WORTH

Progetto finale A.A. 2020/21

Laboratorio di Reti - Corso A

Giuseppe Muschetta

Matricola: 564026

1 Introduzione

Il progetto WORTH consiste nell'implementazione di uno strumento per la gestione di progetti collaborativi ispirato ad alcuni principi della metodologia Kanban. Quest'ultimo è un metodo di gestione "agile" di un insieme di attività che permette di monitorare la loro evoluzione attraverso una vista sulle fasi del progetto.

Il servizio offre funzioni per la creazione e gestione di progetti formati da una serie di attività, che in seguito chiameremo "card", da portare a termine. Inoltre, ogni progetto dispone di una chat utilizzabile solo dai membri che ne fanno parte.

Il sistema è stato progettato seguendo il classico paradigma Client-Server, utilizzando principalmente TCP per la comunicazione e lo scambio di messaggi di richiesta e di risposta.

Per la fase di registrazione è stata usata la tecnologia RMI e il meccanismo delle callback. Le chat sono state implementate in modalità multicast tramite UDP.

Lo stato del sistema è persistente: i dati che riguardano gli utenti registrarti, i progetti creati con i relativi membri e con le relative card persistono sul file system anche quando il server non è in esecuzione.

2 Scelte di progetto

2.1 Interazione con l'utente

L'utente interagisce con il servizio attraverso un'interfaccia a linea di comando (CLI), un menù che guida l'utente nelle funzionalità del sistema attraverso il comando help. Durante l'interazione verranno mostrati all'utente solamente i comandi che possono essere utilizzati in quel momento, ad esempio se l'utente non ha ancora effettuato il login potrà solamente usufruire dei comandi di registrazione e login e non di tutte le altre funzionalità offerte dal servizio. Il menù è modellato dalla classe ClientMenu, che prende in input tutto ciò che l'utente scrive sul terminale, verifica la correttezza del comando e invoca il metodo definito nella classe ClientCore, la quale elabora effettivamente la richiesta dell'utente comunicando con il server.

2.2 Client

Il core del client definisce tutte le operazioni che possono essere richieste dall'utente interagendo con il menù. In particolare, contiene tutti i metodi che implementano la comunicazione con il server, i quali vengono chiamati dalla classe ClientMenu una volta che ha interpretato un comando dell'utente.

2.2.1 Avvio client e connessione

La classe ClientCore implementa il metodo begin() che viene chiamato all'avvio del client nella classe ClientMain. Al suo interno, viene recuperato il riferimento allo stub del server mantenuto nel servizio di registry, per utilizzare tramite RMI i metodi remoti del server utili alla registrazione dell'utente al servizio e del client alle callbacks. In seguito, viene creato l'oggetto ClientMenu e chiamato il relativo metodo che gestisce l'interazione con l'utente. Il client si connette al server quando dall'utente viene richiesta l'operazione di login e si disconnette quando viene invece richiesta l'operazione di logout. Il tutto è racchiuso all'interno di un blocco try che permette al client di terminare in modo sicuro anche quando non si trova in una situazione di operatività normale. Ad esempio, può essere sollevata una IOException, quando il client cerca di connettersi o cerca di comunicare con il server che non è raggiungibile. In questo caso il client annulla l'esportazione del proprio oggetto (che sarebbe servito al server per effettuare le callback), interrompe eventuali thread che erano stati avviati per ricevere i messaggi dalle chat (come sarà descritto in seguito) ed infine chiude il socket.

2.2.2 Comunicazione client-server

Il client comunica con il server utilizzando il protocollo di trasporto TCP, attraverso DataInputStream e DataOutputStream opportunamente bufferizzati concatenando i rispettivi BufferedStream. In questo modo l'utilizzo degli stream risulta più efficiente, minimizzando le chiamate di sistema per l'I/O, che vengono fatte solamente quando si hanno abbastanza byte da leggere e da scrivere. I dati scambiati tra client e server sono incapsulati in un oggetto che viene serializzato al momento dell'invio e deserializzato al momento della ricezione, in formato json.

