Laboratorio di Reti, Corsi A e B

WORTH: WORkTogetHer

Progetto di Fine Corso A.A. 2020/21

1. Descrizione del problema

Negli ultimi anni sono state create numerose applicazioni collaborative, per la condivisione di contenuti, messaggistica, videoconferenza, gestione di progetti, ecc. In questo progetto didattico, WORTH (WORkTogetHer), ci focalizzeremo sull'organizzazione e la gestione di progetti in modo collaborativo. Le applicazioni di collaborazione e project management (es. Trello, Asana) aiutano le persone a organizzarsi e coordinarsi nello svolgimento di progetti comuni. Questi possono essere progetti professionali, o in generale qualsiasi attività possa essere organizzata in una serie di compiti (es. to do list) che sono svolti da membri di un gruppo: le applicazioni di interesse sono di diverso tipo, si pensi alla organizzazione di un progetto di sviluppo software con i colleghi del team di sviluppo, ma anche all'organizzazione di una festa con un gruppo di amici.

Alcuni di questi tool (es. Trello) implementano il metodo Kanban (cartello o cartellone pubblicitario, in giapponese), un metodo di gestione "agile". La lavagna Kanban fornisce una vista di insieme delle attività e ne visualizza l'evoluzione, ad esempio dalla creazione e il successivo progresso fino al completamento, dopo che è stata superata con successo la fase di revisione. Una persona del gruppo di lavoro può prendere in carico un'attività quando ne ha la possibilità, spostando l'attività sulla lavagna.

Il progetto consiste nell'implementazione di **WORkTogetHer (WORTH)**: uno strumento per la gestione di progetti collaborativi che si ispira ad alcuni principi della metodologia Kanban.

2. Specifica delle operazioni

Gli utenti possono accedere a WORTH dopo registrazione e login.

In WORTH, un progetto, identificato da un nome univoco, è costituito da una serie di "card" ("carte"), 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 e una descrizione testuale. Il nome assegnato alla card deve essere univoco nell'ambito di un progetto. Ogni progetto ha associate quattro liste che definiscono il flusso di lavoro come passaggio delle card da una lista alla successiva: TODO, INPROGRESS, TOBEREVISED, DONE. Qualsiasi membro del progetto può spostare la card da una lista all'altro, rispettando i vincoli illustrati nel diagramma in FIG. 1.

Le card appena create sono automaticamente inserite nella lista TODO. Qualsiasi membro può spostare una card da una lista all'altra. Quando tutte le card sono nella lista DONE il progetto può essere cancellato, da un qualsiasi membro partecipante al progetto.

Ad ogni progetto è associata una chat di gruppo, e tutti i membri di quel progetto, se online (dopo aver effettuato il login), possono ricevere e inviare i messaggi sulla chat. Sulla chat il sistema invia inoltre automaticamente le notifiche di eventi legati allo spostamento di una card del progetto da una lista all'altra.

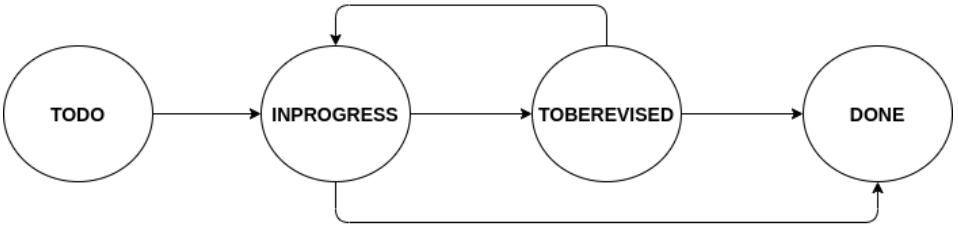

FIG.1

Un utente registrato e dopo login eseguita con successo ha i permessi per:

- recuperare la lista di tutti gli utenti registrati al servizio;
- recuperare la lista di tutti gli utenti registrati al servizio e collegati al servizio (in stato online);
- creare un progetto;
- recuperare la lista dei progetti di cui è membro.

