Università di Catania

DISTRIBUTED SYSTEMS AND BIG DATA
A.A. 2024/2025

Homework 1

Studenti:

Fabio Gallone: 1000001752
Claudio D'Errico: 1000004015

Abstract

L'applicazione, sviluppata su microservizi, offre funzionalità di gestione utenti e analisi di dati finanziari in tempo reale da yfinance.

Il sistema è composto da:

- **Server gRPC**: gestisce le richieste del client per operazioni sugli utenti (registrazione, aggiornamento del ticker associato ad un utente, cancellazione) e fornisce dati finanziari (recuperare l'ultimo valore data l'email di un utente o calcolare la media degli ultimi N valori di uno specifico ticker). Implementa la politica **At-Most-Once** tramite il salvataggio di un identificatore univoco in memoria cache (TTLCache).
- **Data Collector**: è un servizio che raccoglie continuamente informazioni finanziare dall'API esterna yfinance, aggiornandoli sulla tabella del database "financial_data". Integra un **circuit breaker** per monitorare le chiamate a yfinance.
- **Database PostgreSQL**: archivia i dati relativi agli utenti (nella tabella "users") e alle informazioni finanziarie nella tabella "financial_data".
- **Client gRPC**: interfaccia utente per interagire con il sistema. Genera request_id univoci utilizzando un timestamp unito a una stringa casuale per garantire l'idempotenza delle operazioni.
- **Data Cleaner**: un processo automatico che rimuove periodicamente i dati storici più vecchi, mantenendo al massimo 20 record per ogni ticker, per garantire l'efficienza del database e delle prestazioni delle query.

Avendo implementato la politica **at-most-once**, se il client invia una richiesta e non riceve una risposta dal server (ad esempio, perché il server potrebbe essere temporaneamente offline) riprova utilizzando lo stesso request_id per garantire che l'operazione non venga eseguita due volte e per evitare sprechi di risorse andando a interrogare il database inutilmente nel breve periodo. Inoltre, se il server è temporaneamente offline, il client tenta di ristabilire la connessione al server prima di ripetere la richiesta, migliorando la resilienza del sistema. Questo impedisce l'esecuzione di query ridondanti e riduce il carico sul database, un aspetto particolarmente importante in sistemi con un elevato numero di utenti.

Inoltre, è stato implementato un **circuit breaker** per migliorare la resilienza del sistema. In caso di ripetuti fallimenti nelle richieste al servizio **yfinance** (es.: connessione persa), il circuit breaker interrompe temporaneamente le richieste verso tale servizio. Ciò evita richieste inutili e previene il sovraccarico del sistema, migliorando l'esperienza utente.

Fabio Gallone e Claudio D'Errico

Diagramma architetturale

Il diagramma rappresenta l'architettura dell'applicazione, mostrando le principali interazioni tra i microservizi:

- **Server gRPC** comunica con il **Database PostgreSQL** per memorizzare e recuperare dati utente e dati finanziari.
- **Data Collector** aggiorna il **Database PostgreSQL** con i dati finanziari più recenti raccolti da fonti esterne.
- **Data Cleaner** esegue la pulizia periodica dei dati obsoleti nel **Database PostgreSQL**, mantenendo il sistema efficiente.

