu'une liste de fonctionnalités. C'est un prétexte pour maîtriser les compétences qu'on attend réellement d'un développeur Django. À la fin de ce livre, tu sauras :

- Concevoir des modèles de données propres avec l'ORM de Django et des relations **ForeignKey** et **ManyToMany**.
- Créer un utilisateur personnalisé et gérer les rôles avec le système de groupes de Django.
- Construire des vues avec les **Class-Based Views** (*ListView*, *DetailView*, *CreateView*, *UpdateView*).
- Sécuriser une application : authentification, autorisation par rôle, et isolation stricte des données.
- Styliser une interface moderne et responsive avec Tailwind CSS.
- Gérer l'upload de fichiers, les transactions, et la génération de données de test.
- Construire une API REST avec Django REST Framework et l'authentification par jetons JWT.
- Écrire des tests, conteneuriser avec Docker et déployer en ligne.

Pourquoi ce projet vaut de l'or dans ton portfolio

Un ***compteur***, une ***todo-list*** ou même un ***blog*** ne surprennent plus personne en entretien. Un **BugTracker**, si. Parce qu'il touche à tout ce qui compte vraiment dans une application professionnelle : la sécurité des données, la gestion fine des permissions, une architecture qui sépare les responsabilités, et une API pensée pour évoluer.

> **Ce qu'un recruteur regarde**
> Un recruteur ne regarde pas seulement si ton application fonctionne. Il regarde si tu comprends ce que tu as fait. Avec ce projet, tu pourras expliquer comment tu empêches une entreprise de voir les données d'une autre, pourquoi tu as séparé la logique métier des vues, et comment ton API pourrait alimenter une application mobile. C'est cette capacité à raconter ton architecture qui te démarque.

> **En entreprise**
> Dans une vraie équipe, un outil de suivi de bugs est utilisé chaque jour. ***Jira***, ***Azure DevOps***, ***GitHub Issues***, ***Linear***... tous reposent sur les mêmes concepts que ceux de ce livre : des projets, des tickets, des statuts, des rôles et des permissions. En construisant le tien, tu apprends le vocabulaire et les mécaniques que tu retrouveras dès ton premier poste.

La stack technique, et pourquoi ces choix

On ne choisit jamais une technologie au hasard. Voici les outils du livre et, surtout, la raison de chaque choix. Retiens ce réflexe dès maintenant : toujours se demander pourquoi, pas seulement comment.

Django 6+

Django est un ***framework web Python*** dit *batteries incluses*. Traduction : il fournit déjà tout ce dont une application a besoin, sans que tu réinventes la roue. Authentification, interface d'administration, sécurité, gestion de la base de données... c'est déjà là.

> **Comment fonctionne Django**
> Django suit le concept **MVT** : ***Model***, ***Vue***, ***Template***. Le **Modèle** décrit tes données, la **Vue** contient la logique qui répond à une requête, et le **Template** est la page HTML affichée à l'utilisateur. Tout au long du livre, quand tu te demanderas où mettre un bout de code, cette question t'aidera : est-ce que ça décrit une donnée, une logique, ou un affichage ?

PostgreSQL

C'est notre base de données. On aurait pu utiliser SQLite, plus simple à démarrer, mais PostgreSQL est le standard en entreprise pour les applications sérieuses. Il est robuste, gère de gros volumes, et supporte nativement des types avancés comme les UUID que nous utiliserons partout.

> **Bonne pratique pro**
> Développer directement sur la même base que celle utilisée en production évite les mauvaises surprises

au déploiement. Un projet qui marche sur ***SQLite*** mais casse sur ***PostgreSQL*** est un grand classique. On s'épargne ce piège dès le départ.

Django REST Framework (DRF)

DRF transforme ton application en API : au lieu de renvoyer des pages HTML, elle peut renvoyer des données brutes au format JSON. Pourquoi c'est important ? Parce que le jour où tu voudras une application mobile ou une interface ***React*** branchée sur les mêmes données, tout sera déjà prêt. Tu construis une fois, tu réutilises partout.

Tailwind CSS

Tailwind est une bibliothèque de style qui te permet de créer des interfaces modernes sans écrire de CSS à la main. Tu composes ton design directement dans le HTML avec de petites classes. Résultat : un rendu professionnel et cohérent, beaucoup plus vite qu'en partant de zéro.

Les UUID comme identifiants

Par défaut, Django numérote tes enregistrements 1, 2, 3, et ainsi de suite. On va plutôt utiliser des UUID, ces longs identifiants comme a1b2c3d4-... Deux raisons très concrètes. D'abord, une adresse comme */tickets/1* révèle que tu n'as qu'un seul ticket, et permet à un curieux de deviner */tickets/2*. Un UUID rend ça impossible. Ensuite, c'est le choix standard dans les grandes applications distribuées.

> **Ce qu'un recruteur regarde**
> Savoir expliquer pourquoi tu utilises des UUID plutôt que des identifiants séquentiels est exactement le genre de détail qui montre à un Tech Lead que tu penses à la sécurité et à l'échelle, pas seulement à faire marcher le code.

Un utilisateur personnalisé et des Groups

Nous remplacerons l'utilisateur par défaut de Django par le nôtre, pour lui rattacher son entreprise. Et nous gérerons les rôles avec les groupes intégrés de Django plutôt qu'avec un simple champ texte. On expliquera tout ça en détail le moment venu. Retiens juste pour l'instant que ces deux choix sont ceux qu'on ferait dans un grand projet.

> ⚠ **Attention**
> Créer son utilisateur personnalisé se fait au tout début du projet, avant la première migration de la base. Si on l'oublie et qu'on veut le faire plus tard, c'est très pénible à corriger. Nous nous en occuperons donc dès le chapitre sur les modèles. Pas d'inquiétude, je te préviendrai.

Comment ce livre est organisé

Le livre avance en huit parties, du plus simple au plus avancé. Chaque partie regroupe des chapitres courts et concrets. On ne passe jamais à la suite sans avoir vu la fonctionnalité précédente fonctionner à l'écran.