Sugli stream vengono scritti i bytes ottenuti dalla stringa che rappresenta l'oggetto json. L'oggetto usato nella comunicazione viene modellato dalla classe Message, che incapsula tutti i dati che possono essere spediti tra client e server, quindi prima di tutto richieste (lato

client) e responsi (lato server), poi oggetti come, ad esempio, liste di progetti o card che l'utente può richiedere al server per visualizzare lo stato. A causa dell'invio di tipi di oggetti diversi durante la comunicazione, la dimensione dei dati ricevuti non può essere nota a priori. Per questo motivo sia client che server, prima di inviare sullo stream i byte dell'oggetto che incapsula il messaggio, scrivono un numero intero che indica la dimensione effettiva dei dati che si vuole spedire. In questo modo il ricevente effettuerà prima una read dell'intero che indica la dimensione dei dati, per poi allocare un array di byte che andrà a contenere esattamente i byte da leggere corrispondenti al dato ricevuto.

2.2.3 Gestione chat utente

Il client definisce anche alcuni metodi che non utilizzano TCP per soddisfare le richieste dell'utente. Si tratta dei metodi, riguardanti la chat di ogni progetto di cui fa parte l'utente, che utilizzano UDP.

In particolare, ogni messaggio che l'utente vuole scrivere sulla chat viene incapsulato in un datagramma UDP che poi viene spedito attraverso un DatagramSocket all'indirizzo della chat. Per quanto riguarda la ricezione dei messaggi della chat, vengono avviati tanti thread quante sono le chat dell'utente. Questi thread eseguono il task ChatSaver che utilizza un MulticastSocket per iscriversi al gruppo utilizzando l'indirizzo della chat e la porta, quest'ultima uguale per tutte le chat, sfruttando la proprietà reuse socket, settata a true da default. In questo modo ogni thread intercetta tutti i messaggi spediti su quella chat e li salva su una lista che poi l'utente può consultare successivamente. Per ogni thread che viene avviato, il client salva il suo riferimento, per poi utilizzarlo per interromperlo quando non è più utile ricevere e salvare i messaggi da quella determinata chat.

2.2.4 Callbacks di aggiornamento

Il client, definito in particolare dalla classe ClientCore, implementa l'interfaccia remota ClientInterface che contiene i metodi utili al server nelle fasi di callback. Il server invocherà tali metodi da remoto sullo stub del client, che quest'ultimo include nella fase di registrazione per le callback. La registrazione avviene dopo che il client ha ricevuto il responso positivo alla richiesta di login fatta dall'utente. Se ci sono cambiamenti negli utenti registrati o nei progetti creati, al client viene notificato di ricevere le liste aggiornate che poi potrà usare per aggiornare le proprie liste locali contenute nella classe User.

2.3 Server

2.3.1 Avvio del server e persistenza dei dati

All'avvio del server, tramite il metodo begin(), chiamato dal main della classe ServerMain, viene ripristinato lo stato del sistema se esiste già la directory res all'interno della root del progetto. All'interno di questa cartella viene mantenuta la persistenza dello stato del sistema. Se la cartella non esiste viene creata la struttura delle directory e dei file che conterranno i dati.

I dati vengono organizzati con una directory per ogni progetto, in cui sono contenuti i file .json delle card e dei membri del progetto e un file .json che contiene tutti gli utenti registrati al sistema.

Le operazioni di lettura e scrittura da e per tali file sono effettuate utilizzando NIO, dopo aver opportunamente serializzato o deserializzato gli oggetti da scrivere o leggere nei file. Ogni operazione che modifica lo stato del sistema causa l'immediato aggiornamento dei file tramite i metodi saveUsers(), saveProjects().

