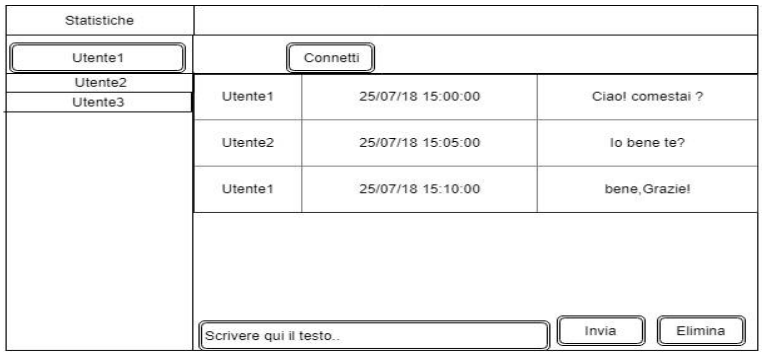
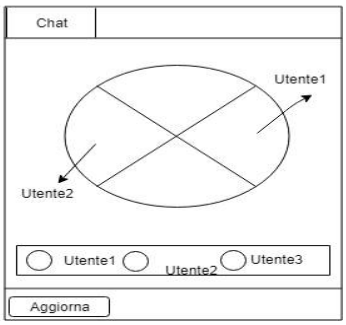
Documento di analisi

- Iacopo Pacini
- Matricola 530649

Mockup:





Documento di analisi: vista dinamica

1. L'utente avvia l'applicazione e inserisce un nome utente nella apposita casella.
   i. Se un altro utente utilizza lo stesso username il sistema glielo notifica scrivendo "nome utente non valido" nella casella di testo.
2. FOR EACH utente collegato il sistema fornisce il nominativo nella colonna a sinistra
3. L'utente clicca su uno o più nomi utente
4. IF l'utente preme invia(Dopo aver scritto il messaggio nella apposita barra) sia destinatari/o che mittente visualizzano il messaggio nella tabella come mostrato sopra.
5. IF uno degli utenti preme logout o chiude la finestra il sistema lo rimuove dall'elenco
6. IF un utente fa il login il sistema lo aggiunge all'elenco
7. IF utente clicca su "Statistiche" si apre l'interfaccia la schermata con il grafo a torta
8. IF utente preme il bottone aggiorna viene aggiornato il grafo che mostra le interazioni con gli utenti.

FILE DI CONFIGURAZIONE LOCALE IN XML

All’avvio il Client legge dal file di configurazione i seguenti dati:

- STILE: font, dimensione del font e colore dello sfondo
- SERVER: IP e porta del server

Il server legge:

- DB: ip,porta e password del database
- Grafo:quanti utenti inserire nel grafo e quanti giorni considerare per il calcolo

ARCHIVIO

Il sistema archivia i seguenti dati testo e data e ora dei messaggi inviati.

File di log remoto in XML

Il sistema invia una riga di log ad ognuno di questi eventi:

- Cambio di scena tra grafo e chat
- Pressione di un pulsante
- Termine dell’applicazione

Ogni riga contiene l’indirizzo IP dell’utente, la data-ora corrente e il nome dell’evento.

Cache Locale

La cache locale permette di ripristinare all'avvio:

- L'ultima vista del grafo
- L'username dell'ultimo utente connesso al servizio nella casella apposita
- gli ultimi messaggi contenuti nella tabella alla chiusura
- Eventuale testo scritto nella casella di invio ma non inviato

Grafo a torta

Il Grafo contiene l'elenco degli N utenti(N configurabile)che hanno ricevuto più messaggi negli ultimi G giorni(G configurabile)

DOCUMENTO DI PROGETTO

Package Client

Classe ClientInterf:

- Classe Front-End
- Si occupa di costruire L'interfaccia dell'applicazione
- Gestisce il caricamento e il salvataggio dei dati sulla cache
- contiene il metodo main
- Crea l'oggetto di tipo GestoreServer

Classe GestoreServer:

- Classe Back-End
- Estensione della classe Thread
- Mantiene una connessione aperta con il Server
- Rimane in ascolto dei messaggi dal server
- Contiene funzioni per l'invio dei messaggi

Classe CacheBinaria

- Classe Back-End
- Rappresenta la cache binaria dell'applicazione

Classe GrafoMessaggiRicevuti

- Classe Front-End
- Estende PieChart
- Contiene funzioni per l'aggiornamento e l'inizializzazione del grafo