Un utente che ha creato un progetto ne diventa automaticamente membro. Può aggiungere altri utenti registrati come membri del progetto. Tutti i membri del progetto hanno gli stessi diritti (il creatore stesso è un membro come gli altri), in particolare:

- aggiungere altri utenti registrati come membri del progetto;
- recuperare la lista dei membri del progetto;
- creare card nel progetto;
- recuperare la lista di card associate ad un progetto;
- recuperare le informazioni di una specifica card del progetto;
- recuperare la "storia" di una specifica card del progetto (vedi seguito per dettagli);
- spostare qualsiasi card del progetto (rispettando i vincoli di Fig.1);
- inviare un messaggio sulla chat di progetto;
- leggere messaggi dalla chat di gruppo;
- cancellare il progetto.

Di seguito sono specificate le operazioni offerte dal servizio. In sede di implementazione è possibile aggiungere ulteriori parametri, se necessario.

register(nickUtente, password): per inserire un nuovo utente, il server mette a disposizione una operazione di registrazione di un utente. Il server risponde con un codice che può indicare l'avvenuta registrazione, oppure, se il nickname è già presente, o se la password è vuota, restituisce un messaggio d'errore. Come specificato in seguito, le registrazioni sono tra le informazioni da persistere.

login(nickUtente, password): login di un utente già registrato per accedere al servizio. Il server risponde con un codice che può indicare l'avvenuto login, oppure, se l'utente ha già effettuato la login o la password è errata, restituisce un messaggio d'errore.

logout(nickUtente): effettua il logout dell'utente dal servizio.

listUsers(): utilizzata da un utente per visualizzare la lista dei nickUtente registrati al servizio e il loro stato (online o offline).

listOnlineusers(): utilizzata da un utente per visualizzare la lista dei nickUtente registrati al servizio e online in quel momento.

listProjects(): operazione per recuperare la lista dei progetti di cui l'utente è membro.

createProject(projectName): operazione per richiedere la creazione di un nuovo progetto. Se l'operazione va a buon fine, il progetto è creato e ha come membro l'utente che ne ha richiesto la creazione.

addMember(projectName, nickUtente): operazione per aggiungere l'utente *nickUtente* al progetto *projectname.* Se l'utente è registrato l'aggiunta come membro è eseguita senza chiedere il consenso a nickUtente, se l'utente non è registrato l'operazione non può essere completata e il servizio restituisce un messaggio di errore.

showMembers(projectName): operazione per recuperare la lista dei membri del progetto.

showCards(projectName): operazione per recuperare la lista di card associate ad un progetto *projectName.*

showCard(projectName, cardName): operazione per recuperare le informazioni (nome, descrizione testuale, lista in cui si trova in quel momento) della card *cardName* associata ad un progetto *projectName.*

addCard(projectName, cardName, descrizione): operazione per richiedere l'aggiunta della card di nome *cardName* al progetto *projectname*. La card deve essere accompagnata da una breve testo descrittivo. La card viene automaticamente inserita nella lista TODO.

moveCard(projectName, cardName, listaPartenza, listaDestinazione): operazione per richiedere lo spostamento della card di nome *cardName* al progetto *projectname* dalla lista *listaPartenza* alla lista *listaDestinazione.*

getCardHistory(projectName, cardName): operazione per richiedere la "storia" della card, ovvero la sequenza di eventi di spostamento della card, dalla creazione allo spostamento più recente.

readChat(projectName)[1]*:* operazione per visualizzare i messaggi della chat del progetto *projectName*

sendChatMsg(projectName, messaggio)[1]*:* l'utente invia un messaggio alla chat del progetto *projectName*

cancelProject(projectName): un membro di progetto chiede di cancellare un progetto. L'operazione può essere completata con successo solo se tutte le card sono nella lista DONE.

Permessi	Utente	Membro di progetto
listUsers	X	
listOnlineusers	X	
listProjects	X	
createProject	X	
addMember		X
showMembers		X
showCards		X
showCard		X

[1] per dettagli sulla chat vedi Sezione 3 Specifiche per l'implementazione

addCard		X
moveCard		X
getCardHistory		X
readChat		X
sendChatMsg		X
cancelProject		X

3. Specifiche per l'implementazione

Nella realizzazione del progetto devono essere utilizzate molte delle tecnologie illustrate durante il corso. In particolare:

