ChainChat

Application de Messagerie Décentralisée basée sur la Blockchain

Projet de Développement

13 août 2025

Table des matières

1 Résumé Exécutif **4**
1.1 Vision . . . 4
1.2 Proposition de Valeur . . . 4

2 Analyse du Marché **4**
2.1 Problématiques Actuelles . . . 4
2.2 Marché Cible . . . 4

3 Architecture Technique **5**
3.1 Stack Technologique . . . 5
3.1.1 Frontend . . . 5
3.1.2 Backend & Infrastructure . . . 5
3.1.3 Smart Contracts . . . 5
3.2 Architecture Système . . . 5

4 Fonctionnalités **6**
4.1 Fonctionnalités Core . . . 6
4.1.1 Messagerie de Base . . . 6
4.1.2 Gestion d'Identité . . . 6
4.2 Fonctionnalités Avancées . . . 6
4.2.1 Économie Intégrée . . . 6
4.2.2 Gouvernance Décentralisée . . . 6

5 Spécifications Techniques Détaillées **6**
5.1 Protocole de Messagerie . . . 6
5.1.1 Format de Message . . . 6
5.1.2 Chiffrement . . . 7
5.2 Smart Contracts . . . 7
5.2.1 Contrat Principal . . . 7

6 Plan de Développement **9**
6.1 Phase 1 : MVP (3 mois) . . . 9
6.2 Phase 2 : Fonctionnalités Avancées (3 mois) . . . 9
6.3 Phase 3 : Écosystème (4 mois) . . . 9
6.4 Phase 4 : Expansion (6 mois) . . . 9

7 Structure du Code **10**
7.1 Architecture Frontend . . . 10
7.2 Architecture Backend . . . 10

8 Sécurité **11**
8.1 Modèle de Menaces . . . 11
8.2 Mesures de Sécurité . . . 11

9 Économie du Token **11**
9.1 Tokenomics CHAT . . . 11
9.2 Modèle Économique . . . 12

10 Défis et Solutions **12**
10.1 Défis Techniques 12
10.2 Défis Adoptation 12

11 Métriques de Succès **12**
11.1 KPIs Techniques 12
11.2 KPIs Business 12

12 Conclusion **13**

1 Résumé Exécutif

ChainChat est une application de messagerie révolutionnaire qui exploite la technologie blockchain pour offrir une communication véritablement décentralisée, sécurisée et résistante à la censure. Contrairement aux applications traditionnelles qui dépendent de serveurs centralisés, ChainChat fonctionne sur un réseau peer-to-peer où chaque utilisateur contribue à l'infrastructure du réseau.

1.1 Vision

Créer un écosystème de communication où les utilisateurs contrôlent entièrement leurs données, leurs interactions et leur identité numérique, sans dépendance à des entités centrales.

1.2 Proposition de Valeur

— **Décentralisation complète** : Aucun point de défaillance unique
— **Propriété des données** : Les utilisateurs contrôlent leurs informations
— **Résistance à la censure** : Impossible de bloquer ou de surveiller
— **Économie intégrée** : Modèle économique basé sur les tokens
— **Confidentialité renforcée** : Chiffrement end-to-end natif

2 Analyse du Marché

2.1 Problématiques Actuelles

1. **Centralisation excessive** : WhatsApp, Telegram, Signal dépendent de serveurs centraux
2. **Contrôle des données** : Les entreprises monétisent les données utilisateurs
3. **Censure** : Possibilité de blocage par les gouvernements ou plateformes
4. **Surveillance** : Métadonnées collectées même avec chiffrement
5. **Dépendance technologique** : Panne de serveurs = service indisponible

2.2 Marché Cible

— Utilisateurs soucieux de leur vie privée
— Activistes et journalistes dans des régimes restrictifs
— Entreprises nécessitant une communication sécurisée
— Early adopters de technologies blockchain
— Communautés crypto et Web3

3 Architecture Technique

3.1 Stack Technologique

3.1.1 Frontend

- **Framework** : React.js avec TypeScript
- **UI/UX** : Material-UI ou Chakra UI
- **State Management** : Redux Toolkit
- **Crypto** : Web3.js, ethers.js
- **Mobile** : React Native (version mobile)

