



UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

LABORATORIO DI RETI, CORSO B  
A.A.2018/2019

**TURING:**  
**disTribUted collaboRative edItiNG**

*Autore:*  
NICCOLÒ CARDELLI

*Matricola:*  
534015

# Contents

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Architettura del sistema</b>	<b>2</b>
2.1	Comunicazioni . . . . .	2
2.2	Server . . . . .	3
2.3	Descrizione delle classi del Server . . . . .	4
2.4	Client . . . . .	6
2.5	Descrizione delle classi del Client . . . . .	6
<b>3</b>	<b>Manuale d'uso</b>	<b>7</b>
3.1	Compilazione, esecuzione e chiusura del Server . . . . .	7
3.2	Compilazione, esecuzione e chiusura del Client . . . . .	8
3.3	Utilizzo di Turing CLI . . . . .	8
<b>4</b>	<b>Info sul progetto</b>	<b>18</b>

# 1 Introduzione

L'applicazione *Turing* è uno strumento di editing collaborativo di documenti di testo. L'utente interagisce con Turing mediante un'Interfaccia a Linea di Comando (**CLI**), con la quale esso può inviare richieste e ricevere risposte rispettivamente al e dal Server.

Un utente, una volta registrato al servizio, può creare, editare e condividere documenti con altri utenti registrati (detti *collaboratori*).

Ogni documento, alla sua creazione, viene diviso in un numero *fissato* di sezioni. Il creatore di tale documento può mandare un invito a collaborare al documento ad un qualsiasi utente registrato, *purchè quest'ultimo non possieda un documento con lo stesso nome*. Tale invito verrà mostrato al nuovo collaboratore quando effettuerà il prossimo login (se era offline) oppure alla sua prossima richiesta verso il Server (se era online).

I collaboratori possono modificare in contemporanea un documento purchè valgano le seguenti condizioni:

- ogni collaboratore si occupi di al massimo un documento alla volta
- ogni collaboratore si occupi di al massimo una sezione alla volta (dello stesso documento)
- una sezione non può essere occupata da più di un utente alla volta

Non appena un creatore o un collaboratore inizia ad editare un documento, viene avviata una chat che permette lo scambio di messaggi tra tutti coloro che stanno editando quel documento. Quando si termina di editare un documento la chat viene chiusa per quell'utente. I messaggi vengono ricevuti solo su richiesta dell'utente tramite un comando apposito.

Per iniziare ad utilizzare Turing e vedere le funzionalità offerte passare direttamente alla sezione Manuale d'Uso.

## 2 Architettura del sistema

In questo capitolo discuterò nel dettaglio quali sono le entità, i protocolli di comunicazione, le strutture dati e le classi utilizzate per realizzare Turing, indicando (quando possibile) i motivi delle varie scelte progettuali.

### 2.1 Comunicazioni

#### Protocolli

A seconda della richiesta da parte del Client verso il Server, la modalità di comunicazione sarà diversa:

- Per la richiesta di registrazione si utilizza RMI (RPC per Java), ovvero il Client comunica con il Server invocando un procedure di un oggetto remoto chiamato *Registratore*.
- Per la richiesta di invio e di ricezione di messaggi su una chat, il Client utilizza un Socket UDP Multicast, il quale permette l'invio e la ricezione di pacchetti DatagramPacket a e da un gruppo di host registrati su un determinato indirizzo. L'indirizzo viene fornito al Client (comunicando via TCP Socket), prima che esso possa effettuare queste tipo di richieste.

- Per tutte le altre operazioni, le comunicazioni sono implementate tramite TCP Sockets. Per questo tipo operazioni il protocollo utilizzato è a domanda-risposta, ovvero non è possibile inoltrare nuove richieste se non si è ricevuta prima la risposta alla richiesta precedente.

## Indirizzi UDP

La creazione di una chat corrisponde all'atto di assegnazione *statica* di un indirizzo Ip ad un dato documento. Questi indirizzi vengono assegnati in modo incrementale a partire dall'indirizzo 239.0.0.0. L'avvio di una chat corrisponde invece al momento nel quale un Client inizia ad editare un documento: tale Client si registrerà nel gruppo di multicast per quell'indirizzo e si deregistrerà una volta finito l'editing.

## 2.2 Server

Il Server è costituito da un solo thread, il quale gestisce le richieste di diversi client eseguendo il multiplexing dei vari canali NIO tramite un Selector: grazie a quest'ultimo i Sockets del Server possono essere configurati come *non-bloccanti*, poichè le operazioni di lettura e scrittura da e verso i buffer dei Socket avvengono solamente in seguito al corrispondente evento segnalato dal Selector.

