Università degli Studi di Pisa

Laboratorio di Reti

A.A. 2018/2019

Relazione Progetto Laboratorio

Autore:
Francesco Kotopulos De Angelis

15 luglio 2019

Indice

1 Architettura del sistema

1.1 Descrizione generale

Il progetto consiste nell'implementazione di TURING (disTribUted collaboRative edItiNG), uno strumento per l'editing collaborativo di documenti che offre un insieme di servizi minimale.
Il sistema si basa su una struttura Client - Server. Tutte le operazioni richieste dai singoli Client vengono inviate al Server e processate. Successivamente il Server, dopo aver elaborato le richieste ricevute, invia una risposta positiva o negativa ai Client per notificare il successo o il fallimento dell'operazione richiesta.
I clienti per poter utilizzare il servizio devono necessariamente effettuare una prima registrazione, fornendo uno username ed una password, ed effettuare un login ad ogni accesso al sistema, con le credenziali fornite in fase di registrazione.
Una volta effettuato l'accesso sarà possibile usufruire delle funzionalità offerte dal servizio.

1.2 Specifica delle operazioni

Di seguito sono elencate le specifiche delle operazioni divise per ambiti di appartenenza.

1.2.1 Registrazione e Login

- **Register (Username,Password):** E' la prima operazione da eseguire per poter accedere al servizio. La registrazione richiede un nome utente univoco, non modificabile ed una password. Nel caso in cui si tenti di registrare un utente con uno username già presente nel sistema, viene sollevata una *AlreadyExistingAccountException* e l'errore viene notificato al client.

- **Login (Username,Password):** E' l'operazione che deve essere eseguita tutte le volte che si desidera accedere al sistema. Per poter effetuare un login correttamente l'utente deve aver eseguito precedentemente una registrazione al servizio. Nel caso in cui il Server non riesca verificare la correttezza delle credenziali inserite viene sollevata una *WrongIdentityException* e simultaneamente il client riceve una notifica dell'errore.

1.2.2 Gestione e Visualizzazione documenti

- **Create (NomeDocumento,NumSezioni):** è l'operazione che dati in ingresso i parametri "Nome Documento" e "Numero Sezioni" permette la creazione di un nuovo documento sul Server. Non è possibile creare documenti con lo stesso nome

sul server: se un documento risulta essere già esistente viene sollevata una *Existing-DocumentException* e l'errore viene notificato al client che ha richiesto l'operazione. Nel caso in cui erroneamente si inserisca il numero di una sezione in un formato non numerico viene lanciata una *FormatException*;

- **List ():** visualizza sul terminale la lista di tutti i documenti creati o condivisi con l'utente corrente. Nel caso in cui l'utente non abbia ancora creato nessun documento o non abbia ricevuto nessun invito per la modifica di documenti creati da altri utenti, viene stampato a schermo un messaggio di errore.
- **Show (NomeDocumento,Sezione) —— (NomeDocumento):** permette di visualizzare il contenuto di un Documento che è stato precedentemente creato. Sono presenti due varianti di questa operazione:
  - **Show (NomeDocumento,Sezione):** visualizza sul terminale solamente la sezione specificata.
  - **Show (NomeDocumento):** visualizza sul terminale tutte le sezioni del documento.

  Nel caso in cui erroneamente si inserisca il numero di una sezione in un formato non numerico viene lanciata una *FormatException*;
- **Share (NomeDocumento,Utente):** permette di condividere un documento con un altro utente registrato nel sistema. Per poter eseguire la condivisione del documento è necessario essere il creatore del documento stesso. Nei casi in cui si stia cercando di condividere un documento del quale non si è i proprietari, l'utente scelto per la condivisione non sia registrato nel sistema o si tenti di condividere con se stessi un documento del quale si è i creatori, viene sollevata un'eccezione e il rispettivo errore viene notificato al client che ha richiesto l'operazione.

1.2.3 Editing e Chat

- **Edit (NomeDocumento,Sezione):** permette di modificare la sezione di un documento creato precedentemente. Se il documento o la sezione non sono presenti sul server o non si possiedono i permessi necessari per eseguire questa operazione, viene sollevata un'eccezione e il rispettivo errore viene notificato al client che ha richiesto l'operazione. Nel caso in cui erroneamente si inserisca il numero di una sezione in un formato non numerico viene lanciata una *FormatException*;
  Durante la modifica di un documento si ha la possibilità di accedere ad una chat che permette di inviare e ricevere messaggi a tutti gli utenti che stanno modificando contemporaneamente lo stesso documento.
  - **Send (Messagge):** invia un messaggio sulla chat del documento;

  - **Receive ():** scarica dalla chat del documento tutti i messaggi non letti.