- la fase di registrazione viene implementata mediante RMI.
- La fase di login deve essere effettuata come prima operazione dopo aver instaurato una connessione TCP con il server. In risposta all'operazione di login, il server invia anche la lista degli utenti registrati e il loro stato (online, offline). A seguito della login il client si registra ad un servizio di notifica del server per ricevere aggiornamenti sullo stato degli utenti registrati (online/offline). Il servizio di notifica deve essere implementato con il meccanismo di RMI callback. Il client mantiene una struttura dati per tenere traccia della lista degli utenti registrati e il loro stato (online/offline), la lista viene quindi aggiornata a seguito della ricezione di una callback, attraverso la quale il server manda gli aggiornamenti: nuove registrazioni, cambiamento di stato di utenti registrati (online/offline).
- 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 visualizzazione della lista degli utenti (*listUsers* e *listOnlineusers*) che usano la struttura dati locale del client aggiornata tramite il meccanismo di RMI callback (come descritto al punto precedente) e le operazioni sulla chat.
- Il server può essere realizzato multithreaded oppure può effettuare il multiplexing dei canali mediante NIO.
- L'utente interagisce con WORTH mediante un client che può utilizzare una semplice interfaccia grafica, oppure una interfaccia a linea di comando, definendo un insieme di comandi, presentati in un menu.
- Deve essere implementata una struttura dati separata per ciascuna lista di progetto (ovvero quattro liste per progetto).
- La chat di progetto deve essere realizzata usando UDP multicast (un client può inviare direttamente i messaggi ad altri client).
- Implementazione della chat: nel caso in cui si decida di implementare l'interfaccia grafica, essa prevederà due semplici aree di testo in cui rispettivamente inserire/ricevere i messaggi testuali

inviati alla chat. In questo caso, i messaggi vengono immediatamente presentati all'utente, mano a mano che vengono ricevuti. Invece, nel caso si preferisca una interazione con WORTH a linea di comando, saranno definiti due comandi per, rispettivamente, inviare nuovi messaggi alla chat/ricevere tutti i messaggi ricevuti a partire dall'ultima esecuzione del comando di visualizzazione messaggi. In questo caso, i messaggi vengono presentati all'utente in modo asincrono, su sua richiesta.

- Il server persiste lo stato del sistema, in particolare: le informazioni di registrazione, la lista dei progetti (inclusi membri, card e lo stato delle liste). Lo stato dei progetti deve essere reso persistente sul file system come descritto di seguito: una directory per ogni progetto e un file per ogni card del progetto (sul file sono accodati gli eventi di spostamento relativi alla card). I messaggi delle chat non devono essere persistiti. Quando il server viene riavviato tali informazioni sono utilizzate per ricostruire lo stato del sistema.

4. Modalità di svolgimento e consegna

Il materiale da consegnare sul moodle del corso di Laboratorio di Programmazione di Rete deve comprendere:

- il codice dell'applicazione e di eventuali programmi utilizzati per il test delle sue funzionalità.
    - Il codice deve essere ben commentato
    - La classe che contiene il metodo main deve contenere “main” nel nome, es. ServerMain.java; per le altre classi non ci sono vincoli, ma nomi mnemonici sono ovviamente apprezzati.
- la relazione in formato pdf che deve contenere:
    - una descrizione generale dell'architettura complessiva del sistema, in cui sono motivate le scelte di progetto;
    - uno schema generale dei threads attivati da ogni componente e delle strutture dati utilizzate, con particolare riferimento al controllo della concorrenza;
    - una descrizione sintetica delle classi definite ed indicazioni precise sulle modalità di esecuzione.
    - una sezione di istruzioni su come compilare ed eseguire il progetto (librerie esterne usate, argomenti da passare al codice, sintassi dei comandi per eseguire le varie operazioni...). Questa sezione deve essere un manuale di istruzioni semplice e chiaro per gli utilizzatori del sistema.
    - L'organizzazione e la chiarezza della relazione influiranno sul voto finale.

Il codice che non compila non verrà considerato. Per essere considerato sufficiente e quindi ammissibile per la discussione, il progetto deve implementare tutte le funzioni precedentemente illustrate.

Relazione e codice sorgente devono essere consegnati su Moodle in un unico archivio compresso. Non è richiesta una particolare estensione, ma è fortemente consigliato attenersi alle più comuni: .zip e .tar (.gz, .rar e .7z come seconde scelte). La cartella compressa contenente la soluzione dell'esercizio deve essere chiamata seguendo il formato: MATRICOLA COGNOME NOME.zip. Per esempio, lo studente Mario Rossi con matricola e 012345 deve sottomettere la soluzione in una cartella compressa nominata 012345 ROSSI MARIO.zip

Per domande di chiarimento sul progetto verrà attivato un forum di discussione su moodle per ciascuno dei moduli di Laboratorio A e B su cui potrete pubblicare le vostre domande.