### Flusso di esecuzione

Non appena viene avviato, il Server effettua varie operazioni di setup:

- Se sono stati passati argomenti al programma Server, questi vengono controllati e utilizzati (vedere sez. 3.1).
- Esegue l'esportazione dell'oggetto remoto Registratore, il quale verrà utilizzato dai Clients per effettuare la registrazione.
- Configura il Socket TCP che si occupa di accettare i Clients in modalità *non-bloccante*.
- Se la cartella `data_server` esiste, questa viene svuotata, altrimenti viene creata (vedere sez. 2.4). Essa conterrà tutti i documenti degli utenti registrati a Turing.
- Configura il Selector registrando il Socket TCP come socket di accettazione.
- Configura il Gestore Sessioni.
- Configura il Gestore Documenti, indicando l'indirizzo iniziale utilizzato per le chat degli utenti (vedere sez. 2.1).

Una volta terminato con successo il setup, il Server entra in un ciclo infinito (e vi rimane fino a che non viene chiuso il processo): al suo interno il Selector controlla periodicamente se tra i Socket registrati ce n'è almeno uno per cui è avvenuto uno tra questi eventi:

1. **Accettazione:** un Client chiede di connettersi al Server.
2. **Lettura:** un Client vuole inviare una richiesta di operazione oppure di invio di un file.
3. **Scrittura:** un Client è pronto a ricevere una risposta ad una richiesta di operazione oppure all'invio di un file.

Nei rispettivi casi il Server si comporta in questo modo:

1. **Accettazione:** se il Socket è valido viene accettato e registrato nel Selector.
2. **Lettura:** se il Socket è valido si controlla (grazie al suo *Allegato*) se c'è un file in attesa di essere ricevuto e in tal caso si riceve da quel Socket. Altrimenti si riceve una richiesta di operazione dal Socket e si esegue: in quest'ultimo caso, a seconda dell'operazione, avverranno le modifiche de:
  - l'Allegato del Socket (vedere sez. 2.3, "Allegato").
  - lo stato del Server (vedere sez. 2.3, "Gestori dei Dati").
3. **Scrittura:** se il Socket è valido ed ha un Allegato, si aggiungono alla risposta della richiesta di operazioni le eventuali notifiche, rivolte all'utente del corrispondente Socket, e si invia tale risposta. Infine, sempre controllando l'Allegato, si inviano uno o più eventuali files all'utente.

## 2.3 Descrizione delle classi del Server

Il Server utilizza diverse classi con varie funzioni differenti:

### Dati

Sono i dati primitivi.

- **Allegato:** Contiene le varie informazioni riguardanti una sessione di un utente, dove per *sessione* si intende il periodo di tempo che intercorre tra l'accettazione di un Socket di un Client e la sua disconnessione. Al suo interno contiene:
  - l'utente collegato
  - il messaggio di risposta ad una richiesta
  - il path di un file da ricevere
  - una lista di path di eventuali files da inviare
- **Documento:** Rappresenta un documento. Al suo interno contiene:
  - il nome del documento
  - il numero di sezioni del documento
  - il path del documento
  - l'utente creatore del documento
  - l'ip della chat del documento
- **Utente:** Rappresenta un utente. Al suo interno contiene:
  - il nome dell'utente
  - la password dell'utente

### Gestori dei Dati

Forniscono dei servizi necessari al Server per effettuare le operazioni e mantengono le strutture principali che costituiscono lo stato del Server.

- **GestoreSessioni:** si occupa di tenere traccia degli utenti loggati e delle notifiche verso gli utenti. Fornisce al Server i servizi per effettuare login e logout, aggiungere, togliere notifiche. Al suo interno contiene:
  - l'insieme degli utenti loggati (implementata con un HashSet)
  - l'insieme delle notifiche da inviare per ogni utente (implementata con una HashMap di ArrayList)
- **GestoreDocumenti:** si occupa di tenere traccia dei documenti di ogni utente, dei collaboratori e degli utenti attivi (stanno editando una sezione) per ogni documento. Fornisce al Server l'accesso a tutti i metadati dei Documenti (il numero di sezioni, il creatore, i collaboratori, gli utenti attivi, il path) e fornisce i metodi per creare e condividere i documenti. Al suo interno contiene:
  - l'insieme dei documenti per ogni utente (implementata con una HashMap di HashMap)
  - l'insieme dei collaboratori per ogni documento (implementata con una HashMap di HashSet)
  - l'insieme degli utenti attivi per ogni sezione per ogni documento (implementata come una HashMap di HashMap)

## Comunicazioni

Sono le classi utilizzate dal Server e dal Client per le comunicazioni via Socket TCP.

- **Messaggio:** Rappresenta un messaggio. Al suo interno contiene un buffer di lunghezza variabile con il contenuto del messaggio.
- **Connessione:** Fornisce i metodi per inviare, ricevere messaggi e files via Socket TCP. I messaggi vengono inviati sotto forma di ByteBuffer, mentre i files vengono trasmessi utilizzando la *copy-0 transfer*: invece che copiare il contenuto di un file in un buffer intermedio è possibile collegare (utilizzando un FileInputStream/FileOutputStream) il canale del Socket direttamente con il file e leggere/scrivere da lì. Questo migliora notevolmente il tempo impiegato per effettuare le operazioni.