1. **Fondations :** préparer l'environnement, créer le projet, installer Tailwind.
2. **Modéliser les données :** entreprise, utilisateur, rôles, projets, tickets.
3. **Authentification et CRUD du cœur métier :** projets, tickets, commentaires, pièces jointes.
4. **Rôles, permissions et sécurité :** appliquer une vraie matrice d'autorisation.
5. **Expérience et présentation :** navigation, landing page, tableau de bord.
6. **API REST avec DRF :** serializers, viewsets, authentification par jetons.
7. **Fonctionnalités avancées :** invitations, inscription d'entreprise, notifications.
8. **Qualité, déploiement et portfolio :** tests, mise en ligne, préparation à l'entretien.

Chaque chapitre suit toujours le même rythme : on explique pourquoi on fait cette étape, ce que tu vas apprendre, puis on construit petit à petit en testant au fur et à mesure. On termine par les erreurs fréquentes, un regard sur la pratique en entreprise, un défi à réaliser seul, et un résumé.

La légende des encadrés

Tout au long du livre, tu croiseras des encadrés colorés. Chacun joue un rôle précis. Voici la légende, garde-la en tête.

> **Astuce**
> Un raccourci, une commande pratique ou une petite chose qui va te faire gagner du temps.

> ⚠ **Attention**
> Un piège classique ou une erreur fréquente. Lis-le attentivement, il t'évitera de perdre une heure à déboguer.

> **Bonne pratique pro**
> La façon dont on ferait vraiment les choses dans un projet professionnel, au-delà du simple fait que ça marche.

> **Comment fonctionne Django**
> Un éclairage sur le fonctionnement interne de Django, pour comprendre la magie plutôt que la subir.

> **Ce qu'un recruteur regarde**
> Le point précis qu'un recruteur ou un Tech Lead remarquera, et que tu pourras mettre en avant en entretien.

> **En entreprise**
> Comment cette fonctionnalité est réellement utilisée et implémentée en entreprise.

> **Note**
> Une information utile, un rappel ou une précision de contexte.

Les outils dont tu auras besoin

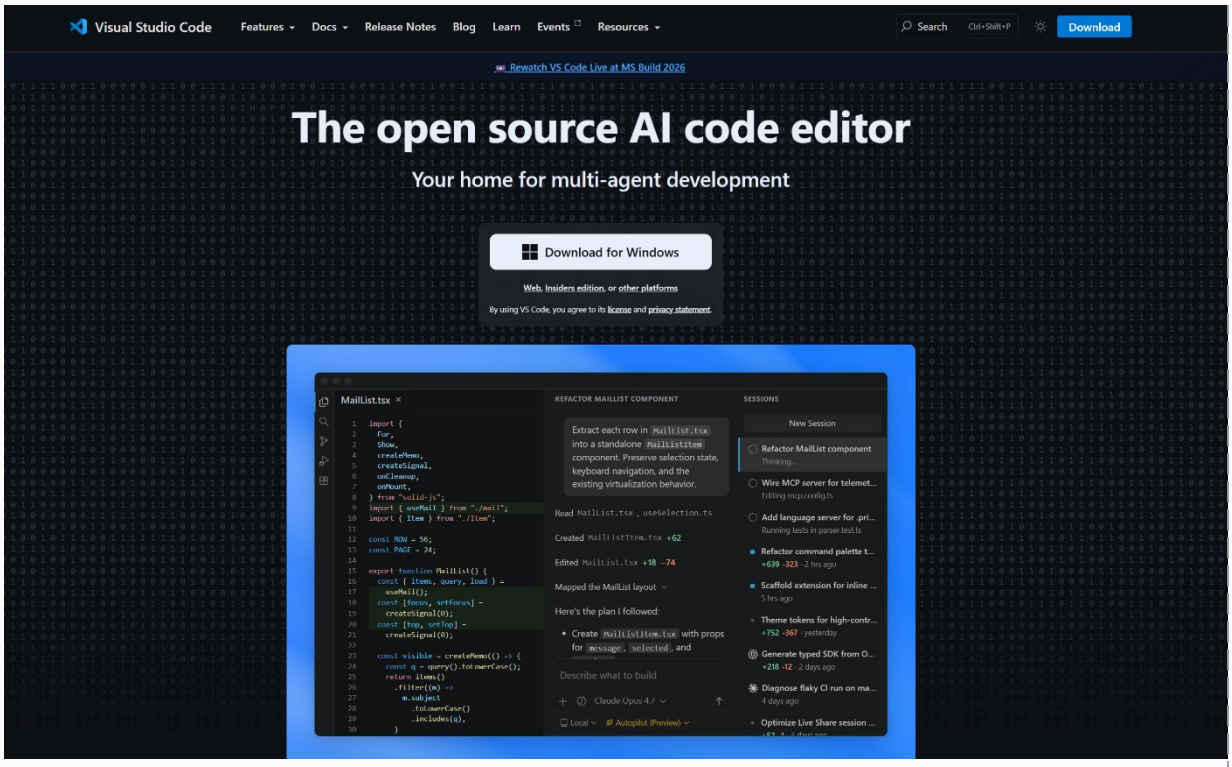
Avant de plonger dans le code au prochain chapitre, installons les outils de base. On les configurera en détail au chapitre 2 ; ici, on se contente de les télécharger et de vérifier qu'ils répondent.

Un éditeur de code : Visual Studio Code

Pour écrire ton code, je te recommande Visual Studio Code, souvent abrégé VS Code. Il est gratuit, léger, disponible sur Windows, macOS et Linux, et c'est l'éditeur le plus utilisé par les développeurs Django. C'est celui que j'utiliserai pour toutes les captures d'écran du livre.

Télécharge-le depuis le site officiel *code.visualstudio.com* et installe-le en gardant les options par défaut.

> **Astuce**
> Tu es libre d'utiliser l'éditeur de ton choix : PyCharm, Sublime Text, Antigravity… Tout ce que tu apprendras ici fonctionne partout. Si tu débutes, reste sur **VS Code** pour suivre les captures sans te poser de questions.



La page de téléchargement de Visual Studio Code sur code.visualstudio.com.