- **End-Edit (NomeDocumento,Sezione):** termina la modifica di una sezione del documento e invia le modifiche al Server. Questa operazione è accessibile solamente nel caso in cui sia stata precedentemente aperta una sezione di un documento per la modifica. Nel caso in cui erroneamente si inserisca il numero di una sezione in un formato non numerico viene lanciata una *FormatException*;

2 Scelte implementative

2.1 Registrazione e notifiche RMI

La registrazione degli utenti al servizio e l'invio e la ricezione di notifiche di condivisione dei documenti tra utenti, sono gestite tramite RMI. Il server crea un oggetto remoto e pubblica un riferimento a tale oggetto. I Client recuperano il riferimento all'oggetto remoto creato dal Server e invocano i metodi disponibili.

2.2 Invio richieste al Server

I client comunicano con il server inviando stringhe con un formato specifico per identificare le singole operazioni richieste. In generale la prima parola della stringa identifica l'operazione che il server dovrà eseguire e le parole successive rappresentano i parametri richiesti per l'esecuzione delle operazioni. Il Server gestisce le richieste provenienti dai singoli Client tramite Multiplexing NIO. La *FormatException* si occupa di gestire i casi in cui il formato delle stringhe inviate al Server non è quello atteso dal Server stesso.

2.3 Persistenza dei dati

Per garantire la persistenza degli utenti e dei documenti nel sistema sono state serializzate alcune strutture dati.
All'interno del progetto sono presenti due file ".Json":

- **UserDB.json:** file che memorizza le informazioni degli utenti registrati al sistema. In particolare viene serializzata l'ArrayList "users".

- **DocDB.json:** file che memorizza le informazioni di tutti i documenti creati nel sistema. In particolare viene serializzato il Vector "docDB".

Per la serializzazione è stato utilizzato un ObjectMapper che si occupa di leggere e scrivere le strutture dati. All'avvio del sistema viene deserializzato il "database" degli utenti e il "database" dei documenti per far si che il Server operi sempre su una base di dati consistente. Inoltre i file "UserDB.json" e "DocDB.json" vengono aggiornati

rispettivamente ogni volta che si effettua la registrazione di un nuovo utente o la creazione di un nuovo documento. Anche le notifiche di condivisione di un documento ad un utente che in quell'istante non è online, vengono aggiunte ad una Messagge Box dell'utente e serializzate per far si che che possano essere ricevute e visualizzate anche dopo un riavvio del Server.

2.4 Chiusura forzata di un client

Al fine di mantenere il sistema, in particolare il Server, in uno stato sempre consistente è stata inserita una gestione per l'interruzione forzata di una connessione da parte di un client. Dal momento che alcune operazioni, come ad esempio la modifica di una sezione di un documento, richiedono un accesso in mutua esclusione alla sezione stessa, è stato necessario implementare un metodo che permettesse di liberare la "lock" sulla sezione. Inoltre se un client si disconnette improvvisamente è necessario che il Server aggiorni la lista dei Client connessi.
Per i motivi sopra citati, nel momento in cui il Server cattura una *IOException* causata dalla disconnessione forzata di un client, vengono eseguite le seguenti operazioni in sequenza.

1. **Chiusura sezioni in modifica:** vengono chiuse le sezioni che eventualmente erano rimaste aperte in modifica fino ad un attimo prima della chiusura della connessione del client.

2. **Logout dell'utente:** il server si occupa di rimuovere dall'HashMap dei client connessi la entry associata alla porta del client che ha chiuso forzatamente la connessione.

3 Thread e strutture dati

3.1 Thread attivati

L'architettura del sistema si basa sul paradigma Client - Server. Il server si occupa di gestire le richieste provenienti dai singoli Client tramite Multiplexing NIO. A questo scopo vengono attivati due thread distinti, uno per ogni Client per gestire l'interazione dell'utente con il sistema ed uno per il Server per gestire le richieste entranti. Inoltre nella fase di Editing viene attivato un thread per ogni client per gestire la chat del documento. Tutti i client che stanno modificando un documento possono inviare e ricevere messaggi sullo stesso indirizzo Multicast associato al documento che si sta editando.

3.2 Strutture dati

Di seguito sono elencate le principali strutture dati utilizzate dal Server per garantire il corretto funzionamento del sistema. Le strutture dati relative alle singole classi sono elencate nella sezione 4.

- **ArrayList** $< User >$ **users:** Lista degli utenti registrati al sistema.
- **Vector** $< Document >$ **docDB:** Lista dei documenti presenti sul Server.
- **HashMap** $< String, Integer >$ **clientMap:** HashMap che contiene tutte le associazioni $< NomeUtente, PortaClient >$ utilizzata dal Server per identificare tutti i client connessi in ogni momento.
- **HashMap** $< String, String >$ **chatMap:** HashMap che contiene tutte le associazioni $< NomeDocumento, IndirizzoMulticast >$ utilizzata dal Server per gestire gli indirizzi Multicast da assegnare univocamente ad ogni documento per avviare le Chat.