Classe TabellaMessaggi

- Classe Front-End
- Costruisce la tabella dove vengo visualizzati i messaggi

Classe CampiTabella

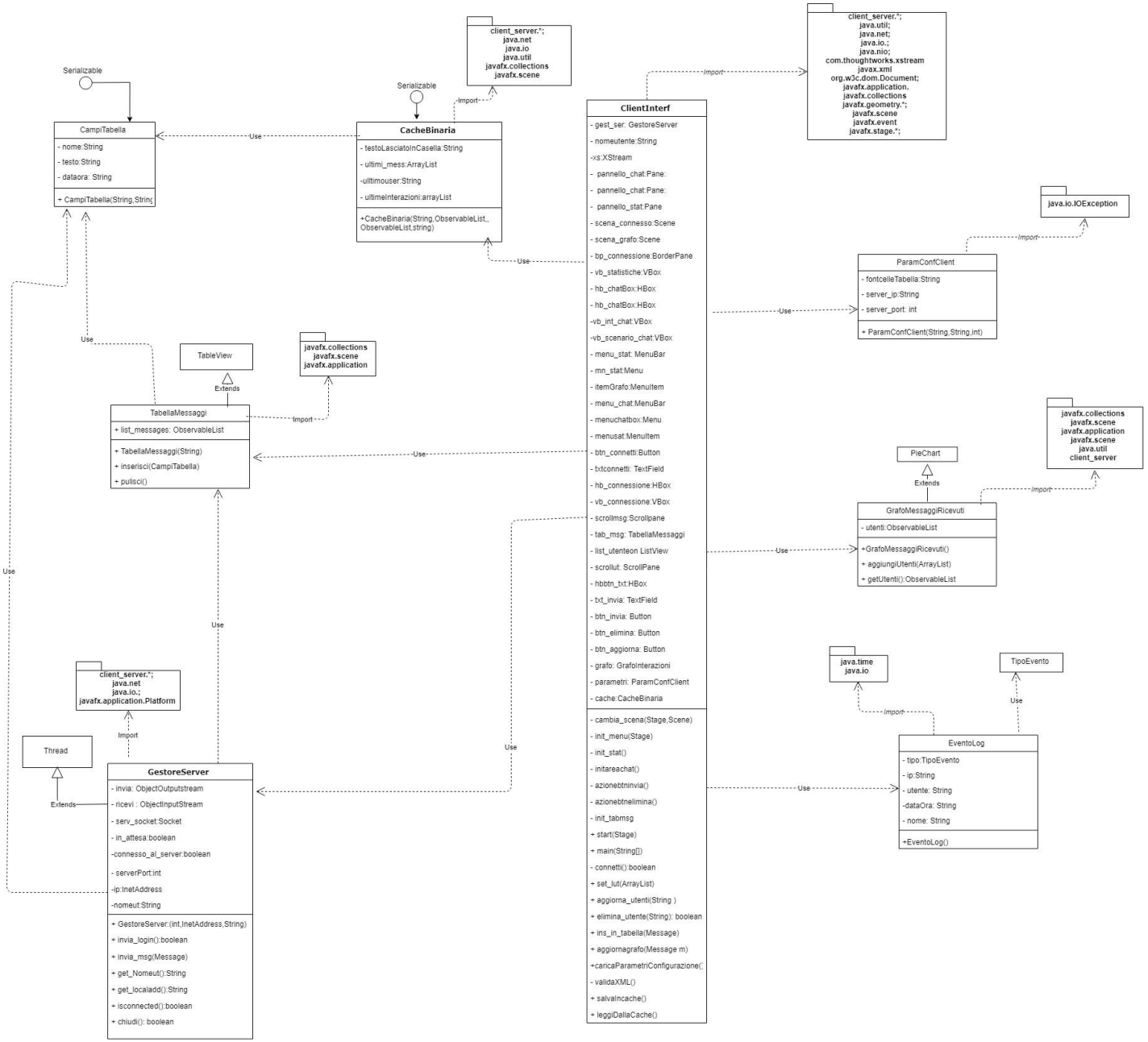
- Classe Front-End
- Descrive il contenuto della TabellaMessaggi

Classe EventoLog

- Classe Back-End
- Descrivi i vari eventi da memorizzare nel file di log remoto in formato xml

