

SAMUELE MOSCATELLI, NICOLÒ PINCIROLI,
ANDREA POZZOLI

GESTIONE DEL TRAFFICO



DOCUMENTAZIONE DEL PROGETTO DI
PROVA FINALE

POLIMI

INGEGNERIA INFORMATICA

A.A. 2018 - 2019

Samuele Moscatelli, Nicolò Pincioli,
Andrea Pozzoli,
Gestione del traffico
Documentazione del progetto di prova finale

E-MAIL:
Samuele Moscatelli - sem.mosca97@libero.it
Nicolò Pincioli - nicolopinci.1997@gmail.com
Andrea Pozzoli - pozzoliandrea97@gmail.com



Figura 1: Un esempio di traffico

INDICE

1	IMPLEMENTAZIONE IN JAVA	1
1.1	Sistema Centrale	1
1.2	Centralina	2
1.3	Applicazione	2
1.4	Principali modifiche rispetto alle fasi del progetto precedenti	3
1.5	Algoritmi significativi	4
1.5.1	Calcolo delle applicazioni da notificare	4
1.5.2	Aggiornamento della tabella di traffico	5
1.5.3	Rilevazione della posizione	6
1.5.4	Calcolo dell'intervallo di tempo della centralina	7
1.5.5	Calcolo dell'intervallo di aggiornamento della centralina	8
1.5.6	Variazione di velocità rilevata dalla centralina	9
1.6	Interfaccia grafica	10
1.6.1	Utilizzo di Swing	10
1.6.2	Login e registrazione	10
1.6.3	Sistema centrale	11
1.6.4	Centralina stradale	12
1.6.5	Applicazione mobile	12
1.7	Formato delle notifiche	14

1.8	Protocollo di comunicazione	14
1.9	Casi di test JUnit	15
1.9.1	Aggiunta e rimozione di utenti - AggiuntaUtentiTest	15
1.9.2	Calcolo dell'intervallo di aggiornamento delle centraline - CalcoloIntervalloTest	15
1.9.3	Verificare che la mappa, inizialmente vuota, venga mostrata - FunzionamentoSistemaCentraleTest	15
1.9.4	Riconoscimento di un amministratore - GestoreAmministratoriTest	16
1.9.5	Funzionamento di creaDatoGenerico - GestoreDatabaseTest	16
1.9.6	Test sulla rilevazione della posizione - PosizioneApplicazioneTest	16
1.9.7	Test sulla gestione delle notifiche ricevute dall'applicazione - SegnalaApplicazioneTest	17
1.9.8	Test sulla ricezione di notifiche applicazione da parte del sistema centrale - SegnalaDatabaseTest	17
1.9.9	Test sulla ricezione di dati di traffico - SegnalaDatabaseSTest	17
1.9.10	Verifica dell'accesso e dell'esistenza di un utente usando la classe GestoreApplicazioni - RegistrazioneUtenteTest	17
1.9.11	Verifica dell'accesso e dell'esistenza di un utente usando la classe GestoreUtenti - RiconoscimentoUtenteTest	18
1.9.12	Test sulla creazione di DatoTraffico - CreaDatoTrafficoTest	18
1.10	Simulazione	18
A	LISTA DI INDIRIZZI POSSIBILI	20
	BIBLIOGRAFIA	24

1

IMPLEMENTAZIONE IN JAVA

Il progetto complessivo è costituito da tre progetti, collegati tramite RMI. I progetti realizzati sono **Sistema centrale**, **Centralina¹** e **Applicazione mobile**.

1.1 SISTEMA CENTRALE

Il sistema centrale ha il compito di raccogliere tutti i dati che arrivano dalle applicazioni e dalle centraline stradali, disporli in una mappa geografica (sotto forma di marcatori di colori diversi) e renderla accessibile agli amministratori, rimuovere le segnalazioni meno recenti (dopo due minuti dalla segnalazione, il punto sulla mappa viene rimosso) e calcolare le applicazioni che devono essere notificate (vengono notificate le applicazioni all'interno di un certo raggio intorno alla segnalazione) e inviare a queste una notifica.

