Documentation sur le Green Code

Aujourd'hui, alors que le changement climatique est une préoccupation pressante, notre responsabilité à tous d'adopter des pratiques durables dans nos activités n'a jamais été aussi grande.

Concernant l'application aujourd'hui livré, voici quelques points expliquant comment cette dernière peut devenir un logiciel durable et se plier aux exigences du green code.

1. **Écrire du code propre**

Un code optimisé consomme moins de ressources CPU, ce qui se traduit par une réduction de la consommation d'énergie et des émissions. Cela implique le maintien des pratiques telles que la minimisation des calculs redondants et de la logique récursive, l'utilisation d'algorithmes appropriés et des classes optimisées, la revue de code pour éliminer les erreurs comme la création de boucles infinies et des bugs.
Une bonne ressource pour en apprendre plus est "Clean Code", de Robert C. Martin Series.

2. **Optimisation du Cloud Computing décentralisé**

Afin d'optimiser l'utilisation des ressources et de coûteux investissements en matériel, il faudrait que l'application repose sur un système de cloud computing sans serveur. En effet, le service de cloud décentralisé permet de distribuer de manière dynamique de l'espace de stockage suivant des algorithmes de balance de charge, de distribution géographique etc. Chaque nœud utilise un logiciel permettant de mettre à disposition de l'espace de stockage. Par exemple, utiliser des services comme AWS Lambda présente des avantages considérables en termes d'économie, de rapidité, de disponibilité et de flexibilité.

3. **Modèles de Conception Sensibles à l'Énergie**

Intégrez des modèles de conception sensibles à l'énergie dans votre architecture logicielle. Par exemple, le modèle de chargement différé (lazy-loading) peut être utilisé pour charger des ressources uniquement lorsque cela est nécessaire, réduisant ainsi la consommation d'énergie inutile. De même, l'optimisation des requêtes de base de données et l'utilisation de techniques de mise en cache peuvent minimiser le besoin de récupération de données répétée. Ces pratiques améliorent l'expérience de l'utilisateur et se traduisent par une utilisation plus efficace des ressources. Par exemple, en utilisant des frameworks comme Hibernate pour gérer efficacement les requêtes de base de données.

4. **Hébergement Vert et Centres de Données**

Choisissez des fournisseurs d'hébergement et des centres de données qui privilégient la durabilité. Recherchez des fournisseurs qui utilisent des sources d'énergie renouvelable, mettent en œuvre des systèmes de refroidissement énergétiquement efficaces et s'engagent à réduire leur impact environnemental. Par exemple, choisir des fournisseurs de cloud comme Google Cloud ou Microsoft Azure, qui ont des engagements forts en matière de durabilité.

5. **Surveillance et Optimisation des Performances**

Afin de connaître les performances de l’application, il est nécessaire d’effectuer de vrais benchmarks et de surveiller la JVM. En effet pouvoir cibler le code qui effectue une surutilisation de la mémoire et des ressources CPU peut nous aider à le supprimer ou le modifier en vue d’optimiser les performances. Pour ce faire, une page web de surveillance ou une API ou l’outillage JVisualVM sont idéales. Par exemple, utiliser des outils de monitoring comme New Relic pour surveiller en temps réel les performances de l'application.

6. **Gestion du Cycle de Vie du Logiciel**

Il faudrait mettre en œuvre des systèmes de contrôle de version et des tests automatisés du logiciel pour garantir des déploiements et des mises à jour sans heurts, réduisant ainsi la probabilité d'erreurs pouvant entraîner une augmentation de la consommation d'énergie. Par exemple, vous pouvez utiliser des outils de CI/CD comme Jenkins pour automatiser les tests et les déploiements.