**Cahier des Charges Technique (CDCT) - NexaCRM**

**1. Introduction**

**Présentation du projet et de son contexte**

Le projet NexaCore est une plateforme SaaS développée par NexaCRM, spécialisée dans la gestion de la relation client (CRM) assistée par l’intelligence artificielle. Cette solution est conçue pour répondre aux besoins des PME, ETI et startups en forte croissance, en leur offrant des outils de gestion et d’analyse avancés pour optimiser leurs interactions clients.

NexaCore se distingue par sa capacité à intégrer des fonctionnalités d’automatisation marketing, de segmentation comportementale et d’intelligence conversationnelle. Son architecture repose sur des technologies modernes et scalables, assurant une expérience utilisateur performante et sécurisée. Le projet répond également aux exigences réglementaires, notamment en matière de conformité RGPD.

Face à une concurrence accrue sur le marché des CRM, NexaCRM cherche à offrir une solution différenciée, facile à intégrer et capable d’évoluer avec les besoins des utilisateurs. Le présent CDCT définit les choix techniques et les contraintes nécessaires pour garantir la réussite du projet.

**Objectifs du projet et enjeux techniques**

L’objectif principal de NexaCore est de fournir une plateforme évolutive, hautement performante et conforme aux standards de sécurité. Elle doit permettre aux entreprises d’améliorer leur gestion des ventes, leurs campagnes marketing et leur support client tout en exploitant des capacités avancées d’analyse de données.

Les enjeux techniques du projet sont multiples. Premièrement, il s’agit de garantir la scalabilité et la haute disponibilité de la plateforme pour gérer des volumes importants de données et d’utilisateurs simultanés. Deuxièmement, l’intégration avec des systèmes tiers tels que les ERP et les outils BI est essentielle pour assurer l’interopérabilité. Troisièmement, la sécurité des données est une priorité, avec des exigences strictes en matière de chiffrement, d’authentification et d’audits réguliers.

Enfin, le projet vise à maintenir une architecture modulaire basée sur des microservices, facilitant l’ajout de nouvelles fonctionnalités et réduisant les temps d’arrêt lors des mises à jour. Le déploiement automatisé et les tests continus garantiront une qualité et une fiabilité optimales tout au long du cycle de vie du produit.

**2. Architecture et Solutions Techniques**

**Description des solutions architecturales envisagées**

La plateforme NexaCore repose sur une architecture distribuée basée sur des microservices. Ce modèle modulaire permet d’assurer une évolutivité, une résilience et une maintenance simplifiée. Chaque microservice est conçu pour remplir une fonction spécifique, facilitant ainsi les mises à jour indépendantes et la gestion des pannes.

Le système adopte un modèle client-serveur avec une interface utilisateur développée en React.js côté client et un backend construit sur Node.js avec Express côté serveur. Le frontend communique avec le backend via des API RESTful et GraphQL, garantissant une flexibilité et une performance optimales pour les échanges de données.

La gestion des données repose sur un double système de bases de données. PostgreSQL est utilisé pour les informations relationnelles structurées telles que les utilisateurs et les opportunités commerciales. MongoDB est intégré pour stocker des données semi-structurées ou non structurées comme les interactions des utilisateurs et les journaux d’activités. Cette combinaison permet une gestion des données adaptée aux différents besoins fonctionnels de la plateforme.

Pour assurer la communication entre les microservices, la plateforme utilise Apache Kafka comme middleware. Il permet de gérer efficacement les flux de données, garantissant la fiabilité et la scalabilité des traitements asynchrones.

L’infrastructure est hébergée sur AWS, utilisant des conteneurs Docker orchestrés par Kubernetes. Ce choix garantit la portabilité des services, l’automatisation des déploiements et une gestion optimisée des ressources en fonction de la charge.

**Justification de chaque choix architectural**

Le choix d’une architecture microservices repose sur la nécessité de fournir une solution évolutive et résiliente. Ce modèle facilite la scalabilité horizontale, permettant d’ajouter ou de modifier des services sans perturber l’ensemble du système. En cas de panne, seuls les services concernés sont affectés, limitant ainsi l’impact sur les utilisateurs.