Il sistema centrale è composto da cinque gestori: `GestoreDatabase`, `GestoreApplicazioni`, `GestoreAmministratori`, `GestoreUtenti`, `GestoreCentraline`. Il `GestoreDatabase` si occupa della gestione di tutte le notifiche delle applicazioni e delle centraline che vengono inviate al sistema centrale. Per uniformare il tipo di notifica esso le trasforma in `DatoGenerico` e le inserisce in una tabella di traffico. Inoltre quando avviene una nuova ricezione, il `GestoreDatabase` fa partire il calcolo delle applicazioni da notificare. Il `GestoreApplicazioni` si occupa di tenere traccia di tutte le applicazioni mobili creando una lista di `App`, di raccogliere le notifiche in arrivo dalle applicazioni e di inviare a queste le notifiche mandate dal `GestoreDatabase`. Il `GestoreAmministratore` gestisce le informazioni relative a tutti gli amministratori, mentre il `GestoreUtenti` quelle relative agli utenti delle applicazioni mobili. Il `GestoreCentraline` possiede la lista delle Cent e riceve i dati di traffico che arrivano dalle centraline stradali.

La classe `FunzionamentoSistemaCentrale` apre una finestra di Login/Registrazione di un amministratore. Quando viene effettuato il login viene aperta la mappa e vengono messi online i server che raccolgono informazioni dalle applicazioni e dalle centraline.

¹ È stata implementata la centralina stradale

1.2 CENTRALINA

La centralina è composta da un `RilevatoreVeicoliS` e da un `RilevatoreVelocita`. Quando la centralina è in funzione, il rilevatore di veicoli ogni tre secondi sceglie casualmente tra 0 e 1, zero indica che non è passato nessun veicolo nei tre secondi trascorsi, mentre uno che ne è passato uno nel medesimo intervallo. Quando un veicolo viene rilevato, il `RilevatoreVelocita` calcola la sua velocità (in un intorno di ± 15 km/h rispetto alla velocità media) e la somma alle velocità dei veicoli precedenti, memorizzando quindi tale valore in un attributo `sommaVelocita`. Allo scadere di un intervallo di tempo, inizialmente fissato e poi calcolato in base al numero di veicoli transitati, viene generato un dato di traffico che si basa sulla velocità media e sul tipo di strada in cui è posizionata la centralina. Il dato di traffico viene poi inviato al `GestoreCentraline` e vengono azzerati i conteggi di veicoli e velocità.

La classe `FunzionamentoCentralinaS` fa partire i run della centralina e del rilevatore veicoli e così facendo si stabilisce anche la connessione RMI (lato client) con il `GestoreCentraline`.

La classe astratta `Centralina` è stata implementata perché nel momento in cui si volesse installare altri tipi di centralina, sarebbe possibile ereditare da questa gli attributi e i metodi fondamentali per una generica centralina.

Anche la centralina è dotata di un'interfaccia grafica per la sua configurazione (è possibile impostare i seguenti parametri: via o piazza, tipo di strada, velocità iniziale impostata manualmente o casuale) e il suo utilizzo. In particolare premendo il bottone "Imponi una velocità" si può cambiare la velocità media del tratto stradale su cui è installata la centralina.

1.3 APPLICAZIONE

L'applicazione svolge due ruoli: ricevere le notifiche dal sistema centrale mostrandole all'utente, e inviare a sua volta notifiche al `GestoreApplicazioni`. Per questo motivo oltre alla classe `ApplicazioneMobile`, sono state implementate le classi `ApplicazioneClient` e `ApplicazioneServer`, per gestire al meglio la comunicazione in entrambi i lati.

La classe `FunzionamentoApplicazione` crea l'applicazione mobile e permette di eseguire i run dell'applicazione client e dell'applicazione server in contemporanea.

Il lato server dell'applicazione riceve le notifiche dal sistema centrale e le mostra all'utente mettendo in alto la più recente. L'utente può decidere di svuotare la lista delle notifiche in qualsiasi momento.

Il lato client invece crea le notifiche applicazione e le invia al sistema centrale.

La scelta della posizione dell'applicazione avviene casualmente tra una lista di posizioni presente in un foglio di calcolo (queste posizioni sono le uniche valide, anche per la configurazione delle centraline). Si può anche decidere di fissare la posizione dell'applicazione nell'ultima posizione rilevata. L'applicazione gestisce la sua posizione anche grazie al sensore GPS del telefono, che qui viene gestito come un rilevatore di posizioni casuali.