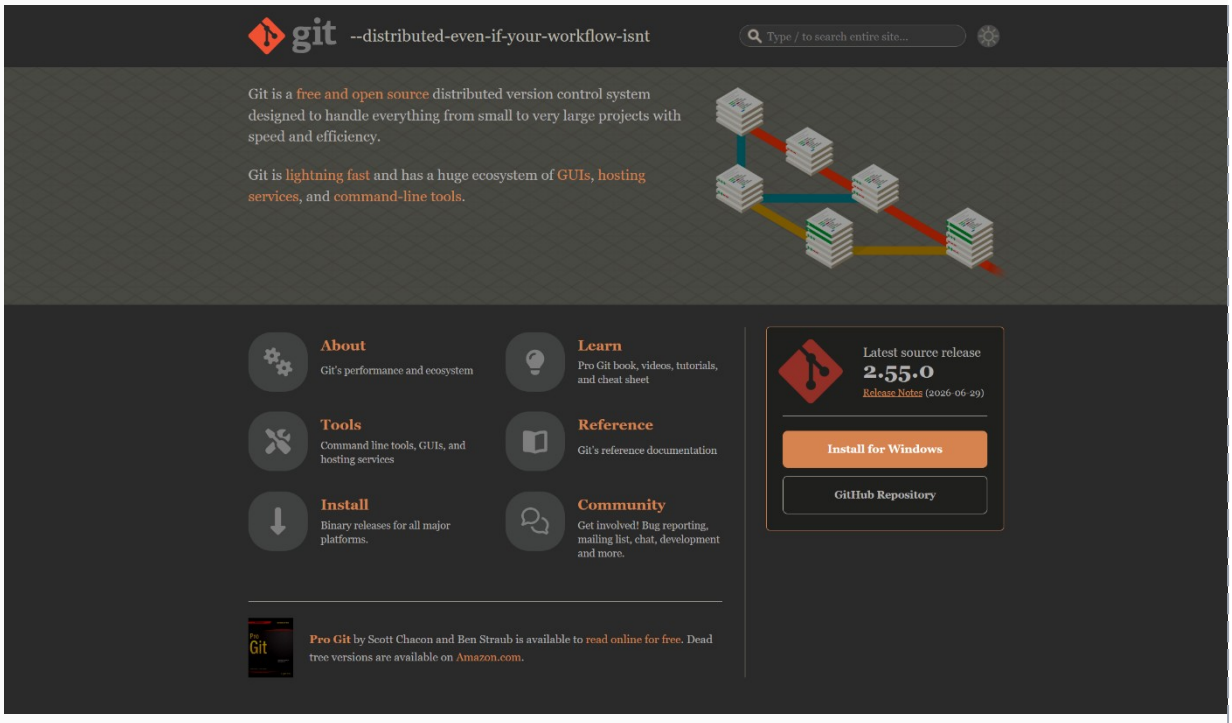
Un terminal : Git Bash sous Windows

On va beaucoup travailler dans un terminal, cette fenêtre où l'on tape des commandes. Sous macOS et Linux, le terminal intégré fait parfaitement l'affaire. Sous Windows, je te recommande d'installer Git Bash plutôt que d'utiliser PowerShell.

Pourquoi Git Bash ? Parce qu'il comprend les mêmes commandes que macOS et Linux. Tu utiliseras donc exactement les mêmes instructions que la majorité des tutoriels et de la documentation, sans avoir à les traduire. En bonus, il installe Git, l'outil qui va suivre l'historique de ton code.

Sous Windows, télécharge Git depuis *git-scm.com*. L'installateur inclut Git Bash. Garde les options par défaut, elles conviennent très bien.

> **Note**
> Dans ce livre, chaque fois qu'une commande diffère selon le système, je te donnerai les deux versions : une pour Git Bash sous Windows et pour les terminaux Linux et macOS, et une pour PowerShell si tu préfères le terminal natif de Windows. Tu n'auras jamais à deviner.



La page de téléchargement de Git sur git-scm.com, avec Git Bash inclus.

Vérifier que tout répond

On ne fait pas encore d'installation Django ici. On vérifie simplement que les outils de base sont là. Ouvre ton terminal (Git Bash sous Windows, le Terminal sous macOS ou Linux) et tape ces commandes. Les versions exactes n'ont pas d'importance pour l'instant.

```
# Git Bash (Windows) ou terminal Linux / macOS
git --version
python --version      # ou parfois : python3 --version
```

```
# PowerShell (Windows)
git --version
python --version
```

Si chaque commande affiche un numéro de version, tu es prêt. Si Python n'est pas reconnu, pas de panique : on installera la bonne version proprement au chapitre 2, c'est justement son rôle.

> ⚠ **Attention**
> Sous Windows, si tu installes Python toi-même, coche impérativement la case ***Add Python to PATH*** pendant l'installation. Sans elle, le terminal ne saura pas où trouver Python et la commande échouera. On y reviendra en détail au chapitre 2.

Comment travailler avec ce livre

Ce livre est fait pour être pratiqué, pas seulement lu. Voici la méthode qui donne les meilleurs résultats.

- Tape le code toi-même. Ne **copie-colle** pas. Le fait d'écrire ancre l'apprentissage bien mieux que la lecture.
- Teste à la fin de chaque chapitre. Ne passe jamais à la suite tant que la fonctionnalité en cours ne marche pas chez toi.
- Enregistre ton avancement avec Git après chaque étape qui fonctionne. C'est ton filet de sécurité.
- Relève le défi de fin de chapitre. C'est là que tu passes de **recopieur** à **développeur**.

> **Bonne pratique pro**
> Prends l'habitude d'enregistrer ton travail dans Git dès qu'une petite étape fonctionne, avec un message clair comme ajout du modele Project. En entreprise, un historique de **commits** propre et régulier est un signe de sérieux. C'est aussi ce qui te sauve quand une modification casse tout : tu peux revenir en arrière en une commande.

> **En entreprise**
> L'erreur numéro un des développeurs juniors est d'écrire beaucoup de code d'un coup avant de le tester. Quand quelque chose casse, impossible de savoir quelle ligne est en cause. La bonne habitude, celle qu'on te demandera en équipe : de petits changements, testés souvent, enregistrés régulièrement.

Ce que nous avons vu

Faisons le point. Dans ce chapitre, tu as découvert le projet que nous allons construire : un traqueur de bugs multi-entreprises, sécurisé, avec une API et un déploiement en ligne. Tu sais désormais pourquoi ce projet est un atout pour ton portfolio, quelles technologies nous allons utiliser et surtout pourquoi. Tu connais la structure du livre, la légende des encadrés, et tu as installé tes outils de base.