5. Esempio di utilizzo interfaccia CLI

Di seguito mostriamo alcuni esempi dell'utilizzo dell'interfaccia a riga di comando (CLI), lato client. In tutti gli esempi assumiamo che il server sia già attivo. Le righe che iniziano con '>' sono esempi di comandi da immettere, le righe che iniziano con '<' sono esempi di risposte. La specifica completa della CLI deve essere inclusa nella relazione ed è caldamente consigliata l'implementazione anche di un comando '`help`' che riassuma la sintassi dei comandi accettati dalla CLI.

Registrazione e login utente

Faccio partire il client ($ indica che sono sul terminale):
```
$ java MainClientClass
```

Richiedo il login di user1, ma l'utente user1 non esiste:
```
> login user1 pass
< errore, utente user1 non esistente
```

Registro l'utente user1:
```
> register user1 pass
< ok
```

Richiedo il login di user1:
```
> login user1 pass
< user1 logged in
```

Richiedo il login di un utente mentre c'è già un utente loggato:
```
> login user2 password
< c'è un utente già collegato, deve essere prima scollegato
```

Eseguo il logout dell'utente attualmente loggato:
```
> logout
< user1 scollegato
```

Creazione e lavoro sul progetto

Aubrey *Drake* Graham decide di organizzare una call skype con i suoi amici per rimanere in contatto con loro, durante questo periodo di quarantena. Decide gentilmente di usare WORTH per fare pubblicità al vostro servizio. Fa partire il client dalla riga di comando:

```
$ java MainClientClass
```

Drake fa il login:
```
> login drake h0tl1ne.BLING
< utente drake logged in
```

Drake crea un progetto:
```
> create_project call_skype_quarantena
< progetto call_skype_quarantena creato
```

Drake aggiunge alcuni task al progetto:
```
> add_card call_skype_quarantena preparare_meme
< card preparare_meme aggiunta al progetto call_skype_quarantena

> add_card call_skype_quarantena preparare_giochi
< card preparare_giochi aggiunta al progetto call_skype_quarantena
```

Drake prepara il torneone di “non ho mai”:

```
> move_card call_skype_quarantena preparare_giochi todo inprogress
< ok
```

Drake, colpito da folgorante illuminazione, prepara alcuni schemi per dei meme (principalmente di tipo drakeposting):

```
> move_card call_skype_quarantena preparare_meme todo inprogress
< ok
```

I meme preparati lo soddisfano, ma decide che vuole farli prima vedere ad alcuni suoi amici per accertarsi che facciano ridere:

```
> move_card call_skype_quarantena preparare_meme inprogress toberevised
< ok
```

Decide di chiedere feedback sui meme alla sua amica Callie

Callie deve registrarsi prima di poter collaborare al progetto:

```
> register Callie CoolGirl
< ok
```

Drake invita Callie a collaborare al progetto:

```
> add_member call_skype_quarantena Callie
< ok
```

Callie fa il login:

```
> login Callie CoolGirl
< ok
```

Callie controlla i meme e li ritiene soddisfacenti, per cui decreta che il compito è finito:

```
> move_card call_skype_quarantena preparare_meme toberevised done
< ok
```

Callie invia un messaggio a Drake per informarlo di aver completato il task:

```
> send call_skype_quarantena “bro, nice job, cya l8r”
< message sent
```

L’ora della chiamata si avvicina, Drake decide di fare un punto della situazione sui task:

```
> show_cards call_skype_quarantena
< card: nome=preparare_giochi, stato=inprogress
< card: nome=preparare_meme, stato=done
```

Drake controlla se ci sono messaggi in chat:

```
> receive call_skype_quarantena

< messaggio da Worth: “Callie ha spostato preparare_meme da inprogress
a done”
< Callie ha detto: “bro, nice job, cya l8r”
< Non ci sono altri messaggi
```

Il torneo di “non ho mai” è pronto, per cui decreta che il compito è finito:

```
> move_card call_skype_quarantena preparare_giochi inprogress done
< ok
```

Tutti i task sono finiti, a questo punto il progetto può essere cancellato:

```
> cancel_project call_skype_quarantena
< progetto call_skype_quarantena chiuso
```

6. Esempio di utilizzo GUI

L'utente può interagire con il servizio Worth anche mediante una semplice interfaccia grafica (GUI). Di seguito mostriamo alcuni esempi.