L'applicazione mobile si presenta inizialmente con una grafica per il Login/Registrazione. Avvenuto il login vengono mostrate le notifiche e i pulsanti per l'utilizzo dell'applicazione.

1.4 PRINCIPALI MODIFICHE RISPETTO ALLE FASI DEL PROGETTO PRECEDENTI

Rispetto alle fasi di analisi dei requisiti e di design del sistema complessivo, sono state apportate alcune modifiche.

Le **connessioni** tra i diversi programmi sono state realizzate utilizzando RMI, e non implementando una connessione TCP/IP come supposto all'inizio. Il motivo di questa scelta è che RMI è supportato nativamente in Java e permette di connettere le parti di un sistema distribuito utilizzando una programmazione ad alto livello.

Rispetto all'analisi dei requisiti proposta inizialmente, è stata scelta l'implementazione della centralina stradale. Inoltre la scelta del metodo di rappresentazione dei dati per l'amministratore è ricaduta sulla mappa, che, rispetto al diagramma, rende più semplice il controllo dello stato del traffico complessivo.

Rispetto al **class diagram**, all'interno del quale erano stati riportati soltanto i metodi e gli attributi più significativi per il funzionamento del sistema, sono stati aggiunti alcuni getter e setter e sono state apportate alcune modifiche ai costruttori, in modo da garantire il funzionamento del sistema ed una maggiore leggibilità del codice anche in presenza di RMI.

L'**interfaccia grafica**, non considerata durante la realizzazione del class diagram, ha richiesto l'implementazione di alcune classi aggiuntive, tra cui `LoginDialog`, `RegistrazioneDlg` e `MappaGrafica`. Inoltre è stato necessario implementare alcuni metodi per poter utilizzare le API di `JMapView`.

1.5 ALGORITMI SIGNIFICATIVI

1.5.1 Calcolo delle applicazioni da notificare

```

if (!(this.listaApplicazioni.isEmpty())) {
    for (App var: this.listaApplicazioni) {
        if ((var.getUtente().getUsername() != null) && (! (var.
            getUtente().getUsername().equals(mittente))) {
            IApplicazioneMobile appServer=this.
                getInterfacciaApp(var.getId());
            double R = 6378.137;
            double lat1=posizione.getLatitude();
            double lat2=appServer.getPosizione().
                getLatitude();
            double lon1=posizione.getLongitude();
            double lon2=appServer.getPosizione().
                getLongitude();
            double dLat = lat1 * Math.PI/180 - lat2 * Math.
                PI/180;
            double dLon = lon1 * Math.PI/180 - lon2 * Math.
                PI/180;
            double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                Math.cos(lat1 * Math.PI/180) * Math.cos(
                lat2 * Math.PI/180) * Math.sin(dLon/2) *
                Math.sin(dLon/2);
            double c = 2 * Math.atan2(Math.sqrt(a), Math.
                sqrt(1-a));
            double d = R * c;

            if (d*1000<this.raggio) {
                NotificaApplicazione notifica;    notifica=
                    appServer.creaNotificaApplicazione(
                    posizione, tipo);
                appServer.aggiungiNotificaInCoda(notifica);
            }
        }
    }
}

```

L'algoritmo mostrato rappresenta il procedimento seguito dal GestoreApplicazioni per determinare quali applicazioni notificare in seguito alla segnalazione di un evento di traffico.

In particolare, se la lista non è vuota, viene eseguito un ciclo su tutti gli elementi dell'array, il quale contiene i dati delle applicazioni notificabili, in modo da trovare quelle a cui si è connesso un utente, il quale quindi ha fatto il login o si è registrato, verificando però che non sia lo stesso utente che ha inviato quella segnalazione.

Per ogni applicazione che rispetti questi criteri viene chiamato il metodo che trova nel registry RMI l'interfaccia di tale applicazione e viene calcolata la distanza in metri tra la posizione dell'evento di traffico e quella dell'applicazione; nel caso in cui tale distanza sia minore del raggio predefinito, e quindi nel caso in cui l'applicazione rientri nella zona circolare critica, questa riceve la notifica appena creata.

