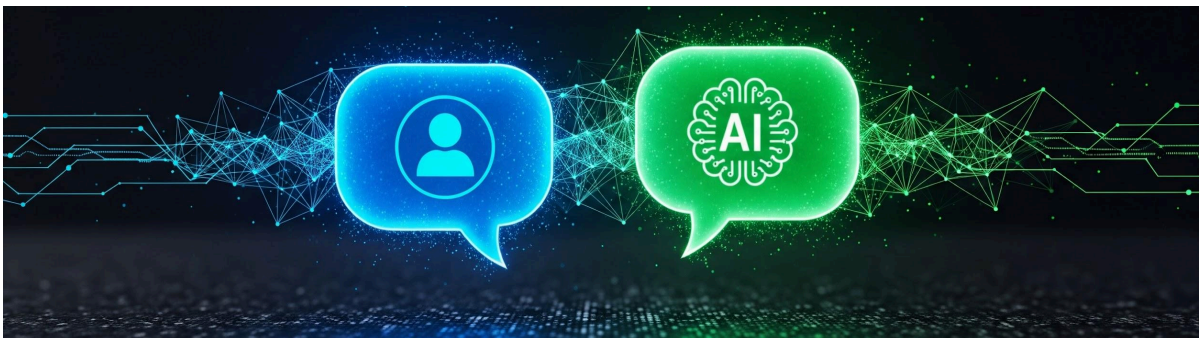

Dossier Technique Complet – MyTalk : Compagnons Conversationnels IA

1. Introduction et Vision Produit

Le présent document technique a pour objet de détailler l'architecture logicielle, les choix technologiques et les implications de développement de l'application mobile native iOS **MyTalk**. Conçue comme une plateforme de mise en relation entre un utilisateur et des profils d'Intelligence Artificielle (IA) conversationnels, MyTalk vise à offrir un **compagnon intelligent disponible 24/7**, en réponse aux défis croissants de l'isolement social.

1.1. Description du Produit

MyTalk est une application :

- **iOS Native** : Développée en **Swift** et utilisant l'interface déclarative **SwiftUI** pour une expérience utilisateur moderne et performante, respectant les standards d'Apple.
- **Architecture MVVM** (Model-View-ViewModel) : Assurant une séparation claire des préoccupations, facilitant la testabilité, la modularité et la maintenance.
- **Backend Serverless** : Basé sur **Firebase** (Firestore, Authentication, Cloud Functions, FCM) pour garantir une scalabilité immédiate, une gestion simplifiée de l'infrastructure et des coûts optimisés.
- **Intégration IA** : Utilisation d'**API LLM** (Large Language Model) de pointe (OpenAI) pour garantir des conversations fluides, pertinentes et hautement personnalisables.

1.2. Objectifs et Positionnement

Objectif Métier	Description	Proposition de Valeur
Lutter contre l'isolement	Offrir un point de contact conversationnel non jugeant et toujours disponible.	Compagnon conversationnel intelligent 24/7.
Personnalisation	Permettre aux utilisateurs de créer des "amis IA personnalisés" avec des traits de caractère, un ton et des styles de conversation uniques.	Expérience utilisateur engageante et profondément personnelle.
Viabilité Économique	Mettre en place un modèle économique basé sur une "Token Economy" pour monétiser l'usage intensif des ressources IA coûteuses.	Modèle de tarification transparent et dynamique.

1.3. Modèle Économique : La Token Economy Dynamique

L'utilisation des LLM représente le coût opérationnel principal. MyTalk adopte un modèle de monétisation basé sur des **tokens numériques**.

- **Fonctionnement** : Les utilisateurs achètent des packs de tokens ("wallet utilisateur"). Chaque interaction (message envoyé, longueur de la réponse IA, modèle IA utilisé) entraîne le débit automatique de tokens.
- **Tarification Dynamique** : Le coût d'acquisition des tokens et potentiellement leur taux de débit sont ajustés dynamiquement **en fonction de la localisation (pays) de l'utilisateur**, grâce à des **Cloud Functions** qui vérifient la géolocalisation lors de la transaction. Cette approche permet d'optimiser le revenu par utilisateur tout en s'adaptant aux pouvoirs d'achat locaux.