Il panel `Worth` può essere utilizzato per l'inserimento delle credenziali dell'utente. Dopo aver digitato username e password nelle apposite *TextBox,* l'utente può cliccare sul relativo bottone per richiedere al server l'operazione di registrazione e login.

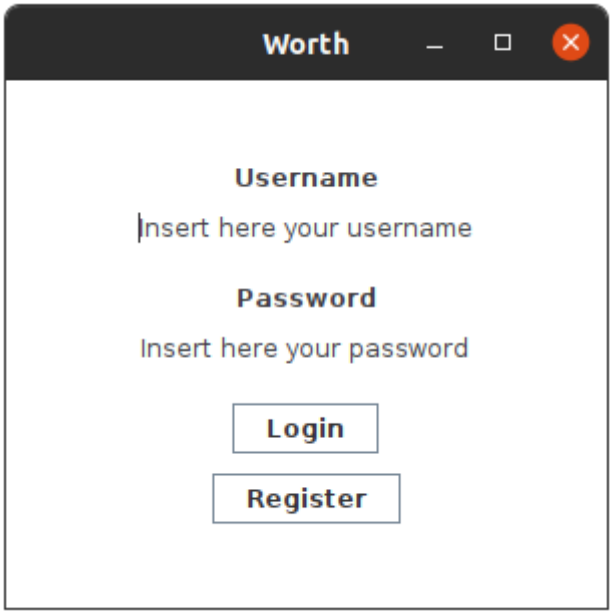

FIG. 2

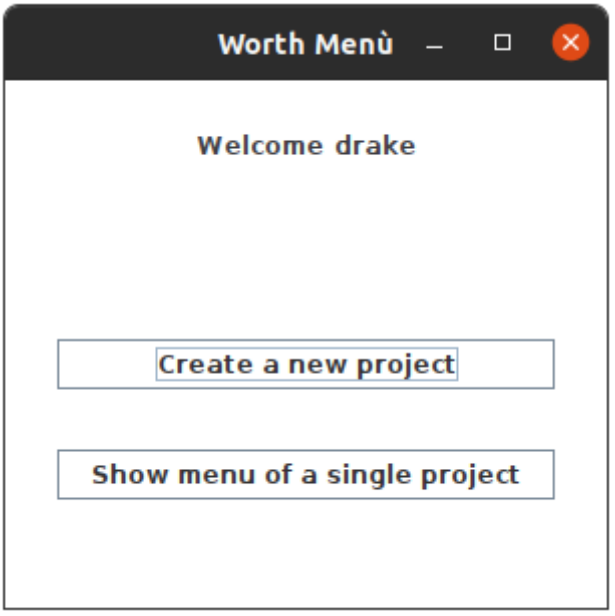

FIG. 3

Quando il server comunica il successo dell'operazione, viene aperto il `Worth Menù` mostrato in figura 3. Attraverso questa interfaccia l'utente può creare un nuovo progetto oppure mostrare l'insieme delle azioni possibili su un progetto esistente.

In figura 4 viene mostrato il menù del progetto *call_skype_quarantena.* Attraverso l'interfaccia relativa ad un singolo progetto un utente può creare e spostare le diverse card.

