TURING

Distributed Collaborative Editing

Progetto individuale di Luca Murgia

Università di Pisa

Marzo 2020

Indice generale

Introduzione

Questo progetto di fine anno, realizzato per il corso di Laboratorio di Reti, prevede la realizzazione di TURING (disTribUted collaboRative edItiNG), un insieme di servizi finalizzato all'editing collaborativo di documenti.

Per questo progetto ho voluto dare molta importanza alla struttura semplice del codice, alla sua modularità e soprattutto all'interfaccia grafica, cercando di rendere il funzionamento del programma il più intuitivo e semplice possibile per l'utente.

A questo proposito ho utilizzato un insieme di JList all'interno dell'interfaccia del menù, l'utente avrà quindi la possibilità di visualizzare in ogni momento una lista di tutti i documenti a cui ha accesso, assieme a tutte le sezioni che li compongono e tutti gli altri utenti che vi possono accedere.

Architettura software e scelte di progettazione

Il software è diviso in due package:

- **client**: Insieme di interfacce grafiche, chat e editor di testo finalizzato alla creazione e all'editing di documenti di testo condivisi.
- **server**: Fornisce un sistema di gestione dati centralizzato, capace di memorizzare e modificare documenti e dati utente.

LATO CLIENT

Il Client prevede tre diverse classi grafiche create utilizzando la libreria javax swing:

- la classe **LoginGUI** contiene l'interfaccia di login
- la classe **MainMenuGUI** contiene l'interfaccia del menù principale
- la classe **EditorGUI** contiene l'interfaccia dell'editor di testo e chat

Possiede una struttura minimale con al centro un'unica classe: **ControlClient**, la quale crea e comunica con le interfacce grafiche precedentemente citate e gestisce la comunicazione con il server.

La **comunicazione Client-Server** avviene tramite invio di messaggi di formato standard per mezzo di una connessione TCP che viene avviata al momento del login del client.

I messaggi spediti e ricevuti sono gestiti in modo univoco per mezzo del **campo ID**: un byte rappresentante l'operazione richiesta da parte del client o la risposta specifica da parte del server.

LATO SERVER

Il cuore del server è costituito dalla classe **ControlServer**, la quale:

- Crea e gestisce il **registro documenti** ed il **registro utenti**, implementati rispettivamente per mezzo di una **HashMap** e una **ConcurrentHashMap**.
- Crea e gestisce la connessione TCP con i vari client: un Threadpool multithread mette in comunicazione le classi ControlClient con il loro specifico **Handler** nel lato server, uno per ogni client, che riceve e gestisce le operazioni che vengono richieste.

Il registro utenti ed il registro documenti vengono memorizzati all'interno di rispettivi file tramite i metodi della classe astratta **SaveLibrary**, invocati alla chiusura del server.

La registrazione degli utenti viene effettuata tramite l'invocazione del metodo remoto register.
Le classi ControlClient e ControlServer, a questo proposito, istanziano due altre classi: la classe **RegistrationClient** invoca il metodo remoto sulla classe **RegistrationServer**, passando come parametro l'username e la password dell'utente da registrare nel **Registro Utenti**.

Thread, Concorrenza e Strutture Dati

In questa sezione sono elencati i vari thread che vengono avviati durante l'esecuzione del programma

LATO CLIENT

Le classi che gestiscono l'interfaccia grafica vengono avviate da **thread separati**, in questo modo più interfacce di visualizzazione possono essere visualizzate contemporaneamente da un solo client.

La classe ControlClient avvia quindi diversi thread grafici che verranno successivamente chiusi tramite il metodo **dispose()** fornito dalla libreria javax swing.

Il thread che gestisce l'editor, ogni volta che viene avviato, genera a sua volta un **thread UDP** in ascolto sul canale relativo al documento aperto.

L'insieme di tutti i thread del client è associato ad un **unico handler** nel lato server che, se necessario, può richiedere la lock su strutture dati condivise.

LATO SERVER

La funzione **main** del server genera un **unico thread grafico** avente un bottone di avvio e un bottone di stop.

Alla pressione del bottone START viene creato un **ControlServer thread** che rimane in ascolto sul canale TCP, generando, tramite un threadpool, tanti **thread Handler** quanti sono i client che hanno effettuato il login.

Un **thread RMI** rimane attivo per gestire le richieste remote di registrazione dei client.

Alla pressione del bottone STOP il thread centrale del server riceve una interrupt

- Vengono salvati tutti i dati utente e documenti
- Il registro RMI, il suo binding e lo stub del ControlServer precedentemente esportati vengono rimossi tramite **unexport()** e **unbind()**
- Una **shutdown()** viene invocata sul threadpool che, a sua volta, interrompe tutti i thread handler in esecuzione su di esso.