Classe ParamConfClient

- Classe Back-End
- Contiene i parametri di configurazione iniziali del client


client_server.*;
java.net
java.io
java.util
javafx.collections
javafx.scene
client_server.*;
java.util;
java.net;
java.io.;
java.nio;
com.thoughtworks.xstream
javax.xml
org.w3c.dom.Document;
javafx.application.
javafx.collections
javafx.geometry.*;
javafx.scene
javafx.event
javafx.stage.*;
Serializable
Serializable
Import
Import
CampiTabella
- nome:String
- testo:String
- dataora: String
+ CampiTabella(String,String
Use
CacheBinaria
- testoLasciatoInCasella:String
- ultimi_mess:ArrayList
-ulltimouser:String
- ultimeInterazioni:arrayList
+CacheBinaria(String,ObservableList,, ObservableList,string)
ClientInterf
- gest_ser: GestoreServer
- nomeutente:String
-xs:XStream
- pannello_chat:Pane:
- pannello_chat:Pane:
- pannello_stat:Pane
- scena_connesso:Scene
- scena_grafo:Scene
- bp_connessione:BorderPane
- vb_statistiche:VBox
- hb_chatBox:HBox
- hb_chatBox:HBox
-vb_int_chat:VBox
-vb_scenario_chat:VBox
- menu_stat: MenuBar
- mn_stat:Menu
- itemGrafo:MenuItem
- menu_chat:MenuBar
- menuchatbox:Menu
- menusat:MenuItem
- btn_connetti:Button
- txtconnetti: TextField
- hb_connessione:HBox
- vb_connessione:VBox
- scrollmsg:Scrollpane
- tab_msg: TabellaMessaggi
- list_utenteon ListView
- scrollut: ScrollPane
- hbbtn_txt:HBox
- txt_invia: TextField
- btn_invia: Button
- btn_elimina: Button
- btn_aggiorna: Button
- grafo: GrafoInterazioni
- parametri: ParamConfClient
- cache:CacheBinaria
- cambia_scena(Stage,Scene)
- init_menu(Stage)
- init_stat()
- initareachat()
- azionebtninvia()
- azionebtnelimina()
- init_tabmsg
+ start(Stage)
+ main(String[])
- connetti():boolean
+ set_lut(ArrayList)
+ aggiorna_utenti(String)
+ elimina_utente(String): boolean
+ ins_in_tabella(Message)
+ aggiornagrafo(Message m)
+caricaParametriConfigurazione(
- validaXML()
+ salvaIncache()
+ leggiDallaCache()
Import
java.io.IOException
ParamConfClient
- fontcelleTabella:String
- server_ip:String
- server_port: int
+ ParamConfClient(String,String,int)
Use
TableView
Extends
javafx.collections
javafx.scene
javafx.application
TabellaMessaggi
+ list_messages: ObservableList
+ TabellaMessaggi(String)
+ inserisci(CampiTabella)
+ pulisci()
Import
Use
javafx.collections
javafx.scene
javafx.application
javafx.scene
java.util
client_server
PieChart
Extends
GrafoMessaggiRicevuti
- utenti:ObservableList
+GrafoMessaggiRicevuti()
+ aggiungiUtenti(ArrayList)
+ getUtenti():ObservableList
Import
Use
Use
Use
Use
Use
client_server.*;
java.net
java.io.;
javafx.application.Platform
Import
Thread
Extends
GestoreServer
- invia: ObjectOutputstream
- ricevi : ObjectInputStream
- serv_socket:Socket
- in_attesa:boolean
-connesso_al_server:boolean
- serverPort:int
-ip:InetAddress
-nomeut:String
+ GestoreServer:(int,InetAddress,String)
+ invia_login():boolean
+ invia_msg(Message)
+ get_Nomeut():String
+ get_localadd():String
+ isconnected():boolean
+ chiudi(): boolean
java.time
java.io
Import
TipoEvento
Use
EventoLog
- tipo:TipoEvento
- ip:String
- utente: String
-dataOra: String
- nome: String
+EventoLog()
Use

Package Server

tutte le classi sono BackEnd

Classe Server:

- Contiene il metodo main
- Responsabile di rimanere in ascolto di nuove richieste di connessione da parte dei client e di creare un oggetto di tipo GestoreUtente per ogni utente richiedente
- Tiene traccia degli utenti connessi attraverso una HashMap che associa il nome utente con il relativo GestoreUtente
- contiene delle funzioni per l'invio dei messaggi.

Classe GestoreUtente

- Estende Thread
- Mantiene una connessione aperta con un determinato utente
- Responsabile dei messaggi da e verso l'utente stesso

Classe GestoreDB

- Responsabile delle interazioni con il DB
- Contiene gli statement per l'aggiornamento dei dati del grafo e per l'inserimento dei messaggi nel db

Classe ParamConfServer

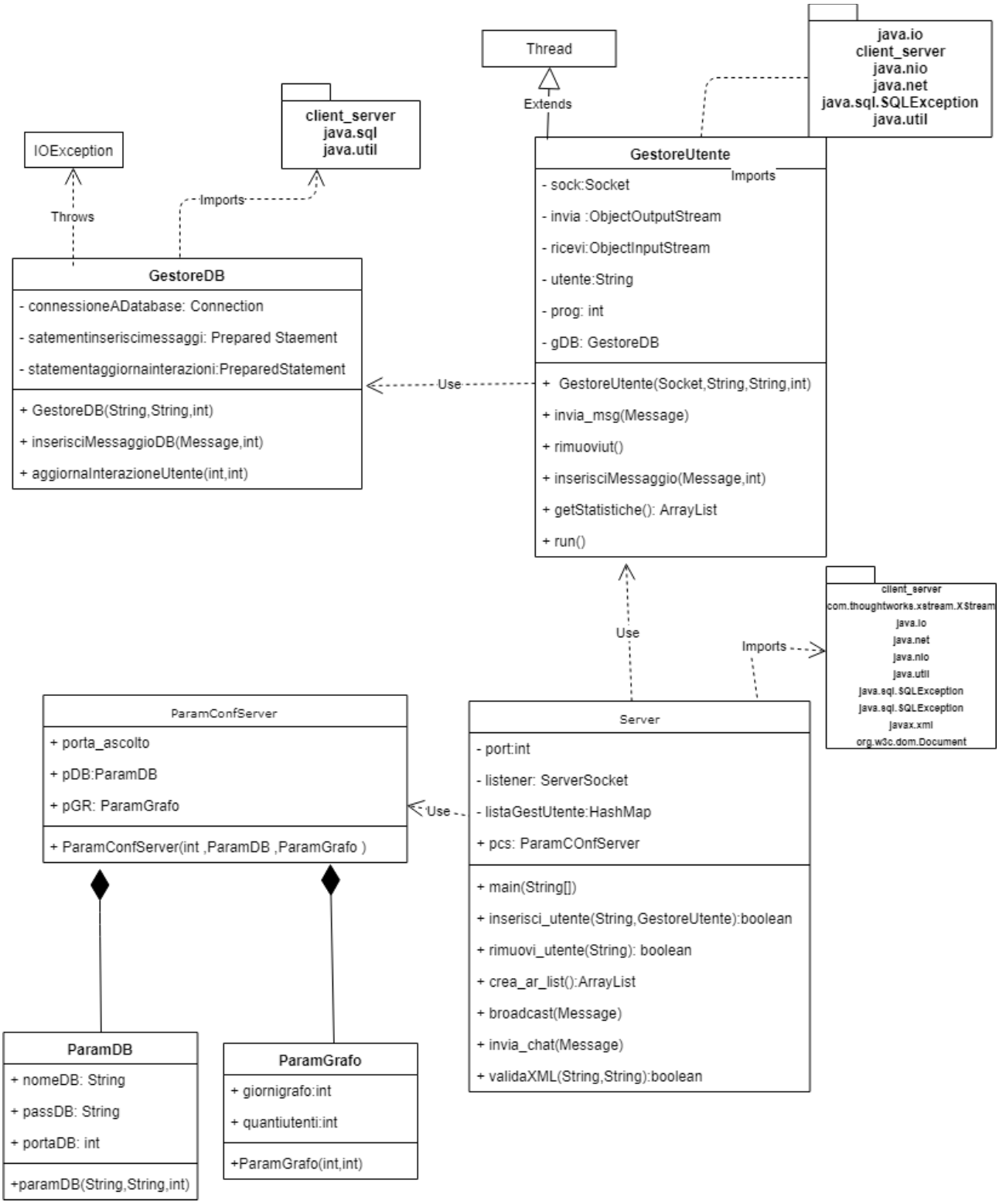
- Contiene i parametri iniziali di configurazione del server