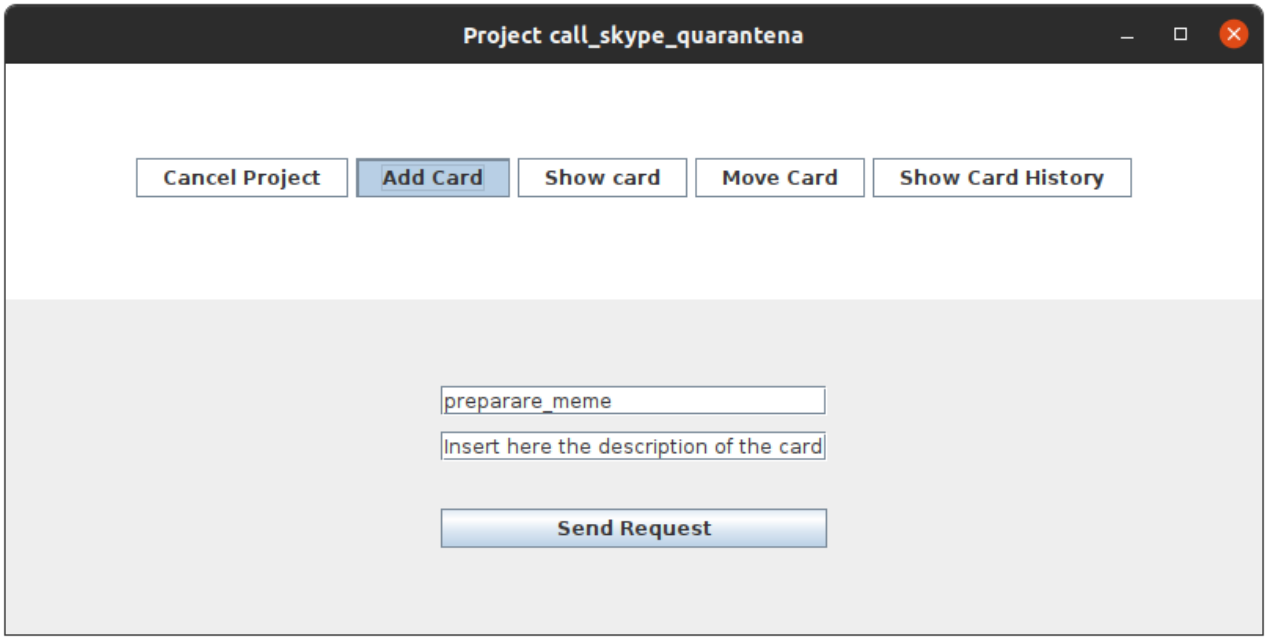

FIG. 4

Nella figura 4 abbiamo riportato una possibile interfaccia grafica per la visualizzazione dell'operazione *AddCard*. Attraverso due semplici *TextBox* l'utente inserisce nome e descrizione della card che desidera inserire. Infine, cliccando il bottone *Send Request*, il client invia automaticamente la richiesta dell'operazione al server.

Successivamente, l'utente decide di spostare la card *prepare_meme* dalla lista *todo* alla lista *inprogress*. In figura 5 viene mostrato un esempio di interfaccia grafica per la gestione della *MoveCard.* In questo caso l'utente inserisce il nome della card, la lista di partenza e la lista di arrivo.