STRUTTURE DATI

Le strutture dati che ho scelto di utilizzare sono:

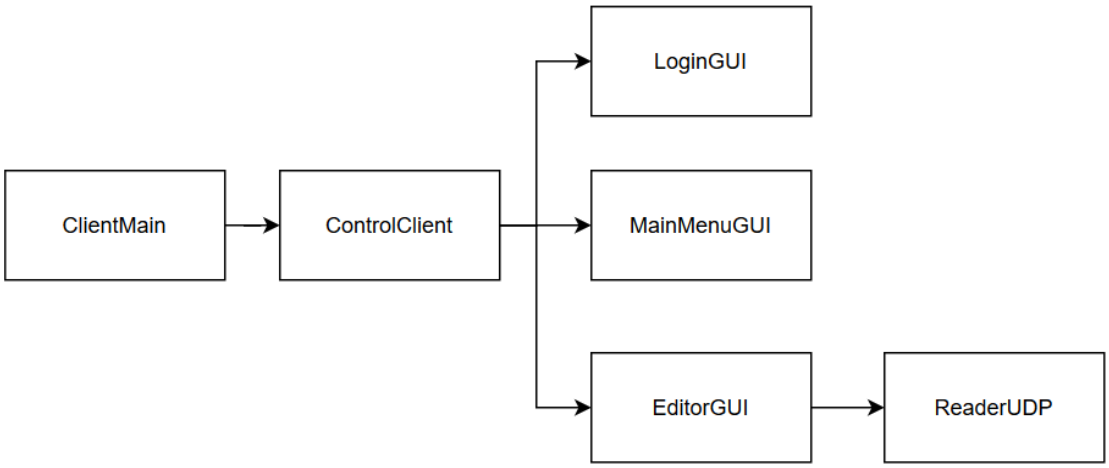
- Una **HashMap** contenente il registro documenti in cui:
  - Il nome del documento ha la funzione di chiave della mappa hash
  - I valori contenuti nel registro appartengono alla classe **Document**
- Una **ConcurrentHashMap** per il registro utenti in cui:
  - Il nome utente ha la funzione di chiave della mappa
  - I valori contenuti nel registro appartengono alla classe **UserData**

Le **lock** sulle sezioni che vengono modificate sono gestite dalla classe **Document**, esiste infatti un array con tante lock quante sono le sezioni di un documento.
Le lock e unlock di una sezione vengono effettuate dall'handler, previa richiesta tramite messaggio da parte del relativo client.

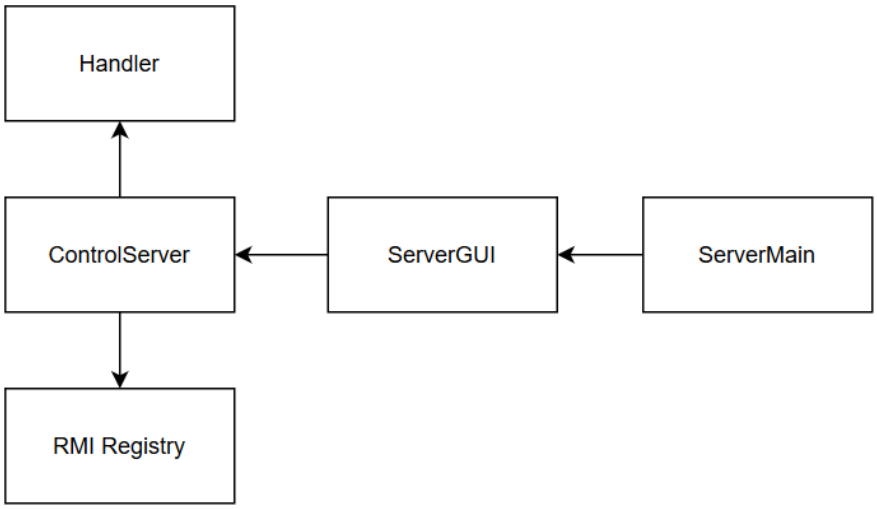
Un particolare Message ID fa sì che l'handler verifichi la presenza di una lock sulla sezione richiesta:

- Restituisce un messaggio di errore se la lock è già acquisita da un altro handler
- Se la lock è libera viene acquisita, un thread editor viene quindi avviato
- La lock viene rilasciata alla chiusura dell'editor.

SCHEMA THREAD: LATO CLIENT



SCHEMA THREAD: LATO SERVER



Classi

In questa sezione viene fornita un elenco delle principali classi del programma, affiancate a una breve descrizione del loro funzionamento.

