Plan Technique

1. Introduction

Le projet "Dream Habitat" redéfinit la décoration intérieure. Capturez l'essence de votre style en utilisant une photo, et laissez notre IA fusionner créativité et tendances. Personnalisez chaque détail pour créer un chez-vous en constante évolution. Ce document technique décrit les technologies utilisées, l'architecture du système, les processus de déploiement, la stratégie de sécurité, le plan de tests, les outils de surveillance et de journalisation, ainsi que la gestion de la documentation.

2. Technologies Utilisées

Backend :

- **Spring Boot FRAMEWORK Java:**
  - Raison du choix :

    Spring Boot permet de gérer un grand nombre de connexions simultanées de manière efficace grâce à son architecture non-bloquante.

    Spring Boot simplifie la gestion des routes et des middlewares, offrant une grande flexibilité pour développer des API robustes et performantes.

Frontend :

- **React.js, Redux, et Axios :**
  - Raison du choix :

    React.js est choisi pour sa capacité à créer des interfaces utilisateur dynamiques et réactives.
    Redux est utilisé pour la gestion de l'état global de l'application, ce qui facilite le développement de fonctionnalités complexes et la gestion des données.
    Axios est préféré pour les requêtes HTTP en raison de sa simplicité d'utilisation et de ses fonctionnalités avancées comme l'interception des requêtes et des réponses.

Mobile:

- **React Native et Expo:**
  - Raison du choix :

    Expo simplifie le développement et permet de tester rapidement à partir de n'importe quel device sans configurer Xcode ou Android Studio.

API :

- **Postman et Swagger :**
  - Raison du choix :

    Postman est utilisé pour tester et documenter les API grâce à son interface conviviale et ses capacités de création de collections de tests.

    Swagger est employé pour générer automatiquement la documentation des API, assurant une cohérence et une mise à jour continue de la documentation.

Documentation :

- **Swagger :**
  - Raison du choix :

    Swagger permet une intégration directe avec les spécifications des endpoints, générant une documentation claire et accessible pour les développeurs.

Partie Mobile:

- **Interface Utilisateur :** Developpé avec React Native pour offrir une expérience Cross platform fluide et interactive. l'application est testé grâce à Expo.

DevOps :

- **Docker et GitLab :**
  - Raison du choix :

    Docker permet d'isoler les environnements de développement, de test et de production, assurant une portabilité et une cohérence des déploiements.

    GitLab est choisi pour la gestion du code source et la mise en place de pipelines CI/CD robustes, facilitant l'intégration continue et le déploiement continu.

Communication :

- **Jira:**
  - Raison du choix :

    Jira est utilisé pour les communications en équipe, assurant une collaboration fluide et efficace.

Maquette :

- **Figma :**
  - Raison du choix :

    Figma est utilisé pour la conception des maquettes de l'application, offrant des outils de collaboration en temps réel pour les designers.

Documents :

- **Microsoft Word et Excel :**
  - Raison du choix **:**

    Microsoft Word et Excel sont utilisés pour générer les documents et les feuilles de calcul nécessaires au projet, facilitant la documentation et le reporting.

3. Architecture du Système

Partie Backend :

- **Structure :** Serveur spring boot pour offrir une structure scalable et performante pour gérer les requêtes et les opérations de l'API.

Partie Frontend :

- **Interface Utilisateur :** Construite avec React.js pour offrir une expérience utilisateur fluide et interactive. Redux gère l'état de l'application de manière centralisée, et Axios permet des interactions efficaces avec l'API.

Partie API :

- **Documentation et Tests :** Les API RESTful sont testées et documentées avec Postman et Swagger, assurant une communication claire et des tests rigoureux.

Partie DevOps :

- **Gestion des Environnements :** Les environnements sont conteneurisés avec Docker, et les pipelines CI/CD sont gérés avec GitLab, permettant des déploiements continus et une intégration fluide des modifications.

4. Déploiement et Gestion des Environnements

Conteneurisation avec Docker :

- **Explication :** Docker isole les applications et leurs dépendances, garantissant que le logiciel fonctionne de manière cohérente quel que soit l'environnement. Cela permet de simplifier le déploiement et la gestion des versions.

CI/CD avec GitLab :

- **Explication :** GitLab CI/CD automatise les tests et les déploiements, réduisant le temps entre les modifications de code et leur mise en production. Cela assure une livraison rapide et fiable des fonctionnalités.

5. Stratégie de Sécurité

Sécurité des Données :

- **Explication :** Le chiffrement des données sensibles protège contre les accès non autorisés. L'utilisation de JWT pour l'authentification et l'autorisation sécurise les communications entre le client et le serveur.

Sécurité des API :

- **Explication :** La validation et le nettoyage des entrées utilisateur préviennent les attaques d'injection. L'utilisation de tokens CSRF protège contre les attaques de falsification de requêtes

intersites.

Surveillance et Réaction :

- **Explication :** Des systèmes de surveillance détectent les comportements anormaux et des audits réguliers identifient les vulnérabilités potentielles, permettant une réaction rapide et efficace.

6. Plan de Tests

Tests Unitaires :

- **Explication :** Les tests unitaires vérifient le bon fonctionnement des composants individuels, assurant une base solide pour l'application.

Tests d'Intégration :

- **Explication :** Les tests d'intégration s'assurent que les différents composants du système fonctionnent bien ensemble, identifiant les problèmes d'interaction.

Tests de Performance :

- **Explication :** Les tests de performance, réalisés avec Postman, évaluent la capacité du système à gérer des charges élevées, garantissant la réactivité de l'application.

Tests de Sécurité :

- **Explication :** Les tests de sécurité identifient et corrigent les vulnérabilités, assurant que l'application est protégée contre les menaces.

7. Outils de Surveillance et de Journalisation

Surveillance :

- **Explication :** La stack ELK (Elasticsearch, Logstash, Kibana) collecte et analyse les logs, offrant une vue détaillée des opérations du système. Prometheus surveille les performances en temps réel, permettant une détection rapide des problèmes.

Journalisation :

- **Explication :** La journalisation détaillée permet de suivre les événements du système, facilitant l'audit et la résolution des problèmes.

8. Gestion de la Documentation

Documentation API :

- **Explication :** Swagger génère automatiquement la documentation des API, assurant une mise à jour continue et une référence claire pour les développeurs.

Documentation Technique :

- **Explication :** La documentation technique couvre le code source et les configurations système, offrant une ressource complète pour le développement et la maintenance.