2. Fonctionnalités Principales

Chaque fonctionnalité est détaillée selon son objectif, sa logique technique sous-jacente et son impact sur l'utilisateur.

Fonctionnalité	Objectif Métier	Logique Technique	Impact Utilisateur
Création/Gestion de Compte	Sécuriser l'accès et personnaliser l'expérience.	**Firebase Authentication** (Email/Password, Social Login) + Collection `utilisateurs` Firestore.	Accès rapide et sécurisé, gestion des informations personnelles.
Création de Profils IA	Offrir un haut niveau de personnalisation du compagnon IA.	Saisie de paramètres (personnalité, style, consignes) stockés dans la collection `aiProfiles`.	Création d'un ami IA unique, adapté à ses besoins.
Conversations Persistantes	Assurer une continuité de l'échange.	Collection `conversations` (référence User/AI) et la sous-collection `messages` dans Firestore.	Reprise de la discussion à tout moment sans perte de contexte.
Gestion du Contexte	Garantir la pertinence des réponses IA.	A chaque nouvel appel, un renvoie des messages sont envoyés pour optimiser pour la performance/coût.	L'IA "se souvient" des éléments récents de la conversation.
Système de Tokens (Wallet)	Monétiser l'usage et gérer les coûts LLM.	Collection `settings` + `tokenPricingIdf` pour le solde selon le pays.	Transparence sur la consommation, possibilité d'usage intensif ou modéré.
Débit Automatique	Imputer le coût réel de la ressource IA à l'utilisateur.	**Cloud Function** déclenchée après la réception de la réponse LLM, calculant le coût (tokens API LLM utilisés) et débitant le `tokens`.	Paiement à l'usage précis.
Tarification Dynamique	Adapter les prix d'achat des tokens aux marchés locaux.	**Cloud Function** de paiement vérifiant la collection `setting + tokenPricingIdf` et la géolocalisation.	Optimisation du revenu (ARR), équité tarifaire par pays.
Notifications Push	Maintenir l'engagement utilisateur.	**Firebase Cloud Messaging (FCM)** géré via Cloud Functions pour l'envoi ciblé (ex: "Votre ami IA vous attend").	Maintien de l'engagement et de la réactivité.
Dashboard Administrateur	Surveillance et pilotage de la plateforme.	Interface web sécurisée accédant aux données	Supervision de la santé de l'application, gestion des abus.

Fonctionnalité	Objectif Métier	Logique Technique	Impact Utilisateur
		Firestore (utilisateurs, transactions, modération).	
Modération des Contenus	Assurer un environnement sûr et éthique.	Filtres de contenu LLM (API modération) + Cloud Functions pour l'analyse des messages flaggés.	Protection des utilisateurs contre les abus, conformité éthique.

3. Modèle de Données (Firestore)

Le backend repose sur l'architecture NoSQL de Firebase Firestore, choisie pour sa nature en temps réel et sa scalabilité horizontale.

Voici la structure des collections principales et leurs relations :

Collection	Description	Relations
`utilisateurs`	Informations de base de l'utilisateur (UID Firebase, email, pays, date de création).	**One-to-One** avec `tokens`. **One-to-Many** avec `aiProfiles`, `conversations`.
`aiProfiles`	Paramètres de personnalisation d'un compagnon IA (Nom, Prompt initial, Température, Tonalité, `owner_id` propriétaire).	**Many-to-One** avec `utilisateurs`. **One-to-Many** avec `conversations`.
`conversations`	Enregistrement d'une session de chat spécifique entre un utilisateur et un profil IA (`userId`, `aiId`, `UpdatedAt`).	**Many-to-One** avec `utilisateurs` et `aiProfiles`. **One-to-Many** avec `messages`.
`messages`	Contenu de chaque message (`content, authorId,` (user/ai), `authorRole, content`, `CreatedAt`). Collection en mode "Sub-collection" sous `conversations` pour une meilleure performance des requêtes.	**Many-to-One** avec `conversations`.