Ton défi

Avant de passer à la suite, prends dix minutes pour ces trois petites actions. Elles te mettront en selle.

1. Installe Visual Studio Code (ou l'éditeur de ton choix), et sous Windows installe Git Bash via git-scm.com.
2. Ouvre ton terminal et vérifie que git --version répond bien.
3. Crée un compte gratuit sur GitHub si tu n'en as pas encore. Nous y publierons ton projet, et c'est le lien que verront les recruteurs.

> **Astuce**
> Pas encore de compte GitHub ? Rends-toi sur github.com et inscris-toi. Choisis un nom d'utilisateur propre et professionnel : c'est une partie de ta vitrine de développeur.

Prochaine étape

Assez de préparatifs, place à la pratique. Au chapitre 2, on installe et on configure tout l'environnement de développement : Python et son environnement virtuel, PostgreSQL et pgAdmin, puis Django lui-même. À la fin du prochain chapitre, tu auras une base de données prête et un socle solide pour créer le projet. On y va.

CHAPITRE 2

Préparer son environnement de travail

On ne construit pas une maison sans avoir d'abord préparé le terrain et sorti les outils. C'est exactement ce qu'on fait dans ce chapitre. Avant d'écrire du code Django, il nous faut un atelier propre : Python installé, un espace de travail isolé pour notre projet, une base de données prête, et Git pour suivre chaque changement.

C'est peut-être le chapitre le moins spectaculaire du livre, mais c'est l'un des plus importants. Un environnement bien préparé t'évitera des heures de bugs bizarres plus tard. On prend le temps de bien faire, une fois, et on n'y revient plus.

Pourquoi cette étape ?

Chaque projet Django a besoin de sa propre bulle : sa version de Python, ses bibliothèques, sa base de données. Si tu installes tout en vrac sur ton ordinateur, deux projets finiront tôt ou tard par se marcher sur les pieds. L'un voudra une version d'une bibliothèque, l'autre une version différente, et c'est le conflit.

La solution professionnelle tient en un mot : l'isolation. On crée pour chaque projet un environnement séparé qui contient uniquement ce dont ce projet a besoin. C'est propre, reproductible, et c'est exactement ce qu'on fait en entreprise.

Ce que tu vas apprendre

À la fin de ce chapitre, tu sauras :

- Installer Python et vérifier qu'il fonctionne.
- Configurer Visual Studio Code pour Python.
- Créer et activer un environnement virtuel, cette fameuse bulle isolée.
- Installer Django 6 dans cet environnement.
- Installer PostgreSQL et pgAdmin, puis créer la base de données du projet.
- Mettre ton projet sous contrôle de version avec Git.

Étape 1 — Installer Python

Django est écrit en Python, donc c'est notre point de départ. Django 6 exige une version récente de Python : prends Python 3.12 ou plus. Tu peux vérifier si Python est déjà présent en ouvrant ton terminal et en tapant la commande de vérification.

```
# Git Bash (Windows) ou Linux / macOS
python --version      # sinon essaie : python3 --version
```

```
# PowerShell (Windows)
python --version
```

Si tu vois s'afficher un numéro comme Python 3.12.x, tu peux passer à l'étape suivante. Sinon, télécharge et installe Python.

- Windows et macOS : télécharge l'installateur officiel sur python.org, rubrique Downloads (*https://www.python.org/downloads*).
- Linux : utilise le gestionnaire de paquets de ta distribution, par exemple ***sudo apt install python3 python3-venv python3-pip*** sur Ubuntu.

> ⚠ **Attention**
> Sous Windows, l'installateur affiche en bas une case Add python.exe to PATH. Coche-la absolument avant de cliquer sur Install. C'est l'oubli numéro un des débutants : sans cette case, ton terminal répondra python n'est pas reconnu, et rien ne fonctionnera. Si ça t'arrive, désinstalle puis réinstalle en cochant la case.

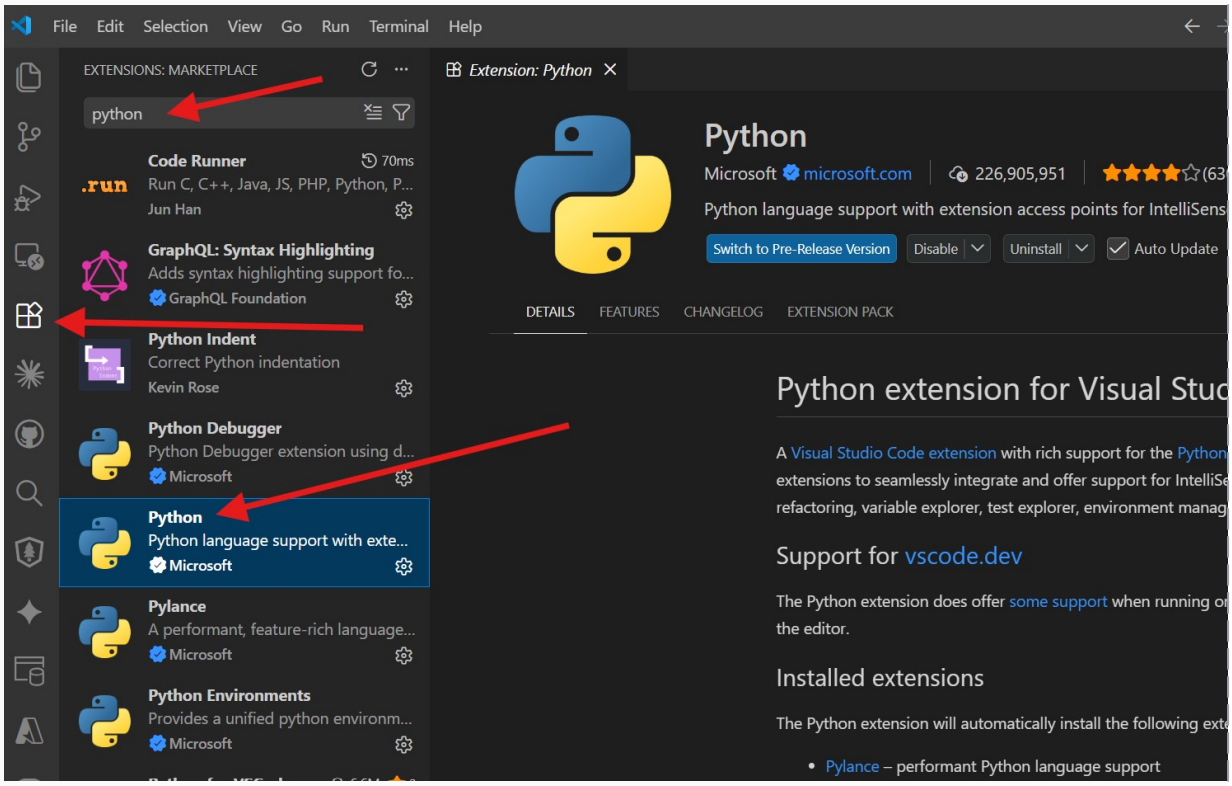
L'installateur de Python sous Windows, avec la case Add python.exe to PATH cochée en bas.