3.1.2 Backend & Infrastructure

- **Runtime** : Node.js avec TypeScript
- **P2P Network** : libp2p (protocole IPFS)
- **Base de données** : IPFS pour le stockage distribué
- **Blockchain** : Ethereum ou Polygon pour les smart contracts
- **Chiffrement** : Signal Protocol, Curve25519

3.1.3 Smart Contracts

- **Langage** : Solidity
- **Framework** : Hardhat pour le développement
- **Oracles** : Chainlink pour données externes
- **Stockage** : Arweave pour l'archivage permanent

3.2 Architecture Système

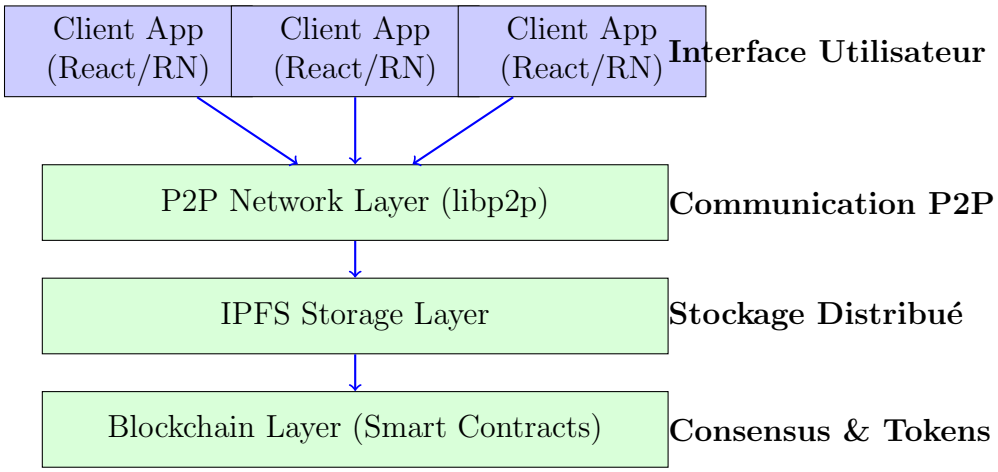

FIGURE 1 – Architecture générale du système ChainChat

4 Fonctionnalités

4.1 Fonctionnalités Core

4.1.1 Messagerie de Base

— Messages texte avec chiffrement end-to-end
— Messages multimédias (images, vidéos, audio)
— Messages vocaux avec transcription automatique
— Accusés de réception cryptographiquement prouvés
— Messages éphémères avec auto-destruction

4.1.2 Gestion d'Identité

— Identité décentralisée (DID) basée sur la blockchain
— Génération automatique de clés publique/privée
— Système de réputation décentralisé
— Vérification d'identité par consensus
— Récupération de compte via seeds phrases

4.2 Fonctionnalités Avancées

4.2.1 Économie Intégrée

— Token natif CHAT pour les transactions
— Micropaiements pour les messages premium
— Récompenses pour l'hébergement de nœuds
— Marché NFT pour avatars et stickers
— Système de tips entre utilisateurs

4.2.2 Gouvernance Décentralisée

— DAO pour les décisions du réseau
— Vote proportionnel au stake de tokens
— Propositions d'amélioration du protocole
— Mécanismes anti-spam communautaires
— Modération décentralisée

5 Spécifications Techniques Détaillées

5.1 Protocole de Messagerie

5.1.1 Format de Message

```
interface ChainChatMessage {
  id: string;                    // Hash unique du message
  sender: string;                // Adresse publique de l'exp diteur
  receiver: string;              // Adresse publique du destinataire
  content: EncryptedContent;     // Contenu chiffr
  timestamp: number;             // Timestamp Unix
  messageType: MessageType;      // TEXT, IMAGE, AUDIO, etc.
  signature: string;             // Signature cryptographique
  nonce: string;                // Nonce pour  viter  les replays
  metadata: MessageMetadata;     // M tadonn es optionnelles
}

interface EncryptedContent {
  data: string;                  // Contenu chiffr  en base64
  algorithm: string;             // Algorithme de chiffrement utilis
  keyHash: string;              // Hash de la cl  de session
}
```

Listing 1 – Structure d'un message ChainChat

5.1.2 Chiffrement

Le système utilise un chiffrement hybride combinant :

- **ECDH** pour l'échange de clés
- **AES-256-GCM** pour le chiffrement symétrique
- **Signal Protocol** pour la forward secrecy
- **Double Ratchet** pour la rotation des clés