CLASSI: LATO CLIENT

- **ClientMain**: Avvia il ControlClient.
- **ControlClient**:
  - Riceve i comandi dai thread di interfaccia
  - Spedisce richieste al server ed elabora le risposte
  - Istanzia la classe RMI che lancia la funzione remota register all'interno del server.
- **Message**: Oggetto contenente i parametri che vengono spediti come messaggio dal client o dal server.
  - il più importante di questi è il campo **ID**: un byte rappresentante l'operazione richiesta dal client o il tipo di risposta del server.
  - Alla ricezione di un messaggio segue l'esecuzione di una funzione, i campi rimanenti del messaggio rappresentano i parametri che verranno applicati alla funzione.
- **ReaderUDP**: Semplice servizio UDP multicast, riceve datagrammi da un gruppo associato al documento, mostra il testo contenuto nei datagrammi alla TextArea di chat dell'editor.
- **RegistrationClient**: classe contenente il registry del server e il suo stub, questa classe può invocare sul server il metodo register, che salva i dati di un nuovo utente all'interno del registro utenti, passando come parametro il nome utente e la password del client.
- **LoginGUI**: interfaccia utente che mostra una semplice schermata di login, possiede:
  - Due campi di testo dove inserire nome utente e password
  - Due bottoni per effettuare le operazioni di login e registrazione.
- **MainMenuGUI**: interfaccia utente che mostra il menù principale, possiede:
  - Bottoni rappresentanti le funzioni invocabili

        - I parametri delle funzioni invocate dai bottoni vengono scelti sia da riga di testo, per mezzo di JOptionPane, che dagli elementi selezionati all'interno delle Jlists
    - Una JList di documenti apribili
    - Una JList contenente le varie sezioni del documento selezionato
    - Una JList contenente gli utenti registrati che possono essere invitati all'editing di un documento
    - Una JList contenente gli utenti che già hanno accesso al documento selezionato.
- **EditorGUI**: interfaccia avente tre campi di testo:
    - Un campo **editor** (JEditorPane) che mostra, ed eventualmente permette di modificare, il contenuto di un documento selezionato o di una singola sezione.
    - Un campo **chatReader** (JTextArea) non modificabilee che mostra i messaggi ricevuti nella chat dal momento in cui l'utente apre il documento.
    - Un campo **chatWriter** (JEditorPane) sempre modificabile, che permette di scrivere ed inviare, tramite pressione del tasto **send** (JButton) messaggi sulla chat.

CLASSI: LATO SERVER

- **ServerMain**: Avvia il ControlServer
- **ControlServer**:
  - Avvia il servizio di registrazione e quello di login
  - Entra in un loop in cui accetta le richieste di login del client e, per ogni richiesta, genera un Handler che viene eseguito da un Threadpool multithread.
  - Ha accesso al Registro dei documenti e al Registro Utenti, che può caricare da file e aggiornare.
- **RegistrationServer**: Offre un servizio RMI con registry e stub del ControlServer, contiene il metodo remoto register().
- **Handler**: gestore del client, possiede un loop continuo in cui accetta le richieste, le elabora e restituisce un messaggio di risposta.
- **SaveLibrary**:
  - Contiene funzioni invocate dal server per il salvataggio su file dei dati utente e dei documenti.
  - I dati salvati sono riutilizzabili dal server in future esecuzioni.
- **ServerGui**: Interfaccia grafica minimale del server, possiede
  - Un EventLog ispezionabile
  - Un bottone START per avviare i servizi del server
  - Un bottone STOP per chiudere il server interrompendo i thread che lo compongono e salvando i dati.
- **UserData**: Contiene i dati relativi agli utenti salvati nel registro utenti
  - Nome utente
  - Password
  - Vettore dei documenti a cui si ha accesso
  - Inviti pendenti

- **Document**: Classe contenuta nel registro documenti, composta da
    - Un array di stringhe, rappresentanti il testo di una sezione e visualizzabili tramite l'editor.
        - Queste vengono passate come parametro dei messaggi al client a seguito di una richiesta di editing o di visualizzazione di un documento.
    - Un array di lock, una per ogni sezione, che viene acceduto al momento dell'editing e rilasciato alla sua fine.
        - **DocumentData**: contenente tutti i metadati del documento, quali
            - Nome del documento
            - Numero di sezioni del documento
            - Owner del documento
            - Vettore di utenti con cui il documento è stato condiviso

Modalità di esecuzione

In questa sezione viene descritta la sequenza di eventi principale, sia dal lato client che dal lato server, nel caso in cui un utente voglia registrarsi al servizio, creare e modificare un documento.

