



Spécification Technique – Calcul Dynamique des Données de Marché

 Référence : ST-MKT-001

 Date : 2025-07-25

Auteur : LaylaHft Engineering Team

 Liée à : SF-MKT-001

1. Objectif technique

Implémenter un service autonome, découplé et résilient capable de calculer les données dynamiques de marché (`CurrentPrice`, `Change24hPct`, `Change7dPct`, `Change30dPct`, `Sparkline`) pour chaque symbole actif à partir des données historiques Binance (Klines). Ce service devra être intégré à la plateforme HFT Layla sans alourdir le module `SymbolDownloader`, avec des hooks observables (SignalR, logs, métriques), en s'appuyant sur la mécanique événementielle de FastEndpoints.

2. Architecture

2.1 Composants clés

- **SymbolDownloader** : responsable du téléchargement initial des métadonnées de symboles. Il ne calcule aucune donnée dynamique.
- **SymbolMarketStatsCalculator** : service responsable du calcul des données dynamiques de marché pour un symbole donné.
- **SymbolMarketStatsHandler** : handler événementiel FastEndpoints qui s'abonne à `SymbolImportedEvent` pour déclencher les calculs.
- **SymbolImportedEvent** : événement implémentant l'interface `IEvent`, publié dès qu'un symbole est téléchargé.

2.2 Flux événementiel

1. `SymbolDownloader` importe un symbole depuis Binance.
2. Une fois le `SymbolMetadata` inséré dans le store, il instancie un `SymbolImportedEvent` et appelle `.PublishAsync(Mode.WaitForAll)` dessus.
3. `SymbolMarketStatsHandler`, abonné à cet événement, est exécuté automatiquement par FastEndpoints.
4. Le handler récupère le symbole concerné depuis le store.
5. Il appelle `SymbolMarketStatsCalculator.CalculateAsync(...)` pour effectuer les calculs de marché.
6. Une fois les données calculées, le symbole est mis à jour dans le store.
7. Une notification `SymbolUpdated` est émise via SignalR pour mise à jour UI ou d'autres modules.

Cette approche garantit la **décorrélation**, la **testabilité**, la **modularité** et l'**extensibilité** de la plateforme.

3. Dépendances techniques

- `BinanceRestClient` (via `IBinanceRestClient`) pour les `klines`
 - `ISymbolStore` pour lecture/écriture des symboles
 - `ILogger<T>` pour traçabilité
 - `IHubContext<SymbolHub>` pour push SignalR
 - `IEvent` et `.PublishAsync()` (FastEndpoints) pour publication d'événements
 - `IEventHandler<TEvent>` (FastEndpoints) pour consommer les événements
 - `IMetrics` pour histogrammes et compteurs (optionnel)
-

4. Interfaces et contrat d'intégration

- **Event DTO** : `SymbolImportedEvent` (implémente `IEvent`, contient `Symbol`, `Exchange`, `QuoteAsset`)
 - **Handler** : `SymbolMarketStatsHandler : IEventHandler<SymbolImportedEvent>`
 - **Service de calcul** : `ISymbolMarketStatsCalculator` exposant `CalculateAsync()` et `CalculateAllAsync()`
-

5. Algorithme de calcul (conceptuel)

- Récupération de l'historique de prix (`klines`) sur 30 jours (intervalle 1d ou 6h)
 - Extraction des valeurs de clôture (`close`)
 - Calcul de :
 - `CurrentPrice` = dernier `close`
 - `Change24hPct`, `Change7dPct`, `Change30dPct` = variation par rapport au `close` N jours avant
 - `Sparkline` = liste des closes normalisée
 - Mise à jour du symbole dans le `SymbolStore`
 - Notification en temps réel via SignalR
-

6. Observabilité

- Traçabilité complète via `ILogger`
 - Exposition de métriques techniques :
 - `symbols.stats.duration.ms`
 - `symbols.stats.failure.count`
 - `symbols.stats.success.count`
 - Possibilité d'alerter via EventBus secondaire ou outil externe (Prometheus, AppInsights, etc.)
-

7. Résilience

- Timeout API Binance : 5 secondes
 - Retry (x3) avec délai exponentiel en cas d'échec
 - Circuit breaker interne (désactivation du handler temporaire en cas d'échecs répétés)
 - Mode fallback :
 - Données laissées inchangées
 - Log d'erreur avec contexte complet (symbol, temps, message)
-

8. Tests & validations

- Tests unitaires :
 - Handler reçoit bien l'événement et déclenche le calcul
 - Mock Binance client pour simuler données valides, incomplètes ou manquantes
 - Calculs de variations sur jeu de données contrôlé
 - Tests d'intégration :
 - Chaîne complète `Downloader -> Event -> Handler -> Calcul -> SignalR`
 - Résilience à une coupure réseau ou données erronées
-

9. Objectifs atteints par cette architecture

- Découplage du calcul de données dynamiques via événements FastEndpoints
 - Réutilisabilité : un symbole peut être recalculé à tout moment (batch, CLI, timer)
 - Maintenabilité : ajout d'autres handlers (ex : log historique, alertes) sans impacter le downloader
 - Observabilité complète du processus
 - Tolérance aux erreurs par symbole sans blocage du reste de la plateforme
-

Souhaites-tu que je crée maintenant l'**EPIC dans le release plan** ?