/\*\*  
 \* @file: src/ai/coordinators/README.md  
 \* Documentation du système de métriques  
 \*/

# Système de Métriques  
  
Ce module fournit une infrastructure complète pour la collecte, l'agrégation et l'analyse des métriques de performance de l'application LSF.  
  
## Caractéristiques  
  
- \*\*Collecte de métriques\*\* avec support pour différents types (gauge, counter, histogram, summary)  
- \*\*Agrégation automatique\*\* des métriques pour l'analyse des tendances  
- \*\*Gestion des seuils\*\* avec notification d'événements  
- \*\*Cache intelligent\*\* pour les métriques fréquemment accédées  
- \*\*Persistance configurable\*\* pour les données historiques  
- \*\*Support des métriques distribuées\*\* avec identification des noeuds  
- \*\*Export vers des systèmes externes\*\* comme Prometheus  
- \*\*Pattern Observable\*\* pour s'abonner aux mises à jour de métriques  
  
## Architecture  
  
Le système utilise une architecture modulaire avec des composants interchangeables :  
  
- `MetricsCollector` : Point d'entrée principal pour l'enregistrement et la récupération des métriques  
- `MetricsStorage` : Stockage des valeurs de métriques (mémoire, fichier, etc.)  
- `MetricsAggregator` : Agrégation des métriques brutes en statistiques utiles  
- `ThresholdManager` : Détection et notification des dépassements de seuils  
- `MetricsExporter` : Export des métriques vers des systèmes externes  
  
## Utilisation  
  
### Initialisation  
  
```typescript  
import { MetricsCollector } from '@ai/coordinators/services/metrics';  
  
// Créer une instance simple  
const metrics = new MetricsCollector('lsf-app');  
  
// Ou avec des options avancées  
const metrics = new MetricsCollector('lsf-app', {  
 config: {  
 maxHistorySize: 5000,  
 retentionPeriod: 7 \* 24 \* 60 \* 60 \* 1000, // 7 jours  
 },  
 nodeId: 'server-01'  
});

### **Enregistrement des métriques**

// Métrique simple  
metrics.recordMetric('api.requests', 1, 'counter');  
  
// Avec tags  
metrics.recordMetric('api.latency', 42.5, 'histogram', {   
 endpoint: '/translate',   
 method: 'POST'   
});  
  
// Mesure de durée  
const start = Date.now();  
// ... opération ...  
const duration = Date.now() - start;  
metrics.recordMetric('operation.duration', duration, 'histogram');

// Dernière valeur  
const latency = metrics.getMetric('api.latency');  
console.log(`Latence: ${latency?.value}ms`);  
  
// Filtrer par tags  
const translationLatency = metrics.getMetric('api.latency', { endpoint: '/translate' });  
  
// Historique  
const latencyHistory = metrics.getMetricHistory('api.latency', 10);  
  
// Statistiques  
const stats = metrics.getMetricStats('api.latency');  
console.log(`Min: ${stats?.min}ms, Max: ${stats?.max}ms, Avg: ${stats?.avg}ms`);

### **Configuration des seuils**

import { ThresholdManager } from '@ai/coordinators/services/metrics/alerts';  
  
const thresholdManager = new ThresholdManager();  
  
// Ajouter un seuil  
thresholdManager.addThreshold({  
 namespace: 'lsf-app',  
 metric: 'api.latency',  
 operator: 'gt',  
 value: 200, // ms  
 duration: 60000, // Doit être dépassé pendant 1 minute  
 severity: 'warning'  
});  
  
// S'abonner aux événements de seuil  
thresholdManager.addObserver({  
 update: (event) => {  
 console.log(`Alerte: ${event.config.metric} dépasse ${event.config.value}`);  
 }  
});  
  
// Fournir le gestionnaire de seuils au collecteur  
const metrics = new MetricsCollector('lsf-app', { thresholdManager });

### **Export vers Prometheus**

import { PrometheusExporter } from '@ai/coordinators/services/metrics/exporters';  
  
const exporter = new PrometheusExporter({  
 endpoint: '<http://prometheus:9091/metrics/job/lsf>',  
 interval: 15000 // 15 secondes  
});  
  
// Intégrer l'exporteur avec le collecteur  
metrics.addObserver({  
 update: (metricUpdate) => {  
 exporter.export([{  
 namespace: metricUpdate.namespace,  
 name: metricUpdate.name,  
 value: metricUpdate.value,  
 timestamp: metricUpdate.timestamp,  
 tags: metricUpdate.tags  
 }]);  
 }  
});

## Plan de migration  
  
Voici un plan de migration progressif pour passer de l'implémentation actuelle à cette nouvelle architecture :  
  
1. \*\*Étape 1 (Base)\*\* : Créer les fichiers de types et les interfaces de base  
2. \*\*Étape 2 (Isolement)\*\* : Implémenter le stockage mémoire et la collecte de base  
3. \*\*Étape 3 (Migration)\*\* : Adapter le système existant pour utiliser la nouvelle architecture  
4. \*\*Étape 4 (Extension)\*\* : Ajouter les fonctionnalités avancées une par une  
5. \*\*Étape 5 (Tests)\*\* : Déployer une batterie de tests exhaustifs  
6. \*\*Étape 6 (Documentation)\*\* : Documenter le système complet  
  
Cette approche nous permet d'améliorer progressivement le système tout en maintenant la compatibilité avec le code existant.