## Registrazione RMI

Sono le classi che si occupano della registrazione degli utenti a Turing.

- IRegistratore: è l'interfaccia che contiene i metodi accessibili da remoto necessari per effettuare la registrazione.
- Registratore: è la classe del Server che implementa il servizio remoto di registrazione. Fornisce ai Client i metodi per registrarsi. Al suo interno contiene:
  - L'insieme degli utenti registrati (implementato con una ConcurrentHashMap). La struttura utilizzata è thread-safe, poichè le invocazioni a metodi remoti di più Client diversi possono essere eseguite concorrentemente.

## Conversione Ip

Classe di utility per conversione di indirizzi ip.

- IpConverter: fornisce i metodi per convertire un indirizzo ip in un intero e viceversa e per iterare su di esso.

## 2.4 Client

Il Client è costituito da un solo thread e, differentemente dal Server, utilizza un Socket *bloccante* poichè esso non può proseguire nella operazioni fino a che non riceve una risposta dal Server (protocollo domanda-risposta).

### Flusso di esecuzione

Non appena viene avviato il Client effettua varie operazioni di setup:

- Se sono stati passati argomenti al programma Client, questi vengono controllati e utilizzati.
- Ricerca l'oggetto remoto Registratore, utilizzato per effettuare la registrazione, e esce in caso di fallimento.
- Configura il Socket TCP che comunicherà con il Server in modalità bloccante.
- Se all'interno della cartella *Turing\_Client* esiste la cartella `data_client_X`, allora crea la cartella `data_client_Y` dove  $Y = X+1$ .

Questa scelta progettuale è data dal fatto che gli utenti possono eseguire più di un Client alla volta sullo stesso host: questo comporta che all'avvio un Client esso non possa sapere se la cartella `data_client` è ancora utilizzata da un altro Client e non può ripulirla. Quindi è necessario aggiungere sempre nuove cartelle numerate diversamente.

- Prova a connettersi al Server ed esce in caso di fallimento.
- Configura lo Stato Client.

Da qui in poi il Client si comporta come se fosse una macchina a stati finiti: nello specifico, quella rappresentata nella figura 2. Il Client entra in un ciclo infinito (e vi rimane fino a chiuso il processo o l'utente abbia richiesto un'operazione di QUIT): al suo interno il processo rimane in attesa di input da tastiera dell'utente. Se l'utente ha inserito una stringa nel formato corretto, allora questa viene divisa in comandi e argomenti.

In base allo Stato Client e al comando inserito, viene eseguita l'operazione specificata: a seconda dell'esito di quest'ultima si determina il nuovo Stato della macchina. Le operazioni consentite in un dato Stato sono solo quelle specificate in figura e descritte nella sezione 3.3: in caso di operazione non consentita questa non verrà eseguita del tutto e il Client rimarrà nello stesso Stato.

## 2.5 Descrizione delle classi del Client

Il Client utilizza diverse classi con due principali funzionalità:

## Stato Client

- **StatoClient**: rappresenta lo Stato del client. Al suo interno contiene:
  - il Registratore remoto
  - il Socket TCP con cui il Client è collegato al Server
  - il Socket UDP Multicast
  - il path della directory data\_client
  - lo Stato dell'utente
  - l'username dell'utente loggato
  - il nome del documento editato
  - il numero della sezione del documento editato
  - l'ip della chat del documento editato

## Comunicazioni

Sono le classi utilizzate dal Server e dal Client per le comunicazioni via Socket TCP.

- **Messaggio**: stessa classe usata dal Server (vedere sez. 2.3, "Messaggio").
- **Connessione**: stessa classe usata dal Server (vedere sez. 2.3, "Connessione").

## 3 Manuale d'uso

In questo capitolo verranno indicate le operazioni necessarie per poter iniziare ad usare Turing e verranno descritte tutte le funzionalità offerte, mostrando degli esempi (non tutti quelli possibili) di comandi corretti ed errati.

**Prima di effettuare qualsiasi operazione** assicurarsi che la struttura della cartella *Turing* sia questa:

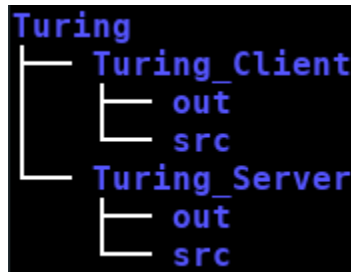


Figure 1: Struttura cartelle iniziale

### 3.1 Compilazione, esecuzione e chiusura del Server

Per la **compilazione** del Server spostarsi sulla cartella *Turing\_Server* ed eseguire il comando:

```
javac -d out -sourcepath src src/Server.java
```

Nella stessa cartella, per **eseguire** il Server utilizzare il comando:



```
java -classpath out Server [-s ser_port] [-r reg_port]
```

Dove:

- *ser\_port* è il numero di porta in cui il Server rimane in ascolto di richieste TCP (se **omesso** verrà utilizzato il numero di porta di default 9999).
- *reg\_port* è il numero di porta in cui il Server rimane in ascolto di richieste RMI, ovvero per accedere al servizio di registrazione a Turing (se omesso verrà utilizzato il numeri di porta di default 6000).