Classe ParamDB

- Contiene porta,nome e password del DB

Classe ParamGrafo

- Contiene il numero degli utenti da visualizzare nel grafo e quanti giorni considerare nell'analisi


Thread
java.io
client_server
java.nio
java.net
java.sql.SQLException
java.util
Extends
client_server
java.sql
java.util
IOException
Imports
Throws
Imports
GestoreUtente
- sock:Socket
- invia :ObjectOutputStream
- ricevi:ObjectInputStream
- utente:String
- prog: int
- gDB: GestoreDB
+ GestoreUtente(Socket,String,String,int)
+ invia_msg(Message)
+ rimuoviut()
+ inserisciMessaggio(Message,int)
+ getStatistiche(): ArrayList
+ run()
GestoreDB
- connessioneADatabase: Connection
- satementinseriscimessaggi: Prepared Staement
- statementaggiornainterazioni:PreparedStatement
+ GestoreDB(String,String,int)
+ inserisciMessaggioDB(Message,int)
+ aggiornaInterazioneUtente(int,int)
Use
Use
Imports
client_server
com.thoughtworks.xstream.XStream
java.io
java.net
java.nio
java.util
java.sql.SQLException
java.sql.SQLException
javax.xml
org.w3c.dom.Document
ParamConfServer
+ porta_ascolto
+ pDB:ParamDB
+ pGR: ParamGrafo
+ ParamConfServer(int ,ParamDB ,ParamGrafo)
Use
Server
- port:int
- listener: ServerSocket
- listaGestUtente:HashMap
+ pcs: ParamCOnfServer
+ main(String[])
+ inserisci_utente(String,GestoreUtente):boolean
+ rimuovi_utente(String): boolean
+ crea_ar_list():ArrayList
+ broadcast(Message)
+ invia_chat(Message)
+ validaXML(String,String):boolean
ParamDB
+ nomeDB: String
+ passDB: String
+ portaDB: int
+paramDB(String,String,int)
ParamGrafo
+ giornigrafo:int
+ quantiutenti:int
+ParamGrafo(int,int)

Package client_server

Classe Message
- Classe BackEnd
- Classe astratta utilizzata per lo scambio di informazioni sia client-client che client-server
- Contiene mittente,destinatario e orario d'invio
- Contiene due funzioni astratte getCampi() e getTesto() implementate opportunamente dalle classi figlie

Classe MessageLOGIN_OUT
- Estende Message
- Viene inviata da un client per connettersi o disconnettersi dal servizio

Classe Message_OK:
- Estende Message
- Inviata dal server ad un client per notificare l'avvenuta registrazione,contiene la lista dei client connessi in quel momento

Classe Message_ERR:
- Estende Message
- Inviata dal server ad un client per notificare la mancata registrazione dovuta ad una collisione sullo username