5.2 Smart Contracts

5.2.1 Contrat Principal

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract ChainChatToken is ERC20, Ownable {

    struct User {
        string publicKey;
        string username;
        uint256 reputation;
        bool isActive;
        uint256 joinedAt;
    }

    struct Node {
        address operator;
        string endpoint;
        uint256 stake;
        uint256 uptime;
        bool isActive;
    }
```

```
24
25      mapping(address => User) public users;
26      mapping(address => Node) public nodes;
27      mapping(string => address) public usernameToAddress;
28
29      uint256 public constant MESSAGE_COST = 0.001 ether;
30      uint256 public constant NODE_MIN_STAKE = 100 ether;
31
32      event UserRegistered(address indexed user, string username);
33      event MessageSent(address indexed sender, address indexed receiver);
34      event NodeRegistered(address indexed operator, string endpoint);
35
36      constructor() ERC20("ChainChat", "CHAT") {
37          _mint(msg.sender, 1000000 * 10**18); // 1M tokens initiaux
38      }
39
40      function registerUser(string memory _username, string memory
    _publicKey)
41          external
42      {
43          require(bytes(_username).length > 0, "Username cannot be empty")
    ;
44          require(usernameToAddress[_username] == address(0), "Username
    taken");
45          require(!users[msg.sender].isActive, "User already registered");
46
47          users[msg.sender] = User({
48              publicKey: _publicKey,
49              username: _username,
50              reputation: 100, // R putation initiale
51              isActive: true,
52              joinedAt: block.timestamp
53          });
54
55          usernameToAddress[_username] = msg.sender;
56          emit UserRegistered(msg.sender, _username);
57      }
58
59      function sendMessage(address _receiver) external payable {
60          require(msg.value >= MESSAGE_COST, "Insufficient payment");
61          require(users[msg.sender].isActive, "Sender not registered");
62          require(users[_receiver].isActive, "Receiver not registered");
63
64          emit MessageSent(msg.sender, _receiver);
65      }
66
67      function registerNode(string memory _endpoint) external {
68          require(balanceOf(msg.sender) >= NODE_MIN_STAKE, "Insufficient
    stake");
69
70          _transfer(msg.sender, address(this), NODE_MIN_STAKE);
71
72          nodes[msg.sender] = Node({
73              operator: msg.sender,
74              endpoint: _endpoint,
75              stake: NODE_MIN_STAKE,
76              uptime: 0,
77              isActive: true
```

```
        });

        emit NodeRegistered(msg.sender, _endpoint);
    }
}
```

Listing 2 – Smart Contract ChainChat

6 Plan de Développement

6.1 Phase 1 : MVP (3 mois)

- Interface utilisateur de base (React)
- Messagerie P2P simple (libp2p)
- Chiffrement end-to-end
- Smart contract de base
- Tests sur réseau de test

6.2 Phase 2 : Fonctionnalités Avancées (3 mois)

- Messages multimédias
- Système de tokens CHAT
- Nœuds récompensés
- Interface mobile (React Native)
- Tests de charge

6.3 Phase 3 : Écosystème (4 mois)

- DAO et gouvernance
- Marché NFT intégré
- API pour développeurs tiers
- Audit de sécurité complet
- Lancement mainnet

6.4 Phase 4 : Expansion (6 mois)

- Intégrations cross-chain
- Fonctionnalités entreprise
- Plugins et extensions
- Optimisations performance
- Adoption massive

7 Structure du Code

7.1 Architecture Frontend

```
src/
          components/           # Composants r utilisables
                Chat/
                      MessageList.tsx
                      MessageInput.tsx
                      ChatWindow.tsx
                Auth/
                      Login.tsx
                      Register.tsx
                UI/
                    Button.tsx
                    Modal.tsx
          hooks/                # Custom hooks React
                useP2P.ts
                useWallet.ts
                useMessages.ts
          services/             # Services externes
                blockchain.ts
                ipfs.ts
                crypto.ts
                p2p.ts
          store/                # Redux store
                slices/
                      authSlice.ts
                      chatSlice.ts
                      networkSlice.ts
                index.ts
          types/                # Types TypeScript
                message.ts
                user.ts
                blockchain.ts
          utils/                # Utilitaires
              encryption.ts
              validation.ts
              constants.ts
```

Listing 3 – Structure des dossiers React

7.2 Architecture Backend