React.js a été retenu pour le développement du frontend en raison de sa réactivité et de sa capacité à créer des interfaces dynamiques et interactives. Il garantit une expérience utilisateur fluide et personnalisable, essentielle pour un CRM.

Node.js avec Express a été sélectionné pour le backend en raison de sa gestion asynchrone des requêtes, offrant une faible latence et une excellente performance. Ce choix s’intègre également parfaitement avec GraphQL, permettant une gestion des requêtes flexible et optimisée.

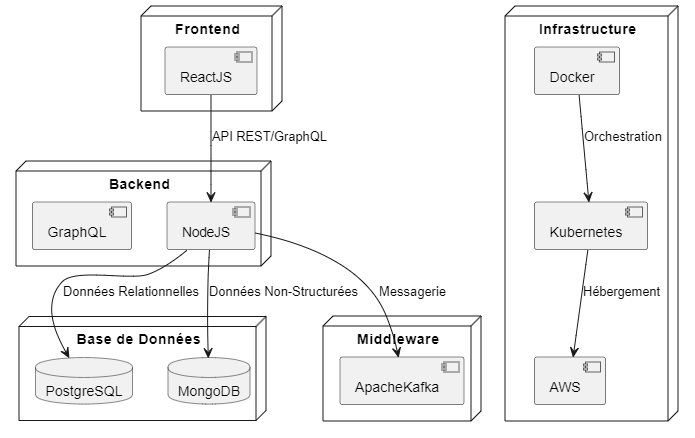
La combinaison de PostgreSQL et MongoDB offre une approche hybride pour la gestion des données. PostgreSQL garantit la cohérence des données structurées, tandis que MongoDB assure une flexibilité pour les données non structurées et semi-structurées, adaptées aux besoins analytiques et aux historiques d’interactions.

Apache Kafka a été choisi pour sa capacité à gérer des volumes importants de données et à orchestrer des flux complexes entre microservices. Il joue un rôle clé dans la gestion des événements et des messages asynchrones.

Docker et Kubernetes sont utilisés pour garantir une gestion automatisée des conteneurs et des déploiements. Cette solution assure la portabilité des services et une adaptation rapide à la demande, essentielle pour une plateforme SaaS évolutive.

L’hébergement sur AWS offre des services fiables et sécurisés, avec des options d’évolutivité automatique et de gestion optimisée des coûts.

**Schéma illustrant l’architecture globale**



**3. Implantations Techniques**

**Détails sur les technologies utilisées**

NexaCore repose sur un ensemble de technologies modernes et éprouvées pour garantir performance, évolutivité et sécurité. Le frontend est développé en React.js, un framework JavaScript largement adopté pour la création d’interfaces utilisateur dynamiques et réactives. Il est complété par Redux pour la gestion des états globaux, assurant une synchronisation fluide des données.

Le backend utilise Node.js avec Express, une solution performante pour gérer les API et traiter les requêtes de manière asynchrone. L’intégration de GraphQL permet une gestion optimisée des requêtes complexes en offrant aux clients la possibilité de récupérer uniquement les données nécessaires.

Les bases de données adoptent une approche hybride avec PostgreSQL pour les données relationnelles et MongoDB pour les données semi-structurées. Cette combinaison garantit une flexibilité dans la gestion des informations, qu’elles soient structurées ou non.

Pour la communication entre microservices, NexaCore intègre Apache Kafka en tant que middleware. Il facilite la gestion des événements et des flux de données en temps réel tout en assurant une haute disponibilité.

L’infrastructure est déployée sur AWS, en utilisant Docker pour la conteneurisation et Kubernetes pour l’orchestration des conteneurs. Ce choix technologique permet de gérer automatiquement les ressources, d’assurer la scalabilité et de faciliter la gestion des déploiements.

**Justification des choix techniques**

React.js a été sélectionné pour sa capacité à créer des interfaces rapides et interactives, tout en offrant une gestion efficace des composants réutilisables. Son intégration avec Redux permet de simplifier la gestion des états complexes, indispensable pour une application CRM riche en fonctionnalités.

