RELAZIONE DEL PROGETTO

“WORTH”

Reti di Calcolatori-Laboratorio

A.A 2020/2021

Studente

Alessandro Di Biase 544338

Docente

Laura Ricci

Sommario

Introduzione 3
Architettura Software 3
WORTH Server 3
ServerMain 3
Progetto 4
GeneretorIp 4
Scheda 4
Utility JAR 4
Protocollo Utilizzato 5
WORTH Client 6
ChatTask 6
GestioneRisposta 7
View 7

Introduzione

Il progetto di fine corso del Laboratorio di programmazione di reti dell'a.a. 2020/2021, riguarda la realizzazione di un servizio denominato WORkTogetHer (WORTH): uno strumento per la gestione di progetti collaborativi che si ispira ai principi di organizzazione della metodologia Kanban, aiutando gli utenti a coordinarsi nello svolgimento di progetti e attività. Per la realizzazione delle componenti software sono state utilizzate tecniche viste durante lo svolgimento del corso, usufruendo del linguaggio di programmazione Java.

Architettura Software

L'architettura generale del servizio WORTH è del tipo Client-Server. La registrazione degli utenti avviene tramite RMI e, solamente in seguito è possibile effettuare il login, che apre una connessione TCP con il server. A seguito del login il client si registra a un servizio di notifica tramite RMI callback. Mentre, per quanto concerne la chat è stata realizzata tramite UDP Multicast.

WORTH Server

ServerMain

Il *ServerMain* all'avvio esegue un setup, in cui verifica la presenza della cartella *MainDir*, che si crea solamente al primo avvio. Questa cartella, si occupa di contenere tutti i progetti che gli utenti desiderano creare. Nel momento in cui *MainDir* contiene almeno un progetto, viene inizializzato lo stato del sistema. Infatti, il *ServerMain*, non solo reperisce le informazioni inerenti ai progetti ma anche quelle degli utenti registrati, persistenti fuori *MainDir*. Per fare ciò, il *ServerMain*, si serve di quattro ConcurrentHashmap:

- *LisProject*: dove a ogni NomeProgetto viene associato un oggetto della classe *Progetto;*
- *UserBase:* in cui a ogni NickUtente viene associata una Password;
- *KeysUserMap:* in cui a ogni chiave del selettore viene associato il Nickname. Utilizzata per identificare da chi è stata fatta la richiesta;
- *LoginMap:* Tiene traccia degli utenti online.

Subito dopo l'inizializzazione dello stato viene esportato l'oggetto remoto RMI *RegisterImpl,* che permette di invocare un unico metodo che consente di registrare un utente nel sistema.

Il *ServerMain* è costituito da un unico selettore principale, che si occupa di attendere le connessioni, leggere i messaggi inviati dal client e scrivere le risposte. Le scritture e le letture vengono effettuate a *blocchi,* scrivendo o leggendo solamente i bytes al momento disponibili. Quando la lettura è terminata si effettua la tokenizzazione del messaggio per identificare e soddisfare l'operazione richiesta. Tale messaggio rispetta lo specifico protocollo definito nella sezione Protocollo Utilizzato.

Progetto

Principalmente la classe *Progetto* si occupa di gestire e persistere le Schede che rappresentano i compiti da svolgere. A ogni progetto è associata una lista di membri, ovvero quegli utenti che hanno i permessi per accedere ai servizi legati al progetto.

Sia le liste di Schede che quella dei membri sono state modellate attraverso l'uso di ArrayList, infatti all'interno della classe progetto vi sono cinque liste:

- ToDo
- InProgress
- ToBeRevised
- Done
- Members

A ogni progetto è associata una cartella, infatti, nel momento in cui si istanzia un oggetto del tipo Progetto si controlla se esiste una cartella con lo stesso nome. Nel caso negativo si crea una nuova cartella con il nome del progetto, successivamente si crea la lista dei membri e la si persiste all'interno della cartella appena creata, e si richiede al *GeneratorIp* un nuovo indirizzo di Multicast mantenuto in una variabile. Infine si istanziano le quattro liste che gestiscono le schede. Viceversa, nel caso in cui la Directory dovesse essere già presente si deserializzano le schede e le si inseriscono nella lista appropriata. Lo stesso viene fatto con la lista membri. Anche in questo caso, viene richiesto un indirizzo di Multicast.

Ogni volta che si sposta una scheda o si aggiunge un nuovo membro al progetto viene persistito lo stato della singola scheda o della lista membri in base all'operazione richiesta.

GeneretorIp

La funzione principale del *GeneratoreIp* è quella di offrire al richiedente un IP Multicast compreso fra 224.0.0.1 e 239.255.255.255 (Classe D) da assegnare a ogni progetto. Esso genera gli indirizzi in maniera sequenziale utilizzando un opportuno algoritmo, nel caso in cui un progetto dovesse essere eliminato l'indirizzo Multicast non va perso, infatti, viene restituito al Generatore che lo immagazzina su uno Stack. Nel momento in cui si richiede un nuovo indirizzo si controlla prima che lo Stack sia non vuoto. Nel caso negativo si restituisce un indirizzo già generato altrimenti se ne genera uno nuovo. Nel caso in cui si dovesse arrivare all'indirizzo 240.0.0.0 verrebbe lanciata un'eccezione del tipo *OutOfRangeExcpetion*.