Riguardo la **chiusura** del Server, non è prevista una chiusura "peaceful", ma occorre inviare un segnale di chiusura (SIGINT, SIGQUIT, SIGKILL) al processo (CTRL+C su Linux).

### 3.2 Compilazione, esecuzione e chiusura del Client

Per la **compilazione** del Client, usando il terminale, spostarsi sulla cartella *Turing\_Client* ed eseguire il comando:

```
javac -d out -sourcepath src src/Client.java
```

Nella stessa cartella, per **eseguire** il Client (Turing CLI) utilizzare il comando:

```
java -classpath out Client [-i ip_addr] [-s ser_port] [-r reg_port]
```

Dove:

- *ip\_addr* è l'indirizzo ip del Server (se **omesso** verrà utilizzato l'indirizzo *Localhost*).
- *ser\_port* è il numero di porta in cui il Server rimane in ascolto di richieste (se **omesso** verrà utilizzato il numero di porta di default 9999).
- *reg\_port* è il numero di porta in cui è in ascolto il registro RMI (in esecuzione sull'host Server), il quale fornisce il servizio di registrazione di Turing.

Riguardo la **chiusura** del Client è prevista una chiusura "peaceful" e una "non peaceful". Per la chiusura "peaceful" occorre utilizzare il comando *quit* (vedere sezione EEEE). Per la chiusura "non peaceful" occorre inviare un segnale (SIGINT, SIGQUIT, SIGKILL) al processo (CTRL+C su Linux).

**ATTENZIONE:** se un Client è stato chiuso con quest'ultimo metodo, per si vuole riavviare l'intera applicazione **da zero** occorre anche eliminare TUTTE le cartelle *data\_client\_X* create durante l'ultima esecuzione del Client. In caso di riavvio del Client *senza questo passaggio* verrà creata una nuova cartella *data\_client\_Y* dove  $Y=X+1$ , lasciando la cartella *data\_client\_X* e il suo contenuto inutilizzabili.

### 3.3 Utilizzo di Turing CLI

Questo schema mostra il funzionamento generale e desiderato di Turing CLI:

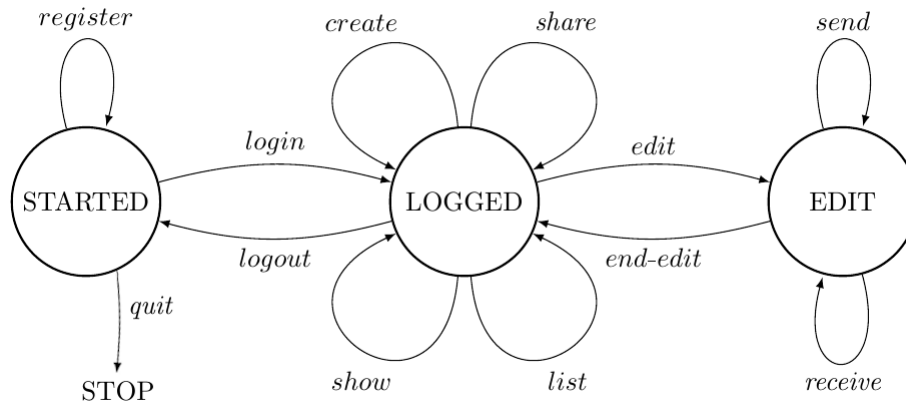


Figure 2: Automa del comportamento del Client Turing

STARTED, LOGGED e EDIT rappresentano i tre stati nel quale Turing si può trovare (STOP indica che Turing CLI non è più in esecuzione). *register*, *login*, *logout*, *quit*, *create*, *show*, *list*, *edit*, *end-edit*, *send* e *receive* sono tutti i possibili comandi.

Se in uno qualsiasi dei tre stati viene eseguito un comando non previsto oppure il comando ha esito negativo, lo stato di Turing CLI rimane lo stesso. Non appena viene avviata, la Turing CLI si trova nello stato **STARTED**. Di seguito sono elencati tutti i comandi che Turing CLI può eseguire.

### Registrazione (*register*)

Un utente può registrarsi a Turing indicando un username e una password, i quali potranno essere utilizzati successivamente per effettuare il login. L'username è alfanumerico, di lunghezza compresa tra 3 e 20 ed *univoco* per ogni utente. La password è alfanumerica e di lunghezza compresa tra 6 e 20.