4 Descrizione delle classi

4.1 Classi principali

Di seguito sono elencate le classi principali che strutturano il progetto.

4.1.1 Client

La classe Client è utilizzata per gestire l'interazione con l'utente. E' presente un Menù principale che permette all'utente di scegliere l'operazione desiderata. Le operazioni disponibili sono tutte quelle offerte dal servizio. Tuttavia alcune di esse sono selezionabili solamente in determinate situazioni. All'interno della classe sono inoltre presenti tutti i metodi, invocati dal menù principale, per inviare le richieste al Server per le singole operazioni.

4.1.2 Server

La classe Server si occupa di gestire tutte le richieste provenienti dai singoli Client. Le richieste di connessione, lettura e scrittura dei canali vengono gestite con Multiplexing. All'interno della classe è presente un selector che si occupa di accettare le connessioni in entrata, leggere le stringhe inviate dai client per identificare le operazioni richieste, scrivere sui canali eventuali risposte e notifiche ai Client.

4.1.3 User

La classe User modella la struttura di un profilo di un qualsiasi utente registrato al servizio.
Al suo interno sono definite le seguenti strutture e variabili:

- **String username:** nome dell' utente;
- **String password:** password dell'utente;
- **Vector** $< String >$ **userDoc:** lista dei documenti creati dall'utente;
- **ConcurrentLinkedQueue** $< String >$ **msgList:** lista delle notifiche ricevute dall'utente quando è offline;
- **int lastDocIndex:** indice dell'ultimo documento aperto per la modifica;
- **String lastSectionIndex:** indice dell'ultima sezione aperta per la modifica.

4.1.4 Document

La classe Document modella la struttura di un qualsiasi documento sul Server.
Al suo interno sono definite le seguenti strutture e variabili:

- **final String creator:** nome del creatore del documento;
- **String name:** nome del documento;
- **Vector** $< String >$ **allowedUsers:** lista degli utenti autorizzati alla modifica del documento;
- **String numSezioni:** numero di sezioni del documento;
- **boolean[ ] isEditing:** array di booleani che identifica le sezioni attualmente in modifica.

In aggiunta ai metodi standard per modificare o ritornare valori delle variabili presenti nella classe Document, sono presenti due metodi per la creazione di un documento e il controllo dei permessi per la modifica del documento stesso.

- **void create():** crea un nuovo documento e la relativa cartella solo nel caso in cui il documento non sia già esistente.
- **boolean checkAllowedUsers(String user):** controlla che l'utente "user" abbia i permessi per modificare il documento.

4.1.5 Chat

La classe Chat modella la struttura delle chat dei documenti. Estende la classe Thread ed al suo interno è presente il metodo run() per permettere la sua esecuzione su di un Thread separato (uno per ogni client che apre un documento per la modifica);
Al suo interno sono definite le seguenti strutture e variabili:

- **String user:** nome dell'utente che ha avviato la chat;
- **InetAddress ia:** indirizzo Multicast per inviare e ricevere messaggi;
- **boolean isRunning:** valore booleano utilizzato per avviare o terminare la chat (default: true);
- **MulticastSocket ms:** multicast socket per connettersi all'indirizzo multicast associato ad un documento;
- **ConcurrentLinkedQueue** $< String >$ **queue:** lista dei messaggi inviati e ricevuti sulla chat;

In aggiunta al metodo run() per avviare la chat è presente un metodo shutDown() utilizzato per terminare la chat.

4.1.6 MyRemoteServer

La classe MyRemoteServer serve per gestire tutte le connessioni RMI. Nel progetto la registrazione di un nuovo utente e l'invio di una notifica ai client per la condivisione di un documento sono gestite tramite RMI.

