Corso: Laboratorio di Reti B

Anno Accademico: 2020/21

Nome: Elia Fagioli

Matricola: 580115

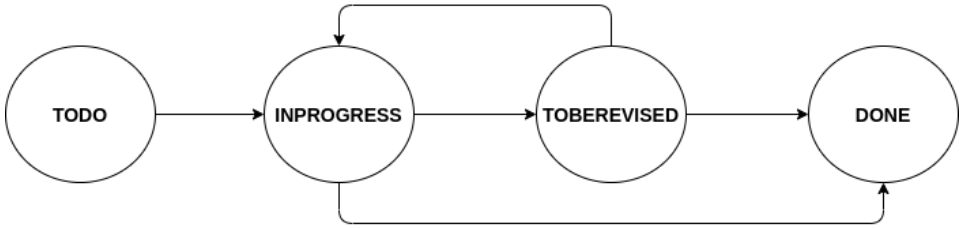
***Relazione progetto di fine corso**: WORTH: WORkTogetHer*

UNIVERSITÀ DI PISA

Indice

Presentazione delle caratteristiche progetto:

Lo scopo del progetto è di realizzare un'applicazione focalizzata sull'organizzazione e la gestione di progetti in modo collaborativo, un tipo di applicazione di collaborazione e project management che aiuta le persone a organizzarsi e coordinarsi nello svolgimento di progetti comuni.
I progetti in questione possono essere progetti professionali oppure semplici attività, essi devono però poter essere organizzati in una serie di compiti (es. to do list) che andranno svolti ad esempio da membri di un gruppo.
Nella descrizione del progetto sono stati presentati come modelli da seguire: Trello e Asana.
Il progetto si basa sulla metodologia **Kanban**, in particolare sulla Kanban Board, ovvero uno strumento di base costituito da una bacheca o lavagna.
Questa "***lavagna***" è divisa in colonne che rappresentano le varie fasi del flusso di lavoro, le varie attività invece vengono chiamate "***card***" e passano da una colonna all'altra in base al loro svolgimento secondo un preciso schema.
In WORTH, un ***progetto***, identificato da un nome univoco, è costituito da una serie di card che rappresentano i compiti da svolgere per portarlo a termine, e fornisce una serie di servizi.
Ad ogni progetto è associata una lista di ***membri***, ovvero utenti che hanno i permessi per modificare le card e accedere ai servizi associati al progetto (es. chat).
Una card è composta da un nome, una lista dei movimenti e una descrizione testuale, *il nome assegnato alla card deve essere univoco nell'ambito di un progetto.*
Ogni progetto ha associate quattro liste (ovvero le colonne della nostra lavagna Kanban) che definiscono il flusso di lavoro come passaggio delle card da una lista alla successiva secondo il seguente metodo:



Qualsiasi membro del progetto può spostare la card da una lista all'altro, rispettando i vincoli illustrati nel diagramma.
L'ultima caratteristica dell'applicativo è che ogni progetto abbia la propria ***chat di gruppo*** dove i membri di esso possono comunicare e mantenere traccia dei progressi del progetto tramite notifica dello spostamento di una card da una lista ad un'altra.

Progettazione

Visualizzazione dei progetti già presenti in commercio

I progetti già in commercio cui fare riferimento, ovvero Trello ed Asana, sono stati analizzati tramite video introduttivi sull'utilizzo e sulle loro varie funzionalità presenti su YouTube, notando che questi applicativi sono molto usati e in moltissimi settori differenti.
Una volta appreso il funzionamento di questi software, la serie di servizi richiesti per il progetto sono stati schematizzati e ho iniziato la realizzazione del progetto tramite ambiente di sviluppo Intellij Community Edition 2021.1.2.