### Esempio uso corretto

- ```
$ turing register pippo 123456
Creato directory /home/nick/Scrivania/Turing/Turing_Client/data
_client_1/pippo
Registrazione eseguita con successo.
```

### Esempio uso errato

- Caratteri non-alfanumerici per l'username
 

```
$ turing register pippo_ 123456
L'username deve contenere solo caratteri alfanumerici.
```
- Lunghezza username  $\geq 20$  o lunghezza username  $\leq 3$

```
$ turing register pi 123456
```

```
L'username deve contenere tra i 3 e i 20 caratteri.
```

- Caratteri non-alfanumerici per la password

```
$ turing register pippo 12345_
```

```
La password deve contenere solo caratteri alfanumerici.
```

- Lunghezza password  $\geq 20$  o lunghezza password  $\leq 6$

```
$ turing register pippo 12345
```

```
La password deve contenere tra i 6 e i 20 caratteri.
```

- Utente già registrato

```
$ turing register pippo 123456
```

```
Creata directory /home/nick/Scrivania/Turing/Turing_Client/data  
_client_1/pippo.
```

```
Registrazione eseguita con successo.
```

```
$ turing register pippo 123456
```

```
Utente già registrato, scegli un altro username.
```

### Login (*login*)

Un utente (precedentemente registrato) può effettuare l'accesso a Turing inserendo correttamente l'username e la password indicati in fase di registrazione. Un utente non può effettuare il login se lo stesso utente ha già effettuato l'accesso a Turing utilizzando una diversa CLI.

#### Esempio uso corretto

- ```
$ turing login pippo 123456
```

```
Login eseguito con successo.
```

#### Esempio uso errato

- Utente non registrato.

```
$ turing login pluto 123456
```

```
Utente inesistente. Prima devi registrarti.
```

- Password errata.

```
$ turing login pippo 123457
```

```
Password errata.
```

- L'utente si è già loggato su un altro host/CLI.

```
$ turing login pippo 123456
```

```
Utente pippo già loggato su un altro host.
```

### Logout (*logout*)

Un utente (precedentemente loggato e registrato) può effettuare il logout da Turing.

### Esempio uso corretto

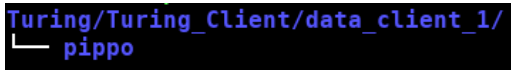
- `$ turing logout`  
Logout eseguito con successo.

### Creazione di un documento (*create*)

Un utente (precedentemente registrato e loggato) può creare un nuovo documento composto da un certo numero di sezioni. Ogni sezione viene memorizzata in un file e quindi il documento può essere considerato come un insieme di files. Il nome del documento è alfanumerico e univoco (*per quell'utente*): non è possibile creare un documento con lo stesso nome di uno creato da o condiviso da terzi con l'utente stesso.

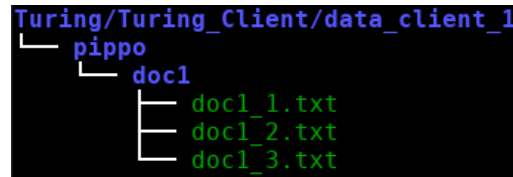
### Esempio uso corretto

- `[pippo]$ turing create doc1 3`  
Creata directory /home/nick/Scrivania/Turing/Turing\_Client/data\_client\_1/pippo/doc1  
Documento creato con successo.



```
Turing/Turing_Client/data_client_1/  
└─ pippo
```

Figure 3: Situazione files prima di *create*



```
Turing/Turing_Client/data_client_1/  
└─ pippo  
   └─ doc1  
      ├── doc1_1.txt  
      ├── doc1_2.txt  
      └── doc1_3.txt
```

Figure 4: Situazione files dopo *create*

### Esempio uso errato

- Caratteri non-alfanumerici per il nome documento

```
[pippo]$ turing create doc_1 3  
Il nome documento deve contenere solo caratteri alfanumerici.
```

- Lunghezza nome documento  $\geq 20$  o lunghezza username  $\leq 3$

```
[pippo]$ turing create d 3  
Il nome documento deve contenere tra i 3 e i 20 caratteri.
```

- Numero di sezioni  $\geq 16$  o numero di sezioni  $\leq 0$

```
[pippo]$ turing create doc1 100  
Il numero di sezioni deve essere un intero compreso tra 1 e 15.
```

- Documento già presente

```
[pippo]$ turing create doc1 3  
Creata directory /home/nick/Scrivania/Turing/Turing_Client/data
```

```

_client_1/pippo/doc1
Documento creato con successo.
[pippo]$ turing create doc1 3
Documento già presente.

```

### Condivisione di un documento (*share*)

Un utente (precedentemente registrato e loggato) può condividere uno dei documenti *che ha creato* con un qualsiasi altro utente registrato, purchè quest'ultimo non possieda un documento con lo stesso nome e non sia già un collaboratore.

#### Esempio uso corretto

- [pippo]\$ turing share doc1 topolino  
Documento condiviso con successo.  
Verrà inviato un invito a topolino.

Quando l'utente invitato effettuerà il login o una qualsiasi altra operazione *mentre* è loggato egli vedrà questo:

- \$ turing login topolino 123456  
[NUOVA NOTIFICA!]  
pippo ti ha invitato a collaborare al documento doc1 composto da 3 sezioni.  
Ora puoi accedere e modificare il documento.  
Creata directory /home/nick/Scrivania/Turing/Turing\_Client/data\_client\_2/topolino/doc1  
Login eseguito con successo.