4.2 Eccezioni

All'interno del progetto sono state aggiunte alcune eccezioni per gestire situazioni particolari che si possono verificare durante l'utilizzo del sistema. Alcune di queste eccezioni inviano anche un codice di errore per far si che il Client possa riconoscere e distinguere queste situazioni anomale.
Di seguito sono elencate le eccezioni in questione:

- **AlreadyExistingAccountException:** eccezione che viene lanciata nel caso in cui si tenti di registrare un nuovo account con un nome gia presente all'interno del database degli utenti;
- **MissingAccountException:** eccezione che viene lanciata nel caso in cui si tenti di effettuare un login o una condivisione con un utente che non è ancora registrato al servizio (e quindi non è presente nel database degli utenti);

- **WrongIdentityException:** eccezione che viene lanciata quando non si inseriscono correttamente le credenziali di accesso nella fase di login;
- **OnlineException:** eccezione che viene lanciata quando si tenta di eseguire un login anche se è già stato effettuato in precedenza;
- **OfflineException:** eccezione che viene lanciata quando si tenta di eseguire delle operazioni senza aver prima effettuato il login;
- **MissingDocumentException:** eccezione che viene lanciata quando si tenta di accedere ad un documento non esistente;
- **ExistingDocumentException:** eccezione che viene lanciata quando si tenta di creare un documento con lo stesso nome di un documento già esistente;
- **EditingException:** eccezione che viene lanciata quando si tenta di editare una sezione che è già in fase di modifica da un altro utente;
- **NotAllowedSharingException:** eccezione che si verifica quando si tenta di condividere con un altro utente un documento del quale non si è il creatore;
- **NotAuthorizedUserException:** eccezione che viene lanciata quando si tenta di modificare una sezione di un documento che non è stato condiviso con l'utente corrente;
- **FormatException:** eccezione che viene lanciata quando si inseriscono parametri in un formato errato.

5 Compilazione ed esecuzione

All'interno del progetto vengono serializzate alcune strutture dati per garantire la persistenza dei dati degli utenti e dei documenti presenti nel Server. A questo scopo prima di procedere alla compilazione del progetto è necessario aggiungere alcune librerie esterne come indicato nella sezione dedicata. Completata questa operazione si può procedere ordinatamente alla compilazione e all'esecuzione delle classi Server e Client.
La prima esecuzione del Server solleverà un'eccezione per la deserializzazione delle strutture dati. Quest'eccezione non è da considerarsi un errore di esecuzione ma solamente un avviso da parte del Server che le strutture dati che sta cercando di deserializzare sono in realtà ancora vuote. Una volta effettuata una registrazione ed una creazione di un documento, i file ".Json" non risulteranno essere più vuoti e verranno deserializzati correttamente dal Server.

5.1 Parametri d'ingresso

Il progetto prevede il settaggio di alcuni parametri d'ingresso per il Client e per il Server. Il Client richiede ordinatamente i seguenti parametri d'ingresso:

- **Port:** porta utilizzata dal Client per connettersi al Server;
- **Editor:** nome dell'editor predefinito utilizzato nel sistema operativo sul quale viene eseguito il progetto. Questo parametro viene utilizzato per scegliere l'editor di testo con il quale la funzione "Edit" dovrà aprire i documenti.
  Di seguito sono elencati i parametri di "Editor" da passare al Client per i sistemi operativi più conosciuti:
  - **Windows:** notepad;
  - **Mac OS:** open;
  - **Linux:** gedit;

Il Server richiede come unico parametro d'ingresso la porta sulla quale dovrà rimanere in ascolto di possibili connessioni da parte dei Client. E' necessario che le porte utilizzate per il Server e i Client siano le stesse.

5.2 Sintassi dei comandi

In questa sezione sono elencati tutti i comandi con la relativa sintassi per la loro corretta esecuzione.

- **Register (Username,Password):** registrazione di un nuovo utente;
- **Login (Username,Password):** login di un utente;
- **Create (NomeDocumento,NumSezioni):** creazione di un nuovo documento;
- **List ():** visualizzazione di tutti i documenti creati dall'utente o condivisi con l'utente stesso;
- **Show (NomeDocumento,Sezione):** visualizzazione di una sezione di un documento;
- **Show (NomeDocumento):** visualizzazione di tutte le sezioni di un documento;
- **Share (NomeDocumento,Utente):** condivisione di un documento con un altro utente;
- **Edit (NomeDocumento,Sezione):** modifica di una sezione di un documento;

- **Send (Messagge):** invio di un messaggio sulla chat del documento;
- **Receive ():** ricezione di tutti i messaggi non letti sulla chat del documento;
- **End-Edit (NomeDocumento,Sezione):** terminazione della modifica di una sezione di un documento.

5.3 Librerie esterne

Affinchè i dati fossero persistenti, alcune strutture sono state serializzate. La serializzazione viene effettuata su un file ".JSON" con l'ausilio di un ObjectMapper. Inoltre, per una corretta serializzazione dei dati, sono state utilizzate le "Jackson Annotation" per specificare la struttura del file ".JSON".
Per garantire un corretto funzionamento del sistema è necessario aggiungere al progetto le seguenti librerie:

- **jackson-core-2.9.7;**
- **jackson-databind-2.9.7;**
- **jackson-annotations-2.9.7.**

E' possibile scaricare le librerie utilizzate nel progetto cliccando ***Qui.***