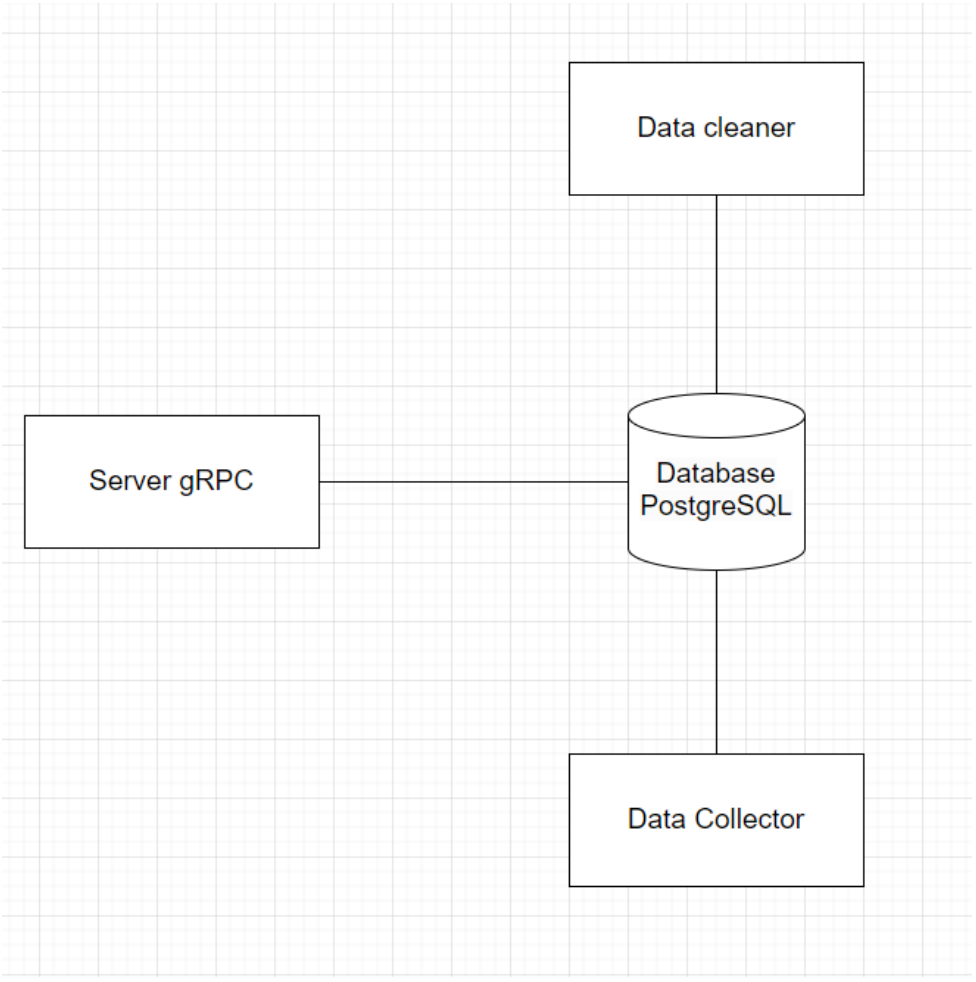
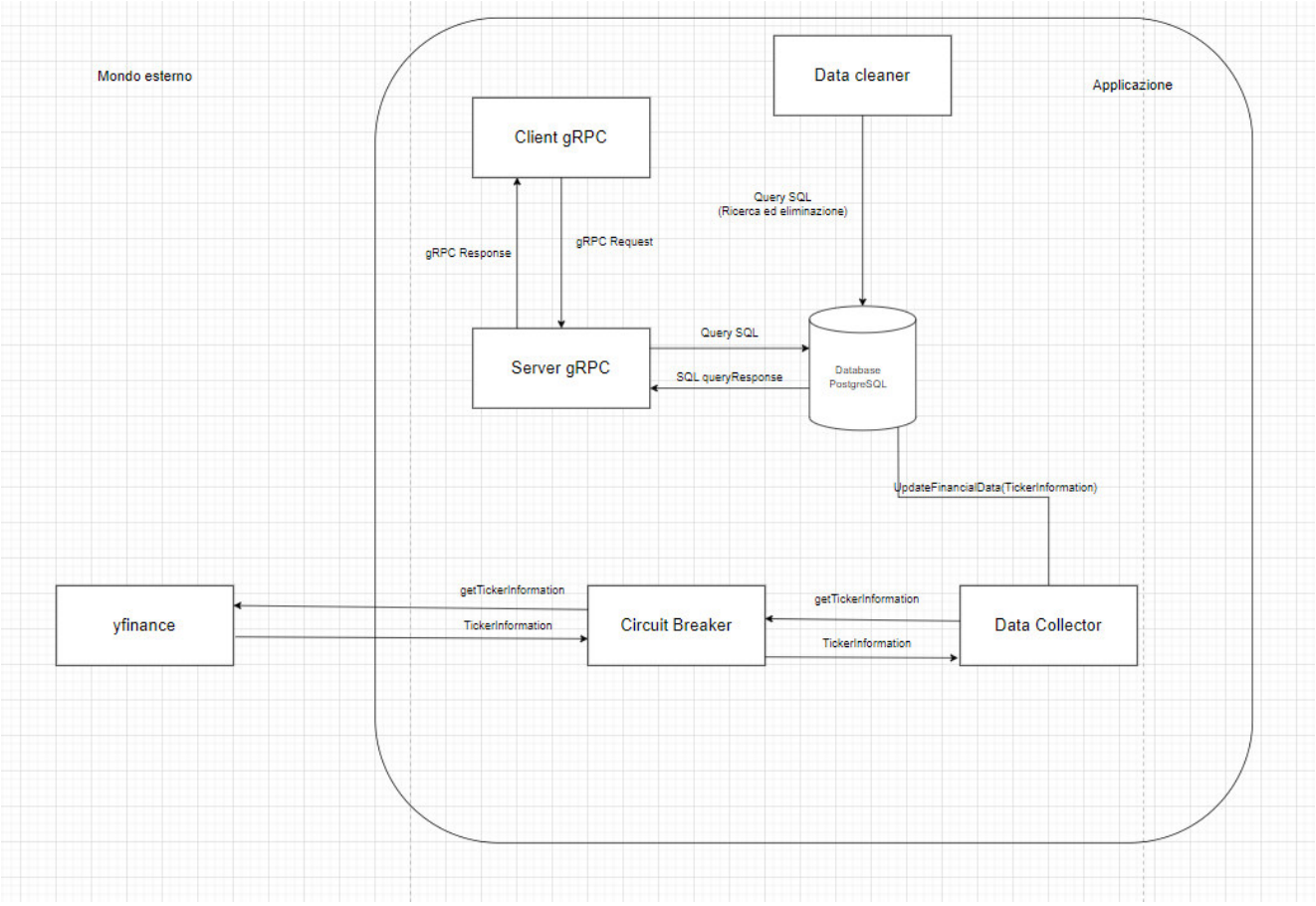