Étape 2 — Configurer Visual Studio Code

Tu as installé VS Code au chapitre précédent. Ajoutons-lui une extension qui va te faciliter la vie : l'extension Python de Microsoft. Elle apporte l'autocomplétions, la coloration intelligente, le débogage et la détection automatique de ton environnement virtuel.

1. Ouvre VS Code.
2. Clique sur l'icône Extensions dans la barre de gauche (le carré composé de petits carrés).
3. Cherche Python, puis installe l'extension officielle éditée par Microsoft.

> **Astuce**
> Installe aussi l'extension ***Tailwind CSS*** IntelliSense. Elle proposera l'autocomplétions des classes Tailwind quand nous stylerons nos pages. Ce n'est pas indispensable maintenant, mais tu me remercieras plus tard.



L'onglet Extensions de VS Code avec l'extension Python de Microsoft installée.

Étape 3 — Créer le dossier du projet

Choisis un endroit clair sur ton disque pour ranger le projet, puis crée un dossier. Évite les chemins qui contiennent des accents ou des espaces, ils causent parfois des soucis avec certains outils.

```
# Git Bash (Windows) ou Linux / macOS
cd ~/Documents
mkdir bugtracker
cd bugtracker
```

```
# PowerShell (Windows)
cd $HOME\Documents
mkdir bugtracker
cd bugtracker
```

Tu peux maintenant ouvrir ce dossier dans VS Code, soit depuis le menu Fichier puis Ouvrir le dossier, soit directement en ligne de commande.

```
code .        # ouvre le dossier courant dans VS Code
```

Étape 4 — Créer et activer l'environnement virtuel

Voici notre fameuse bulle isolée. La commande venv, incluse dans Python, crée un dossier caché qui contiendra une copie de Python et toutes les bibliothèques propres à ce projet. On l'appelle traditionnellement .venv.

```
# Créer l'environnement (identique partout)
python -m venv .venv        # Linux/macOS : parfois python3 -m venv .venv
```

L'environnement est créé, mais il faut maintenant l'activer pour l'utiliser. C'est l'étape que tout le monde oublie au début. L'activation diffère selon ton terminal.

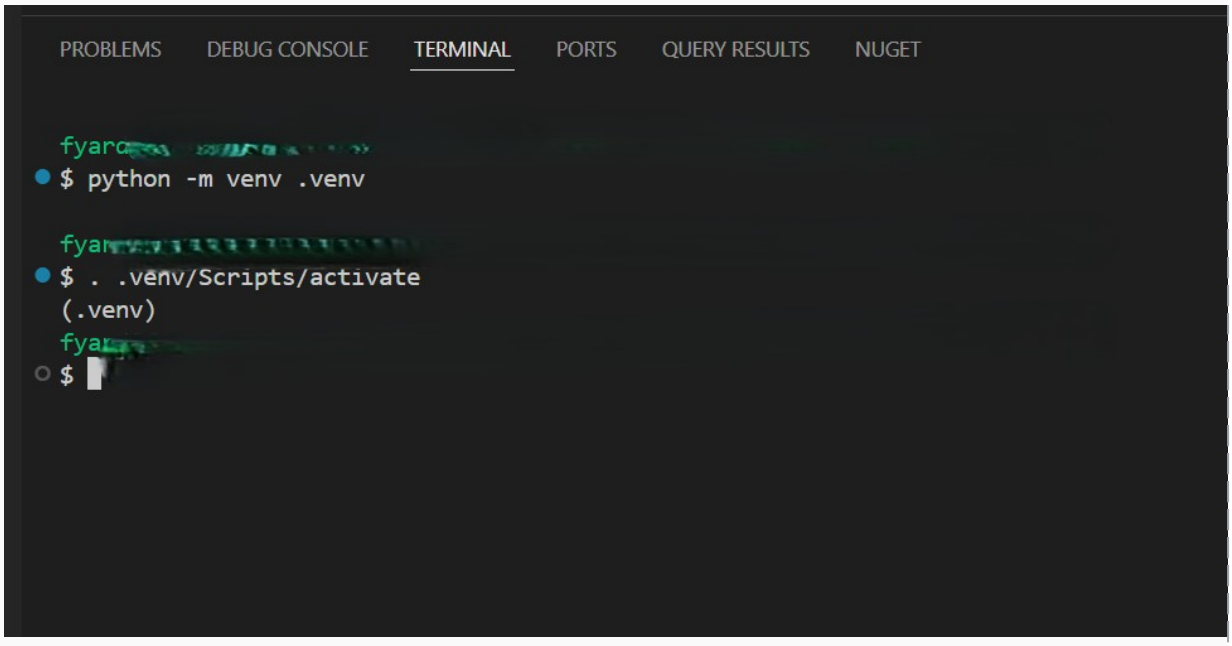
```
# Activer — Git Bash (Windows)
source .venv/Scripts/activate

Ou encore:
. .venv/Scripts/activate

# Activer — Linux / macOS
source .venv/bin/activate
```

```
# Activer — PowerShell (Windows)
.venv\Scripts\Activate.ps1
```

Quand l'environnement est actif, ton invite de commande affiche (.venv) au début de la ligne. C'est ton signal visuel : tant que tu vois ce préfixe, tu travailles bien dans la bulle du projet.



Le terminal avec le préfixe (.venv) affiché en début de ligne, signe que l'environnement est actif.