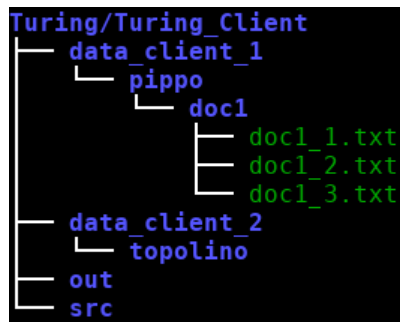


Figure 5: Situazione files prima di *share*

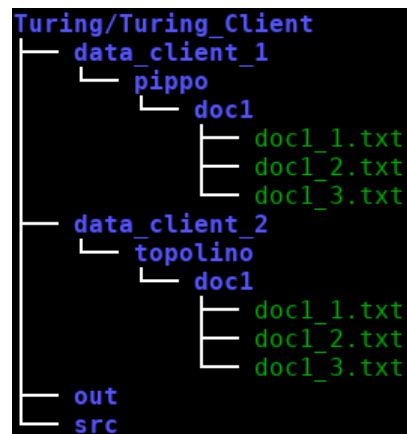


Figure 6: Situazione files dopo *share*

### Esempio uso errato

- L'utente non possiede il documento che vuole condividere.

```
[pippo]$ turing share doc2 topolino
Non possiedi alcun documento chiamato doc2.
```

- Utente invitato non registrato.

```
[pippo]$ turing share doc1 topolino
L'utente invitato non risulta registrato.
```

- L'utente invitato è lo stesso che il mittente stesso.

```
[pippo]$ turing share doc1 pippo
Non puoi autoinvitarti.
```

- L'utente invitato è già un collaboratore oppure possiede un documento con lo stesso nome di quello che si vuole condividere.

```
[pippo]$ turing share doc1 topolino
L'utente invitato possiede un documento con lo stesso nome del tuo.
```

- L'utente è solo un collaboratore e non il creatore del documento.

```
[topolino]$ turing share doc1 minnie
Non hai i permessi per condividere il documento.
Solo il creatore puo' dividerlo.
```

### Download di una sezione di un documento (*show*)

Un utente (precedentemente registrato e loggato) può scaricare una sezione di un documento da lui ha creato o di cui è collaboratore, purchè il numero di sezione sia corretto. La sezione che verrà scaricata non comprende le modifiche attualmente in atto (effettuate da un altro collaboratore) su quella sezione. Utilizzando questo comando viene anche visualizzato il nome dell'utente che, eventualmente, sta editando il documento.

### Esempio uso corretto

- [pippo]\$ turing show doc1 2  
Sezione 2 del documento doc1 scaricata con successo.

oppure se un altro utente sta editando quella sezione

- [pippo]\$ turing show doc1 2  
L'utente topolino sta editando la sezione.  
Sezione 2 del documento doc1 scaricata con successo.

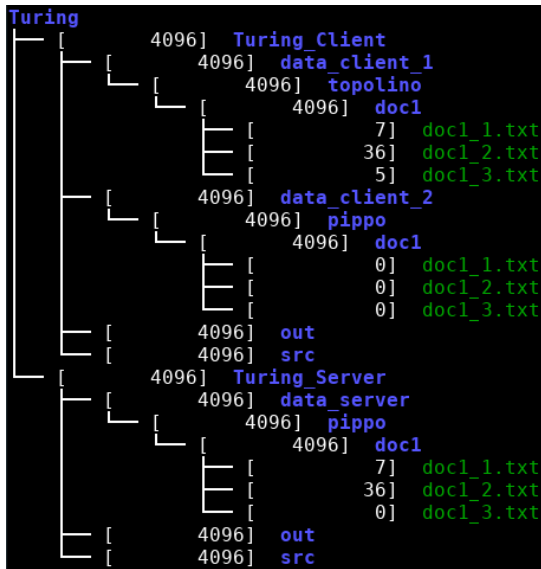


Figure 7: Situazione files prima di *show*

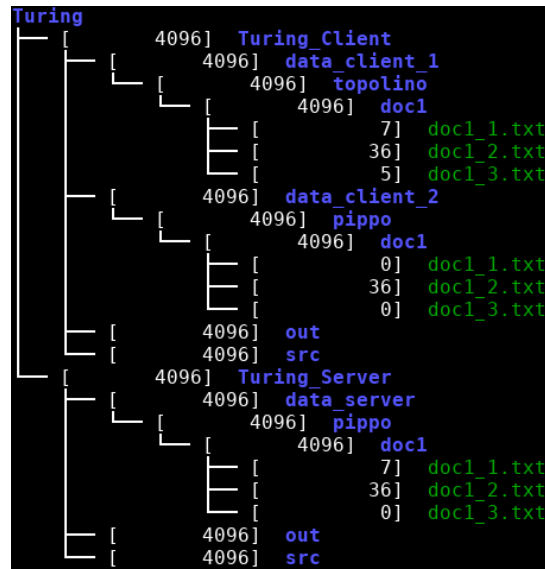


Figure 8: Situazione files dopo *show*

### Esempio uso errato

- L'utente non possiede il documento che vuole scaricare.

```
[pippo]$ turing show doc2 2
Non possiedi alcun documento chiamato doc2.
```

