Plan de test : Extrait

Introduction :

Ce plan détaillera notre stratégie de test, y compris les tests qui devraient être réalisés, l'environnement de test où ces tests seront effectués, et les cas de test spécifiques qui seront réalisés dans le cadre du projet. En suivant ce plan, nous visons à assurer que notre produit atteint les normes de qualité les plus élevées et répond aux besoins de nos utilisateurs.

Rôle et Responsabilités	Deux développeurs logiciels chargés de créer le programme et de rédiger les tests unitaires. Nous avons également un responsable qualité qui se charge du développement des tests et de leur exécution. Les deux développeurs sont là pour apporter leur aide en cas de besoin.
Stratégie de test	Pour garantir que chaque partie de l'application est correctement testée, il est crucial d'adopter une stratégie ordonnée et efficace. Les tests unitaires, tant avec TestCase dans test_models.py que les fixitures pytest dans test_models.py avec fixitures, doivent être exécutés en premier pour vérifier le bon fonctionnement isolé des modèles, surtout après chaque modification de code source. Ensuite, les tests d'URLs dans test_urls.py s'assurent que les routes mappent correctement aux vues, particulièrement après des changements dans les fichiers d'URLs ou de vues. Les tests d'intégration dans test_search.py suivent pour vérifier que les différentes parties de l'application fonctionnent bien ensemble après des modifications ou ajout de fonctionnalités, de vues, modèles ou templates. Enfin, les tests avec mocks dans test_views.py permettent de simuler des réponses externes, utiles pour tester des comportements spécifiques des vues. Pour chaque type de modification – que ce soit des modèles, des vues, ou des routes – les tests appropriés doivent être exécutés en conséquence, et lors des régressions ou déploiements, l'ensemble des tests doit être exécuté pour s'assurer de la stabilité de l'application.
Environnement de test :	L'environnement de test utilise Django et pytest-django pour exécuter des tests unitaires, d'intégration, et des tests avec mocks. Cette configuration permet d'assurer que chaque composant de l'application fonctionne correctement individuellement et ensemble.

Cas de test	Dans test_models.py, les tests unitaires sont essentiels après chaque modification dans les modèles, afin de vérifier que les attributs, méthodes et relations entre les modèles fonctionnent comme prévu. Dans test_urls.py, les tests d'URLs sont pertinents après des changements dans les fichiers d'URLs ou de vues pour garantir que les routes sont correctement configurées et mappées. Les tests d'intégration dans test_search.py sont cruciaux après des modifications dans les vues, les modèles ou les templates, afin de valider le bon fonctionnement global de l'application. Enfin, dans test_views.py, les tests avec mocks sont utiles pour simuler des réponses de la base de données ou d'autres services externes pour tester des scénarios spécifiques, comme les erreurs de base de données ou les réponses inattendues de services externes. En exécutant ces tests dans les scénarios appropriés, on garantit que l'application fonctionne correctement et que les modifications apportées n'ont pas introduit de régressions.