pre-condizioni: Il server è attivo al momento in cui il client viene avviato

LATO CLIENT: REGISTRAZIONE, LOGIN E RICHIESTA DI EDITING

- **ClientMain** comincia l'esecuzione, istanzia e avvia **ControlClient**
- **ControlClient** mostra l'interfaccia di login avviando **LoginGUI** in un thread separato.
  - Il **bottone register** viene premuto
    - Vengono prelevati **username** e **password** dai relativi campi di testo
    - La registrazione viene avviata nel server con i parametri prelevati, attraverso **RegistrationClient**
    - I dati inseriti in fase di registrazione possono ora essere utilizzati per effettuare il login al servizio.
  - Il **bottone Login** viene premuto
    - **Username** e **Password** vengono prelevati dai relativi campi di testo
    - **ControlClient** incapsula i dati in un apposito **Message.LOGIN**
    - Se il messaggio di riscontro del login è positivo (**Message.LOGIN_OK**) il ControlClient chiude **LoginGUI** lanciando una dispose() e apre **MainMenuGUI**.
- Viene premuto il bottone **New** in **MainMenuGUI**
  - La pressione del bottone viene catturata dal suo **actionListener**
  - Vengono richiesti, tramite **JOptionPane**, il **nome del documento** da creare e il suo **numero di sezioni**
  - I dati vengono passati al **ControlClient**, incapsulati in un **Message.CREATE_DOCUMENT**, serializzati e inviati al server via Stream
  - Il nome del documento creato viene quindi aggiunto al **UserData** del client, che viene aggiornato
  - la funzione **update** del MainMenuGUI mostrerà ora il documento fra la lista dei documenti modificabili

- Quando un documento viene selezionato la **OnSelectionListener** della **documentList** fa sì che i dati nella **sectionList** e nella **adminList** vengano aggiornati, mostrando le sezioni del documento selezionato e gli utenti che sono in grado di modificarlo
- Una volta selezionato un documento e una sezione è possibile premere sul bottone **Edit**.

- Viene premuto il bottone **Edit**
  - Alla sua pressione viene inviato al server un **Message.LOCK** e un **Message.EDIT** contenente il nome del documento e il numero della sezione da modificare.
  - Se la sezione richiesta non è in fase di modifica da parte di un altro utente, **MainMenuGUI** lancia una dispose() alla propria finestra e viene aperto al suo posto **EditorGUI**.
  - Nell'interfaccia di editing viene caricato e modificato il testo della sezione
    - Durante questa fase è possibile chattare con gli utenti attivi sullo stesso documento.
- Alla chiusura del documento viene mostrato un JOptionPane offre l'opzione di salvare o meno il documento
  - Viene scelto di salvare il documento
  - Viene inviato un **Message.END_EDIT** al server con la stringa da modificare.
  - **EditorGUI** lancia una **dispose()**, viene chiusa la finestra.
- ControlClient avvia di nuovo **MainMenuGUI**.

LATO SERVER: AVVIO, GESTIONE RICHIESTE E SHUTDOWN

- Quando il server viene avviato la classe **ServerMain** istanzia la classe **ServerGUI**, viene mostrata l'interfaccia grafica del server.
- Viene premuto il bottone **RUN**
  - Una istanza della classe **ControlServer** viene avviata
  - ControlServer scarica dai file **DocumentMemory** e **UserMemory** il **registro documenti** e il **registro utenti** rispettivamente.
    - Se i registri non sono presenti vengono creati.
  - Viene attivato il servizio RMI di registrazione, tramite la classe **RegistrationServer**, esportando uno **stub di ControlServer**
    - Viene ricevuta la **richiesta di registrazione** di un client
      - l'**Username** usato come parametro della funzione register() viene confrontato con quelli presenti nel registro utenti
      - nel caso non ci fosse un match viene creato un **UserData** relativo al nuovo utente e inserito nel registro.
  - Viene creato un socket e un **ThreadPool**.
- **ControlServer** entra in un loop fornendo il **servizio di login**, rimane in attesa di richieste di Login da parte dei client
  - Viene lanciata la **Server.accept()** a seguito di una connessione da parte di un client
  - Il socket del client viene quindi passato al suo **Handler** appena istanziato
  - **Handler** viene eseguito da uno dei thread del pool.
- **Handler** entra in un loop di attesa richieste.
- Viene ricevuta una richiesta **Message.CREATE_DOCUMENT**
  - **Handler** inoltra la richiesta al **ControlServer**, che confronta il nome documento con quelli inseriti all'interno del registro
  - In caso non ci fosse un riscontro, viene inserito un nuovo **Document**, avente tante sezioni quante quelle specificate nel messaggio
  - Assieme a **Document** viene istanziata la classe **DocumentData** con i metadati del documento stesso
  - La **UserData** relativa all'utente che ha effettuato la richiesta viene aggiornata all'interno del **RegistroUtenti**, aggiungendo il nome del documento alla lista dei documenti posseduti