Scheda

Nella classe *Scheda* vi sono le informazioni da persistere, ossia il nome della scheda, la descrizione e la storia delle transizioni fra gli stati precedenti, in cui si è scelto di utilizzare lo Stack come struttura dati in quanto l'ultimo stato, ovvero quello corrente, si trova in cima. Per persistere la scheda si è scelto di utilizzare il meccanismo di serializzazione interno a Java implementando Serializable.

Utility JAR

Il JAR Utility definisce alcune funzioni ausiliari che servono solo a rendere il codice più leggibile e quindi meno verboso. Le funzioni servono rispettivamente per la lettura e la scrittura sul disco, per l'invio di messaggi a un client specifico e per inviare messaggi a un gruppo Multicast.

Si è scelto di creare questo JAR esclusivamente per esercitazione e il codice è leggibile nella cartella Utility.

Protocollo Utilizzato

Richiesta

LOGIN *Nickname Password*

LOGOUT

LISTPROJECTS

CREATEPROJECT *NomeProgetto Nickname*

ENTER *NomeProgetto*

SHOWMEMBERS *NomeProgetto*

CANCELPROJECT *NomeProgetto*

ADDCARD Card *Descrizione NomeProgetto*

SHOWCARDS *NomeProgetto*

SHOWCARD *Card NomeProgetto*

MOVECARD *Card From To NomeProgetto*

ADDMEMBER *NomeProgetto NomeMembro*

*Per rendere più leggibile il protocollo è stato omesso il carattere a capo e rimpiazzato dallo spazio.

Risposta

201 Login effettuato con successo

202 Lista Progetti

203 Progetto creato con successo

204 Messaggio di Ok, Stampa del messaggio

205 Progetto Rimosso con successo

206 Lista Membri

207 Informazioni Scheda

208 Entrato nel progetto

300 Logout

Errori

401 Errore nel passaggio dei parametri

402 Password o Nickname errati

403 Progetto non esistente

404 Errore Generico, Stampa del messaggio

WORTH Client

L'utente può interagire con il servizio tramite il Client che è composto principalmente da due Thread: il Thread principale che avvia la GUI e uno secondario che gestirà i messaggi della chat.

Un requisito fondamentale del Client è quello di avviare prima il server, infatti, nel caso in cui non fosse così, verrebbe arrestato subito dopo aver mostrato il messaggio di terminazione.

Viceversa, dopo il suo avvio, il client inizializza la routine per accedere alla registrazione RMI e infine vengono istanziati gli oggetti *Model*, *View* e *Controller* per realizzare il conosciuto omonimo pattern. Fondamentalmente il pattern è basato sulla separazione dei compiti fra i componenti che interpretano i tre ruoli fondamentali:

- Il *Model* fornisce i metodi per accedere e modificare i dati dell'applicazione. Più nello specifico, tiene traccia dell'utente che ha effettuato l'accesso, della lista degli utenti (costantemente aggiornata dal server tramite callback) e infine il nome e l'indirizzo di Multicast IP del progetto che si sta visualizzando.
- Il *View* visualizza i dati, i quali possono essere contenuti nel *Model* oppure richiesti al Server. Inoltre, si occupa dell'interazione con gli utenti.
- Il *Controller* riceve i comandi dell'utente, interagisce direttamente con il server tramite la connessione TCP e modifica sia il *Model* (aggiorna lo stato dei dati) che il *View* (aggiorna gli elementi grafici).

ChatTask

Di fondamentale importanza è la classe *ChatTask*, definita per ricevere i messaggi UDP Multicast ricevuti sia dal sistema (come ad esempio il cambiamento dello Stato della scheda oppure la notifica dell'aggiunta di un nuovo membro nel gruppo) che da altri membri. Infatti, nel momento in cui il client decide di creare o entrare in un progetto, il server oltre al codice di risposta invia anche l'indirizzo Multicast del progetto. Il client lo salva nel model e successivamente istanza un Thread che esegue questo task.

Il task rimane in esecuzione fin quando il Thread non viene interrotto. Questa interruzione è generata da due eventi diversi:

1. L'utente che vuole tornare nella schermata principale utilizza il bottone Go Back. In questo caso l'interruzione avviene all'interno del client stesso.
2. Un membro sceglie di eliminare il progetto. In questo caso, il server, prima di eliminare il progetto e tutte le informazioni all'interno, notifica tramite Multicast la cancellazione del progetto. Il Thread, alla ricezione di questa comunicazione, esce dal ciclo di ricezione di messaggi provocando la terminazione del Thread.