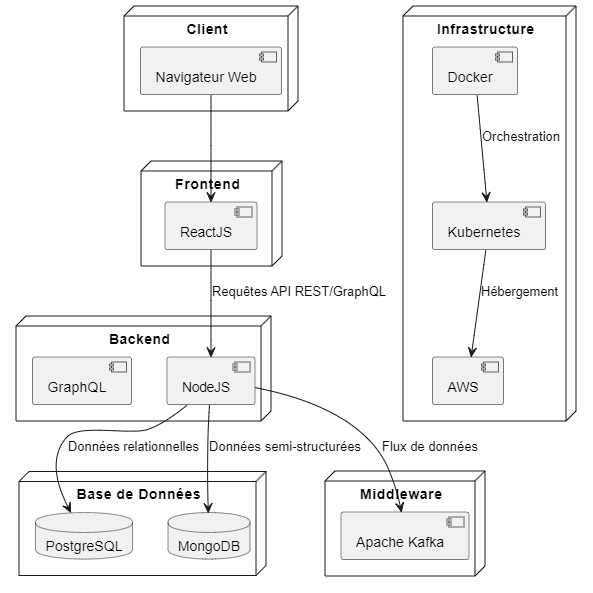
Node.js et Express sont utilisés pour leurs performances élevées et leur prise en charge de requêtes asynchrones. Ce choix est particulièrement adapté aux API modernes et aux applications nécessitant des interactions en temps réel. GraphQL a été ajouté pour offrir une flexibilité supplémentaire, permettant aux clients de structurer leurs requêtes de manière précise et efficace.

Le choix de PostgreSQL garantit la cohérence et l’intégrité des données relationnelles critiques, tandis que MongoDB offre la flexibilité nécessaire pour les données semi-structurées comme les journaux d’activité ou les interactions des utilisateurs. Cette approche mixte répond aux besoins variés de stockage et d’analyse de données.

Apache Kafka a été retenu pour sa capacité à gérer des flux de données complexes et volumineux, tout en garantissant la fiabilité et la persistance des messages échangés entre microservices. Cette technologie facilite l’évolutivité et la tolérance aux pannes.

L’utilisation de Docker et Kubernetes pour la gestion des conteneurs et des déploiements assure une infrastructure scalable, avec des déploiements automatisés et une gestion simplifiée des ressources. AWS offre une infrastructure robuste avec des services de sécurité intégrés et une flexibilité dans la gestion des coûts.

**Diagramme de déploiement**



**4. Dictionnaire de Données (selon SQL ou NoSQL)**

**Liste des entités (tables) de la base de données**

**PostgreSQL (SQL - Données Relationnelles)**

1. Utilisateurs : Gère les informations des utilisateurs de la plateforme.
2. Contacts : Stocke les informations des prospects et clients.
3. Opportunités : Suivi des opportunités commerciales.
4. Campagnes : Gestion des campagnes marketing.
5. Tickets : Suivi des demandes de support client.
6. AuditLogs : Journal des activités pour la conformité et la sécurité.

**MongoDB (NoSQL - Données Semi-Structurées)**

1. Interactions : Historique des échanges avec les utilisateurs (emails, appels).
2. ChatbotLogs : Enregistrements des conversations avec le chatbot IA.
3. PerformancesCampagnes : Données analytiques pour mesurer les résultats marketing.
4. LogsSysteme : Informations sur les erreurs et les journaux système.