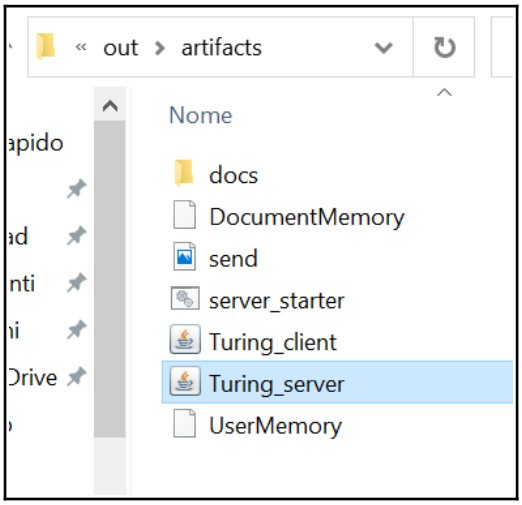
    - Viene inviato un messaggio di conferma al client, così che possa aggiornare la propria **UserData**.
- Viene ricevuto un **Message.LOCK**
    - Il ControlServer verifica che la lock relativa alla sezione richiesta non sia già stata acquisita
    - Viene acquisita la lock dall'Handler che ha portato avanti la richiesta
    - Viene restituito un **Message.LOCK_OK** al client.
- Viene ricevuto un **Message.EDIT**
    - **Handler** verifica di aver acquisito la lock sulla sezione richiesta
    - Viene richiesta dal registro documenti la stringa contenente il **testo del file** specificato
    - Il testo viene passato al client in un messaggio di risposta.
- Viene ricevuto un **Message.END_EDIT**
    - **ControlServer** sostituisce il **testo** all'interno del **Document** corrispondente nel registro
    - Viene scritto un **file** all'interno della cartella docs/'nomeDocumento' contenente il testo appena modificato
    - viene effettuata l'**unlock** della sezione.
- Viene premuto il tasto **STOP** nell'interfaccia del server
    - Il **ControlServer** riceve una interrupt
    - I **registri** vengono salvati in dei file
    - Viene lanciata una **Unexport** verso tutti gli oggetti remoti esportati nel **RegistrationServer**
    - Viene lanciata una **Unbind** verso il registro
    - I thread in esecuzione nel **ThreadPool** ricevono una interrupt
    - il server smette di accettare richieste di login e registrazione, termina la sua esecuzione.

Manuale Utente

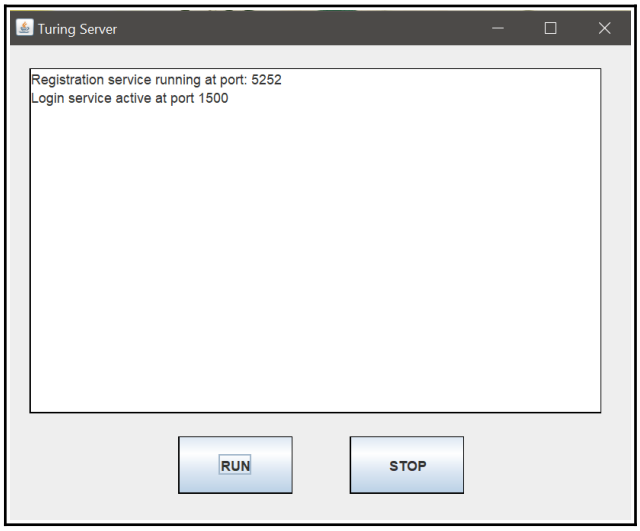
In questa sezione è presente un insieme sintetico di istruzioni per avviare e utilizzare il software

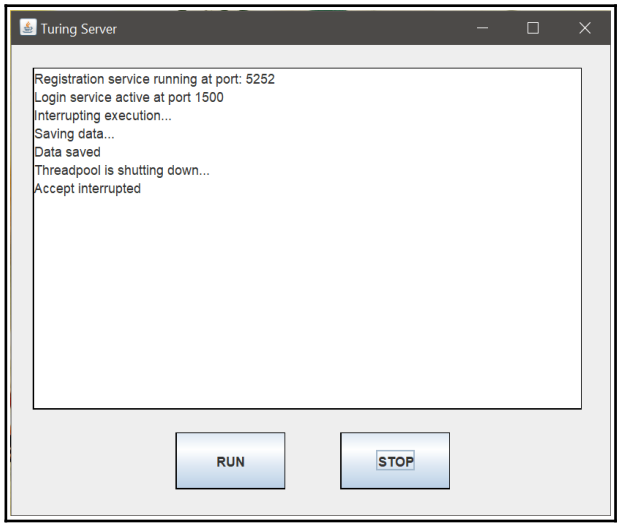
LATO SERVER

Per avviare il server aprire il file Turing_server.jar nella cartella artifacts.