> ⚠ **Attention**
> Sous Windows, si PowerShell refuse d'exécuter le script d'activation avec un message d'erreur sur les stratégies d'exécution, lance une fois cette commande puis réessaie : *Set-ExecutionPolicy -Scope CurrentUser RemoteSigned*. C'est précisément ce genre de tracas qui explique pourquoi je te recommande Git Bash sous Windows : tu n'as pas ce problème.

> ⍰ **Comment fonctionne Django**
> Pourquoi un environnement virtuel plutôt que d'installer Django sur tout l'ordinateur ? Parce que chaque projet fige ainsi ses propres versions. Dans six mois, tu pourras revenir sur ce projet et il fonctionnera toujours, même si tu as installé des versions plus récentes ailleurs. On enferme les dépendances dans la bulle du projet plutôt que de les répandre partout. C'est un principe qu'on appelle l'isolation des dépendances.

Étape 5 — Installer Django 6

Maintenant que la bulle est active, tout ce qu'on installe reste dedans. Installons Django. On utilise ***pip***, l'outil qui télécharge et installe les bibliothèques Python.

```
# L'environnement (.venv) doit être actif
pip install django
```

Vérifie que l'installation a réussi en demandant sa version à Django.

```
django-admin --version       # doit afficher 6.x.x
```

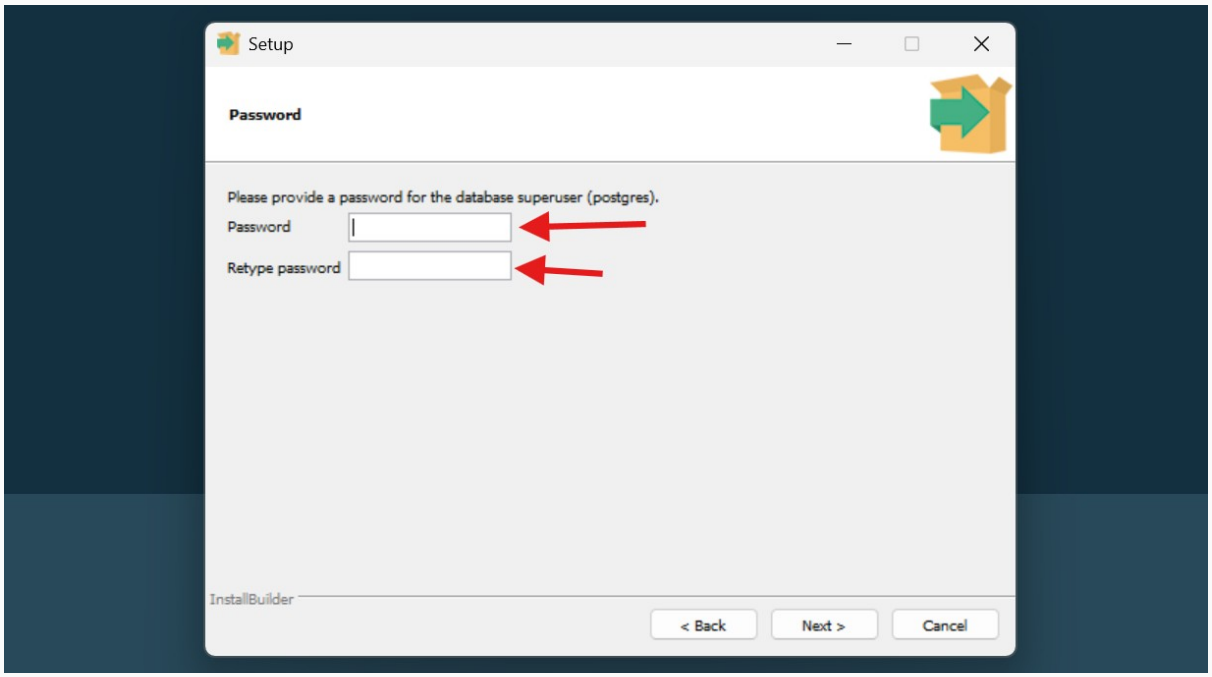
> **Bonne pratique pro**
> Prends l'habitude de figer tes dépendances dans un fichier. La commande ***pip freeze > requirements.txt*** enregistre la liste exacte des bibliothèques et de leurs versions. C'est ce fichier qui permettra, plus tard, de reconstruire le même environnement sur le serveur de déploiement ou sur l'ordinateur d'un collègue. On le fera proprement au fil du livre.

Étape 6 — Installer PostgreSQL et pgAdmin

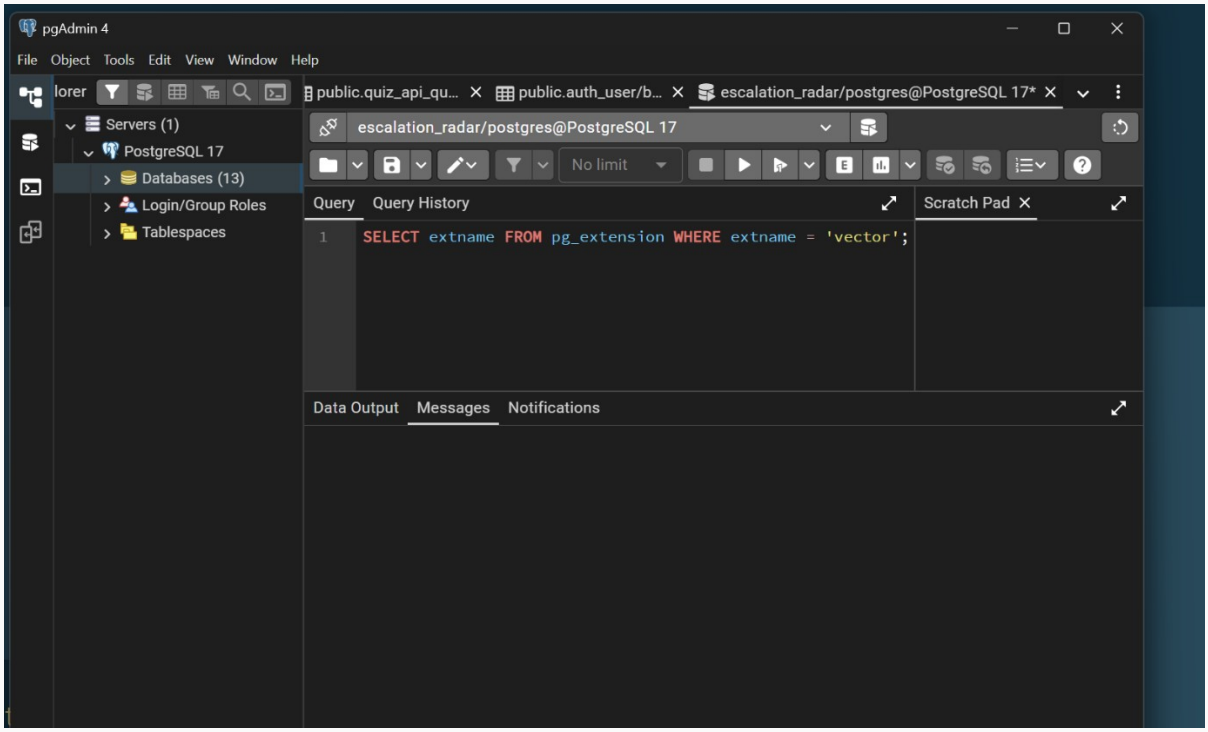
Notre application a besoin d'un endroit où ranger ses données : la base de données. On a choisi PostgreSQL, le standard des applications professionnelles. Il vient avec pgAdmin, une interface graphique bien pratique pour voir et gérer tes données sans taper de commandes.