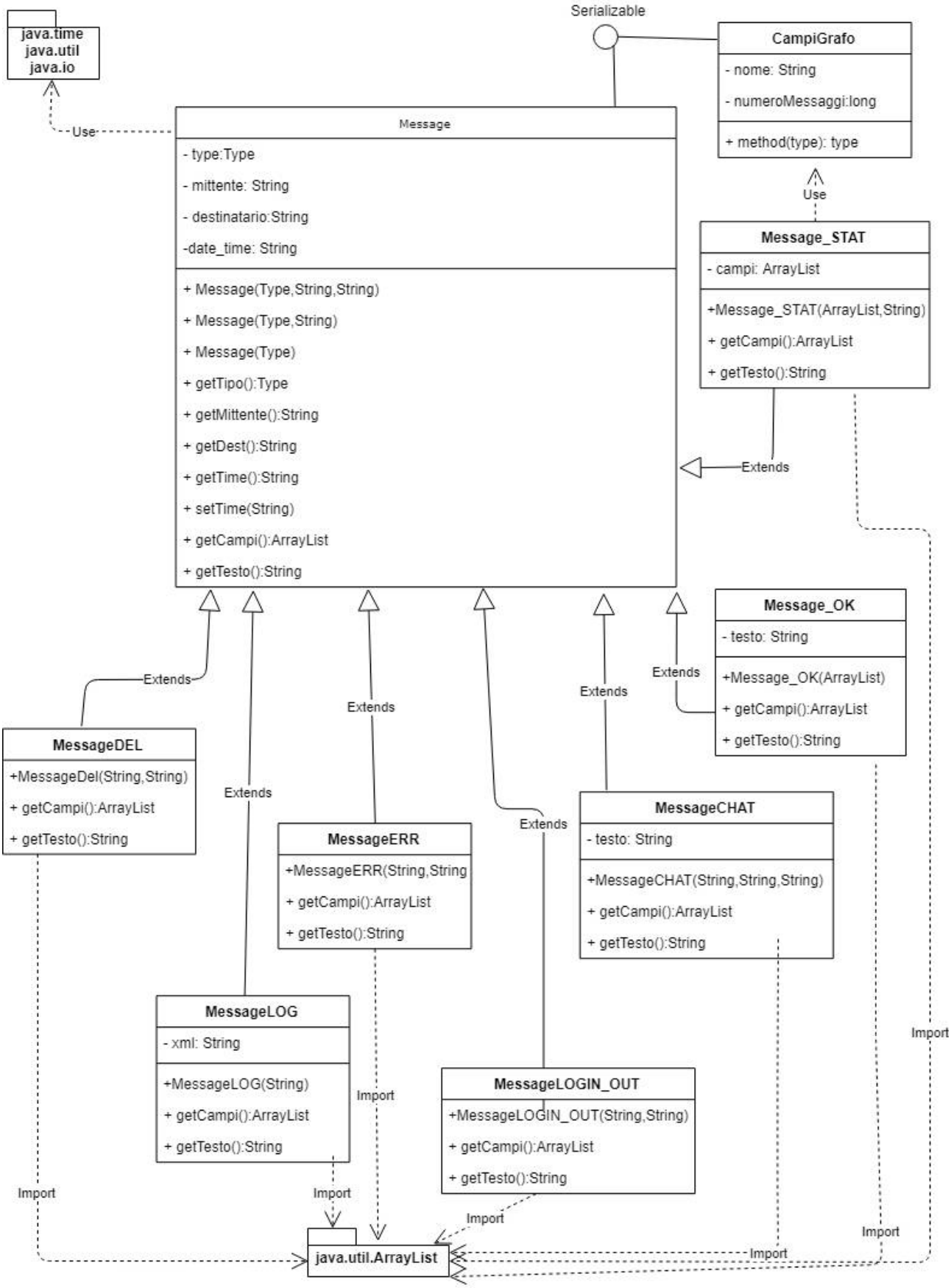
Classe Message_CHAT:
- Estende Message
- Utilizzata dai client per scambiarsi messaggi testuali contenuti nel campo testo

Classe MessageLog:
- Estende Message
- Contiene la riga di log da scrivere nel file di log remoto nel campo testo

Classe MessageSTAT:
- Estende Message
- Contiene nella variabile campi i dati necessari per l'aggiornamento del grafo

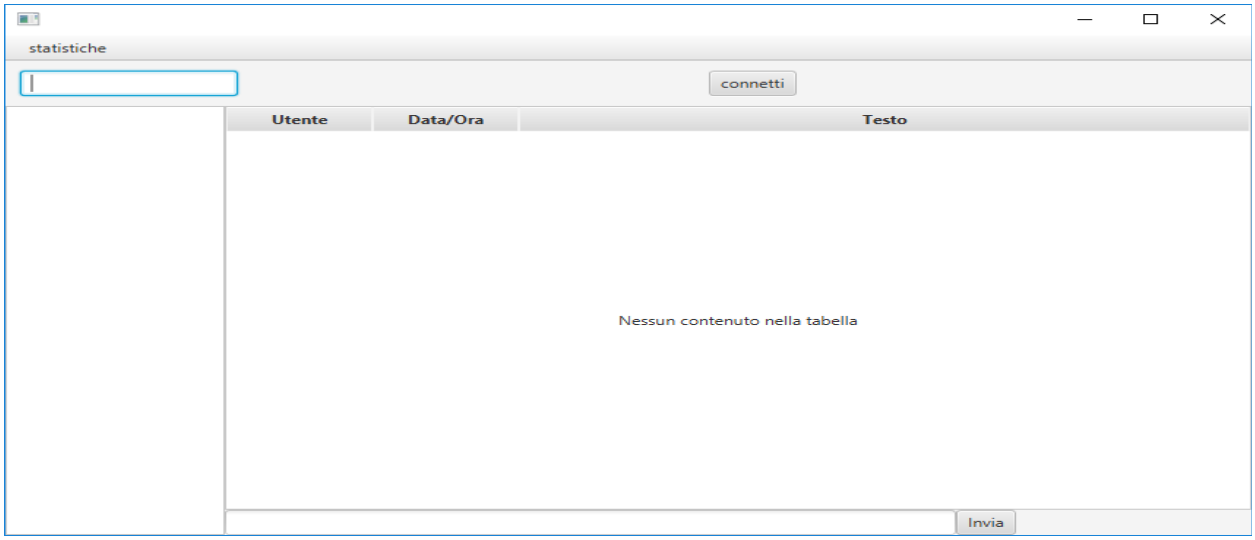
Classe CampiGrafo:
- Utilizzata per trasferire i parametri per l'aggiornamento del grafo