Premere il bottone RUN nell'interfaccia e aspettare che vengano visualizzati i messaggi di avvenuto avvio.

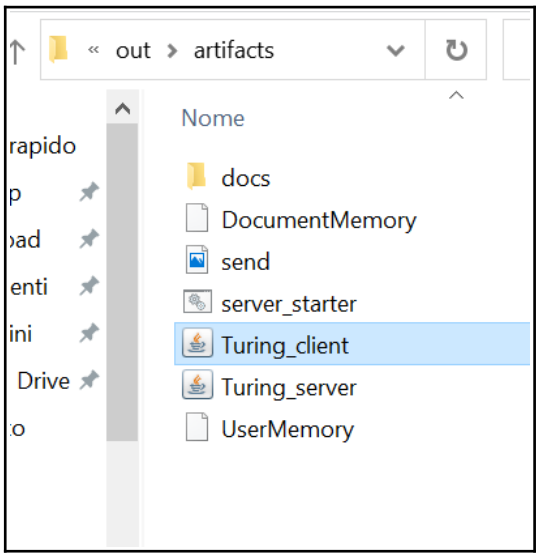




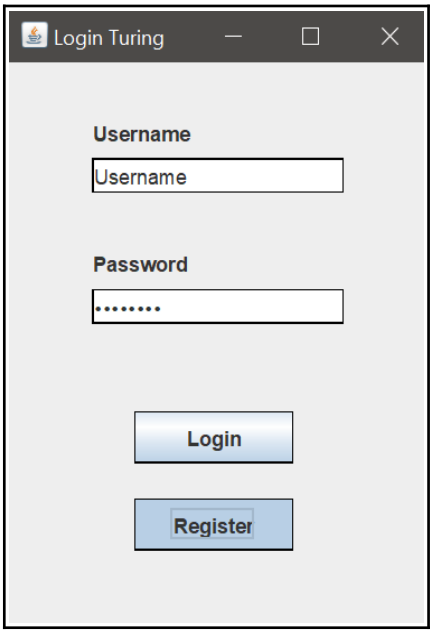
Per interrompere l'esecuzione e terminare il programma salvando i dati premere sul bottone STOP e successivamente chiudere la finestra.

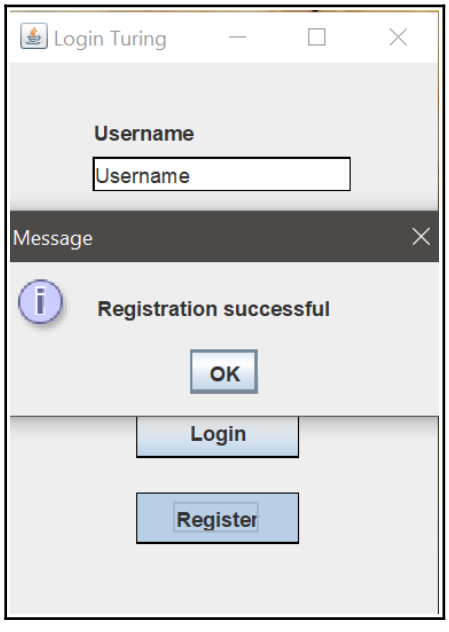
LATO CLIENT

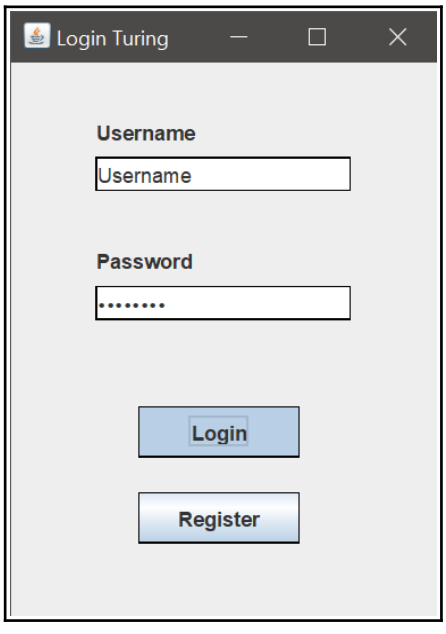
Per avviare il client aprire il file Turing_client nella cartella artifacts.



Verrà visualizzata la schermata di Login, per effettuare la registrazione digitare il proprio nome utente e la propria password e premere il bottone Register.