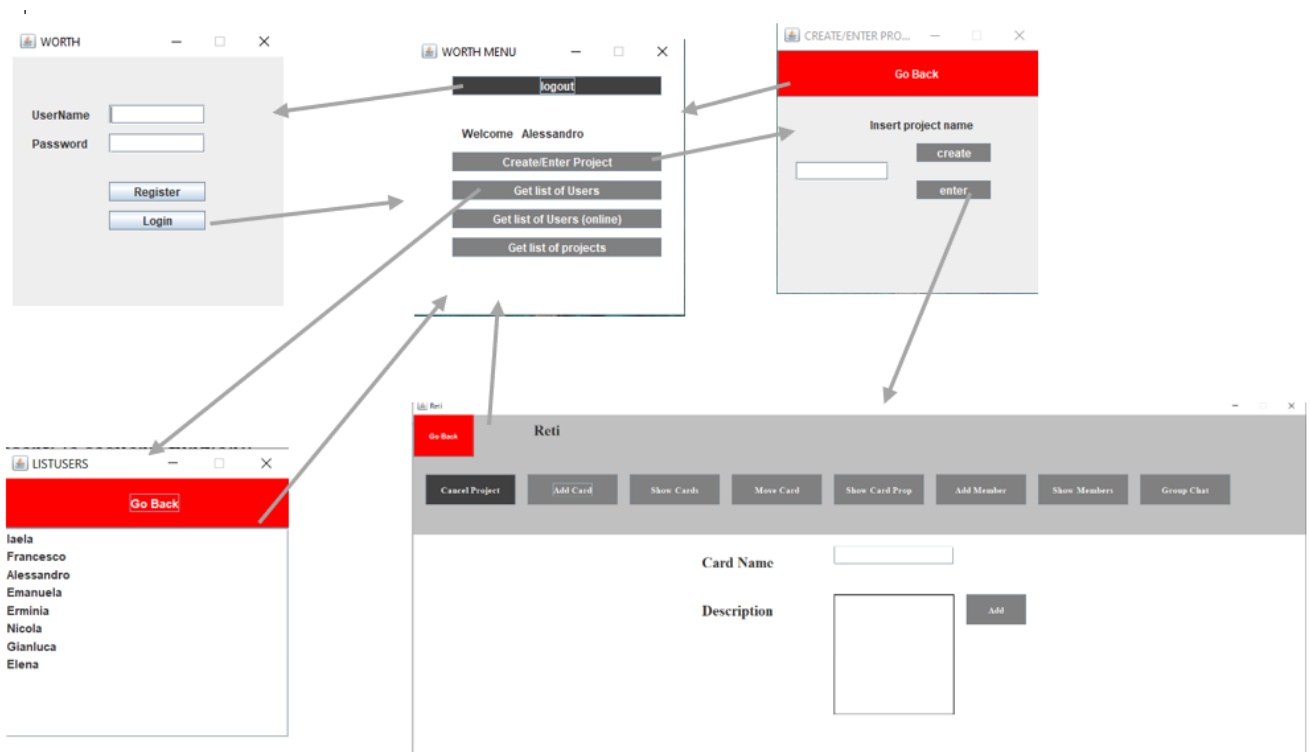
GestioneRisposta

La funzione principale della classe *GestioneRisposta* è quella di dividere in token il messaggio ricevuto, descritto nella sezione Protocollo Utilizzato, e di notificare al Client l'esito dell'operazione richiesta tramite l'uso della View.

View

Più nello specifico, la *View* definisce e gestisce tutti gli elementi grafici. Per avere una User Experience migliore si è deciso di implementare cinque differenti schermate (vedere Schema di Navigazione):

- Schermata Iniziale: è il punto iniziale dell'applicativo Client, che contiene gli elementi principali per la registrazione e il login;
- Worth Menù: è possibile sia ritornare indietro tramite il logout, sia andare nella schermata dove si può creare o entrare in un nuovo progetto, che accedere alla visualizzazione della lista desiderata (progetti, utenti online, utenti online/offline);
- List View: lo scopo principale di questa schermata è quello di far visualizzare le diverse liste richieste dall'utente;
- Create/Enter *View*: questa schermata permette di creare un progetto nuovo o di accedere ad un progetto già esistente;
- Project View: definisce gli elementi per interagire con un progetto nello specifico (inserire una scheda, visualizzare le schede presenti, cambiare lo stato di una scheda, aggiungere un nuovo membro, entrare nella chat di gruppo ed infine cancellare il progetto che si sta visualizzando).



Schema di Navigazione

Compilazione/Esecuzione

Per compilare ed eseguire il progetto sono stati realizzati degli script (sia per Windows che per Linux) per rendere più rapida la procedura.

WORTHServer:

Posizionarsi in WORTHServer;
(Windows) Aprire una nuova Powershell eseguire: ./Compile.ps1;

(Linux) Aprire un nuovo terminale eseguire: bash Compile.sh;

Esecuzione:

(Windows) ./Execute.ps1

(Linux) bash ./Execute.sh

Nel caso in cui si dovesse terminare l'esecuzione del server e successivamente rieseguire, posizionarsi con changedirectory in WORTHServer.

WORTHClient:

Posizionarsi in WORTHClient;
Aprire una nuova Powershell eseguire: ./Compile.ps1

Per l'esecuzione: ./Execute.ps1

Nel caso in cui si dovesse terminare l'esecuzione del server e successivamente rieseguire, posizionarsi con changedirectory in WORTHClient.