Schematizzazione architettura progetto

Nel punto 3 del progetto, ovvero Specifiche per l'implementazione, la prima cosa che viene specificata è che le prime due operazioni che possono essere effettuate nell'utilizzo di Worth sono ***login*** e ***registrazione*.**

PRIMA OPERAZIONE:

Al lancio di un Client, viene effettuata la scelta tra accedere (se già registrati) oppure registrarsi.
Nel primo caso (**login**) vengono inviati username e password, tramite connessione TCP/IP con il server Worth presso indirizzo IP e porta noti, attraverso una funzione booleana che ritornerà true se l'accesso è avvenuto con successo, altrimenti il server restituirà un messaggio di errore e la funzione ritornerà false.
Nel caso di **registrazione**, , tramite paradigma RMI, il client crea un riferimento agli oggetti remoti pubblicati dal server su un registry noto: questi saranno necessari per l'operazione di registrazione al servizio, e verranno invocati su richiesta dell'utente.
La classe RMI verificherà prima se l'username è già in uso, in caso contrario effettuerà la registrazione del nuovo utente nel database utenti e successivamente tramite Client verrà automaticamente effettuato il login al servizio Worth.
La scelta di accedere subito dopo la registrazione, anche se non richiesta esplicitamente nella consegna nel progetto, è stata fatta perché frequentemente un utente che si registra ad un servizio viene fatto accedere subito dopo.

SECONDA OPERAZIONE:

Come specificato nella consegna del progetto ci sono due tipi di operazioni: quelle che può fare ogni utente e quelle che possono effettuare su un progetto solo i membri di quest'ultimo:

OPERAZIONI UTENTE DOPO L'ACCESSO:

Ogni utente può:

1. Effettuare il logout e terminare il client
2. Recuperare la lista degli utenti registrati al servizio con il relativo status: ONLINE/OFFLINE
3. Recuperare la lista degli utenti online
4. Creare un nuovo progetto e diventarne automaticamente membro
5. Stampare la lista dei progetti di cui si è membro
6. Selezionare un progetto per effettuare le operazioni che possono svolgere solo i membri di quest'ultimo

OPERAZIONI MEMBRO PROGETTO*: dopo averne selezionato uno nella scelta precedente*

0. Uscire dalle operazioni di progetto e tornare indietro;
1. Stampare la lista membri progetto, per scelta ogni progetto contiene un file XML con all'interno i membri di esso.
2. Aggiungi membro al progetto, il membro deve essere un utente registrato di Worth

3. Cancellare il progetto, viene chiesto di confermare, gli altri utenti di Worth, se all'interno del progetto, nell'effettuare delle operazioni vedranno riscontrato un messaggio di errore e torneranno al menu principale.
   Rispettando la consegna, un progetto può essere cancellato solo quando tutte le card sono presenti nella lista, e quindi nella directory DONE.
4. Stampa delle liste del progetto, che per consegna sono rappresentate da directory con all'interno le card in forma di file XML.
5. Creare una nuova card
6. Selezionare una card per effettuare le operazioni di modifica o visualizzazione su di essa
7. Inviare messaggio alla chat di progetto tramite UDP multicast, ogni utente, una volta connesso, riceverà i messaggi su ogni progetto di cui è membro a partire dal momento in cui effettua l'accesso o viene aggiunto a un progetto fino a quando il progetto non viene cancellato o il progetto viene eliminato.
8. Stampare la lista di messaggi e di notifiche del progetto, le chat vengono cancellate alla terminazione del server (non sono tra le informazioni da persistere) oppure quando un progetto viene eliminato.

OPERAZIONI MEMBRO PROGETTO – OPERAZIONI CARD*: dopo averne selezionata una nella scelta precedente, punto 6:*

In questa parte del programma un membro può:

0. Tornare indietro alle operazioni di progetto
1. Spostare la card selezionata in un'altra lista, seguendo il diagramma degli spostamenti
2. Stampare le info della card selezionata: nome lista attuale e descrizione
3. Stampare la storia della card, ovvero la lista degli spostamenti tra le liste.