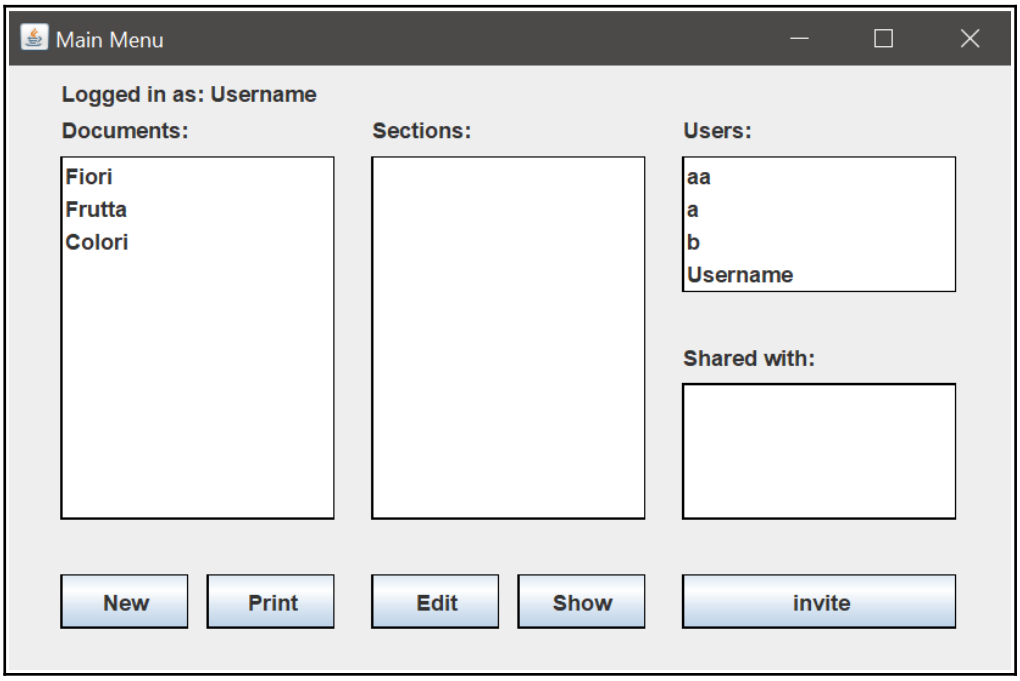




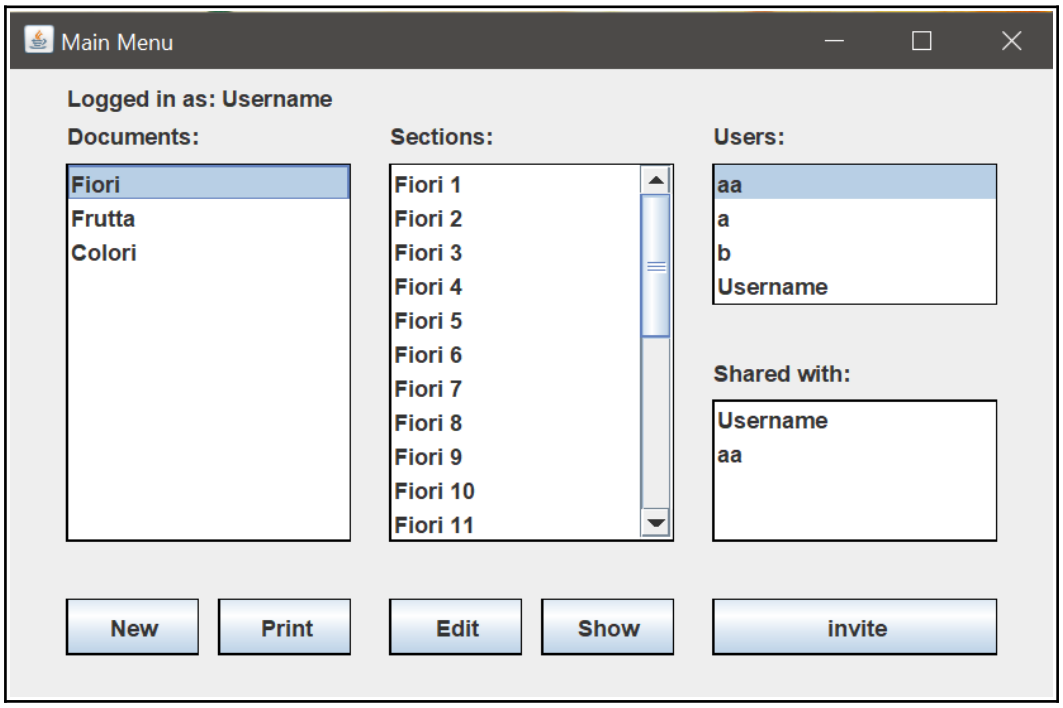
Da questo momento in poi potrà essere effettuato il login con le credenziali inserite in fase di registrazione.

Una volta effettuato il login verrà mostrata la schermata principale del programma.

- Premendo il tasto New è possibile creare un nuovo documento.