**Description des entités**

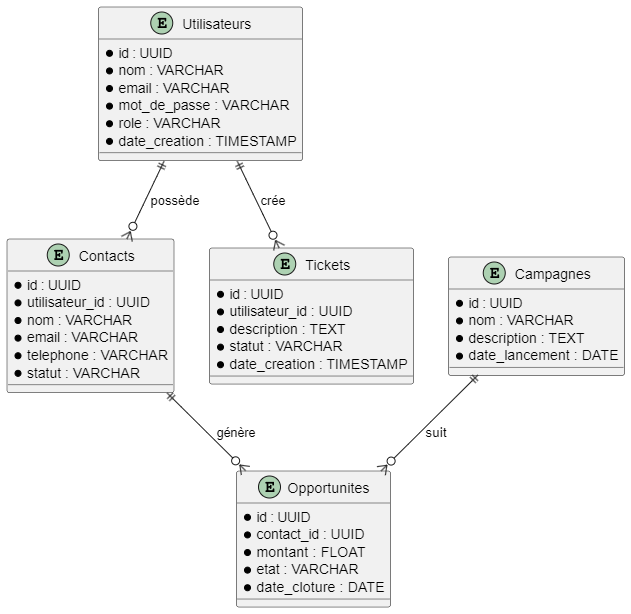
**Table : Utilisateurs (PostgreSQL)**

* id (UUID, clé primaire) : Identifiant unique de l’utilisateur.
* nom (VARCHAR) : Nom complet.
* email (VARCHAR, unique) : Adresse email de connexion.
* mot\_de\_passe (VARCHAR) : Mot de passe hashé.
* role (VARCHAR) : Rôle dans la plateforme (admin, commercial, support).
* date\_creation (TIMESTAMP) : Date de création du compte.

**Collection : Interactions (MongoDB)**

* \_id (ObjectID) : Identifiant unique.
* utilisateur\_id (UUID) : Référence à l’utilisateur concerné.
* type\_interaction (STRING) : Type d’interaction (email, appel, chatbot).
* contenu (TEXT) : Contenu du message ou de l’appel.
* horodatage (DATE) : Date et heure de l’interaction.

**Schéma ER (Entité-Relation)**



**5. Arborescence de Fonctionnalités**

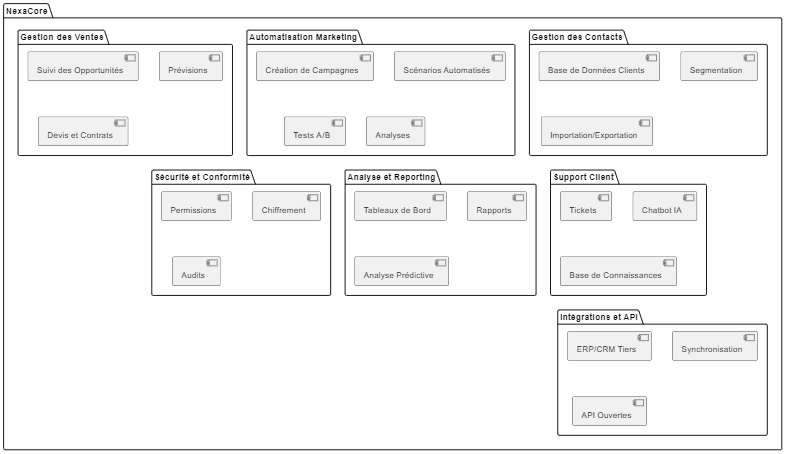
**Hiérarchie des Fonctionnalités**

La plateforme NexaCore propose une organisation hiérarchique des fonctionnalités, structurée autour des modules principaux et de leurs sous-fonctionnalités. Cette organisation permet d’optimiser l’accès aux différentes fonctionnalités tout en garantissant une navigation intuitive pour les utilisateurs.

**Niveau 1 : Modules Principaux**

1. **Gestion des Contacts et Prospects**
   * Centralisation des informations clients.
   * Historique des interactions.
   * Segmentation dynamique.
   * Importation et exportation des données.
2. **Automatisation Marketing et Campagnes**
   * Création de campagnes.
   * Planification et scénarios automatisés.
   * Tests A/B et analyses de performance.
3. **Gestion des Ventes et Pipeline Commercial**
   * Suivi des opportunités.
   * Prévisions de ventes.
   * Gestion des devis et contrats.
4. **Support Client et Service Après-Vente**
   * Gestion des tickets.
   * Chatbot IA.
   * Base de connaissances et enquêtes de satisfaction.
5. **Analyse et Reporting**
   * Tableaux de bord interactifs.
   * Rapports personnalisés.
   * Analyse prédictive et sentimentale.
6. **Sécurité et Conformité (NexaGuard)**
   * Gestion des accès et permissions.
   * Chiffrement des données.
   * Journaux d’audit et alertes.
7. **Intégrations et API**
   * Connecteurs ERP et CRM tiers.
   * Synchronisation bidirectionnelle.
   * API ouvertes pour développements personnalisés.