Architettura Client/Server e classi utilizzate

Dopo previa login effettuata con successo, l'utente interagisce, secondo il modello client-server (richieste/risposte), con il server sulla connessione TCP creata, inviando i comandi elencati in precedenza. Tutte le operazioni sono effettuate su questa connessione TCP, eccetto:

- la registrazione (RMI)
- le operazioni di recupero della lista degli utenti (listUsers e listOnlineusers) che usano la struttura dati locale del client (ricevuta come risposta dopo il login e aggiornata tramite il meccanismo di RMI callback)
- Operazioni sulla chat. (UDP Multicast)

L'interazione Client/Server è effettuata su una connessione TCP/IP con il server Worth presso indirizzo IP e porta noti, in quanto progetto di corso universitario più che progetto destinato alla vendita, ho ritenuto, anche per comodità di testing durante l'elaborazione, che le porte per la connessione TCP, per RMI e per la chat Multicast fossero implementate di default:

- SOCKET_PORT = 6789; //porta connessione TCP col server
- RMI_PORT = 5000; //porta RMI
- CHAT_PORT = 1234; //porta Chat Multicast

Le due classi main: MainClassServer e MainClassClient servono rispettivamente per:

- MainClassServer: inizializzare un server Worth passandogli come argomenti SOCKET_PORT ed RMI_PORT e start.
- MainClassClient:
    - Inizializzare un ChatRegister con CHAT_PORT e stringa vuota per l'indirizzo.
    - inizializzare un Thread ChatStarter con il ChatRegister precedente
    - inizializzare un Client con SOCKET_PORT, RMI_PORT e il ChatRegister
    - start di client e thread ChatStarter.

L'interazione richiesta/risposta tra client e server è stata sviluppata tramite invio e ricezione di String.
Il Client Worth invia le proprie stringhe tramite due funzioni, una void e una boolean, secondo la seguente sintassi:

OPERAZIONE dato1 dato2 dato3…

La stringa è unica ma ogni spazio separa le informazioni, nella ricezione il server effettuerà uno split(" ") sulla stringa ricevuta ottenendo un array di stringhe ottenuto dalle stringhe separate da uno carattere spazio.
Esempio: per effettuare il login, il client invia tramite funzione boolean:

"login username password"

In caso di utilizzo di una funzione boolean, il server risponderà con delle informazioni (dopo la login il server invia la lista degli utenti) oppure con un messaggio di errore in caso non sia andata a buon fine l'operazione.
I messaggi di errore si distinguono perché la stringa di risposta inizia con ERROR.
La connessione TCP sulla quale viaggiano le richieste è mantenuta aperta, quando il client invia la richiesta di logout il server effettua il logout dell'utente e chiude la connessione TCP, a questo punto anche il client terminerà, chiudendo la connessione invocando il metodo close sul proprio socket TCP.
Il funzionamento del tutto tramite utilizzo della funzione split è basato sull'utilizzo di Scanner.next() che prende in input la stringa fino al carattere " " permettendo il corretto inserimento da tastiera di un dato solo, è stata per l'appunto fatta la scelta di utilizzare una unica stringa (fino allo spazio) per username, password, nomi di progetti e nomi di card.