1.5.2 Aggiornamento della tabella di traffico

```
DatoGenerico datoGenerico=creaDatoGenerico(pos, tipo, data,
    ora, minA, oraA);
GregorianCalendar dat = new GregorianCalendar();
for (int i=this.tabellaTraffico.size()-1;i>=0;--i) {

    int oraAttuale=dat.get(Calendar.HOUR);
    int minAttuale=dat.get(Calendar.MINUTE);
    int minArr=this.tabellaTraffico.get(i).getMinA();
    int oraArr=this.tabellaTraffico.get(i).getOraA();

    if ((oraAttuale==oraArr&&(minAttuale-minArr>2)) || (
        oraAttuale>oraArr&&minArr<58) || (oraAttuale==0&&
        oraArr==11&&minArr<58)) {
        try {
            FunzionamentoSistemaCentrale.getMappa().
                rimuoviMarcatore(this.tabellaTraffico.get(i)
                    .getPosizione().getLatitudine(), this.
                    tabellaTraffico.get(i).getPosizione().
                    getLongitudine());
        } catch(Exception e) {
            JOptionPane.showMessageDialog(null, "La mappa
                non e' disponibile.");
        }
    }

    if (this.tabellaTraffico.get(i).getPosizione().equals(
        pos)) {
        this.tabellaTraffico.remove(i);
        try {
            FunzionamentoSistemaCentrale.getMappa().
                rimuoviMarcatore(pos.getLatitudine(), pos.
                    getLongitudine());
        } catch(Exception e) {
            JOptionPane.showMessageDialog(null, "La mappa
                non e' disponibile.");
        }
    }
}
```

```

        }
    }
    this.tabellaTrafico.add(datoGenerico);

    try {
        FunzionamentoSistemaCentrale.getMappa().aggiungiPunto(
            datoGenerico);
    }
    catch(Exception e) {
        JOptionPane.showMessageDialog(null, "La mappa non e' disponibile.");
    }

    if (!(datoGenerico.getTipo().endsWith("Trafico nella norma")))
    {
        GestoreApplicazioni.getInstance().
            calcolaApplicazioniDaNotificare(mittente, pos, tipo);
    }
}

```

L'algoritmo mostrato rappresenta il metodo con cui viene aggiornata la tabellaTrafico (array contenente tutti i dati ricevuti) e di conseguenza la mappa.

In particolare, una volta creato il dato di tipo DatoGenerico, si scorre la tabella di traffico con un ciclo for al fine di verificare per prima cosa se sono presenti dati obsoleti², e quindi cancellarli, e in secondo luogo per verificare se sono già presenti dati relativi alla posizione della nuova notifica ricevuta e nel caso sostituirli. Dopodiché viene inserito il dato nella mappa sotto forma di marcatore con un colore dipendente dal tipo di evento e, successivamente, nel caso in cui il dato descriva un evento di traffico critico, viene chiamata la funzione calcolaApplicazioniDaNotificare() (descritta nell'algoritmo precedentemente mostrato) al fine di notificare le applicazioni nella zona critica.

1.5.3 Rilevazione della posizione

```

double latitudine=0;
double longitudine=0;
String via;
String percorsoCorrente = System.getProperty("user.dir");

Workbook wb= Workbook.getWorkbook(new File(percorsoCorrente
+ "/vie3.xls"));

Sheet sheet = wb.getSheet(0);

```

² Un dato è obsoleto se è stato ricevuto da più di due minuti

```

Random random = new Random();
int min = 0;
int max = 543;
int viaCasuale = ((max-min) + 1);
int miavar = random.nextInt(viaCasuale) + min;

Cell cella = sheet.getCell(0,miavar);
via = cella.getContents();
cella=sheet.getCell(1,miavar);
latitudine=Double.valueOf(cella.getContents());
cella=sheet.getCell(2,miavar);
longitudine=Double.valueOf(cella.getContents());

this.posizione=new Posizione(via,latitudine,longitudine);
return this.posizione;

```

L'algoritmo mostrato indica il metodo con cui viene rilevata la posizione dell'applicazione.