**Organigramme des Fonctionnalités**

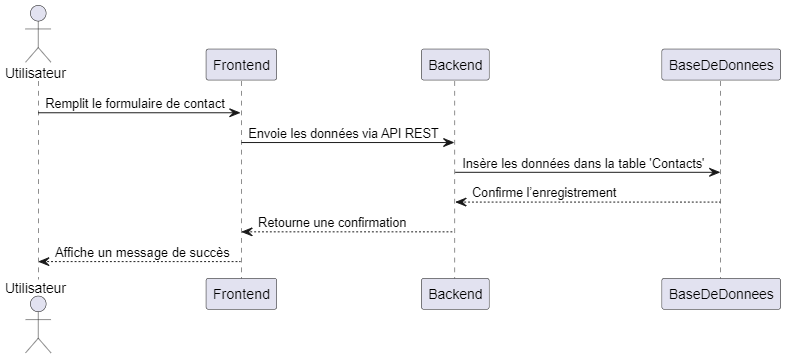


**6. Transitions et Flux de Données**

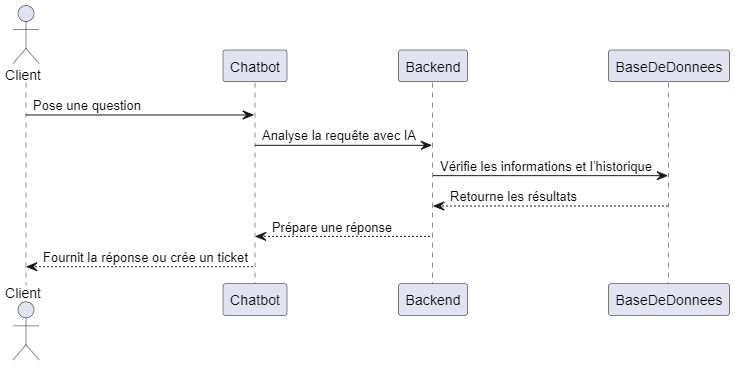
**Diagrammes de Séquence**

Ces diagrammes permettent de visualiser les interactions dynamiques et d’identifier les dépendances critiques.

**Exemple : Création d’un Contact**

****

**Exemple : Gestion des Tickets avec Chatbot IA**

****

**Flux de Données entre les Modules**

La gestion des flux de données repose sur des échanges asynchrones et synchrones entre les composants, optimisés par des microservices et des API RESTful/GraphQL. Les flux sont orchestrés par Apache Kafka pour la gestion des événements en temps réel et garantissent la cohérence des données entre les modules.

**Description des Flux**

1. **Gestion des Contacts** : Les informations sont envoyées depuis le frontend vers le backend et enregistrées dans PostgreSQL. Des événements sont déclenchés pour notifier les autres modules en cas de modification.
2. **Automatisation Marketing** : Les workflows automatisés utilisent des requêtes GraphQL pour récupérer des segments de contacts et planifier des actions spécifiques.
3. **Suivi des Opportunités** : Les mises à jour des opportunités sont suivies en temps réel et synchronisées avec les rapports et les prévisions via Kafka.
4. **Support Client** : Les tickets et les interactions du chatbot sont enregistrés dans MongoDB et synchronisés avec le backend pour les réponses automatisées et les escalades.
5. **Analyse et Reporting** : Les données collectées sont envoyées vers les tableaux de bord interactifs en utilisant des API pour générer des visualisations en temps réel.

**7. Déploiement**

**Stratégie de Déploiement**

La stratégie de déploiement de NexaCore repose sur un processus automatisé utilisant l’intégration et le déploiement continus (CI/CD). Ce modèle garantit des livraisons rapides et fiables, tout en minimisant les risques d’erreurs humaines. L’infrastructure est basée sur des conteneurs Docker orchestrés par Kubernetes, offrant une gestion flexible et évolutive.