RMI e RMI Callback

Le specifiche dell'implementazione richiedono che le funzionalità di registrazione di nuovi utenti Worth avvenga tramite RMI e che i clienti online vengano notificati tramite RMI Callback in tempo reale quando un altro utente accede o effettua il logout.
Inoltre, dovendo implementare un servizio di chat, Il meccanismo delle callback è stato utilizzato per unirsi alle chat di progetto all'accesso, oppure quando si viene aggiunti ad un progetto ed anche quando lo si crea.
Le funzionalità RMI del server sono comprese nella classe **RMIEventManager** (presente nella directory RMI) che estende la classe RemoteServer e implementa l'interfaccia RMIEventManagerInterface.
La classe viene istanziata nel server, all'interno del metodo start() della classe WorthServer, tramite metodo rmiRegister(), dopo la creazione di un nuovo oggetto, il metodo provvede anche ad effettuare il bind di un nuovo registry e ad avviare il server RMI.
Il costruttore di RMIEventManagerInterface richiede come argomenti la ConcurrentHashMap<String, String> che rappresenta la lista degli utenti come *chiavi* con relativo *valore* il loro stato attuale, e una stringa che sta ad indicare il path del file utenti.xml che verrà utilizzato in fase di registrazione RMI perché è un'informazione che deve essere persistita.
Tutti i metodi della classe sono synchronized dato che non è possibile sapere a priori quanti e quali thread RMI eseguiranno i metodi della classe.
Il metodo boolean **register** controlla che nella HashMap non sia già presente l'username ricevuto dal client nella registrazione (in caso di riscontro ritorna null) altrimenti lo aggiunge all'HashMap e tramite SAXBuilder e XMLOutputter va ad aggiornare il file utenti.xml.
La classe contiene due metodi per registrarsi e cancellarsi dalla callback (registerForCallback e unregisterFromCallback) che vengono richiamati dal client quando si effettuano login e logout e due metodi di notifica (updateUsers e updateChat) per notificare il cambio di stato di un utente e l'unione di un membro alla chat di progetto.
Ci si registra e cancella dalla callback tramite un oggetto RMIClient istanziato su ogni client e passato come argomento nei metodi register e unregister, la suddetta classe contiene i metodi utilizzati nei metodi di notifica di RMIEventManager per notificare il cambio di stato e l'accesso in una chat di progetto da parte di un membro.

Server

Il server come detto in precedenza è istanziato in una classe MainClassServer e contiene i vari path per il file degli utenti e per la directory dei progetti oltre alle ConcurrentHashMap di **worthUsers**<username, status> e **worthProjects**<nome progetto, indirizzo multicast chat>.
Nel costruttore del server si ricevono le porte per RMI e connessione TCP e inoltre si verifica l'esistenza del file utenti.xml (altrimenti lo si crea) e se ne tiene una copia in un Document nella classe che verrà aggiornato per effettuare le operazioni.
Nel costruttore si inizializzano anche le due ConcurrentHashMap citate precedentemente.
Per quanto riguarda la ConcurrentHashMap worthProjects gli indirizzi multicast sono generati tramite l'utilizzo di una classe MulticastGenerator e associati ai progetti già presenti nella directory Progetti.
Per questo progetto, si poteva scegliere tra implementare un server multi-threaded o uno che effettua il multiplexing dei canali tramite la libreria Java NIO, si è optato per la seconda in modalità non bloccante.
L'implementazione del server attraverso NIO effettua il multiplexing delle richieste ed è basata su tre componenti fondamentali:

- un oggetto della classe ServerSocketChannel che rimane in ascolto delle connessioni in arrivo
- un SocketChannel per ogni connessione attiva
- un Selector per iterare sulle connessioni attive ed effettuare il multiplexing di queste.

Al momento della creazione del server viene istanziato un Selector e il ServerSocketChannel, tramite l'invocazione del metodo open(), per il corretto funzionamento in modalità non bloccante, l'oggetto ServerSocketChannel è registrato sul selettore appena creato.
In seguito all'invocazione del metodo start(), il server entra in un ciclo while(true); ad ogni iterazione `e verificata la presenza o meno di canali pronti per effettuare qualche operazione.
Se il selettore non è vuoto (ovvero, restituisce un valore diverso da zero invocando su di esso il metodo select()):
if (sel.select() == 0) continue;
si crea un oggetto di tipo Iterator<SelectionKey> per iterare sul set di chiavi corrispondenti ai canali disponibili.
Per ogni chiave restituita dall'iteratore, si individua il tipo di evento pronto da essere elaborato:

