

RELAZIONE DEL PROGETTO

“WORTH”

Reti di Calcolatori-Laboratorio

A.A 2020/2021

Studente

Alessandro Di Biase 544338

Docente

Laura Ricci

Introduzione

Il progetto di laboratorio di fine corso a.a 2020/2021 di reti riguarda la realizzazione di un servizio denominato WORkTogetHer (WORTH): uno strumento per la gestione di progetti collaborativi che si ispira ad alcuni principi della metodologia Kanban, utilizzando il 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 solo dopo essersi registrati è possibile effettuare il login che apre una connessione TCP con il server (qui si è scelto di utilizzare un protocollo in-band-signaling, quindi sia i dati che le operazioni richieste viaggiano sullo stesso canale). 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 per inizializzare lo stato del sistema, infatti, non solo reperisce le informazioni inerenti ai progetti ma anche quelle degli utenti che utilizzano il sistema. Per fare ciò si serve di 3 ConcurrentHashmap:

* *LisProject*: dove ad ogni NomeProgetto viene associato un Ogetto della classe *Progetto.*
* *UserBase:* Ad ogni NickUtente viene associata una Password
* *KeysUserMap:* Ad ogni NickUtente (loggato) viene associata una chiave del selettore.

Subito dopo viene esportato l’oggetto remoto RMI *RegisterImpl,* 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 l’operazione richiesta, Il messaggio rispetta uno 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. Ad ogni progetto è associato una lista di membri, ovvero utenti che hanno i permessi per accede 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 5 liste:

* ToDo
* InProgress
* ToBeRevised
* Done
* Members

Ad ogni progetto è associato una Directory, infatti, nel momento in cui si istanzia un oggetto del tipo Progetto si controlla se esiste una directory con lo stesso nome, nel caso negativo si crea una nuova Directory con il nome del progetto, si crea la lista dei membri e la si persiste all’interno della cartella, si richiede al GeneratorIp un nuovo indirizzo di multicast mantenuto in una variabile ed infine si istanziano le 4 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.

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 ad 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 quindi si controlla prima che lo stack sia non vuoto nel caso negativo si restituisce un indirizzo già generato altrimenti si genera uno nuovo.

Nel Caso in cui si dovesse arrivare all’indirizzo 240.0.0.0 verrebbe lanciata un’eccezione OutOfRangeExcpetion.

**Scheda**

**Utility JAR**

Il JAR Utility definisce cinque funzioni auto esplicative che servono solo a rendere il codice più leggibile e quindi meno verboso.  
Si è scelto di creare un JAR esclusivamente per esercitazione.

Protocollo Utilizzato

Richiesta

LOGIN*Nickname**Password*

LOGOUT

LISTPROJECTS

CREATEPROJECT *NomeProgetto Nickname*

ENTER *NomeProgetto*

SHOWMEMBERS  NomeProgetto

CANCELPROJECT  NomeProgetto

ADDCARD  Card  Descrizione  NomeProgetto

SHOWCARDSNomeProgetto

SHOWCARDCard NomeProgetto

MOVECARDCard FromTo NomeProgetto

ADDMEMBERNomeProgettoNomeMembro

\*Per rendere più leggibile il protocollo ho emesso il carattere a capo che è rimpiazzato dallo spazio.

Risposta

201 Login effettuato con successo  
202 Progetto creato con successo

Errori

401 Errore nel passaggio dei parametri  
402 Password o Nickname errati  
403 Progetto già esistente

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 ed 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 ed 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:

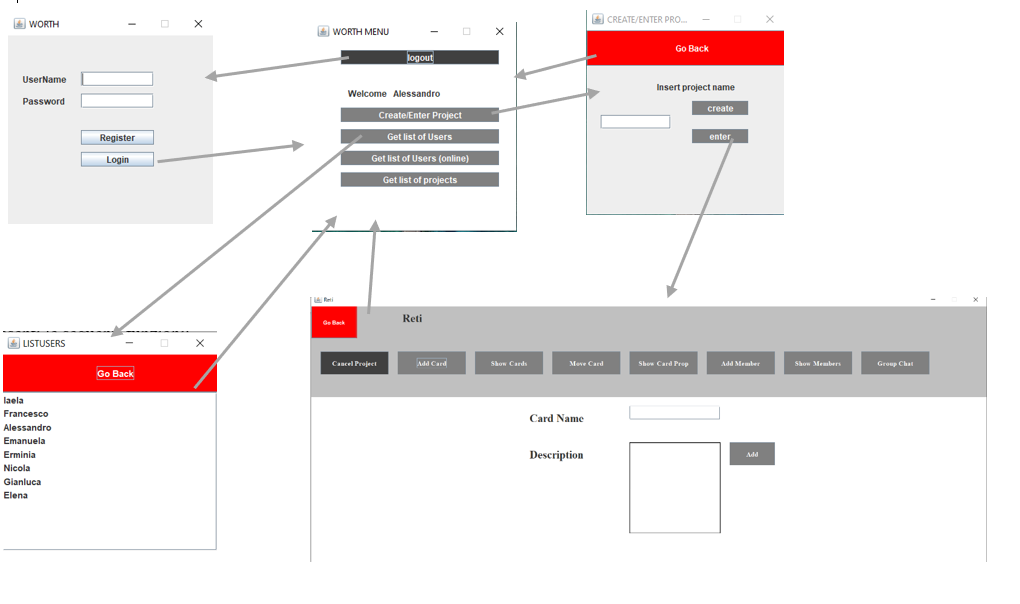
* Il Model fornisce i metodi per accedere e modificare i dati dell’applicazione, più nello specifico tiene traccia dell’utente che ha effettuato l’accesso, della lista degli utenti (costantemente aggiornata dal server tramite callback) ed infine il nome e l’indirizzo di Multicast IP del progetto che 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

View

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

* Schermata Iniziale: il punto iniziale dell’applicativo client, esso contiene gli elementi principali per la registrazione ed il login.
* Worth Menù: dove è possibile sia ritornare indietro tramite il logout, creare un nuovo progetto o entrare in uno già esistente ed infine accedere alla vista di 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.



Schema di Navigazione

Compilazione

Windows

Per la compilazione ed esecuzione si è deciso di scrivere uno script in powershell per rendere meno verbosa l procedura.

Server:

Posizionarsi in WORTHServer;   
Aprire una nuova Powershell eseguire: ./CompileAndExecute.ps1;

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

Client:

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

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

Ubuntu

Conclusioni