Génération de données, linters, git hooks, outils de build

Génération de données

La génération de données dans le développement apporte de nombreux avantages. Bien réalisée, elle permet de travailler sur une application dont les données et le volume ressemblent au produit en production. Elle est également indispensable pour les tests unitaires et fonctionnels.

Si le sujet est traité sérieusement et tôt dans le développement, il apporte un confort important et permet d'explorer des cas d'usage qui pourraient être oubliés avant la mise en production (encodage et caractères spéciaux, gestion des doublons, problèmes d'affichage en cas de données volumineuses, etc.).

Plus les données générées sont proches de la réalité, plus les tests seront fiables et plus les développeurs seront à l'aise avec l'application. Il ne faut pas hésiter à générer une grande variété et un volume élevé de données, y compris des cas dits « exotiques ».

Un autre avantage est la possibilité de mettre rapidement le projet en place en local, pour favoriser le travail collaboratif ou faire des démonstrations. Nous allons ici nous concentrer sur la génération de données avec PHP, en débutant par l'utilisation d'un framework dédié avant d'aborder son intégration dans un projet plus complet.

Pourquoi générer des données ?

- Rendre l'application plus réaliste pour les tests.
- Tester la performance, la robustesse, la gestion d'erreurs et les comportements.
- Détacher les environnements de test de la production.

À éviter

- L'utilisation répétée de valeurs comme "test" : peu réaliste et peu représentatif.

Peupler la bdd :

Dans un projet, il est important de peupler la base de données avec des données cohérentes. Plusieurs méthodes existent :

1. **Récupérer la base de production**
   + Données réelles, problèmes réels facilement reproduits
   - Il faut avoir accès à une base de production
   - Il faut avoir le droit de la récupérer **-** Risque de fuite de données confidentielles **Conclusion** : idéalement à éviter

2. **Construire un jeu de données statique**
   + Peut être modelé exactement pour reproduire des comportements souhaités **+** Facilité de partage via script d'import
   - Fastidieux à maintenir
   - Risque de ne pas couvrir tous les cas d'usage

- Ne contiendra jamais beaucoup de données
- Tous les utilisateurs couvrent les mêmes cas

Conclusion : démarche intéressante, mais limitée

3. **Prévoir une commande générant automatiquement des données**
   + Facile à écrire et maintenir
   + Script versionné ce qui permet de travailler sur des évolutions sans impacter la branche principale
   + Possibilité de générer rapidement un grand volume de données
   + Facilite les tests

   Conclusion : solution idéale à privilégier

L'idéale est donc de rédiger un script qui permet de générer des données fictives.

Symfony et Laravel proposent respectivement les **Datafixtures** et **Seeders** qui remplissent ce rôle : peupler la base via une simple commande. Si vous n'utilisez pas ces frameworks, il est recommandé de s'en inspirer.

Générer des données à utiliser comme on veut

FakerPHP est une **librairie puissante** qui simplifie considérablement la génération de données. Elle inclut des générateurs pour de nombreux types de données courantes (prénoms, adresses, mails, coordonnées, etc.).
Elle supporte également plusieurs localisations, ce qui permet d'obtenir des données adaptées (formats d'adresse et dates françaises, par exemple) et de gérer plusieurs pays dans une même application, avec des formats adaptés.

Linters

Les linters sont des outils d'analyse de code statique. Ils lisent votre code et, grâce à un ensemble de règles définies, signalent les corrections à apporter. Ces règles sont éditables et leur fichier de configuration est partageable.
Les sujets traités couvrent les erreurs de développement (variables non déclarées, oubli de point-virgule, etc.) ainsi que de nombreuses règles de bonnes pratiques (espacement, style, etc.).

Les avantages sont nombreux : le code devient plus "propre", plus lisible et cohérent, notamment si plusieurs personnes travaillent sur un projet.

Selon le linter et le langage, ils peuvent répondre à différents besoins :

- Analyse les types, les variables, les signatures et les retours de fonctions.
- Détecte les incohérences avant exécution.
- Vérifie le respect des standards
- Analyse la structure, l'indentation et les conventions.

GIT hooks

Les git hooks sont simplement des scripts bash exécutés à un moment précis (avant commit, avant push, etc.). Ils interrompent l'action si la commande retourne une erreur. Cela évite de pousser du code incorrect et permet (normalement) de réussir à chaque fois la CI s'il y en a une. Parmi les principaux hooks :

- `pre-commit` : avant l'enregistrement d'un commit
- `post-commit` : après le commit
- `pre-receive` / `post-receive` : avant/après un push
- `update` : avant/après la mise à jour d'un dépôt
- `pre-rebase`, `post-rebase` : avant/après un rebase

Placez ces scripts dans le dossier `.git/hooks` du dépôt. Vous pouvez adapter un exemple ou écrire votre propre hook.

Outils de build

Les outils de build JavaScript sont des logiciels qui transforment le code source (JavaScript, styles, ressources) en une version optimisée, prête à être exécutée dans le navigateur, en gérant par exemple la minification et la transpilation. Ils offrent également un environnement de développement confortable, avec serveur local, rechargement automatique des pages et parfois des outils d'analyse de code.