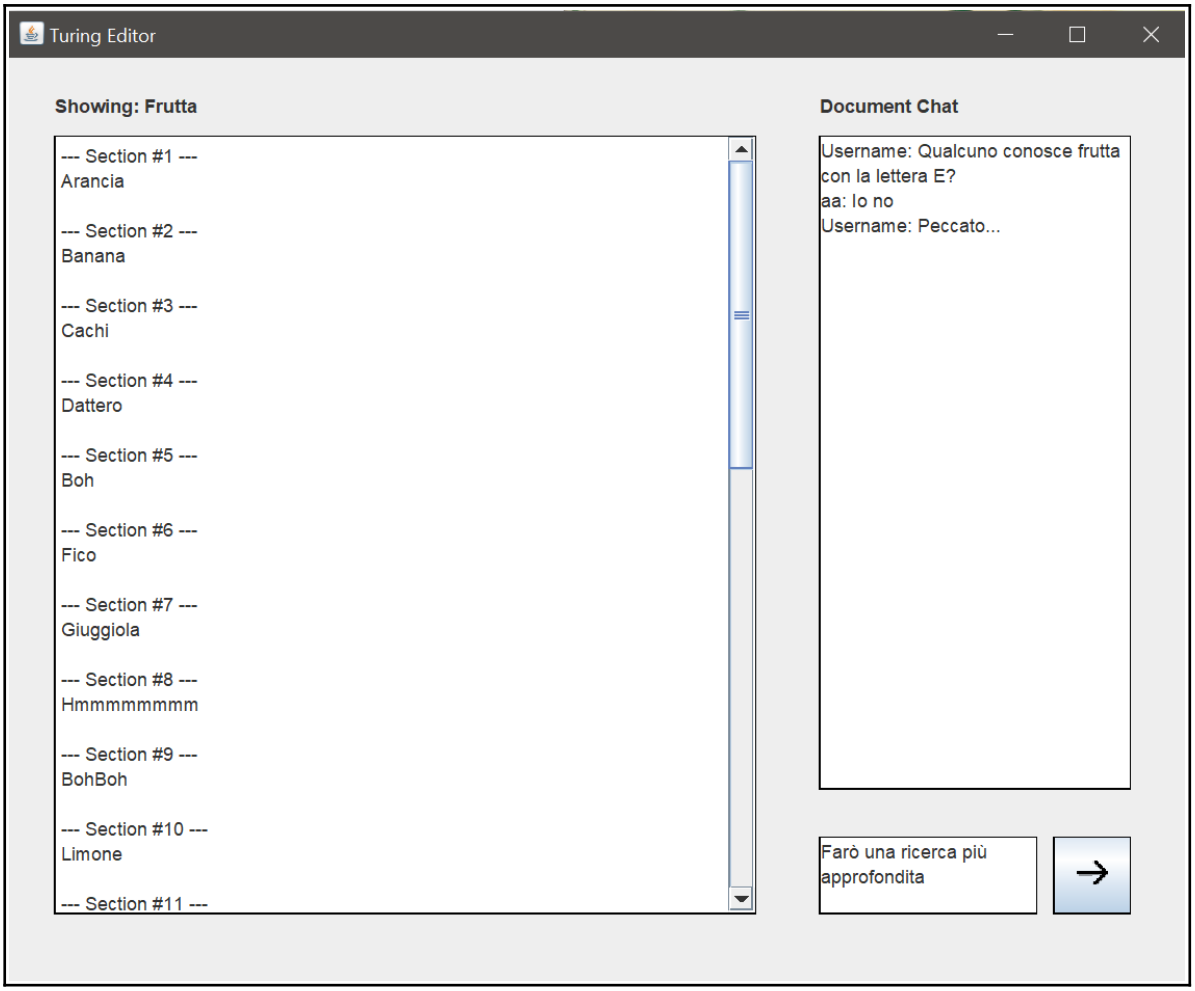


Quando un documento viene selezionato è possibile visualizzare la lista delle sezioni che lo compongono e la lista degli utenti che hanno il permesso di modificarle.



- Il tasto **Print** permette di visualizzare interamente il contenuto del documento selezionato.
- Il tasto **Edit** permette di modificare la sezione selezionata.
- Il tasto **show** permette di mostrare la sezione richiesta, anche se questa è in fase di editing.
- Il tasto **Invite** permette di invitare l'utente selezionato dalla lista Users al documento selezionato nella lista Document.

Premendo uno dei tasti Edit/Show/Print è possibile visualizzare la schermata di editing



All'interno di questa schermata è possibile chattare con tutti gli utenti che, allo stesso momento, stanno modificando o visualizzando lo stesso documento.

Chiudendo l'editor previa apertura tramite tasto Edit è possibile scegliere se salvare o meno il documento appena modificato. Premere Yes per non perdere le modifiche apportate.