IDEMPOTENT RECEIVER

Idempotent Receiver

- Intento: Identifica le richieste dei client univocamente così da poter ignorare le richieste duplicate quando il client ritenta
- Problema
  - Il client potrebbe mandare una richiesta ma non ricevere una risposta; per il client è impossibile sapere se la risposta è andata persa, o il server si è bloccato prima di elaborare la richiesta
  - Il client quindi deve inviare nuovamente la richiesta. Se il server aveva già ricevuto la richiesta, l'aveva elaborata e dopo si era bloccato, allora il server riceverà una richiesta duplicata dal client

Idempotent Receiver - Esempio

- Un servizio di pagamento riceve la richiesta: "Addebita 100 Euro a Bob"
- La rete è instabile quindi il messaggio viene ritrasmesso una seconda volta
- Se il servizio non è idempotente allora Bob paga 200 Euro
- Molti sistemi usano il pattern "at-least-once delivery", ovvero un messaggio è mandato ***non esattamente una volta*** ma ***almeno una volta*** per garantire affidabilità e ridurre i tempi di attesa, di conseguenza la ricezione duplicata è inevitabile
- Pertanto, il receiver deve essere progettato per essere idempotente
- **Una operazione idempotente è una operazione che può essere eseguita tante volte con lo stesso effetto di essere eseguita una sola volta**. Per es. impostare la variabile a un certo valore è una operazione idempotente

Soluzione

- Il receiver deve far sì che l'elaborazione di un messaggio più volte produca lo stesso effetto della sua elaborazione una sola volta
- Passi della soluzione
  1. Il client crea un ID univoco per la richiesta per es. numerando la richiesta, o tramite hash dei dati che costituiscono la richiesta; oppure chiede al server di creare un ID della richiesta
  2. Il server crea una sessione per immagazzinare le risposte alle richieste dei client registrati e traccia il tempo dell'ultimo accesso del client
  3. Quando il server riceve una richiesta, anzitutto controlla se la richiesta con quell'ID e da quel client è stata già elaborata. Se il server trova la richiesta conservata manda la stessa risposta al client (senza elaborare nuovamente la richiesta), altrimenti elabora la richiesta e ne conserva il risultato

Rimozione Richieste Conservate

- Le richieste già elaborate e conservate potrebbero essere cancellate dal server dopo un timeout
- Le richieste del client potrebbero essere numerate (in modo progressivo), il server può quindi rimuovere le richieste salvate che hanno numero minore a quella appena ricevuta dallo stesso client

Struttura

- MessageReceiver: riceve e valida il messaggio, elabora il messaggio solo se non ancora gestito
- MessageDB: conserva l'ID dei messaggi già elaborati (in memoria per migliorare le prestazioni, o su database)

Conseguenze

- Benefici
  - Garantisce l'effetto di exactly-once anche con at-least-once delivery
  - Migliora l'affidabilità e la consistenza dell'applicazione
  - Facilita l'integrazione in architetture a microservizi. Quando ci sono più microservizi, uno potrebbe servire una richiesta già servita da altri, tramite la condivisione del DB dei messaggi si creano microservizi idempotenti
- Compromessi
  - Maggiore occupazione di memoria e tempo d'esecuzione più lungo per la necessità di cercare gli ID delle richieste su storage
  - Oppure, richieste salvate in memoria per accorciare i tempi di esecuzione
  - Necessità di gestire la scadenza delle richieste registrate
  - Necessità di messaggi con ID univoci

Esempio

- Service consente di leggere e aggiornare i dati di un utente. Per far sì che le operazioni siano idempotenti, i messaggi sono numerati e ServiceImpl dovrà conservare i messaggi ricevuti