Dopo aver ripristinato l'eventuale stato del sistema, il server esporta il suo riferimento e lo registra nel servizio di registry appena creato. Grazie a questo, i client potranno utilizzare i metodi remoti del server. Infine, il server entra in un loop infinito in cui si blocca sull'operazione di accept() del welcome socket (listening socket), con la quale aspetta nuove connessioni da parte dei client.

2.3.2 RMI e il meccanismo delle callbacks

Il server con la classe ServerCore implementa l'interfaccia remota ServerInterface che definisce i metodi che vengono utilizzati tramite RMI dai client per registrarsi alle callback e per registrare l'utente al sistema.

Ogni client che effettua il login con il proprio utente, chiama il metodo remoto registerForCallbacks() (dualmente chiama il metodo unregisterForCallbacks() quando fa il logout) sullo stub del server e come argomento passa il proprio riferimento. In questo modo il server lo aggiunge alla propria lista di riferimenti ai client a cui dovrà fare le callback. All'interno dello stesso metodo il server esegue il metodi privati updateClientUsers() e updateClientChats() che invocano i metodi remoti sugli stub dei client per effettuare le callback. Questi metodi vengono utilizzati dal server quando lo stato del sistema cambia, ad esempio quando un utente si registra o effettua il login oppure quando viene creato o cancellato un progetto.

2.3.3 Gestione richieste client

Nel momento in cui un client si connette viene creato un socket per la comunicazione che viene passato al task ServerThread che implementa la gestione della richiesta del client. Il task viene sottomesso al threadpool che definisce la natura multithreading del server. Ad ogni client viene associato un thread del pool, seguendo le politiche del cachedThreadPool. La gestione della richiesta consiste nel leggere dallo stream associato al socket del client e deserializzare l'oggetto Message ricevuto. La richiesta del client viene interpretata, e sulla base di questa vengono chiamati i metodi definiti all'interno della classe WorthCore. Quest'ultima classe implementa le operazioni che permettono ai client di modificare e visualizzare lo stato del sistema, gestisce le strutture dati che definiscono lo stato del progetto: la lista degli utenti registrati e la lista dei progetti creati. Il task riceve il risultato dell'elaborazione della richiesta che corrisponde ad un oggetto Message. Quest'oggetto, contenente il responso ed eventuali altri dati richiesti, viene serializzato e inviato al client, il quale potrà leggerlo e dare la risposta per l'operazione richiesta dall'utente.

2.3.4 Disconnessione di un client

Può succedere che al momento della lettura della richiesta del client dallo stream, quest'ultimo sia disconnesso. In questo caso viene lanciata una EOFException che viene gestita dal task completando in modo sicuro la disconnessione del client: viene fatto il logout dell'utente e viene chiuso il socket. Un altro problema che si può riscontrare in seguito alla disconnessione di un client riguarda il relativo riferimento contenuto nella lista dei client registrati per le callback. Questa situazione viene gestita dal thread principale del server all'interno dei metodi updateClientUsers() e updateClientChats() eliminando il riferimento dalla lista quando viene sollevata un'eccezione in seguito al tentativo di effettuare la callback.

2.4 Implementazione del servizio WORTH

Come anticipato precedentemente, la classe WorthCore implementa il cuore dello stato e delle funzionalità del sistema. Essa implementa l'interfaccia WorthInterface la quale definisce le operazioni che il servizio deve garantire. I metodi implementati da questa classe modificano le strutture dati che contengono gli utenti registrati e i progetti creati e pertanto al loro interno vengono effettuate le chiamate ai metodi saveUsers() e saveProjects(), implementati nella classe ServerCore, per rendere subito effettive le

modifiche. Inoltre, vengono gestiti gli accessi concorrenti alle strutture in modo da evitare inconsistenze sui dati.

2.4.1 Gestione concorrenza

Dato che questi metodi possono essere utilizzati in modo concorrente da più thread del pool che gestisce le richieste dei client, si è ritenuto opportuno sincronizzare le operazioni che accedono e modificano le liste degli utenti e dei progetti. Tramite appositi blocchi sincronizzati sulle strutture dati di cui sopra, vengono garantiti accessi atomici evitando in toto inconsistenze sui dati.