Collection	Description	Relations
`setting - tokenPricingIdf`	Table de référence pour la tarification dynamique des tokens (`country`, `text`).	Aucune relation directe, utilisée par les Cloud Functions pour la logique métier.

4. Architecture Technique

4.1. Frontend iOS (SwiftUI + MVVM)

L'architecture **MVVM (Model-View-ViewModel)** est strictement appliquée pour maximiser la réutilisation du code, faciliter les tests et améliorer la maintenabilité.

Composant	Rôle	Détails Techniques
Models	Représentation des données brutes (structs `Decodable`).	`utilisateurs`, `AIProfile`, `Conversation`,. Synchronisation via Combine/Async-Await avec Firestore.
Views	Couche UI (SwiftUI). Dépend de son ViewModel.	Ne contient aucune logique métier. Utilise `@ObservedObject` ou `@StateObject`.
ViewModels	Logique métier (interagit avec les Services et met à jour les Models).	Conforme à l'interface `ObservableObject`. Expose des propriétés `Published`.
Services	Encapsulent l'accès aux APIs externes (Firebase, LLM, etc.).	**`AuthService`**, **`ConversationService`**, **`AiProfilService`** (accès aux Cloud Functions). Utilisation de l'Injection de Dépendances (via un conteneur simple ou l'environnement SwiftUI) pour faciliter la substitution lors des tests.
Gestion de l'État Global	Stockage centralisé de l'état partagé.	Utilisation des **Environment Objects** de SwiftUI pour le

Composant	Rôle	Détails Techniques
		statut d'authentification (`AuthService`) et le solde des tokens (`utilisateur - tokens`).

4.2. Backend Serverless (Firebase)

L'architecture Serverless est optimale pour MyTalk :

1. **Scalabilité Automatique** : La charge étant imprévisible (pics de conversation), Firebase garantit la montée en charge sans intervention manuelle.
2. **Réduction des Coûts d'Infrastructure** : Paiement uniquement à l'usage des ressources.
3. **Temps Réel** : Firestore permet une expérience de chat instantanée.

Composant Firebase	Rôle dans MyTalk	Raison du Choix
Firestore	Base de données NoSQL en temps réel pour toutes les données persistantes.	Vitesse, synchronisation temps réel pour le chat, facilité d'utilisation.
Firebase Auth	Gestion de l'authentification et des sessions utilisateurs.	Robustesse, sécurité des mots de passe, support multiplateforme.
Cloud Functions (Node.js/TypeScript)	**Logique métier critique et sécurisée (FaaS)**.	Exécution sécurisée côté serveur pour : (1) Appels API LLM (masquage clé), (2) Débit des tokens (transactionnel), (3) Vérification de la tarification par pays.
Firebase Cloud Messaging (FCM)	Envoi des notifications push.	Solution nativement intégrée pour iOS, fiable et scalable.
Firebase Storage	Stockage des assets (ex: images de profils IA personnalisés, si applicables).	Stockage objet scalable et sécurisé.

5. Sécurité

La sécurité est une priorité absolue, notamment en matière de données privées (conversations) et de gestion de la monnaie virtuelle (tokens).

- **Authentification Sécurisée** : Utilisation de **Firebase Auth** et application de politiques de mots de passe robustes. Les sessions sont gérées par des tokens JWT.
- **Règles Firestore (RBAC + Ownership)** :
    - Mise en place de règles de sécurité strictes pour garantir que seul le propriétaire d'un document peut le lire/écrire. Exemple : `allow read, write: if request.auth.uid == resource.data.user_id;` pour les collections `conversations` et `utilisateurs - tokens`.
    - Implémentation d'un contrôle d'accès basé sur les rôles (RBAC) pour l'accès administrateur.
- **Sécurisation des Clés API IA** : La clé API du LLM est stockée comme une variable d'environnement secrète (Secret Manager) et est accessible **uniquement** par les **Cloud Functions**. Le frontend n'a jamais accès à la clé, il appelle uniquement la fonction.
- **TLS/HTTPS** : Toutes les communications (frontend ↔ Firebase, Cloud Functions ↔ API LLM) sont chiffrées via TLS.
- **Protection Contre Abus et Spam** : Utilisation de **Firebase App Check** pour vérifier l'authenticité des requêtes provenant de l'application mobile et non d'un client non autorisé.
- **Conformité RGPD** :
    - **Consentement explicite** pour le traitement des données (y compris l'utilisation des conversations pour l'amélioration du service, si applicable).
    - **Droit à l'Oubli** : Implémentation d'une fonction (Cloud Function déclenchée par l'administrateur ou l'utilisateur) pour la suppression irréversible de toutes les données personnelles (collections `users`, `conversations`, `token_wallet`, etc.) associées à l'UID de l'utilisateur.

