🚀 Piano di Refactoring e Monitoraggio: Ottimizzazione Simula.cs

Questo documento funge da **Master Tracker** per il processo di refactoring della classe `Simula.cs`. Serve a monitorare lo stato di avanzamento di ogni singolo intervento, garantendo che ogni modifica sia mappata e verificata.

📊 Stato di Avanzamento Generale

Progresso Totale: `[                    ] 0%`

Fase	Obiettivo	Stato
Fase 1	**Async Optimization** (Eliminazione Sync-over-Async)	🟣 NOT STARTED
Fase 2	**Throttling I/O** (Dirty Check & Rate Limiting)	🟣 NOT STARTED
Fase 3	**Jittering & Staggering** (Disallineamento istanze)	🟣 NOT STARTED
Fase 4	**Verbosity Control** (Logging Optimization)	🟣 NOT STARTED
Fase 5	**Verifica & Test** (Stress Test & Monitoraggio CPU)	🟣 NOT STARTED

🛠️ Dettaglio Interventi

1. ⚡ Fase 1: Async Optimization (Priorità Massima)

Obiettivo: Eliminare il context switching e il Thread Pool Starvation.

- [ ] **[TASK-1.1] Refactoring `IobGetDataFromServerToFtp`**
  - *Problema:* Uso di `.GetAwaiter().GetResult()` (Line 1670).
  - *Azione:* Trasformare il metodo in `async Task` e usare `await` per le chiamate `HttpService`.
- [ ] **[TASK-1.2] Refactoring `IobGetSendDossierKepware`**
  - *Problema:* Uso di `.GetAwaiter().GetResult()` (Line 1802).
  - *Azione:* Trasformare in `async Task`.
- [ ] **[TASK-1.3] Audit `Task.Run` (Fire-and-Forget)**
  - *Problema:* Potenziali fughe di thread o eccezioni silenziate (Line 158, 1003, 1597, 1748).
  - *Azione:* Implementare un wrapper `SafeTaskRun` con gestione log errori centralizzata.

2. 🛡️ Fase 2: Throttling I/O & Dirty Check

Obiettivo: Ridurre il traffico verso Redis e i servizi esterni.

- [ ] **[TASK-2.1] Implementazione Dirty Check `upsertKey`**
  - *Problema:* Invio continuo di dati anche se invariati (Line 232, 298, 327, 978).
  - *Azione:* Creare un `Dictionary<string, object> _lastSentValues` per validare i dati prima di `upsertKey`.
- [ ] **[TASK-2.2] Rate Limiting `decodeToBaseBitmap`**

    - *Problema:* Operazione pesante invocata troppo frequentemente (Line 612).
    - *Azione:* Implementare un controllo temporale (es. max 1 esecuzione ogni 500ms).
- [ ] **[TASK-2.3] Throttling `trySendPzCountBlock`**
    - *Problema:* Possibile saturazione chiamate HTTP per i contapezzi (Line 1246).
    - *Azione:* Limitare la frequenza di invio tramite timer o flag di stato.
- [ ] **[TASK-2.4] Ottimizzazione chiamate `HttpService`**
    - *Problema:* Chiamate REST ripetitive e non ottimizzate.
    - *Azione:* Implementare un meccanismo di caching temporaneo per i dati che non cambiano rapidamente.

3. 🎲 Fase 3: Jittering & Disallineamento

Obiettivo: Distribuire il carico computazionale nel tempo tra le diverse VM.

- [ ] **[TASK-3.1] Jitter di Inizializzazione**
    - *Problema:* Ritardo fisso o troppo breve all'avvio (Line 70).
    - *Azione:* Estendere il range del `Random startDelay` e assicurarsi che non blocchi il thread principale.
- [ ] **[TASK-3.2] Jittering dei Periodi di Polling**
    - *Problema:* Sincronizzazione dei task tra istanze diverse.
    - *Azione:* Applicare un rumore statistico (jitter) a ogni iterazione dei cicli di polling/timer.

4. 📝 Fase 4: Verbosity Control

Obiettivo: Ridurre l'overhead di I/O disco e CPU per il logging.

- [ ] **[TASK-4.1] Pulizia Log ad alta frequenza**
    - *Problema:* Log massivi di stringhe e memoria nei cicli critici (Line 972-975).
    - *Azione:* Spostare i log di debug/trace in modalità non invasiva o eliminarli se ridondanti.

🧪 Fase 5: Verifica e Validazione

Test Case	Risultato Atteso	Stato	Note
Stress Test CPU	Riduzione % CPU rispetto al baseline	🟣	
Monitoraggio Context Switch	Riduzione numero switch/sec	🟣	
Test Correttezza Dati	I dati inviati a Redis/MES sono identici	🟣	
Test Error Handling	Le eccezioni nei Task asincroni vengono loggate	🟣	

📓 Note e Osservazioni

Aggiungere qui osservazioni durante lo sviluppo.