WORTH

Simone Ianniciello

A.A. 2020/2021

Descrizione generale

Introduzione

L'applicazione WORTH fornisce quasi tutti i servizi tramite interazione **Client-Server** fatta esclusione della chat di progetto che lavora in modalità **Peer-to-Peer**.

Tipi di connessione utilizzati

Java RMI

Tramite una funzione `register(username, hash)` il client richiede la registrazione di un nuovo utente al servizio.

Java RMI Callback

Il server fornisce al client la lista degli utenti registrati e notifica i cambiamenti di stato `(online - offline)`.

UDP Multicast

Ogni progetto ha associata una chat tramite la quale i membri possono comunicare, l'indirizzo di questa chat viene fornito dal server tramite la connessione TCP ma i messaggi vengono poi trasferiti direttamente da un client agli altri.

TCP

Tramite una connessione TCP persistente (creata al momento della richiesta di accesso al servizio) un client puo' effettuare delle operazioni sui dati sui quali ha accesso sul server come:creazione di un progetto, aggiunta di un membro, movimento di una card...
I comandi TCP sono codificati come descritto all'interno del file

`worth/common/CMD.java`

Un esempio di comando e': `0x10<username>0x03<hash>0x03`

`0x10` Codice identificativo del comando di login.

`<username>` Nome utente codificato come sequenza di byte

`0x03` Identifica la fine di un parametro.

`<hash>` hash della password codificato come sequenza di byte

Scelte d'implementazione

Interfaccia L'utente finale interagisce con il server tramite un'interfaccia grafica sviluppata tramite la piattaforma JavaFX.

Persistenza dei file di progetto Ogni volta che viene fatta una modifica ad un progetto viene ricreato e scritto su file-system un file json contenente il nome del progetto, le card associate, e i membri di esso.

Persistenza dei file delle card Ogni volta che viene fatta una modifica ad una card virene ricreato e scritto su file-system un file json contenente il nome della card, la descrizione, e lo storico di tutti i suoi movimenti.

Identificazione utenti Per non spedire e memorizzare le password in chiaro, il client, in fase di registrazione e di login, la trasforma in un hash. Questo metodo non e' comunque molto sicuro, quindi la priorità per un ipotetico prossimo aggiornamento sarebbe modificare questa cosa.

Persistenza degli utenti registrati Gli usernames e gli hash associati sono salvati sul file-system come una sequenza di valori

`<username>:<hash>`

E' stato scelto di non utilizzate un file JSON per questo parametro perché altrimenti all'aggiunta di un nuovo utente sarebbe stato necessario riscrivere l'intero file da capo.

Threads e controllo di concorrenza

Lato server

Il server, al momento, non attiva alcun thread supplementare rispetto al main dato che gestisce le connessioni TCP tramite multiplexing dei canali con NIO. La classe che si occupa del multiplexing pero' e' stata implementata in modo da poter essere attivata come nuovo thread permettendo di eseguire altre operazioni sul thread main.

Per gestire la concorrenza tutte le strutture dati principali utilizzano classi thread safe come la `ConcurrentHashMap`

Lato client

Il client attiva un thread per la gestione della GUI e, al momento dell'apertura di una chat, un thread che si occupa della ricezione e dell'invio dei messaggi sul gruppo multicast.

Descrizione delle classi

La struttura del progetto e' divisa in 3 packages principali:

client Contiene tutte le classi utilizzate per il funzionamento del client.

server Contiene tutte le classi utilizzate per il funzionamento del server.

common Contiene le interfacce per la comunicazione RMI e le strutture dati che devono restare immutate tra client e server.

client

```
client
├── chat
│   └── Chat.java
├── gui
│   ├── Card.java
│   ├── GUI.java
│   ├── LoginController.java
│   ├── MenuController.java
│   ├── login.fxml
│   └── menu.fxml
├── tcp
│   ├── Commands.java
│   ├── Connection.java
│   └── Response.java
├── users
│   ├── Callback.java
│   └── Users.java
├── utils
│   ├── Arguments.java
│   ├── Const.java
│   ├── NoSelectionModel.java
│   └── Utils.java
└── clientMain.java
```

clientMain.java Main class

chat si occupa della ricezione e dell'invio dei messaggi nel giusto gruppo multicast.

gui Le classi all'interno di questo package si occupano di mostrare l'interfaccia grafica all'utente e di gestire le richieste provenienti da essa.

tcp Contiene le classi di supporto alla comunicazione TCP per l'invio delle richieste al server.

users Package che si occupa di mantenere le liste degli utenti e il loro stato, e aggiornarle tramite RMI callback.

utils Contiene tutte le classi di supporto.

server

```
server
├── projects
│   ├── Card.java
│   ├── Projects.java
│   └── Project.java
├── tcp
│   └── Connection.java
├── users
│   ├── Users.java
│   ├── User.java
│   ├── RegisterService.java
│   └── Callback.java
├── chat
│   └── IPGen.java
├── utils
│   ├── Utils.java
│   ├── Arguments.java
│   ├── ConcurrentHashSet.java
│   └── Const.java
└── mainServer.java
```

mainServer.java Main class

projects Contiene le classi utili alla creazione e modifica di tutti i dati dei progetti.

tcp Contiene la classe `Connection.java`, utilizzata per l'apertura della socket per l'accettazione delle richieste TCP e per l'interpretazione dei comandi in arrivo.

users Contiene il *database* degli utenti registrati e tutte le operazioni per essi, oltre alle classi utilizzate per la notifica degli utenti al client tramite callback.

chat La classe `IPGen.java` e' utilizzata per la creazione di indirizzi multicast unici per ogni progetto.

utils Contiene tutte le classi di supporto.

common

```
common
├── CMD.java
├── Status.java
├── CallbackInterface.java
├── CallbackServerInterface.java
└── RegisterServiceInterface.java
```

CMD.java Contiene la descrizione e il codice dei comandi utilizzabili sulla connessione TCP.

Status.java Enum usato per distinguere gli stati in cui possono trovarsi le card.

***Interface.java** Interfacce per la definizione dei metodi usati da RMI.

Guida all'utilizzo

Tutte le librerie sono gia presenti nella cartella /lib

Server

Librerie utilizzate

- CommandLine Utilizzata per il parsing dei comandi su riga di comando. (source)
- JSONSimple Utilizzata per convertire oggetti in stringhe JSON. (source)

Client

Librerie utilizzate

- CommandLine Utilizzata per il parsing dei comandi su riga di comando. (source)
- JavaFX Utilizzata per la creazione della GUI. (Windows download)