### Synthèse des Refactorisations et Optimisations

#### 1. Activation du Mode Strict

* **Ajout de** declare(strict\_types=1); au début des fichiers pour activer le mode strict de PHP 7.1, assurant des types de données stricts et réduisant les erreurs de type.

#### 2. Injection de Dépendances

* **Utilisation de l'injection de dépendances via le constructeur** plutôt que d'utiliser des singletons, améliorant la testabilité et réduisant le couplage des composants.

|  |
| --- |
| public function \_\_construct(  QuoteRepository $quoteRepository,  SiteRepository $siteRepository,  DestinationRepository $destinationRepository,  ApplicationContext $applicationContext  ) {  // ...  } |

#### 3. Typage Strict

* **Ajout de types stricts** dans les signatures de méthode pour les paramètres et les valeurs de retour, augmentant la robustesse et la lisibilité du code.

#### 4. Séparation des Responsabilités

* **Découpage des méthodes complexes en méthodes plus petites et spécifiques** pour rendre le code plus lisible et plus facile à maintenir. Par exemple, la méthode computeText() a été divisée en plusieurs méthodes spécifiques.

|  |
| --- |
| private function replaceQuoteVariables(string $text, Quote $quote): string  private function replaceUserVariables(string $text, User $user): string |

#### 5. Optimisation des Remplacements de Texte

* **Utilisation de** strtr() **pour les remplacements multiples** au lieu de plusieurs appels à str\_replace(), ce qui améliore l'efficacité.

|  |
| --- |
| $replacements = [  '[quote:destination\_name]' => $destination->countryName,  // ...  ];  return strtr($text, $replacements); |

#### 6. Simplification des Conditions

* **Utilisation de l'opérateur de coalescence nulle (??)** et de vérifications de type plus concises pour simplifier la logique conditionnelle.

|  |
| --- |
| $quote = $data['quote'] ?? null;  if ($quote instanceof Quote) {  // ...  } |

#### 7. Suppression de Code Mort et Variables Non Utilisées

* **Suppression des variables non utilisées et des conditions redondantes**, améliorant la clarté et la maintenabilité du code.

#### 8. Utilisation de Méthodes Privées

* **Extraction de la logique complexe dans des méthodes privées**, comme la création du lien de destination, pour améliorer la lisibilité et la maintenabilité.

#### 9. Méthodes Setter Retournant self

* **Modification des méthodes setter pour retourner** self, permettant le chaînage des méthodes et rendant le code plus fluide et lisible.

#### 10. Cohérence et Standardisation des Noms de Variables

* **Standardisation des noms de variables** pour plus de clarté et de conformité avec les principes de clean code (par exemple, renommer $\_quoteFromRepository en $quoteFromRepository).

#### 11. Utilisation de use et Importations Explicites

* **Utilisation de** use **pour les classes** comme DateTime et autres dépendances, évitant les utilisations répétées de backslashes (\) et améliorant la lisibilité.
* **Importations explicites de classes** comme Faker\Factory et Faker\Generator, simplifiant le code et rendant les dépendances plus claires.

### Bénéfices de ces Modifications

Ces modifications rendent le code :

* **Plus propre et lisible** : Suppression des éléments redondants, réorganisation des méthodes et standardisation des noms de variables.
* **Plus robuste et sûr** : Utilisation du typage strict et encapsulation appropriée avec des getters et setters.
* **Plus flexible et maintenable** : Introduction de l'injection de dépendances et des mocks, permettant une meilleure isolation des tests et une gestion plus facile des dépendances.
* **Conforme aux bonnes pratiques modernes de PHP**, facilitant la compréhension et la maintenance par d'autres développeurs.

Ces changements améliorent considérablement la qualité, la lisibilité et la maintenabilité. Ils rendent le code plus modulaire, plus facile à tester et à maintenir, tout en améliorant potentiellement les performances et en facilitant l'ajout de nouvelles fonctionnalités ou la modification de la logique existante.