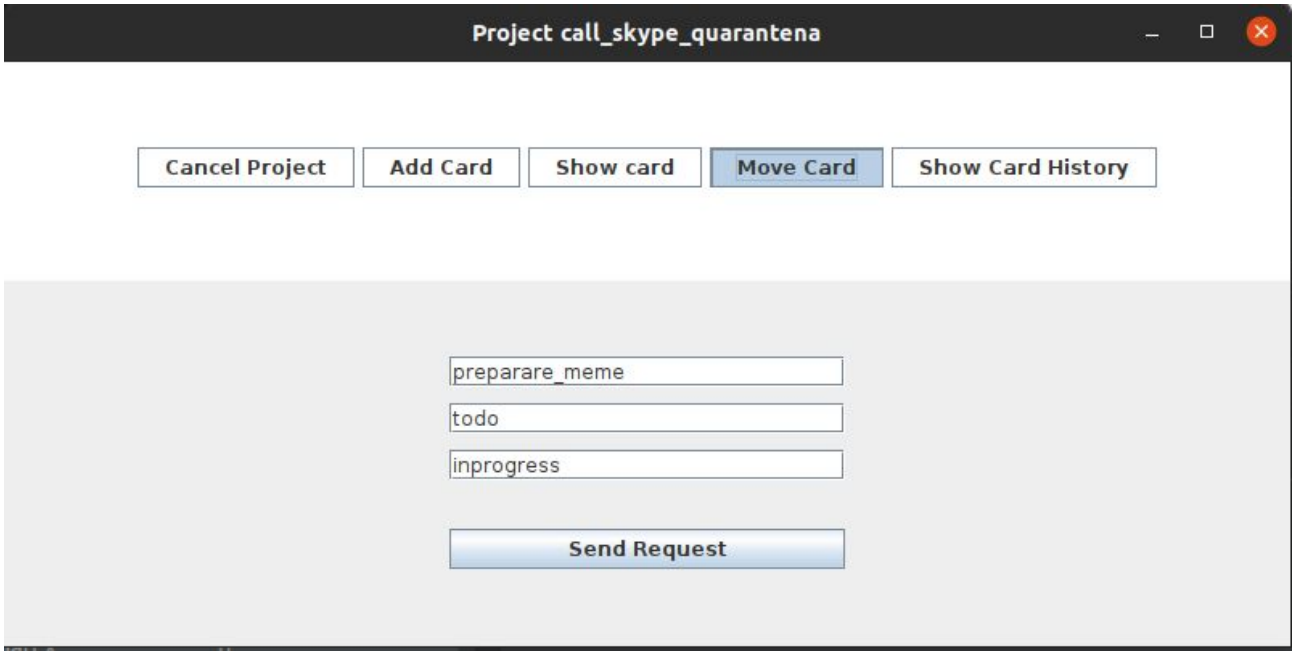

FIG. 5

Durante la modifica delle varie liste del progetto, l'utente può inviare messaggi sulla chat del team attraverso l'apposita interfaccia grafica mostrata in figura 6.

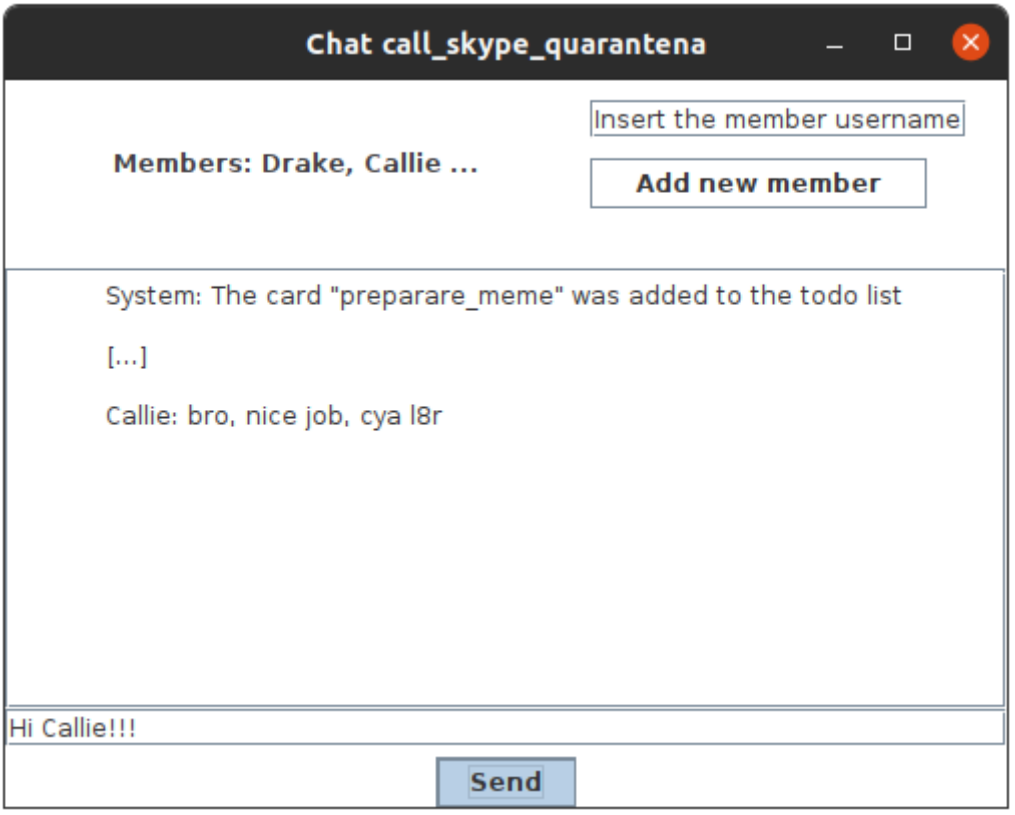

FIG. 6

Buon divertimento Drake per la tua call!
P.S. Sotto, uno dei meme che Drake ha mostrato ai suoi amici prima di iniziare il torneo.