- isAcceptable(): se la chiave corrisponde ad un evento di tipo OP ACCEPT, vi `e una nuova connessione in arrivo e si invoca il metodo privato accept(), che si occupa di creare un nuovo SocketChannel per la connessione, di registrarlo sul selettore e di allocare un buffer per la memorizzazione dei dati.
- isWritable() per inviare messaggi al client (lancia sendAnswer)
- isReadable quando riceve messaggi dal client (lancia ReadClientMessage), legge il messaggio e fa come prima cosa lo split sul carattere " ", la prima stringa conterrà l'operazione richiesta dal client e sarà gestita da uno switch. Al termine dello switch verrà restituita una stringa come risposta al client contenente i dati richiesti o un messaggio di errore.

In caso di terminazione improvvisa di un Client, l'evento è gestito nel catch IOException, riconducendosi all'username corrispondente (registrato in una ConcurrentHashMap sul Server) e viene effettuato il logout forzato.

Client

Il client viene istanziato lanciando una classe MainClassClient e nel costruttore richiede SOCKET_PORT, RMI_PORT e il riferimento al ChatRegister. Il client come precedentemente spiegato nella parte dell'architettura del software si muove nelle seguenti parti:

- Login/Sign in
- Menu utente
- Menu progetto
- Menu card

Il client comunica con il server attraverso la funzione void sendData e quella boolean sendData2, la prima è usata principalmente quando non posso ricevere messaggi di errore di ritorno, ad esempio nell'effettuare il logout, la seconda invece in base al messaggio di risposta invia un certo ritorno.
Il meccanismo di comunicazione con il server è sempre: operazione seguita da dati separati da spazio.

Implementazione chat IP Multicast tramite datagrammi UDP

La richiesta della consegna è di poter inviare messaggi tra membri dello stesso progetto e notifiche di operazioni svolte all'interno di esso, i messaggi non passano attraverso il Server.
Per la lettura dei dati dal canale, è utilizzato un thread che invoca periodicamente il metodo readDatagramChannel(), in modo da avere la lista dei messaggi per ogni progetto sempre aggiornata e gli accessi concorrenti sono garantiti dall'utilizzo di strutture dati concorrenti.
Una volta connesso al servizio, il client invia una richiesta al Server che risponde con una stringa contenente la lista dei progetti di cui è membro e i relativi indirizzi Multicast per progetto, che vengono generati e assegnati dal server nella ConcurrentHashMap worthProjects.
Una volta recuperati progetti e indirizzi, il client effettua il joinChat su ogni indirizzo multicast ricevuto tramite la classe ChatRegister, successivamente il client entra nella parte di Operazioni utente.
Quando un utente seleziona un progetto ed entra nella parte operazioni membro, può inviare un messaggio tramite l'operazione del menù.
Verrà richiesto di inserire una stringa che sarà inviata tramite il metodo sendMessage della classe ChatRegister, passandogli come parametri il nome del progetto, il nome del mittente e il corpo del messaggio.
La classe ChatRegister tramite il nome del progetto recupererà nella ConcurrentHashMap l'indirizzo Multicast del suddetto progetto e invierà il messaggio sul DatagramChannel all'indirizzo Multicast recuperato e sulla CHAT_PORT.
Quando un utente richiede di leggere tutti i messaggi della chat richiama il metodo della classe ChatRegister getProjectMessages passandogli come argomento il nome del progetto, il metodo ritorna una lista di stringhe, ognuna nel formato:
usernameMittente: corpo del messaggio.
Viene mantenuta in memoria la coppia (projectName, MembershipKey), in modo da poter revocare la sottoscrizione ad un gruppo multicast in caso di cancellazione di un progetto.
Quando un progetto viene eliminato, il Client che lo elimina richiama il metodo deleteGroup della classe ChatRegister che lo rimuove dalle ConcurrentHashMap cancellando anche i messaggi e rimuovendo la membershipKey.

