Comment éviter une injection SQL ou NoSQL ?

Pour éviter les injections SQL, j'utilise systématiquement les **requêtes préparées** via Eloquent ou Query Builder de Laravel. Exemple :
php
```
DB::table('users')->where('email', $email)->first();
```
Cela empêche l'interpréteur SQL d'exécuter du code malveillant injecté dans les paramètres.
Pour les bases NoSQL comme MongoDB, j'évite les requêtes dynamiques et je valide toujours les entrées avec des schémas stricts (ex: Mongoose avec Node.js).
En complément, j'applique une **validation côté serveur**, je filtre les entrées (`strip_tags`, `filter_var`) et je journalise les requêtes suspectes pour analyse.

Quelle est la différence entre hashing (bcrypt, Argon2) et encryption (AES) ?

Le hashing est irréversible et utilisé pour stocker des données sensibles comme les mots de passe.

bcrypt et Argon2 ajoutent du salage et sont conçus pour résister aux attaques par dictionnaire ou brute force.

On ne peut pas "déchiffrer" un hash — on compare avec Hash::check() dans Laravel.

L'encryption (comme AES) est réversible : on peut chiffrer et déchiffrer avec une clé.

Utilisé pour protéger des données qu'on doit pouvoir relire (ex: tokens, fichiers, messages).

Laravel propose Crypt::encrypt() et Crypt::decrypt() pour ça.

En résumé :

Hashing = stockage sécurisé, non réversible

Encryption = transport ou stockage temporaire, réversible avec clé

Comment protéger une API contre le brute force ?

Je protège mes endpoints sensibles (ex: `/login`) avec plusieurs stratégies :

- **Rate limiting** via middleware `throttle:5,1` dans Laravel → max 5 requêtes par minute par IP
- **Delai progressif** après plusieurs échecs (cooldown)
- **Captcha ou challenge** après X tentatives
- **Monitoring des logs** pour détecter des patterns anormaux

En production, j'ajoute un **reverse proxy** comme Nginx ou Cloudflare pour bloquer les IP abusives et limiter les attaques par bot.

Comment stocker les secrets (API keys, tokens) en production ?.

Je ne stocke jamais les secrets en dur dans le code.

- En Laravel, j'utilise le fichier `.env` pour les clés API, tokens, credentials.
- En production, ces variables sont injectées via le système d'environnement du serveur (Docker, CI/CD, etc.)
- J'applique des **permissions strictes** sur le fichier `.env` (`chmod 600`)
- Je chiffre les secrets sensibles si nécessaire avec `Vault`, `AWS Secrets Manager`, ou `Laravel Envoyer`

Enfin, je m'assure que le `.env` n'est jamais versionné (`.gitignore`) et que les accès sont limités aux rôles techniques autorisés.