1. Télécharge l'installateur sur *postgresql.org*, rubrique Download, en choisissant ton système.
2. Lance l'installation. Elle propose d'installer PostgreSQL et pgAdmin ensemble : garde les deux cochés.
3. À un moment, l'installateur te demande de choisir un mot de passe pour l'utilisateur administrateur postgres. Choisis-en un et note-le précieusement, on en aura besoin.
4. Garde le port par défaut, 5432. Retiens ce numéro, il reviendra dans notre configuration.

> ⚠ **Attention**
> Le mot de passe de l'utilisateur postgres te sera redemandé plus tard, notamment au chapitre suivant pour connecter Django à la base. Note-le quelque part de sûr dès maintenant. Beaucoup de débutants le choisissent à la va-vite pendant l'installation, puis passent une demi-heure à le chercher.



L'installateur PostgreSQL demandant de définir le mot de passe de l'utilisateur postgres.



La fenêtre principale de pgAdmin après l'installation, avec le serveur PostgreSQL local dans le panneau de gauche.

Étape 7 — Créer la base de données du projet

PostgreSQL est installé, mais il est vide. On va créer une base de données dédiée à notre BugTracker, ainsi qu'un utilisateur qui aura le droit d'y accéder. Pourquoi un utilisateur dédié plutôt que le tout-puissant compte postgres ? Parce qu'en production, on donne à chaque application juste les droits dont elle a besoin, pas plus. C'est un réflexe de sécurité de base, autant le prendre tout de suite.

Le plus simple est d'utiliser l'outil en ligne de commande de PostgreSQL, SQL shell psql. Ouvre-le et connecte-toi en tant que postgres avec le mot de passe choisi pendant l'installation.

```
# Se connecter à PostgreSQL (le mot de passe postgres sera demandé)
psql -U postgres
```

> **Astuce**
> Tu peux aussi utiliser la commande dans ton terminal VS Code ou terminal GitBash : *psql -U postgres*. Lors de l'utilisation de cette commande, si tu reçois un message d'erreur comme *bash: psql: command not found*, cela veut simplement dire que tu dois ajouter le chemin de postgres dans PATH de ton System de variables d'environnement. Exemple de chemin : « *C:\Program Files\PostgreSQL\17\bin* »

Une fois connecté, l'invite change et affiche postgres=#. Tape maintenant ces trois commandes SQL, une par une. Elles créent la base, l'utilisateur, et donnent à cet utilisateur les droits sur la base.

```
CREATE DATABASE bugtracker_db;
CREATE USER bug_user WITH PASSWORD 'motdepasse_a_changer';
GRANT ALL PRIVILEGES ON DATABASE bugtracker_db TO bug_user;
\q                          -- quitte psql
```

> **Astuce**
> Tu préfères le clic à la ligne de commande ? Tu peux faire exactement la même chose dans **pgAdmin** : clic droit sur **Databases** puis **Create** pour la base, et ***Login/Group Roles*** pour l'utilisateur. Le résultat est identique. Choisis la méthode dans laquelle tu es le plus à l'aise.

> **⚠ Attention**
> Un piège classique de PostgreSQL 15 et plus. Depuis cette version, le schéma public n'autorise plus la création de tables par défaut pour les utilisateurs qui n'en sont pas propriétaires. Ton ***GRANT ALL PRIVILEGES ON DATABASE bugtracker_db TO bug_user*** donne des droits sur la base, mais pas sur le schéma ***public*** à l'intérieur de cette base. Quand Django tente de créer sa table de migrations, tu recevras un message d'erreur « **permission denied for schema public** », pour éviter que cela nous arrive, la correction se fait en une fois, côté PostgreSQL.
>
> - Connecte-toi en tant que ***postgres*** et donne à ***bug_user*** la propriété de la base et du schéma : ***psql -U postgres***

➢ Puis, dans l'invite psql :

```
ALTER DATABASE bugtracker_db OWNER TO bug_user;
\c bugtracker_db
GRANT ALL ON SCHEMA public TO bug_user;
ALTER SCHEMA public OWNER TO bug_user;
\q
```

Comment fonctionne Django

Retiens ces quatre informations, elles forment la carte d'identité de connexion à ta base : le nom de la base (bugtracker_db), l'utilisateur (bug_user), son mot de passe, et le port (5432). Au chapitre 3, on donnera exactement ces valeurs à Django pour qu'il sache où et comment se connecter. On ne les écrira jamais en dur dans le code : elles iront dans un fichier de configuration à part.

Étape 8 — Mettre le projet sous contrôle de version avec Git

Dernière étape de préparation, et pas la moindre : Git. C'est l'outil qui va photographier ton code à chaque étape importante. Si une modification casse tout, tu reviens à la dernière photo qui marchait. Et c'est aussi ce qui te permettra de publier ton projet sur GitHub, ta vitrine de développeur.