- Il numero della sezione richiesta è maggiore del numero di sezioni di quel documento

```
[pippo]$ turing show doc1 13
Numero di sezione errato.
```

### Download di un intero documento (*show*)

Un utente (precedentemente registrato e loggato) può scaricare tutte le sezioni di un documento da lui creato o di cui è collaboratore. Le sezioni scaricate non comprendono le modifiche attualmente in atto (effettuate dai collaboratori) sulle sezioni. Utilizzando questo comando viene anche visualizzata la lista degli utenti che, in quel momento, stanno editando una qualsiasi delle sezioni del documento.

### Esempio uso corretto

- [pippo]\$ turing show doc1  
Numero di sezioni da scaricare: 3.  
Documento doc1 scaricato con successo.

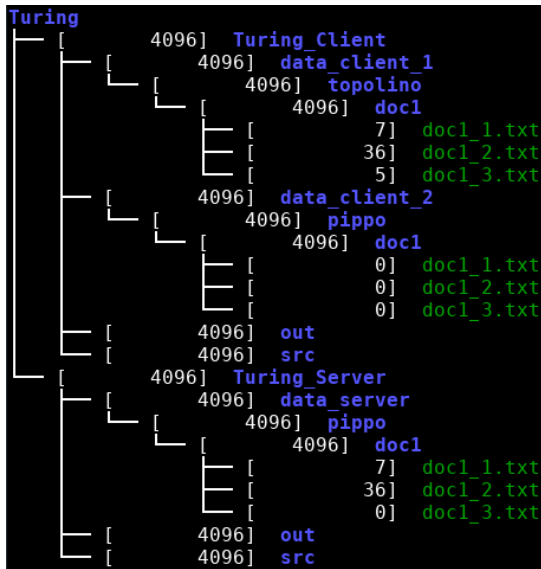


Figure 9: Situazione files prima di *show*

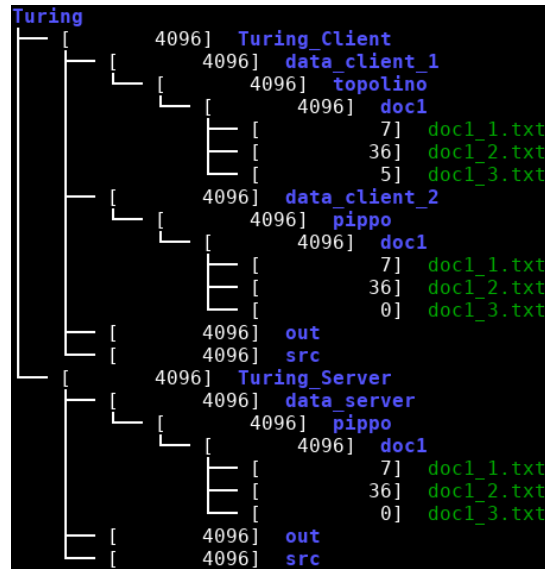


Figure 10: Situazione files dopo *show*

### Esempio uso errato

- L'utente non possiede il documento che vuole scaricare.

```
[pippo]$ turing show doc2
Non possiedi alcun documento chiamato doc2.
```

- Il numero della sezione richiesta è maggiore del numero di sezioni di quel documento

```
[pippo]$ turing show doc1 13
Numero di sezione errato.
```

### Lista dei documenti (*list*)

Un utente (precedentemente registrato e loggato) può richiedere la lista dei documenti da lui creati o di cui è collaboratore. Per ogni documento viene mostrato il nome dello stesso, il nome del creatore e i nomi dei collaboratori.

### Esempio uso corretto

- [topolino]\$ turing list  
Lista ricevuta con successo.  
doc2:  
Creatore: topolino  
doc1:



```
Creatore: pippo  
Collaboratori: topolino
```

### Inizio editing di un documento (*edit*)

Un utente (precedentemente registrato e loggato) può richiedere iniziare a modificare una sezione di un documento da lui creato o di cui è collaboratore, purchè il numero di sezione indicato sia corretto e la sezione in questione non sia già occupata da un altro utente.

#### Esempio uso corretto

- [pippo]\$ turing **edit** doc1 2  
Sezione 2 del documento doc1 scaricata con successo.  
Inizio editing sezione 2 del documento doc1.

A questo punto occorre utilizzare un editor di testo (esterno a Turing CLI) per effettuare le modifiche al documento editato (il path del documento di testo è quello comunicato alla creazione del documento o alla ricezione dell'invito a collaborare).