Diagramma delle Interazioni

Il diagramma illustra il flusso delle interazioni tra i componenti principali dell'applicazione:

- **Client gRPC** invia richieste al **Server gRPC** e riceve risposte per le operazioni sugli utenti e sui dati finanziari.
- **Server gRPC** interagisce con il **Database PostgreSQL** per leggere e scrivere dati.
- **Data Collector** raccoglie dati finanziari da yfinance tramite un **Circuit Breaker** e aggiorna il **Database PostgreSQL**.
- **Circuit Breaker** protegge il sistema da errori nelle chiamate a yfinance.
- **Data Cleaner** esegue una pulizia periodica nel Database **PostgreSQL**, eliminando i dati finanziari obsoleti.



Lista delle API implementate

Nome API	Descrizione	Messaggio di Richiesta	Messaggio di Risposta
RegisterUser	Registra un nuovo utente con un ticker di interesse.	**RegisterUserRequest** – email (string) – ticker (string) – request_id (string)	**RegisterUserResponse** – message (string)
UpdateUser	Aggiorna il ticker di un utente esistente.	**UpdateUserRequest** – email (string) – ticker (string) – request_id (string)	**UpdateUserResponse** – message (string)
DeleteUser	Elimina l'account di un utente.	**DeleteUserRequest** – email (string) – request_id (string)	**DeleteUserResponse** – message (string)
LoginUser	Esegue il login di un utente tramite l'email.	**LoginUserRequest** – email (string)	**LoginUserResponse** – message (string) – success (bool)
GetLatestValue	Recupera l'ultimo valore disponibile per il ticker dell'utente.	**GetLatestValueRequest** – email (string)	**GetLatestValueResponse** – email (string) – ticker (string) – value (double) – timestamp (string)
GetAverageValue	Calcola la media degli ultimi X valori per il ticker dell'utente.	**GetAverageValueRequest** – email (string) – count (int32)	**GetAverageValueResponse** – email (string) – ticker (string) – average_value (double)