Le déploiement se déroule en plusieurs étapes :

1. Préproduction : Les nouvelles versions sont d’abord déployées dans un environnement de préproduction pour des tests approfondis.
2. Validation : Des tests automatisés sont exécutés pour vérifier la stabilité, les performances et la conformité.
3. Production : Une fois validée, la version est déployée progressivement en production à l’aide de stratégies comme le déploiement bleu/vert et les déploiements canary.
4. Surveillance Post-Déploiement : Une surveillance en temps réel est activée pour détecter les anomalies et garantir une performance optimale dès le lancement.

**Outils Utilisés**

* GitLab CI/CD : Automatisation des pipelines d’intégration et de déploiement.
* Docker : Conteneurisation des services pour un déploiement reproductible et isolé.
* Kubernetes : Orchestration et gestion des conteneurs pour assurer la scalabilité et l’automatisation.
* Prometheus : Surveillance et alertes en temps réel.
* Grafana : Visualisation des métriques et des performances.

**Procédures de Roll-Back**

En cas de problème lors du déploiement, des procédures de retour en arrière (roll-back) sont prévues pour rétablir rapidement une version stable.

1. Détection des erreurs : Les erreurs sont identifiées grâce à des tests automatisés et à la surveillance des journaux en temps réel.
2. Activation de la version précédente : Kubernetes permet de restaurer immédiatement la version précédente des conteneurs en cas de défaillance.
3. Isolation des problèmes : Les services problématiques peuvent être désactivés sans affecter l’ensemble du système, grâce à l’architecture microservices.
4. Validation post-roll-back : Après restauration, des tests sont effectués pour vérifier la stabilité et la fonctionnalité complète de la plateforme.
5. Analyse des causes : Un rapport est généré pour identifier les causes de l’incident et éviter sa répétition.

**8. Tests**

**Tests d'Acceptance**

Les tests d'acceptance sont conçus pour vérifier que la plateforme NexaCore répond aux exigences fonctionnelles et aux attentes des utilisateurs. Ils sont réalisés en étroite collaboration avec les parties prenantes et les utilisateurs finaux.

* Tests Fonctionnels : Vérification des fonctionnalités clés comme la gestion des contacts, la création de campagnes et la gestion des opportunités.
* Tests d’Interface Utilisateur : Validation de la navigation et de l’expérience utilisateur.
* Tests de Cas d’Utilisation : Scénarios réalistes simulant des workflows complets pour s’assurer que chaque fonctionnalité fonctionne comme prévu.
* Validation des Données : Contrôle de l’intégrité et de la cohérence des données entre les modules SQL et NoSQL.

**Tests dans le Cycle DevOps**

Le processus DevOps intègre des tests automatisés tout au long du cycle de développement pour détecter et résoudre rapidement les problèmes. Les tests incluent :

* Tests Unitaires : Réalisés à chaque étape du développement pour vérifier chaque module et fonction isolément.
* Tests d’Intégration : Vérification des interactions entre les modules et les systèmes tiers.
* Tests de Régression : Tests automatisés pour garantir que les mises à jour n’introduisent pas de nouvelles erreurs.
* Déploiement Continu : Intégration avec GitLab CI/CD pour tester automatiquement chaque nouvelle version avant déploiement.

**Tests de Performance et de Compatibilité**

Des tests de performance et de compatibilité sont réalisés pour s’assurer que la plateforme fonctionne correctement sous différentes conditions.

* Tests de Charge : Simulation d’utilisateurs simultanés pour vérifier la scalabilité (jusqu’à 10 000 utilisateurs).
* Tests de Stress : Identification des limites du système sous forte pression.
* Tests de Scalabilité : Évaluation de la capacité à gérer des augmentations de trafic et de données.
* Tests Multi-Navigateurs : Vérification de la compatibilité avec Chrome, Firefox, Safari et Edge.
* Tests Multi-Plateformes : Validation sur Windows, macOS, iOS et Android.

**Tests de Déploiement**

Le processus de déploiement est systématiquement testé pour garantir une transition fluide entre les environnements de développement, de préproduction et de production.

* Tests de Déploiement Automatisé : Vérification des pipelines CI/CD pour s’assurer que les mises à jour sont déployées sans interruption.
* Tests Post-Déploiement : Surveillance des performances et des journaux pour détecter les anomalies immédiatement après la mise en production.
* Tests de Roll-Back : Simulation de scénarios de restauration pour vérifier l’efficacité des procédures de retour en arrière.