```
backend/
          src/
                p2p/            # Couche P2P
                      node.ts
                      discovery.ts
                      messaging.ts
                blockchain/     # Interactions blockchain
                      contracts.ts
                      events.ts
                      wallet.ts
                storage/        # Couche stockage IPFS
                      ipfs.ts
                      database.ts
```

```
                crypto/          # Cryptographie
                      encryption.ts
                      signatures.ts
                      keys.ts
                api/            # API REST/WebSocket
                    routes.ts
                    websocket.ts
          contracts/          # Smart contracts
                ChainChat.sol
                ChatToken.sol
                ChatDAO.sol
          scripts/           # Scripts de d ploiement
                deploy.ts
                migrate.ts
          tests/             # Tests
              unit/
              integration/
              e2e/
```

Listing 4 – Structure Node.js

8 Sécurité

8.1 Modèle de Menaces

1. **Attaques man-in-the-middle** : Mitigées par le chiffrement E2E
2. **Spam et flood** : Contrôlés par les coûts en tokens
3. **Attaques Sybil** : Limitées par le système de réputation
4. **Compromission de clés** : Forward secrecy avec Double Ratchet
5. **Attaques 51%** : Diversification sur plusieurs blockchains

8.2 Mesures de Sécurité

— Audit de code par des experts en sécurité
— Bug bounty program
— Chiffrement quantique-résistant (préparation)
— Isolation des clés privées (hardware wallets)
— Tests de pénétration réguliers

9 Économie du Token

9.1 Tokenomics CHAT

— **Supply total** : 1 milliard de tokens
— **Distribution initiale** :
  — 40% - Récompenses réseau (10 ans)
  — 25% - Équipe de développement (4 ans vesting)
  — 20% - Vente publique

  - 10% - Réserves DAO
  - 5% - Marketing et partenariats
- **Mécanismes de burn** : 50% des frais sont brûlés
- **Staking** : Récompenses de 5-15% APY pour les nœuds

9.2 Modèle Économique

- Messages gratuits jusqu'à 1000/mois
- Frais de 0.001 CHAT par message supplémentaire
- Messages premium (prioritaires) : 0.01 CHAT
- Stockage étendu : 1 CHAT/Go/mois
- Services entreprise : abonnements en CHAT

10 Défis et Solutions

10.1 Défis Techniques

- **Scalabilité** : Layer 2 et sharding
- **Latence** : Cache local et prédiction
- **Consommation** : Optimisation énergétique
- **Synchronisation** : Protocoles de consensus efficaces

10.2 Défis Adoptation

- **UX complexe** : Interface simplifiée, onboarding guidé
- **Coûts gas** : Subsidisation initiale, Layer 2
- **Performance** : Optimisations continues
- **Interopérabilité** : Passerelles vers apps traditionnelles

11 Métriques de Succès

11.1 KPIs Techniques

- Temps de livraison des messages ¡ 2 secondes
- Disponibilité du réseau ¿ 99.9%
- Débit : ¿ 10,000 messages/seconde
- Nœuds actifs : ¿ 10,000

11.2 KPIs Business

- Utilisateurs actifs mensuels : 1M en 18 mois
- Messages envoyés : 100M/mois
- Rétention J30 : ¿ 60%
- NPS (Net Promoter Score) : ¿ 70

12 Conclusion

ChainChat représente l'évolution naturelle de la messagerie vers un modèle décentralisé et orienté utilisateur. En combinant les meilleures technologies blockchain, P2P et cryptographiques, nous créons une plateforme qui redonne le contrôle aux utilisateurs tout en offrant une expérience utilisateur exceptionnelle.

Le projet s'appuie sur des technologies éprouvées (React, Node.js, Ethereum) tout en intégrant des innovations de pointe (libp2p, IPFS, Signal Protocol). Cette approche équilibrée garantit à la fois la faisabilité technique et l'adoption par les utilisateurs.

Avec un plan de développement échelonné sur 16 mois et une équipe expérimentée, ChainChat est positionné pour devenir le standard de la messagerie décentralisée et ouvrir la voie à un internet plus libre et respectueux de la vie privée.

Pour plus d'informations techniques ou pour rejoindre le projet, contactez l'équipe de développement.