Spécification Technique Détaillée – US01 : Initialisation du service de fond

Contexte général

Cette spécification décrit en profondeur la mise en œuvre technique de la User Story 01 du module `MarketDataCollector`, dont l'objectif est d'initialiser un service de fond .NET en charge de configurer le flux temps réel des données de marché. L'objectif est de récupérer les timeframes à surveiller, la liste des symboles actifs, de construire dynamiquement l'URL WebSocket Binance multiplexée, d'initialiser les buffers circulaires en mémoire pour chaque couple (symbol, timeframe), et de démarrer une session WebSocket opérationnelle.

---

1. Architecture technique

1.1 Service de fond

- Le point d'entrée technique est un `BackgroundService` nommé `MarketDataCollectorWorker`, enregistré via `IHostedService` au démarrage de l'application.
- Le service s'exécute tant que l'App Service est en ligne ou jusqu'à l'annulation via `CancellationToken`.

1.2 Déploiement

- Le service est intégré à l'API existante contenant `SymbolDownloader`, partageant les mêmes configurations, logs, services injectés et système de monitoring.
- Un seul processus Azure App Service, déployé via Docker ou pipeline GitHub Actions.

1.3 Technologies utilisées

- .NET 9 SDK
- Microsoft.Extensions.Hosting
- WebSockets (client .NET natif ou 3rd party)
- In-memory circular buffer
- appsettings.json pour la configuration

---

2. Dépendances internes

Composant	Rôle
`ISymbolStore`	Fournit la liste actuelle des symboles à surveiller
`IConfiguration`	Permet de lire les timeframes, limites, buffers
`ILogger<MarketDataCollectorWorker>`	Logger structuré intégré à App Insights
`IWebSocketClient`	Client de connexion WebSocket Binance multiplexée

Composant	Rôle
`ICandleBufferStore`	Gère les buffers FIFO internes pour chaque symbole et timeframe

---

3. Étapes techniques détaillées

3.1 Lecture des timeframes depuis la configuration

- Clé : `Timeframes` (format `List<string>`, exemple `"M1", "M5", "M15"`)
- En cas d'absence ou de mauvaise configuration, log erreur + throw `InvalidConfigurationException`

3.2 Chargement des symboles actifs

- Appel asynchrone à `SymbolStore.GetActiveSymbolsAsync()`
- Application d'une limite de taille maximale `MaxSymbols` (par défaut 200)
- Format standard Binance attendu : `BTCUSDT`, `ETHUSDT`, etc.
- Si résultat vide/null : arrêt immédiat du service avec log `Critical`

3.3 Construction de l'URL WebSocket Binance

- Format cible :

```
wss://stream.binance.com:9443/stream?streams=btcusdt@kline_1m/
ethusdt@kline_1m/btcusdt@kline_5m/...
```

- Génération dynamique à partir du produit cartésien symbol × timeframe
- Limite max : 1024 streams (sinon log + tronquage ou fail)

3.4 Initialisation des buffers FIFO internes

- Pour chaque (symbol, timeframe), allocation de 3 buffers en RAM :
- `Volume[]`, `Close[]`, `High-Low[]`
- Implémentation : `CircularBuffer<T>` avec fenêtre `WindowSize` configurable (par défaut 5)
- Les buffers sont stockés dans un dictionnaire indexé `(symbol, timeframe)`

3.5 Ouverture de la session WebSocket

- Utilisation de `ClientWebSocket` ou wrapper personnalisé injecté
- Connexion asynchrone vers l'URL construite
- Attente de premier message pour valider la connexion
- Logs d'état : succès/échec, tentative de reconnexion

---

4. Gestion du cycle de vie

4.1 Au démarrage

- Les méthodes sont séquentielles : configuration → symboles → URL → buffers → connexion
- En cas d'erreur bloquante : arrêt du service avec log explicite

4.2 En fonctionnement

- Le service passe en mode `await foreach (var message in WebSocketStream)`
- Chaque message est délégué à un parseur (non concerné par cette US)

4.3 À l'arrêt

- Le `CancellationToken` est respecté et permet de :
- Fermer proprement la connexion WebSocket
- Libérer les buffers
- Logger l'extinction

---

5. Sécurité, fiabilité et résilience

- Aucune authentification requise sur WebSocket Binance
- Une politique de retry progressive est appliquée sur les erreurs réseau (3 tentatives max)
- Les erreurs de configuration sont considérées comme fatales
- Les buffers sont régénérés à chaque démarrage ou rotation

---

6. Tests techniques attendus

Test	Objectif
Config timeframes vide ou absente	Doit générer une erreur critique
SymbolStore renvoie null	Le service ne démarre pas
Plus de 1024 streams dans l'URL	Le service doit refuser le démarrage
Buffer bien initialisé par symbol/timeframe	Taille exacte, valeurs nulles au départ
Connexion WS réussie	Reçoit un premier message
Connexion WS échouée	Retente 3 fois puis abandonne

---

7. Logging & Observabilité

- Logs `Information` pour chaque phase :
- timeframes lus, symboles chargés, URL générée
- Logs `Warning` pour symboles inconnus, configuration incomplète
- Logs `Error` pour WebSocket ou SymbolStore KO
- KPIs à exposer :

- `symbols_loaded_total`
- `websocket_connection_success`
- `buffers_initialized_total`

---

8. Configuration dans appsettings.json

```
{
  "Timeframes": ["M1", "M5", "M15"],
  "MaxSymbols": 200,
  "BufferWindow": 5,
  "ReconnectTimeoutSeconds": 10
}
```

---

9. Limites connues

- Aucun fallback si le SymbolStore échoue
- Aucun stockage persistant, redémarrage = reset total
- L'ordre des symboles n'est pas garanti
- Si Binance limite un flux (HTTP 429), aucun plan de mitigation n'est prévu dans cette US

Souhaites-tu que je décline cette spec en tâches unitaires ou user stories techniques ?