**9. Documentation**

**Interface de Documentation**

La documentation de NexaCore est conçue pour être accessible et centralisée, offrant une interface intuitive pour répondre aux besoins des utilisateurs finaux, des développeurs et des administrateurs système.

La plateforme utilise Confluence pour structurer et organiser la documentation technique et fonctionnelle. Confluence permet une navigation par catégories, facilitant l’accès aux guides utilisateurs, aux procédures techniques et aux FAQ. Chaque section est accompagnée d’illustrations, de schémas UML et de captures d’écran pour améliorer la compréhension.

Pour les développeurs, un portail Swagger est intégré pour documenter et tester les API REST et GraphQL. Ce portail permet une exploration interactive des endpoints, facilitant les tests et les intégrations avec des outils tiers. Les manuels d’installation, les journaux de versions et les guides d’intégration sont hébergés sur GitLab Wiki, garantissant une versionnage clair et structuré.

Les vidéos tutoriels, créées à l’aide d’outils comme Loom et Camtasia, sont intégrées dans la base de connaissances et accessibles via la plateforme de support client basée sur Zendesk.

**Politique de Maintenance de la Documentation**

La documentation suit un processus de maintenance continue afin de refléter l’évolution des fonctionnalités et des corrections apportées au système.

1. Mises à Jour Régulières : Chaque nouvelle version de NexaCore entraîne une mise à jour des guides utilisateurs, des API et des manuels techniques. Les mises à jour sont validées par l’équipe qualité avant publication.
2. Révision Semestrielle : Un audit complet de la documentation est réalisé deux fois par an pour identifier les incohérences, ajouter des améliorations et archiver les informations obsolètes.
3. Processus Collaboratif : Les mises à jour sont soumises à des revues collaboratives impliquant les développeurs, les chefs de projet et les responsables qualité. Les feedbacks des utilisateurs sont collectés via Zendesk pour affiner la clarté et la pertinence des contenus.
4. Archivage et Versionnage : Les anciennes versions des documents sont archivées et restent accessibles pour référence. Les modifications critiques sont tracées et associées à des numéros de version.

**10. Conclusion**

**Récapitulation des Points Clés**

Le présent Cahier des Charges Technique (CDCT) définit l’ensemble des aspects techniques nécessaires à la conception, au développement, au déploiement et à la maintenance de la plateforme NexaCore.

Nous avons couvert l’architecture modulaire basée sur des microservices et le choix des technologies modernes telles que React.js, Node.js, PostgreSQL, MongoDB, Docker et Kubernetes. Chaque décision technique a été justifiée pour garantir la scalabilité, la sécurité et la flexibilité du système.

Des schémas UML ont été fournis pour représenter les flux de données, les interactions entre les composants et les processus critiques comme la gestion des contacts et l’automatisation des campagnes. Les tests, intégrés au cycle DevOps, ont été détaillés pour assurer la qualité et la stabilité de la plateforme à travers des scénarios d’acceptance, de performance et de déploiement.

La documentation joue un rôle central dans l’accompagnement des utilisateurs et des développeurs, avec une gestion organisée à travers Confluence, GitLab Wiki et Swagger, garantissant une accessibilité et une mise à jour continue.

**Perspectives d’Évolution et d’Amélioration**

NexaCore a été conçu pour être une plateforme évolutive. Les perspectives d’évolution incluent l’ajout de fonctionnalités avancées telles que l’intelligence artificielle pour des recommandations plus précises, des outils de gestion des abonnements, et l’intégration de nouvelles API pour étendre les capacités de personnalisation.

À moyen terme, des améliorations sont envisagées sur la sécurité avec un renforcement des audits et des tests d’intrusion réguliers. L’optimisation des performances sera également poursuivie avec des tests de charge avancés et des simulations pour assurer la gestion d’un volume croissant d’utilisateurs.

Enfin, NexaCore mettra en place un suivi des retours clients pour prioriser les futures améliorations en fonction des besoins réels. Une feuille de route produit détaillant ces évolutions sera mise à jour régulièrement.