Classe MessageDEL:
- Estende Message
- Inviata da un client quando elimina un proprio messaggio dalla tabella e serve ad eliminarlo anche dalla tabella dei destinatari.

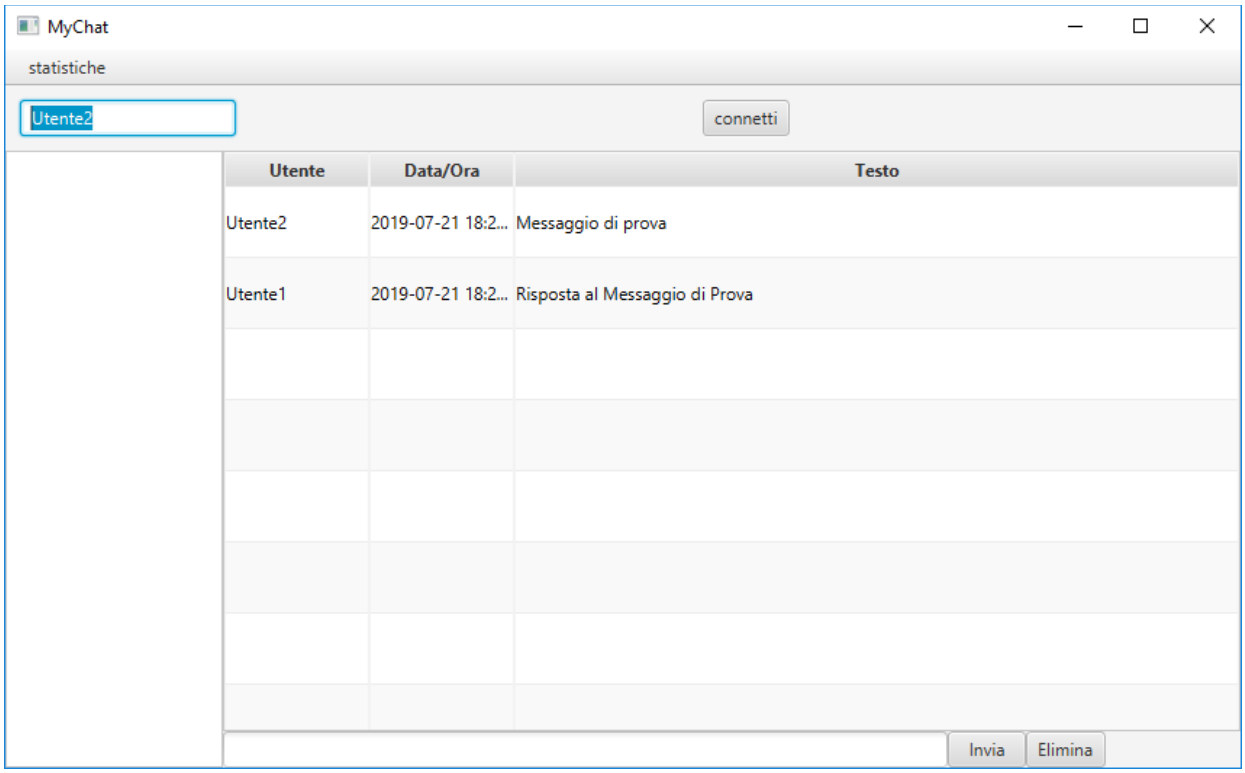

java.time
java.util
java.io
Use
Serializable
CampiGrafo
- nome: String
- numeroMessaggi:long
+ method(type): type
Use
Message
- type:Type
- mittente: String
- destinatario:String
-date_time: String
+ Message(Type,String,String)
+ Message(Type,String)
+ Message(Type)
+ getTipo():Type
+ getMittente():String
+ getDest():String
+ getTime():String
+ setTime(String)
+ getCampi():ArrayList
+ getTesto():String
Message_STAT
- campi: ArrayList
+Message_STAT(ArrayList,String)
+ getCampi():ArrayList
+ getTesto():String
Extends
Message_OK
- testo: String
+Message_OK(ArrayList)
+ getCampi():ArrayList
+ getTesto():String
Extends
Extends
Extends
Extends
MessageDEL
+MessageDel(String,String)
+ getCampi():ArrayList
+ getTesto():String
Extends
MessageERR
+MessageERR(String,String
+ getCampi():ArrayList
+ getTesto():String
Extends
MessageCHAT
- testo: String
+MessageCHAT(String,String,String)
+ getCampi():ArrayList
+ getTesto():String
MessageLOG
- xml: String
+MessageLOG(String)
+ getCampi():ArrayList
+ getTesto():String
MessageLOGIN_OUT
+MessageLOGIN_OUT(String,String)
+ getCampi():ArrayList
+ getTesto():String
Import
Import
Import
Import
Import
Import
Import
java.util.ArrayList