#### Esempio uso errato

- L'utente non possiede il documento che vuole scaricare.

```
[pippo]$ turing edit doc2  
Non possiedi alcun documento chiamato doc2.
```

- Il numero della sezione richiesta è maggiore del numero di sezioni di quel documento

```
[pippo]$ turing edit doc1 13  
Numero di sezione errato.
```

- La sezione richiesta è già occupata da un altro utente.

```
[pippo]$ turing edit doc1 2  
Impossibile modificare sezione 2 del documento doc1.  
L'utente topolino la sta modificando.
```

### Fine editing di un documento (*end-edit*)

Un utente (precedentemente registrato, loggato e che abbia iniziato l'editing) può terminare di modificare una sezione di un documento che stava editando e inviare la copia aggiornata al Server.

#### Esempio uso corretto

- [pippo]\$ turing **end-edit** doc1 2  
Verra' inviato il file /home/nick/Scrivania/Turing/Turing\_Client/data\_client\_2/pippo/doc1/doc1\_2.txt.  
Fine editing sezione 2 del documento doc1.

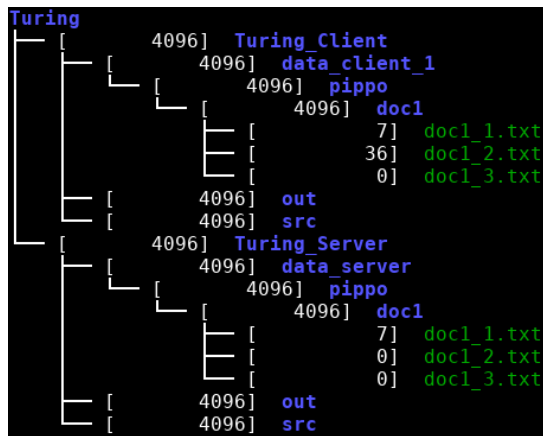


Figure 11: Situazione files prima di *end-edit*

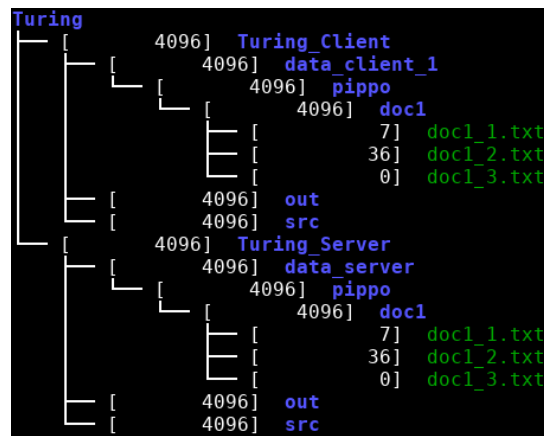


Figure 12: Situazione files dopo *end-edit*

### Esempio uso errato

- Il nome del documento o il numero di sezione non corrispondono con quelli del documento e sezione editata.

```
[pippo]$ turing end-edit doc 1
Stai modificando la sezione 2 del documento doc1.
```

### Invio di un messaggio sulla chat (*send*)

Un utente (precedentemente registrato, loggato e che abbia iniziato l'editing) può inviare un messaggio sulla chat di quel documento. Tutti gli utenti che abbiano iniziato l'editing (di una qualsiasi delle sezioni) prima dell'invio riceveranno il messaggio.

### Esempio uso corretto

- [pippo]\$ turing send "ciao come va?"  
Messaggio inviato sulla chat.

### Ricezione dei messaggi sulla chat (*receive*)

Un utente (precedentemente registrato, loggato e che abbia iniziato l'editing) può richiedere di visualizzare tutti i messaggi *non ancora visualizzati* e *inviati successivamente all'inizio dell'editing*, inoltre i messaggi visualizzati comprendono anche quelli inviati dallo stesso utente.

### Esempio uso corretto

- [pippo]\$ turing receive  
pippo: ciao, come va?  
topolino: tutto bene

### Uscita "peaceful"

Un utente non loggato può richiedere la chiusura del Client. Vengono eliminati tutti i documenti di tutti gli utenti e la cartella principale *data\_client*.

### Esempio uso corretto

- ```
$ turing quit
Chiudo il Client.
Cartella /home/nick/Scrivania/Turing/Turing_Client/data_client_1/pippo eliminata.
Cartella /home/nick/Scrivania/Turing/Turing_Client/data_client_1 eliminata.
```

## 4 Info sul progetto

Il progetto è stato sviluppato e testato interamente in ambiente Linux. L'IDE utilizzato è IntelliJ Community Edition. Le conoscenze richieste per la realizzazione di questo progetto, o almeno per gran parte, sono fornite dal materiale presente sulla pagina del Corso di Laboratorio di Reti. Inoltre sono state utili le conoscenze dei concetti imparati nei corsi di Sistemi Operativi, Programmazione 2, Ingegneria del Software. Il sorgente è disponibile su Github: <https://github.com/sadmozer/Turing>