Controllo concorrenza

Dalla consegna del progetto: il server poteva essere realizzato multithreaded oppure effettuare il multiplexing dei canali mediante NIO e come già spiegato nel punto 3 ho optato per la seconda opzione.
Per il controllo della concorrenza nelle operazioni sono state utilizzate principalmente strutture dati Concurrent o comunque thread-safe in aggiunta all'utilizzo di metodi synchronized in quanto non sempre è possibile fare assunzioni sull'invocazione da parte dei thread Main o dai thread RMI.

Librerie esterne usate

Le librerie esterne utilizzate sono:

- Jdom-2.0.6 per interagire con i file XML utilizzati come quello dei membri, degli utenti, e le card dei progetti.
- Commons-io-2.10.0 per lavorare con stream, readers, writers e file. In particolare, si necessita di questa libreria nella cancellazione dei progetti.

Istruzioni per la compilazione e l'esecuzione del progetto

Il progetto è stato sviluppato tramite l'ambiente di sviluppo IntelliJ IDEA.

IntelliJ

Il progetto è stato principalmente sviluppato e testato in ambiente Windows tramite l'utilizzo di IntelliJ, facendo inizialmente una "*build project*" e successivamente mandando in esecuzione prima il Server tramite MainClassServer e successivamente uno o più Client (impostando la possibilità di creare istanze multiple) tramite MainClassClient presenti rispettivamente nelle directory Server e Client.

Il tutto è stato strutturato tramite porte di default per semplificare il testing, in alternativa le porte avrebbero potuto essere specificate da linea di comando.

Il progetto è stato sviluppato per essere facilmente compilato tramite l'utilizzo di Maven presente sul sistema.

Installazione Maven:

Windows:

Necessaria il download dello zip di Maven: https://maven.apache.org/download.cgi

Unzip del file scaricato e aggiunga del path della sottodirectory bin alle variabili di ambiente.

Ubuntu*:*

Per installare Maven su Ubuntu tramite apt:

- aggiornare il pacchetto index:

  sudo apt update

- Successivamente per installare Maven utilizzare:

  sudo apt install maven

Utilizzo di Maven

Per compilare il codice sorgente, incluse le dipendenze, sarà sufficiente, da riga di comando, posizionarsi nella directory ./Worth-Maven e lanciare il comando:

mvn package

Maven scaricherà ed installerà le dipendenze sul sistema, compilerà il codice e creerà i pacchetti .jar per server e client contenenti il codice e le librerie esterne, tutto ciò all'interno di una cartella **target** che realizzerà.

Per eliminare la directory target:

mvn clean

Una volta compilato il codice sarà possibile eseguire il Server e il Client:
(Utilizzare "\" come file separator in ambiente Windows invece di "/")

java -jar ./target/Server-jar-with-dependencies.jar

java -jar ./target/Client-jar-with-dependencies.jar

Per il server e per ogni Client se eseguiti da terminale è raccomandata l'esecuzione dei singoli in finestre differenti.

Il server stamperà le connessioni con i vari Client e le operazioni di ricezione e risposta mentre i Client le operazioni disponibili e le notifiche di registrazioni o cambiamenti di status (online/offline) degli utenti Worth.

In alternativa all'utilizzo di Maven, sarà possibile importare il codice come un nuovo progetto su un IDE.

Se nell'IDE scelto le dipendenze non vengono inserite in automatico dal file pom.xml, esse sono presenti all'interno della sottodirectory /lib.

Errori e problemi riscontrati

Andando a testare l'applicativo su macchina virtuale Ubuntu tramite VirtualBox, il corretto funzionamento della chat UDP è stato bloccato dalle impostazioni della macchina virtuale relative alle comunicazioni, che rimanevano permesse solo su range di indirizzi locali (localhost).
È consigliata dunque l'esecuzione del progetto non su macchina virtuale.