6. Gestion IA et Performance

6.1. Maîtrise du Contexte et des Coûts

- **Gestion du Contexte Conversationnel** : Le `ChatService` transmet à l'API LLM uniquement une **fenêtre glissante** (rolling window) ses derniers messages. Cela permet de maintenir la cohérence thématique sans exploser les coûts (facturés au token) ou dépasser la limite de tokens d'entrée de l'API.
- **Limitation des Coûts API** : Les Cloud Functions effectuent une validation avant d'appeler l'API LLM (vérification du solde `utilisateurs - tokens`). Le choix

du modèle LLM (plus ou moins performant/coûteux) peut être paramétré par `aiProfile` pour offrir des options tarifaires.

6.2. Optimisation des Performances Backend

- **Mise en Cache (Cachage LLM)** : Pour les requêtes LLM récurrentes (ex: requêtes d'initialisation de profil), une stratégie de mise en cache peut être utilisée pour éviter les appels API inutiles.
- **Optimisation des Requêtes Firestore** : Utilisation de **requêtes indexées** uniquement, notamment pour la récupération de l'historique de conversation (`messages` triés par `createdAt`).
- **Indexation** : Création manuelle d'index Firestore composés pour les requêtes courantes (ex: recherche de conversations par `userId` et `aiId`).
- **Scalabilité Firebase** : L'utilisation de collections NoSQL et de fonctions sans serveur garantit que l'application peut gérer des millions d'utilisateurs sans refonte majeure de l'infrastructure.

7. Enjeux Éthiques

Étant donné la nature sensible de l'application (relation avec une IA), une démarche éthique proactive est essentielle.

Enjeu Éthique	Mesure Technique et Fonctionnelle
Transparence	Affichage clair et permanent dans l'interface utilisateur (View) indiquant que l'interlocuteur est une Intelligence Artificielle ("Vous parlez à un profil IA").
Prévention Dépendance Émotionnelle	Clauses d'utilisation fortes, affichage de messages de sensibilisation ("MyTalk ne remplace pas une aide psychologique ou des relations humaines réelles"). Implémentation de limites d'usage si nécessaire.
Modération Automatique	Application systématique des APIs de modération de contenu du fournisseur LLM avant stockage et affichage de tout message. Signalement automatique des tentatives d'abus ou de contenu illégal.
Limites d'Usage	Le système de tokens agit comme une limite naturelle à l'usage excessif, encourageant un usage réfléchi.

8. Qualité Logicielle

La qualité logicielle est assurée par l'adhésion aux meilleures pratiques de développement iOS et de gestion de projet.