2.4.2 Range indirizzi multicast per le chat

La classe infine definisce il range degli indirizzi che possono essere assegnati alle chat dei progetti e anche come vengono assegnati. Alla creazione del server viene creato un nuovo oggetto WorthCore che nel proprio costruttore inizializza le liste degli utenti, dei progetti e crea un TreeSet a cui viene passato un Comparator che permette di avere tale insieme ordinato in ogni momento.

Subito dopo l'insieme viene riempito con tutte le stringhe corrispondenti agli indirizzi multicast disponibili per l'assegnazione alle chat, fissando il range dall'indirizzo 239.255.224.0 all'indirizzo 239.255.255.255, in modo da avere la possibilità di creare al più 8K chat che corrispondono anche al numero massimo di progetti. Infatti, nell'eventualità che vengano creati tutti i progetti possibili, la creazione di un ulteriore progetto è fermata dall'impossibilità di assegnare un indirizzo alla chat. Infatti, il set si riduce man mano che vengono assegnati indirizzi alle chat, fino ad arrivare possibilmente ad essere vuoto e questo vuol dire che si è arrivati al numero massimo di chat/progetti possibili.

La procedura di assegnazione di un indirizzo corrisponde all'operazione di pollFirst() dall'insieme, che rimuove e restituisce il primo indirizzo disponibile nel range che sarà anche il più piccolo dato l'ordinamento automatico dell'insieme. La creazione di un nuovo progetto, nel caso in cui tutti gli indirizzi sono stati assegnati, deve attendere la cancellazione di uno dei progetti già creati. Quando un progetto viene cancellato, viene annullata l'assegnazione dell'indirizzo alla propria chat. Tale indirizzo viene riaggiunto all'insieme, tornando ad essere disponibile per altre chat.

3 Descrizione classi

Descrizione breve delle classi e della loro funzione all'interno del sistema.

- User : definisce l'utente gestito dal client
- Project : definisce un progetto del servizio
- Chat : definisce la chat di un progetto del servizio
- Card : definisce una attività da portare a termine in un progetto
- ServerMain : contiene il main da cui avviare il server. Scrive su un file le porta su cui è in ascolto e la porta del servizio di registry
- ServerInterface : interfaccia remota usata dal client per la registrazione al sistema e alle callback utilizzando RMI sull'istanza del server recuperata dal registry
- ServerCore : implementa l'interfaccia remota ServerInterface e definisce la struttura multithreading del server. Gestisce le richieste di connessioni da parte di più client che successivamente delega ai task ServerThread eseguiti dal thread pool. Gestisce la persistenza dei dati e il meccanismo delle callback tramite RMI.
- ServerThread : definisce il task che viene eseguito da un thread del pool del server, per soddisfare le richieste del client assegnato. Gestisce la comunicazione con il client attraverso stream, anche nel caso in cui il client si disconnetta senza preavviso.
- WorthInterface : interfaccia del servizio che include le operazioni offerte dal servizio WORTH
- WorthCore : implementazione dell'interfaccia WorthInterface, definisce lo stato e le funzionalità che garantisce il servizio WORTH. Gestisce la consistenza dei dati e il range degli indirizzi delle chat
- ClientMenu : gestisce l'interazione con l'utente. Riceve l'input dal terminale e lo interpreta controllando il formato del comando inserito. Chiama i metodi forniti da ClientCore in base all'operazione richiesta ed infine mostra il risultato all'utente
- ClientMain : contiene il main da cui avviare il client
- ClientInterface : interfaccia remota usata dal server per effettuare le callback. Tramite RMI il server chiama i metodi sull'istanza del client esportata, notificando il client in seguito a cambiamenti avvenuti sulle liste degli utenti e dei progetti

