1. Performance

1.1 Temps de réponse

- Le temps de réponse pour l'envoi et la réception de messages texte ne doit pas dépasser 1 seconde dans des conditions réseau normales
- Le temps de réponse pour le chargement initial de l'application ne doit pas dépasser 3 secondes sur un réseau 4G
- La livraison des messages en attente doit être effectuée dans un délai maximal de 30 secondes après reconnexion

1.2 Traitement des médias

- L'analyse de modération par IA des images doit être complétée en moins de 5 secondes
- Le téléchargement et l'envoi d'images (jusqu'à 5 MB) doit s'effectuer en moins de 10 secondes dans des conditions réseau normales
- La compression des médias doit être effectuée côté client avant l'envoi pour optimiser le transfert

1.3 Capacité et charge

- Le système doit supporter jusqu'à 1000 utilisateurs actifs simultanément dans la phase initiale
- Le système de messagerie doit gérer jusqu'à 10 000 messages par heure

2. Architecture Microservices et Évolutivité

2.1 Architecture distribuée

- Organisation en 6 microservices principaux:
  - **auth-service** (Node.js + TypeScript): Authentification et gestion des sessions
  - **user-service** (Node.js + TypeScript): Gestion des profils et fonctionnalités de réseau social
  - **messaging-service** (Elixir + Phoenix): Communication temps réel et gestion des messages
  - **media-service** (Node.js + TypeScript): Gestion et stockage des fichiers multimédias
  - **notification-service** (Elixir): Gestion des notifications push
  - **moderation-service** (Python): Services d'intelligence artificielle pour la modération

2.2 Spécificités par microservice

- **messaging-service**:
  - Utilisation des capacités d'Elixir/Erlang pour la distribution et la concurrence
  - Implémentation du modèle Actor avec supervision pour la gestion des défaillances
  - Utilisation de Phoenix Channels pour la communication temps réel via WebSockets
  - Distribution des messages via PubSub d'Elixir pour le scaling horizontal
- **user-service**:
  - Gestion des relations sociales (contacts, blocages, listes)
  - Gestion des paramètres de confidentialité et préférences utilisateur
  - API pour la recherche et découverte d'utilisateurs
- **moderation-service**:
  - Optimisé pour l'inférence de modèles de vision par ordinateur
  - Traitement asynchrone des tâches de modération via une file d'attente
  - API pour l'apprentissage continu à partir du feedback
- **auth-service** et **media-service**:
  - Architecture REST/GraphQL pour les opérations CRUD
  - Optimisé pour les opérations I/O intensives avec Node.js
- **notification-service**:
  - Utilisation d'Elixir pour gérer de grandes quantités de connexions simultanées
  - Support FCM/APNS pour les notifications mobiles

2.3 Optimisation des performances

- Utilisation possible de Rust compilé en WebAssembly pour les parties critiques en performance:
  - Traitement des médias dans le media-service
  - Algorithmes de matching dans le user-service
  - Opérations cryptographiques dans l'auth-service
- Compilation Ahead-of-Time (AOT) pour les modules WebAssembly
- Benchmarking pour identifier les goulots d'étranglement nécessitant l'optimisation via Rust/WASM

2.4 Communication entre microservices

- Implémentation de gRPC pour toutes les communications inter-services
- Utilisation de Protocol Buffers pour la sérialisation efficace des données
- Services découplés avec contrats d'API bien définis
- Event sourcing pour la communication asynchrone entre services (optionnel)

2.5 Déploiement Kubernetes sur GCP

- Conteneurisation de tous les services avec Docker
- Orchestration via Google Kubernetes Engine (GKE)
- Utilisation du tier gratuit de GCP quand possible
- Configuration minimale initiale : 3 nœuds pour garantir la haute disponibilité
- Namespace dédié pour chaque environnement (dev, staging, prod)
- Utilisation des services managés de GCP pour réduire la charge opérationnelle

3. Disponibilité et Fiabilité

3.1 Uptime

- L'application doit maintenir une disponibilité de 99% (environ 7h d'indisponibilité par mois permise)
- Les opérations de maintenance doivent être planifiées et annoncées au moins 24h à l'avance

3.2 Tolérance aux pannes

- Le système doit continuer à fonctionner en cas de panne d'un nœud
- Les messages doivent être persistés pour éviter toute perte de données en cas de défaillance du système
- Les clients doivent mettre en cache localement les messages non envoyés pour les retransmettre après reconnexion

3.3 Reprise après sinistre

- Des sauvegardes quotidiennes de la base de données doivent être effectuées
- Le temps de récupération après sinistre majeur (RTO) ne doit pas dépasser 24 heures
- Le point de récupération des données (RPO) ne doit pas dépasser 1 heure de perte de données

4. Sécurité

4.1 Authentification et autorisation

- Mise en place d'une authentification forte basée sur mTLS (TLS mutuel)
- Utilisation du service Certificate Authority de Google Cloud Platform pour gérer la PKI
- Infrastructure de clés publiques (PKI) pour la gestion des certificats clients et serveurs
- Rotation automatique des certificats avec périodes configurables