Documento Di Collaudo

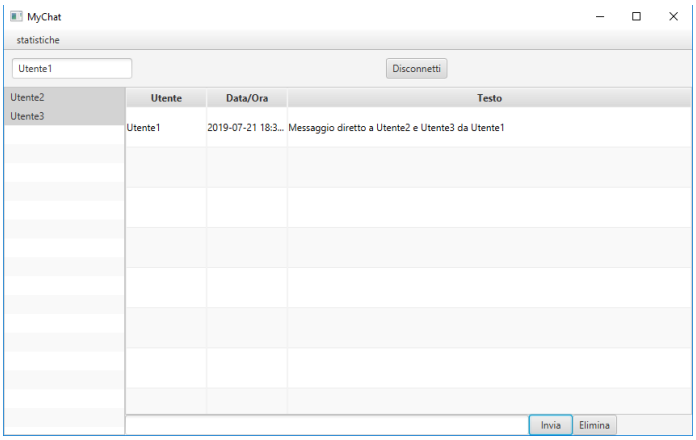
1. L'utente apre l'applicazione, scrive il suo nome sulla casella apposita e preme Connetti.

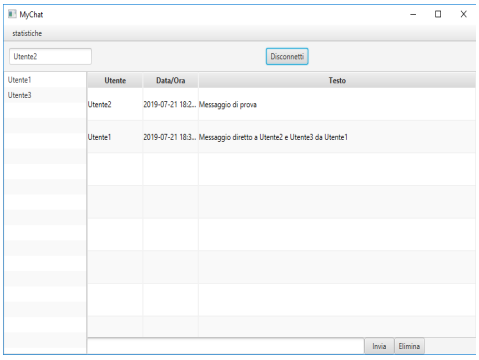


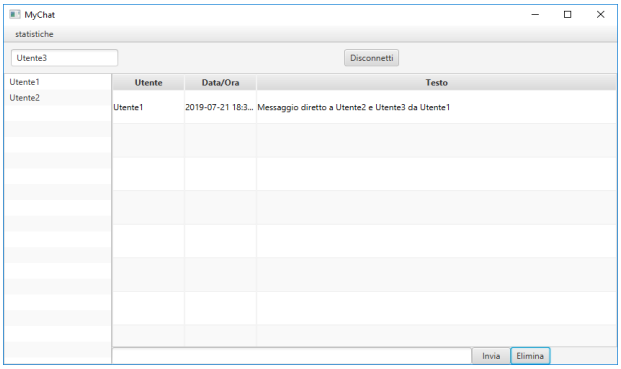
2. Ad una successiva riapertura l'utente trova compilata la casella del nome utente,la casella di invio(se ha lasciato del testo al momento della chiusura),la tabella con gli ultimi messaggi inviati e il grafo come lo ha visualizzato l'ultima volta



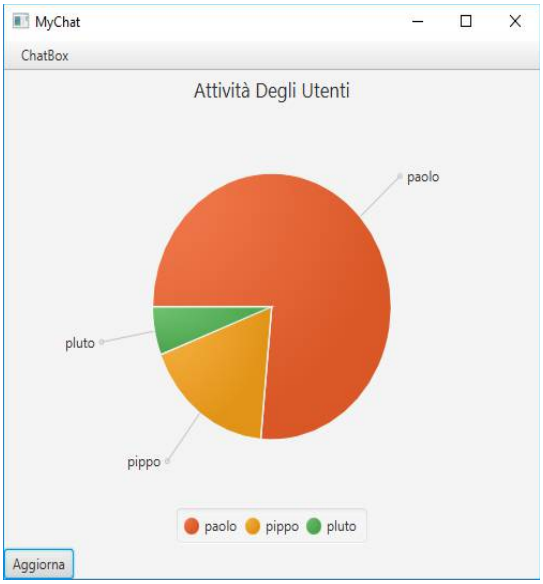
3. L'utente clicca su uno o più utenti nella apposita barra, scrive il messaggio e preme il bottone invia. A quel punto il messaggio viene visualizzato da tutti i destinatari





