



Spécification Fonctionnelle – Calcul Dynamique des Données de Marché

 Référence : SF-MKT-001

 Date : 2025-07-25

Auteur : LaylaHft Team

 Statut : À valider

1. Objectif

L'objectif de cette fonctionnalité est d'enrichir dynamiquement les symboles de la plateforme LaylaHft avec des données de marché calculées et mises à jour régulièrement. Ces données incluent le prix actuel (`CurrentPrice`), les variations en pourcentage sur plusieurs périodes (`Change24hPct`, `Change7dPct`, `Change30dPct`) et une courbe temporelle visualisable sous forme de sparkline (`Sparkline`).

Ce module doit offrir une vue à jour de l'évolution des prix de chaque actif numérique (crypto-monnaie) pour des usages variés : affichage sur l'interface utilisateur, prise de décision par les bots de trading, contrôle de cohérence par les opérateurs, ou encore génération d'alertes internes.

Les données doivent être : - Précises (basées sur les dernières données disponibles), - Cohérentes (alignées avec la logique temporelle), - Résilientes (fonctionner en cas de coupure de l'API Binance), - Observables (loggées, traçables et métriquées).

Cette spécification s'applique à l'ensemble de la plateforme HFT (haute fréquence), et servira de base à des déclinaisons par module technique (Downloader, Store, Notifier, UI, Risk, Analytics).

2. Contexte

Actuellement, le module `SymbolDownloader` importe les symboles de l'API Binance en appelant `GetExchangeInfoAsync()`, puis les stocke dans un cache mémoire (`ISymbolStore`) et un fichier persistant local.

Chaque symbole contient des métadonnées statiques (nom, base/quote asset, tick size, step size, etc.). Ces métadonnées sont essentielles mais insuffisantes pour de nombreux composants de la plateforme, notamment :

- Les interfaces graphiques nécessitent d'afficher des variations de prix et des mini-graphes (sparklines).
- Le moteur de trading a besoin de connaître le `CurrentPrice` à tout moment.
- Le gestionnaire de risques analyse les hausses et baisses sur plusieurs jours.

- Les utilisateurs souhaitent visualiser les évolutions pour faire des choix.

Or, ces données dynamiques ne sont pas présentes à l'import initial, ni mises à jour périodiquement.

Cette spécification vient donc compléter l'existant en introduisant une couche de **calculs dynamiques de données de marché** sur chaque symbole actif.

3. Fonctionnalités attendues

3.1 Calcul des données de marché par symbole

Pour chaque symbole actif, les données suivantes doivent être calculées :

- `CurrentPrice` : le dernier prix connu (dernier `close` d'un `kline`, ou `lastPrice` d'un ticker Binance)
- `Change24hPct` : variation en % entre maintenant et 24h avant
- `Change7dPct` : variation en % entre maintenant et 7 jours avant
- `Change30dPct` : variation en % entre maintenant et 30 jours avant
- `Sparkline` : liste de prix représentative de l'évolution des 30 derniers jours (ex. 1 point par jour ou 1 point toutes les 6h)

Les valeurs doivent être normalisées, cohérentes dans leur logique temporelle, et suffisamment précises pour être utilisées dans des interfaces ou algorithmes.

3.2 Fréquence de mise à jour

- Les données sont recalculées à la fin du chargement initial des symboles
- Une tâche planifiée relance périodiquement le calcul (toutes les 3h, 6h ou autre, à paramétrer)
- En cas de besoin, le service de calcul doit pouvoir être invoqué manuellement par API ou CLI

3.3 Robustesse et résilience

- En cas d'erreur d'appel à l'API Binance (timeout, dépassement de rate limit, indisponibilité), le système doit passer en **mode fallback** :
- Utiliser les dernières données valides en cache si possible
- Logger l'incident avec gravité
- Redémarrer les appels après une période définie (via timer ou circuit breaker)
- Le système ne doit pas planter ou geler si un seul symbole échoue (isolation par symbole obligatoire)