- ClientCore : implementa l'interfaccia remota ClientInterface e definisce le operazioni che l'utente può richiedere e la conseguente comunicazione con il server
- HashPassword : effettua l'hash di una password, utile per non salvare in chiaro le password degli utenti e nella fase di login
- Message : definisce i campi e i vari setter e getter dei messaggi scambiati tra client e server
- Request : enumerazione che definisce i tipi di richiesta che può fare il client al server
- Response : enumerazione che definisce i tipi di responso che il server può mandare al client
- ChatSaver : definisce il task che riceve e salva i messaggi inviati sulla chat, per future consultazioni

4 Threads attivati e strutture dati

In questa sezione vengono specificati i thread attivati dal client e dal server quando sono in esecuzione.

4.1 Threads attivati dal client

Il client attiva i seguenti thread:

- Thread principale: avviato dal main della classe ClientMain, gestisce l'interazione con l'utente e la comunicazione TCP con il server utilizzando le classi ClientCore, ClientMenu e User.
- Threads Savers: vengono avviati dal main thread del client uno per ogni chat di cui l'utente fa parte. Ogni thread esegue il task ChatSaver che legge e salva i messaggi della chat. I thread restano sempre in esecuzione e si bloccano sulla receive del multicast socket in attesa di nuovi messaggi. I thread vengono interrotti quando l'utente effettua il logout, oppure vengono interrotti singolarmente quando l'utente non più fa parte della relativa chat.
- Thread RMI: si attiva subito dopo l'esportazione dello stub del client e resta in attesa di invocazioni sui metodi remoti dell'interfaccia ClientInterface

4.2 Strutture dati del client

Le strutture principali gestite dal client sono le liste all'interno della classe User che mantengono la lista degli utenti iscritti al sistema e la lista delle chat di cui l'utente fa parte. Tali liste vengono aggiornate alla ricezione di una callback grazie alla quale il client riceve dal server le liste aggiornate. Un'altra struttura dati importante è la lista dei riferimenti ai thread attivati per ricevere i messaggi dalle chat. Il client utilizza questa lista per interrompere i task quando non è più necessario ascoltare e salvare i messaggi della chat come descritto nel paragrafo 2.2.3.

4.3 Threads attivati dal server

Il server attiva i seguenti thread :

- Thread principale: avviato dal main della classe ServerMain, accetta le connessioni dei nuovi client, delegando la gestione della richiesta e del responso al threadpool che esegue il task ServerThread.
- Threads del cached thread pool : ad ogni client connesso viene associato un thread del pool che esegue il task ServerThread, che legge dallo stream del socket del client la Request incapsulata nell'oggetto Message inviato dal client. Una volta interpretata la richiesta, il thread si avvale dei metodi della classe WorthCore per elaborarla ottenendo così un Message che incapsula il Response da inviare al client. Questo thread, quindi, gestisce tutta la fase di comunicazione con il client e, se sono state richieste operazioni che modificano lo stato del sistema all'interno dei metodi di WorthCore, vengono effettuate le chiamate ai metodi saveUsers() e saveProjects() della classe ServerCore che permettono di salvare immediatamente le modifiche. Infine, viene gestita anche l'eventuale disconnessione senza preavviso del client, disconnettendo l'utente e chiudendo il socket.
- Thread RMI: si attiva subito dopo l'esportazione dello stub del server rimanendo in attesa di invocazioni sui metodi remoti dell'interfaccia ServerInterface

4.4 Strutture dati del server

Le strutture principali gestite dal server sono la lista degli utenti registrati al sistema, la lista dei progetti creati, l'insieme che mantiene gli indirizzi multicast disponibili per

l'assegnazione alle chat, la lista dei riferimenti ai client registrati per le callback ed infine il cached thread pool utilizzato per la gestione della comunicazione con i client.

Le prime tre strutture dati fanno parte della classe WorthCore che contiene anche i metodi sincronizzati che operano su di esse. In particolare, queste strutture definiscono lo stato del sistema in ogni momento e sono accedute dai thread del pool assegnati ai client per elaborare le richieste di la modifica e visualizzazione dei dati contenuti.