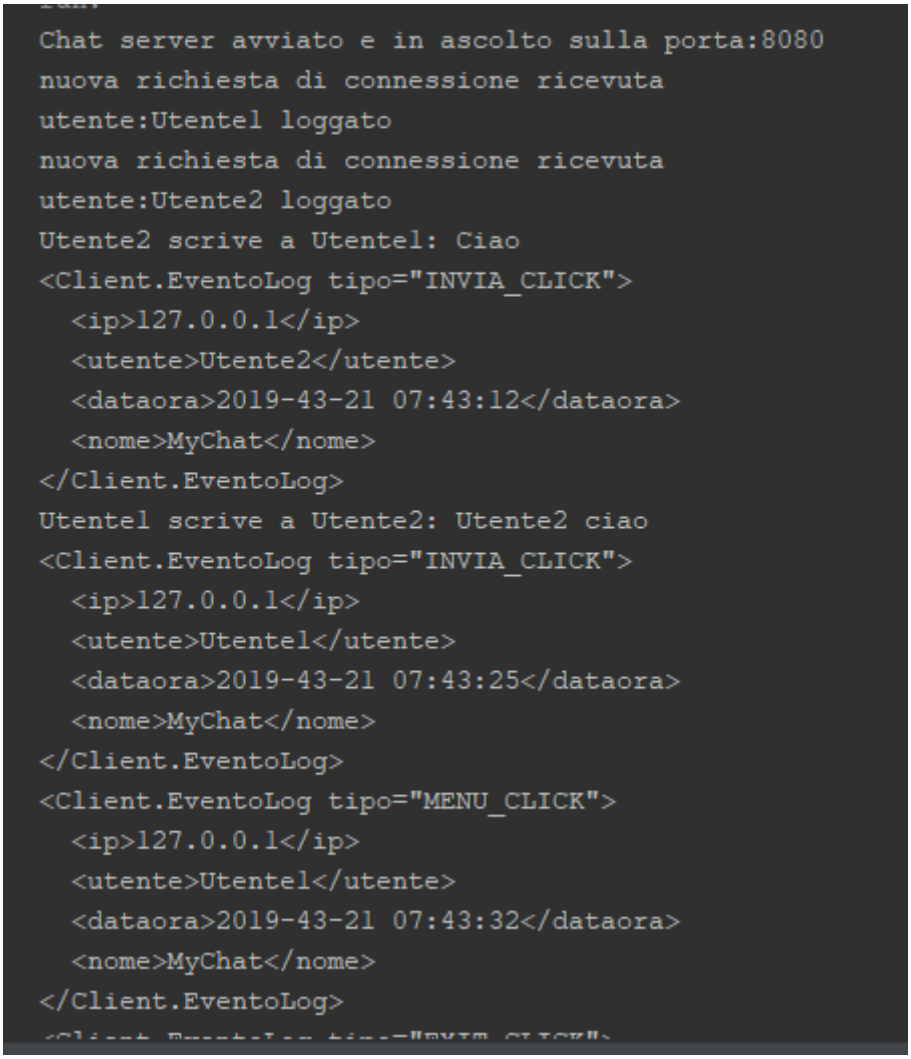


4. Cliccando su "Statistiche" in alto a sinistra si apre un menu che permette di accedere all'interfaccia del grafo



5. Se l'utente seleziona un messaggio altrui sulla tabella e preme il bottone Elimina quest'ultimo viene eliminato dalla tabella. Mentre se elimina un proprio messaggio quest'ultimo viene eliminato anche dalla tabella dei destinatari.

6. Il Server smista i messaggi e aggiorna il file di log



Vista della schermata del server

File di configurazione XML

```
<Server.ParamConfServer>
  <porta_ascolto>8080</porta_ascolto>
  <pDB>
    <nomeDB>prog_av</nomeDB>
    <passDB></passDB>
    <portadb>3306</portadb>
  </pDB>
  <pGR>
    <giornigrafo>15</giornigrafo>
    <quantiutenti>5</quantiutenti>
  </pGR>
</Server.ParamConfServer>
```

Vista dei parametri Server

```
<Client.ParamConfClient>
  <fontCelleTabella>
  -fx-background-color:TRANSPARENT;
  -fx-border-width : 0 1 0 1;-fx-border-color: #F3F3F3;
  -fx-text-fill : black;
  -fx-padding : 16 0 16 0 ;
  </fontCelleTabella>
  <server_ip>127.0.0.1</server_ip>
  <server_port>8080</server_port>
</Client.ParamConfClient>
```

Vista dei parametri Client

File Di Log remoto

```
<EventoLog tipo="INVIA_CLICK">
  <nome>MyChat</nome>
  <ip>127.0.0.1</ip>
  <utente>pluto</utente>
  <dataora>2019-02-03 11:26:17</dataora>
</EventoLog>
```

Esempio di riga di Log XML

Database

Il database Contiene una tabella Messaggi che memorizza i messaggi inviati tra i vari utenti. La chiave primaria è formata da NomeMittente,DataInvio ed un progressivo gestito dalla classe GestoreUtente.

Prog	NomeMittente	NomeDestinatario	DataInvio	Testo
1	pluto	pippo	2019-07-17 ...	dddd
1	pippo	pluto	2019-07-17 ...	Ciao pluto
1	paolo	pippo	2019-07-20 ...	ssss
2	paolo	pippo	2019-07-20 ...	ddd
1	pippo	paolo	2019-07-20 ...	ffff
1	pippo	paolo	2019-07-20 ...	hhhh
2	pippo	paolo	2019-07-20 ...	hhh
1	pippo	paolo	2019-07-20 ...	
2	pippo	paolo	2019-07-20 ...	
3	pippo	paolo	2019-07-20 ...	