3.4 Observabilité / Monitoring

- Chaque calcul doit être loggé : durée, succès/échec, valeur produite
- Des métriques doivent être exposées (nombre de symboles traités, moyenne de variation, temps de réponse Binance, etc.)
- En cas d'erreur critique, une alerte doit être générée (EventBus ou Alerting externe)

3.5 Notifications temps réel

- Une fois les données calculées, un événement `SymbolUpdated` est émis via SignalR
 - L'UI peut ainsi se mettre à jour sans polling
 - Le moteur de trading peut s'abonner à des symboles stratégiques
-

4. Règles de gestion

1. Ne traiter que les symboles actifs (`Status == Trading` et `isSpotTradingAllowed == true`)
 2. Le prix de référence pour les variations est le `close` de la `kline` correspondant à la date cible (J-1, J-7, J-30)
 3. Si les données de référence sont absentes (ex : listing récent), la valeur est laissée à `null`
 4. Le calcul des données de marché est **asynchrone et découplé** du téléchargement initial
 5. En cas de redondance d'informations (ticker et kline), la priorité est donnée aux `klines`
-

5. Critères d'acceptation

- [] Le champ `CurrentPrice` est présent et exact pour chaque symbole actif
 - [] Les champs `Change24hPct` , `Change7dPct` , `Change30dPct` sont cohérents, précis, et non biaisés (même base de calcul)
 - [] Le champ `Sparkline.Data` contient une liste de points numériques récents (30 minimum)
 - [] Le calcul est déclenché automatiquement après l'import initial, et planifiable en tâche récurrente
 - [] Une erreur sur un symbole n'impacte pas les autres
 - [] Une perte de connectivité à Binance est détectée, loggée, et le fallback s'active
 - [] Des logs sont générés pour chaque traitement de symbole (succès ou échec)
 - [] L'UI reçoit un signal temps réel via SignalR à chaque mise à jour d'un symbole
-

6. # Hypothèses / Contraintes

- L'API Binance est la source unique de vérité pour les prix historiques (Klines) et les prix live (Ticker)
 - Le `SymbolDownloader` reste responsable uniquement des données statiques (métadonnées)
 - Le calcul dynamique est géré par un composant à part (`SymbolMarketStatsCalculator`)
 - L'architecture de la plateforme permet de publier des événements via EventBus ou SignalR
 - Les composants consommateurs peuvent tolérer une latence de quelques secondes à minutes
 - Aucun recalcul temps réel haute fréquence n'est requis pour cette première version (pas de 1s ou 5s)
 - Les données sparkline ne nécessitent pas une précision absolue, mais une **tendance visuelle exploitable**
-

7. Cas limites à prendre en compte

- Symbole nouvellement listé (moins de 7 ou 30 jours d'historique)
 - Données manquantes côté Binance pour une date précise
 - Fluctuation extrême du prix (ex : variation > +5000%)
 - Réponse partielle de Binance (limite de rate, ou pagination)
 - Stockage local corrompu ou inaccessible
 - Conflits d'écriture concurrente sur le `SymbolStore`
-

8. Exemple de valeur calculée (BTCUSDT fictif)

- `CurrentPrice` : 58_120.45 USDT
 - `Change24hPct` : -3.22%
 - `Change7dPct` : +6.91%
 - `Change30dPct` : +12.45%
 - `Sparkline` : [51200, 51930, 52800, 52150, ..., 58120]
-

9. Prochaine étape

- Valider cette spécification fonctionnelle avec l'équipe produit
 - Rédiger la **spécification technique** (ST-MKT-001)
 - Créer une **EPIC dans le release plan** : `EPIC-MKT-CALCULATION`
 - Décliner en **user stories** unitaires : traitement, rafraîchissement, résilience, observabilité, interface, tests
-

Souhaites-tu que je passe maintenant à la **spécification technique** ?