RELAZIONE ASSIGNMENT 4 PCD

Semprini Luca - 0000854447
Riciputi Jacopo

Questo assignment richiedeva di implementare un servizio di chat distribuito, prendendo come riferimento le architetture orientate ai servizi e reattive, e aspetti metodologici correlati.

Si è scelto di utilizzare il linguaggio Scala per l'implementazione dell'esercizio, integrandolo con la parte grafica scritta con JavaFX. L'applicativo si suddivide in due parti: Server e Client. Lato Server si utilizza il framework Vert.x (nella versione Core e Web per la creazione del server http) e la tecnologia di Redis per la parte di database; lato Client è stato utilizzato il framework Akka per la comunicazione fra GUI e modello e la libreria **Vert.x web client** per le chiamate REST.

SERVER

La parte Server prevede un backend sviluppato in Scala, che si appoggia sul framework **Vert.x** per la realizzazione di un server HTTP e a Redis per la parte di database e scambio di messaggi, grazie al protocollo PUB/SUB implementato da quest'ultimo.

Il **server HTTP** si occupa di rispondere a determinate chiamate, data la semplicità e l'utilizzo "limitato", gestisce solamente chiamate effettuate con metodo **GET** o **POST**, su determinati URL, configurati appositamente per rispondere alle esigenze avanzate dal client. È possibile infatti dialogare con il server tramite chiamate **RESTFul** così da ottenere o impostare dati presenti su di esso. La scelta di **Vert.x** è dettata dal suo rapido deploy e dalla comodità offerta per la gestione di questo tipo di chiamate e inizialmente anche per la presenza di una libreria client per Redis, successivamente scartata per **redisscala**, ritenuta maggiormente adeguata, inoltre essendo anch'essa non bloccante si sposa perfettamente con il framework scelto.
Per la parte di database invece si è optato per **Redis**, per vari motivi, ma principalmente per le sue caratteristiche NoSQL che si sposano perfettamente a scalabilità e flessibilità del sistema, le capacità prestazionali che offre e per il protocollo PUB/SUB, utilizzato per sviluppare lo scambio di messaggi tra gli utenti all'interno dell'applicazione e per il ricevimento di notifiche da parte del server.

I due blocchi per lo sviluppo del backend sono entrambi funzionanti in un ambiente cloud.
Il server di Vert.x è stato configurato in un dyno di Heroku e avviato, mentre l'istanza Redis è "hosted" sulla piattaforma offerta da Redis Labs.

CLIENT

Il modello del client è strutturato prevalentemente in maniera Actor-based. Si usa infatti il framework Akka per definire i principali comportamenti del sistema, dalle interazioni REST alla GUI. L'attore `RestClient` si occupa di effettuare le chiamate http al Server e risponde all'occorrenza all'attore chiamante (prevalentemente `GUIActor`).
Gli attori responsabili del comportamento della GUI sono `GUIActor` e `PreGUIActor`, i quali si relazionano rispettivamente con le classi di controllo `MainViewController` e `InitialWindowController`. `PreGUIActor` si occupa principalmente di comunicare al `RestClient` l'intenzione da parte dell'utente di accedere al sistema, con uno username

specificato; se lo username non verrà trovato all'interno del database, si richiederà all'utente se desideri creare un nuovo account o inserire un differente username. Nel caso la richiesta dell'utente abbia successo (nuovo account creato oppure login effettuato con uno username esistente), l'attore si occuperà di caricare la finestra principale; se la richiesta fallisce verrà mostrato un messaggio di errore e sarà possibile per l'utente riprovare.
Una volta caricata la View principale, verrà mandato un messaggio di attivazione al `GUIActor`, che, grazie a questo trigger, si occuperà di richiedere al `RestClient` la lista delle chat globali (quindi tutte le chat presenti sul database, non solo quelle a cui è iscritto l'utente loggato) e, una volta ricevute le esporrà nell'apposita lista, su cui sarà possibile fare una operazione di filtraggio se si desidera visualizzare unicamente le chat a cui si è iscritti. L'attore GUIActor interagisce quasi continuamente con il RestClient, in particolare nelle operazioni di creazione di una nuova chat, di rimozione di una chat, di adesione ad una chat esistente di cui non si faceva parte, di abbandono di una chat di cui si faceva parte e di invio di un messaggio su una chat selezionata. Quest'ultima operazione richiede l'intervento di un ulteriore attore (`ChatActor`), che è wrappato all'interno della classe di modello `ChatWrapper`, e che gestirà l'invio dei messaggi sulla chat e la notifica di eventuali errori.

NOTE FINALI

- **DEPLOY:** Per il Server e il Client sono richieste alcune variabili di ambiente da impostare:
  - **REDIS_HOST**=redis-16418.c3.eu-west-1-2.ec2.cloud.redislabs.com;
  - **REDIS_PORT**=16418;
  - **REDIS_PW**=6Ez3LCurX2hQl6S0SQNyNNQjQ5vLvXCu;
  - **PORT**=4700 (Solo per il server)

- **LIMITI**: L'applicazione a livello di modello non presenta limitazioni, il numero di chat, il numero di utenti o di messaggi è libero, vi è solo un piccolo limite dettato da Redis Labs per cui non è possibile avere più di 30 client connessi contemporaneamente all'istanza, dato che si tratta di una versione di prova gratuita. Perciò, dato che lato applicativo si ha per ogni chat un client di Redis, connesso direttamente al database, in caso di login da parte di più utenti e di un numero di chat non limitato, si potrebbe incappare in un blocco dell'applicazione dovuto al rifiuto di collegamento da parte di chi fornisce l'istanza del database.

- **UTENTI PER L'UTILIZZO:** Oltre alla possibilità di creare nuovi utenti, possono essere usati i seguenti (già creati) per effettuare delle prove di utilizzo dell'applicativo:
  - **Jacopo47**
  - **lucasempr**
  - **prova**

Link ai repository:
https://github.com/Jacopo47/assignment04-chat-server
https://github.com/lucasemprini/Assignment04-chat-client