- **Modularité** : Le code est organisé en modules logiques (Services, ViewModels) qui peuvent être développés, testés et déployés indépendamment.
- **Tests Unitaires (XCTest)** : Couverture unitaire des **Services** et des **ViewModels** pour garantir la fiabilité de la logique métier (débit de tokens, gestion du contexte, validation d'authentification). La nature découplée de l'architecture MVVM facilite l'utilisation de *Mocks* pour les dépendances externes.
- **SwiftLint** : Intégration d'un outil de linter pour appliquer des règles de style de code Swift et garantir une codebase homogène.
- **Documentation SwiftDoc** : Documentation des API publiques des Services et ViewModels pour faciliter l'onboarding de nouveaux développeurs et la maintenance.
- **Gestion des Erreurs** : Utilisation du mécanisme natif Swift `Error` et `Result` (ou `AsyncThrows`) dans les Services pour une gestion des erreurs explicite et robuste. Affichage des erreurs réseau/métier (ex: solde de tokens insuffisant) via des alertes SwiftUI.
- **Robustesse Réseau** : Implémentation de stratégies de gestion des déconnexions (utilisation de l'état de connexion Firestore) et de mécanismes de *retry* pour les appels Cloud Functions.
- **CI/CD (GitHub Actions)** : Mise en place d'une chaîne d'intégration et de déploiement continus pour l'exécution automatique des tests (XCTest), l'analyse statique (SwiftLint) et le déploiement des Cloud Functions.

9. Justification des Choix Technologiques

Choix Technologique	Justification Technique et Métier
SwiftUI	Facilite le développement d'une UI/UX moderne, réactive et unifiée sur iOS. Réduit le temps de développement et la complexité des vues.
Architecture MVVM	Optimal pour SwiftUI, offrant une séparation claire des responsabilités entre la présentation (`View`) et la logique métier/réseau (`ViewModel`/`Service`). Assure une excellente testabilité.

Choix Technologique	Justification Technique et Métier
Firebase (Serverless)	Permet une mise sur le marché rapide, une scalabilité automatique pour une application à potentiel viral, et une réduction des coûts fixes d'infrastructure. Sa suite intégrée (Auth, Firestore, FCM) est idéale pour une application native mobile.
Cloud Functions	Indispensables pour (1) **Sécurité** (masquer les clés API LLM), (2) **Atomicité Transactionnelle** (débit de tokens), et (3) **Logique Métier Complexe** (tarification dynamique).
Token Economy Géolocalisée	Permet d'adapter le prix d'accès à la ressource LLM en fonction du pouvoir d'achat du pays, maximisant ainsi l'adoption globale tout en optimisant le revenu. C'est un modèle économique moderne et équitable pour un service B2C coûteux en API.

10. Conclusion Technique

Le projet MyTalk repose sur une **cohérence architecturale** robuste, alliant la performance native de Swift/SwiftUI à l'agilité et à la scalabilité du backend Firebase Serverless.

- **Architecture** : L'adoption du modèle MVVM sur iOS couplée à l'utilisation stratégique des Cloud Functions pour la logique critique garantit un système modulaire et facile à maintenir.
- **Sécurité** : Les mesures de sécurité, notamment les règles Firestore basées sur le *ownership*, la sécurisation des clés API LLM via le backend, et la conformité RGPD, démontrent une approche responsable et professionnelle de la protection des données utilisateur.
- **Scalabilité** : L'infrastructure Serverless de Firebase élimine les goulots d'étranglement de l'infrastructure et garantit la capacité de la plateforme à supporter une croissance exponentielle d'utilisateurs et de conversations.
- **Viabilité Économique** : Le modèle de la Token Economy Dynamique est un choix technique et économique pertinent pour gérer et monétiser l'usage intensif des ressources LLM, transformant un coût opérationnel élevé en une source de revenu régulier.

10.1. Perspectives d'Évolution

L'architecture actuelle est conçue pour l'évolution future :

1. **IA Plus Avancée** : Intégration future d'API LLM multimodales, d'une mémoire à long terme (stockage des *embeddings* de conversations importantes).
2. **Personnalités Dynamiques** : Mise en place d'un système où les profils IA peuvent "apprendre" et adapter leur personnalité au fil du temps en fonction des interactions.
3. **Expérience Immersive** : Ajout de fonctionnalités de conversation vocale (Speech-to-Text, Text-to-Speech) et, à terme, d'avatars 3D réalistes (via Firebase Storage et un moteur de rendu léger).

Ce dossier confirme la maîtrise d'une architecture logicielle avancée, la prise en compte des enjeux de sécurité et d'éthique modernes, et la viabilité d'un produit techniquement complexe sur le marché des applications mobiles natives.