- Mécanisme de révocation en cas de compromission

4.2 Chiffrement avancé

- Chiffrement des messages en transit avec TLS 1.3
- Chiffrement de bout en bout obligatoire utilisant le protocole Signal ou équivalent
- Double ratchet pour la rotation des clés et la confidentialité persistante
- Vérification des clés par comparaison de codes de sécurité ou scan QR
- Chiffrement du stockage local sur les appareils des utilisateurs

4.3 Communication sécurisée entre microservices

- Utilisation de gRPC pour toutes les communications inter-services
- Authentification mutuelle TLS (mTLS) entre microservices
- Circuit breakers implémentés pour gérer les défaillances de communication

4.4 Sécurité de l'API

- API Gateway sécurisé avec mTLS pour toutes les connexions client
- Limitation du taux de requêtes (rate limiting) pour prévenir les abus
- Protection contre les attaques CSRF, XSS et injection
- Validation des entrées pour toutes les API
- Utilisation du tier gratuit de Google Cloud Platform pour l'hébergement et la sécurité

5. Utilisabilité

5.1 Interface utilisateur

- L'interface doit être responsive et s'adapter aux écrans de smartphone, tablette et desktop
- Les temps de chargement perçus doivent être minimisés par des indicateurs de progression
- L'application doit suivre les principes de conception Material Design ou équivalent

5.2 Accessibilité

- L'application doit respecter les normes WCAG 2.1 niveau AA
- Support des lecteurs d'écran et des fonctionnalités d'accessibilité du système d'exploitation
- Support du mode sombre et ajustement de la taille des polices

5.3 Internationalisation

- L'interface utilisateur doit supporter le français et l'anglais dans la version initiale

- L'architecture doit permettre l'ajout simple de nouvelles langues

6. Compatibilité

6.1 Plateformes

- Application mobile : iOS 14+ et Android 9+
- Application web : Chrome, Firefox, Safari, Edge (deux dernières versions majeures)

6.2 Connectivité

- L'application doit fonctionner avec une connectivité dégradée (2G/3G)
- Mode hors ligne permettant de consulter les conversations précédentes et de composer des messages

7. Maintenabilité

7.1 Code

- Le code doit être modulaire et suivre les principes SOLID
- Standards de codage spécifiques à chaque langage:
  - TypeScript: ESLint avec configuration stricte, interfaces bien définies
  - Elixir: Formatage avec mix format, documentation avec ExDoc
  - Python: PEP 8, type hints, documentation avec docstrings
  - Rust (pour WASM): Rustfmt, documentation avec rustdoc
- Couverture de tests unitaires d'au moins 70% pour tous les services
- Tests d'intégration entre les microservices
- Benchmarking des modules critiques, notamment ceux implémentés en Rust/WASM

7.2 Infrastructure

- Infrastructure as Code (IaC) pour la configuration du cluster Kubernetes
- Monitoring avec collecte de métriques et logs centralisés
- CI/CD pour automatiser les tests et le déploiement
- Pipeline spécifique pour chaque microservice permettant des déploiements indépendants

7.3 Modèle de Modération (moderation-service)

- Interface de rétroaction pour améliorer continuellement le modèle de modération
- Logs détaillés des décisions de modération pour analyse et audit
- Architecture permettant le déploiement de nouvelles versions du modèle sans interruption

- Support pour A/B testing de différentes versions du modèle de modération

8. Performances du Modèle de Modération (moderation-service)

8.1 Précision

- Le modèle doit atteindre une précision minimale de 85% dans la détection de contenu inapproprié
- Le taux de faux positifs ne doit pas dépasser 5%
- Les seuils de confiance doivent être ajustables par les administrateurs
- Capacité à catégoriser le contenu en différentes classes (violence, nudité, etc.)

8.2 Efficacité

- Le modèle doit être optimisé pour s'exécuter avec des ressources limitées
- Temps de traitement maximum de 5 secondes par image
- Optimisation pour GPU si disponible, avec fallback CPU
- File d'attente pour gérer les pics de demande
- Utilisation des ressources du tier gratuit de GCP quand possible

8.3 Adaptabilité

- Interface API simple pour soumettre des corrections aux prédictions
- Pipeline d'entraînement incrémental à partir des données étiquetées
- Versionnement des modèles avec possibilité de rollback
- Architecture permettant de mettre à jour le modèle sans redéploiement complet

9. Limitations et Contraintes

9.1 Utilisation des ressources

- Le client mobile ne doit pas consommer plus de
  - Batterie : augmentation max de 15% de la consommation quotidienne
  - Données mobiles : max 50MB/jour en utilisation normale
  - Stockage : max 500MB par défaut (configurable par l'utilisateur)

9.2 Capacité de stockage

- Limite de taille des fichiers : 100MB par fichier

- Quota de stockage initial par utilisateur : 1GB
- Rétention des messages : illimitée (dans les limites du quota)

9.3 Limites Opérationnelles

- Nombre maximum d'utilisateurs par groupe : 200
- Nombre maximum de messages par seconde pour un utilisateur : 10
- Nombre maximum de médias analysés par jour par utilisateur : 100 (pour limiter les abus et les coûts de traitement)