Initialise Git dans le dossier du projet.

```
git init
```

Avant de photographier quoi que ce soit, on doit dire à Git ce qu'il ne doit surtout pas suivre : l'environnement virtuel, les fichiers temporaires de Python, et bientôt notre fichier de secrets. On crée pour cela un fichier nommé ***.gitignore*** à la racine du projet, avec ce contenu.

Tu peux également créer le fichier en utilisant le terminal de VS Code avec la commande suivante :

```
touch .gitignore
```

Dans ce fichier on ajoute tout ce que Git ne doit surtout pas suivre. Pour notre projet bugtracker avec django, il y a une astuce très utilisé pour s'assurer de rien oublier.

- Ouvre le lien suivant dans to navigateur web préfère : *https://www.toptal.com/developers/gitignore*
- Dans la barre de recherche tape : ***django***, puis presse la touche « ***Enter*** » de ton clavier
- Clique sur le bouton « ***Créer*** » ou « ***Create*** », selon la langue qui s'affiche.
- Copie/colle tout ce qui s'affiche dans ton fichier ***.gitignore***
- Tu auras au final quelque chose qui ressemble à ceci :

```
# Created by https://www.toptal.com/developers/gitignore/api/django
# Edit at https://www.toptal.com/developers/gitignore?templates=django

### Django ###
*.log
*.pot
*.pyc
__pycache__/
local_settings.py
db.sqlite3
db.sqlite3-journal
media
...
# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/
```

> **Astuce**
> Tu peux accéder directement à ce document en utilisant le lien suivant :
> ***https://www.toptal.com/developers/gitignore/api/django***

> **⚠ Attention**
> La ligne ***.env*** est capitale. Ce fichier contiendra bientôt tes mots de passe et clés secrètes. S'il se retrouve publié sur GitHub, n'importe qui peut lire tes secrets. L'ajouter au ***.gitignore*** dès maintenant, avant même de le créer, est le bon réflexe. On ne publie jamais de secrets, jamais.

On peut maintenant prendre la première photo de notre projet.

```
git add .
git commit -m "Preparation de l'environnement du projet"
git branch -M main
```

> **Ce qu'un recruteur regarde**
> Un recruteur qui ouvre ton dépôt GitHub regarde souvent l'historique des ***commits*** avant même le code. Des messages clairs et une progression régulière racontent ton sérieux et ta méthode de travail. À l'inverse, un unique commit intitulé projet fini envoie un très mauvais signal. Commite petit, commite souvent, et écris des messages qui décrivent ce que tu as fait.

Les erreurs fréquentes

Voici les trois pièges qui font perdre le plus de temps à ce stade. Les connaître, c'est déjà les éviter.

- Oublier d'activer l'environnement virtuel. Si une commande comme django-admin est introuvable, ton premier réflexe : regarde si (.venv) apparaît bien dans ton terminal. Sinon, réactive-le.
- Python absent du PATH sous Windows. Le message python n'est pas reconnu vient presque toujours de la case Add to PATH non cochée à l'installation. La solution est de réinstaller en la cochant.
- Se tromper de mot de passe PostgreSQL. Si Django n'arrive pas à se connecter au chapitre suivant, c'est neuf fois sur dix le mot de passe de bug_user ou de postgres qui ne correspond pas.

En entreprise

> **En entreprise**
> Le tout premier réflexe d'un développeur qui rejoint une équipe est de reproduire l'environnement du projet sur sa machine. Environnement virtuel, base de données locale, variables d'environnement, dépôt Git (Git repository): ce sont exactement les gestes que tu viens d'apprendre. Beaucoup d'équipes automatisent même cette préparation, notamment avec Docker, que nous verrons au moment du déploiement. Ce que tu fais ici à la main, tu comprendras plus tard comment l'industrialiser.

Ce que nous avons construit

Ton atelier est prêt. Tu as installé Python, configuré VS Code, créé une bulle isolée pour le projet et y as installé Django 6. Tu disposes d'une base de données PostgreSQL avec sa propre base bugtracker_db et son utilisateur dédié. Enfin, ton projet est sous Git avec un .gitignore correct et un premier commit. C'est une fondation propre et professionnelle.

Ton défi

Mets en pratique ce que tu viens d'apprendre pour bien l'ancrer.

1. Désactive puis réactive ton environnement virtuel. Pour le désactiver, tape simplement ***deactivate***. Observe le préfixe (***.venv***) disparaître, puis réactive-le.
2. Dans ***pgAdmin***, retrouve visuellement ta base ***bugtracker_db*** dans l'arborescence de gauche. Confirme qu'elle existe bien.
3. Crée un dépôt vide sur ton compte GitHub, puis relie-le à ton projet local. Indice : GitHub t'affiche les deux commandes à copier, ***git remote add origin*** … suivie de ***git push***. C'est ainsi que ton code arrive en ligne.

> **Astuce**
> Si le défi GitHub te bloque, ce n'est pas grave : nous reviendrons sur la publication du dépôt plus tard dans le livre. L'important pour l'instant est que ton environnement local fonctionne.

Prochaine étape

L'atelier est monté, les outils sont prêts. Au chapitre 3, on entre enfin dans le vif du sujet : on crée le projet Django, on met en place ses applications, et on le connecte à la base de données PostgreSQL que tu viens de préparer. À la fin du prochain chapitre, tu verras la fameuse page de bienvenue de Django s'afficher dans ton navigateur. Le projet prendra vie.

VOUS AVEZ LU L'EXTRAIT

La suite vous attend

Le livre complet compte 339 pages et vous mène jusqu'à la mise en ligne de votre application.

Ce que contient la version complète

- 34 chapitres, du modèle de données au déploiement
- Sécurité par rôles, API REST (DRF + JWT), tests et CI
- 7 bonus, dont la préparation à l'entretien technique
- Les captures d'écran en couleur, en PDF séparé
- Le code source complet du projet
- La Partie 9 sur l'IA, offerte par email

Essayez l'application avant d'acheter

https://bugtrackerapp.up.railway.app/ · Comptes de démonstration en un clic

Obtenir le livre complet

Téléchargement immédiat en PDF
Mises à jour gratuites

https://livrestek.lemonsqueezy.com/

10 % de réduction avec le code : CODELANCEMENT10

à saisir au moment du paiement