Le restanti strutture fanno parte della classe ServerCore e sono accedute dal thread principale. Il threadpool è, come si è già potuto intuire, instanziato in modalità cached, utile al server per delegare l'intera fase successiva alla connessione di un client, in modo da essere sempre pronto ad accettare nuove connessioni.

Ad ogni client viene assegnato un nuovo thread del pool se non ci sono thread liberi, altrimenti viene riutilizzato uno che è inattivo da meno di 60 secondi. Al passare di questo tempo un thread del pool che non riceve nuove richieste da client viene terminato. La lista dei riferimenti agli stub dei client è utile per il meccanismo delle callback. Nello specifico, il server itera su questa lista quando c'è stato un cambiamento di stato nelle liste degli utenti registrati ed in quella dei progetti creati. Per ogni client che si è registrato per le callback (tutti i client si registrano al momento del login dell'utente) vengono invocati i metodi remoti della ClientInterface ai quali sono passati come argomento le liste aggiornate.

5 Librerie esterne utilizzate

Le librerie esterne utilizzate sono jackson-annotations, jackson-databind e jackson-core, tutte in versione 2.9.7. Le librerie sono state utilizzate per effettuare la serializzazione degli oggetti da Java a Json e la deserializzazione da Json a Java. In particolare, sono state utili per la serializzazione dell'oggetto Message utilizzato nella comunicazione TCP tra il client e il server e per la serializzazione dei dati da scrivere nei file che mantengono la persistenza dello stato del sistema. Nell'ultimo caso la lettura e scrittura dei file è stata fatta manualmente utilizzando canali e buffer di NIO, ma si sarebbero potuti utilizzare i metodi writeValue() e readValue() di Jackson che in una istruzione scrivono e leggono rispettivamente l'oggetto dal file facendo anche la serializzazione e deserializzazione. I file .jar, relativi alle librerie Jackson, necessari per la compilazione e l'esecuzione del progetto sono inclusi nella cartella lib presente nella root.

6 Istruzioni per compilazione e avvio

Il progetto è stato sviluppato e testato con le versioni 8 e 15 di Java, sia in ambiente UNIX sia in ambiente Windows. Per la compilazione è necessario posizionarsi sulla root del progetto ed eseguire il comando javac indicando nel classpath il percorso di ogni file .jar, in modo da risolvere tutte le dipendenze. Per il corretto funzionamento del sistema bisogna avviare prima il server e poi i client, altrimenti il client termina immediatamente avvisando che il servizio non è disponibile.

Di seguito sono elencati i comandi da digitare sul terminale per la compilazione ed esecuzione rispettivamente del client e del server :

6.1 Ambiente UNIX

Compilazione :

javac -d out -cp lib/jackson-annotations-2.9.7.jar:lib/jackson-databind-2.9.7.jar:lib/jackson-core-2.9.7.jar src/*.java

Avvio Server :

java -cp out:lib/jackson-annotations-2.9.7.jar:lib/jackson-databind-2.9.7.jar:lib/jackson-core-2.9.7.jar ServerMain

Avvio Client :

java -cp out:lib/jackson-annotations-2.9.7.jar:lib/jackson-databind-2.9.7.jar:lib/jackson-core-2.9.7.jar ClientMain

6.2 Ambiente Windows

Compilazione :

javac -d out -cp lib\jackson-annotations-2.9.7.jar;lib\jackson-databind-2.9.7.jar;lib\jackson-core-2.9.7.jar src*.java

Avvio Server :

java -cp out;lib\jackson-annotations-2.9.7.jar;lib\jackson-databind-2.9.7.jar;lib\jackson-core-2.9.7.jar ServerMain

Avvio Client :

java -cp out;lib\jackson-annotations-2.9.7.jar;lib\jackson-databind-2.9.7.jar;lib\jackson-core-2.9.7.jar ClientMain