In particolare, viene fatto riferimento ad un foglio di calcolo contenente le vie della città di Como prese in considerazione. Dopo aver generato un numero casuale indicante il numero della riga da prelevare, vengono a loro volta estratti il nome della via (nella prima colonna), la latitudine (nella seconda colonna) e la longitudine (nella terza colonna), necessari per creare un nuovo oggetto posizione che viene poi assegnato all'applicazione in esame.

1.5.4 Calcolo dell'intervallo di tempo della centralina

```

Registry registry = null;
try {

    registry = LocateRegistry.createRegistry(12345);
    registry.rebind("gestApp", GestoreApplicazioni.
        getInstance());

} catch (Exception e) {
    JOptionPane.showMessageDialog(null, "Il sistema e' 
        gia' in uso");
}

```

La porzione di codice mostrata rappresenta il caricamento dell'interfaccia server del gestore applicazioni nel registry RMI. Il motivo principale per il quale viene mostrato è però un altro: come si può notare nella catch, in caso di eccezione, viene mostrato un pannello di errore che indica che il sistema è già in uso. Ciò è importante in quanto impedisce di poter aprire due o più volte il sistema centrale, che è unico.

1.5.5 Calcolo dell'intervallo di aggiornamento della centralina

```

float temp=((float)numeroVeicoli)/((float)this.
    intervalloDiTempo);

if (temp > 0.0) {
    this.intervalloDiTempo=(int) (this.intervalloDiTempo
        *this.rapporto/temp);

    if (this.intervalloDiTempo < this.intervalloMinimo)
    {
        this.intervalloDiTempo = this.
            intervalloMinimo;
    }

    this.rapporto=temp;
}
else {
    this.intervalloDiTempo=this.intervalloDiTempo*2;
}

```

L'algoritmo, come da specifica, varia l'intervallo di tempo di segnalazione del traffico da parte della centralina in base alla quantità di veicoli rilevati durante l'intervallo appena scaduto. In particolare, se vengono rilevati veicoli, `temp` sarà maggiore di zero, di conseguenza il nuovo intervallo di tempo dipenderà dall'intervallo di tempo precedente (impostato dall'interfaccia della centraline), dal rapporto (inizialmente a zero) e da `temp` stesso.

Più nello specifico, `rapporto` indica il numero di veicoli al secondo e, in particolare, nella formula usata per aggiornare l'intervallo di tempo della centralina esso indica il valore relativo all'intervallo precedente. La variabile `temp` ha lo stesso significato di `rapporto`, tuttavia indica il numero di veicoli al secondo relativo all'intervallo appena scaduto. In questo modo, se passano più vetture, l'intervallo diminuisce, così che vengano inviate più spesso le notifiche relative alla situazione nel tratto di strada considerato, mentre se ne passano meno l'intervallo aumenta.

L'intervallo di tempo ha un valore minimo, sotto il quale non è possibile scendere. Se non vengono rilevati veicoli, invece, viene raddoppiato l'intervallo di tempo considerato.

All'aumentare del numero di veicoli per unità di tempo rilevati, quindi, diminuisce l'intervallo di tempo della rilevazione successiva.

1.5.6 Variazione di velocità rilevata dalla centralina

```

Random random = new Random();
int min;
int max;
if (this.velocita>15) {
    min = this.velocita-15;
}
else {
    min =0;
}
if(this.velocita<95) {
    max = this.velocita+15;
}
else {
    max=110;
}
int intorno = ((max-min) + 1);
int vel = random.nextInt(intorno) + min;
this.sommaVelocita=this.sommaVelocita+vel;

```

Una centralina stradale deve rilevare la velocità dei veicoli che transitano sul tratto di strada a cui si riferisce.

Dal momento che risulterebbe poco realistico se la centralina rilevasse una velocità molto diversa rispetto a quella della misurazione precedente, la simulazione impone che questa possa variare fino ad un massimo di 15 km/h (in più o in meno) da una rilevazione a quella successiva.

Nel caso in cui la velocità dovesse essere inferiore a 15 km/h, il valore minimo consentito per la rilevazione successiva sarà pari a 0 km/h³.

Il valore minimo, il valore massimo e la differenza tra i valori minimo e massimo vengono indicati dalle variabili `min`, `max` e `intorno`.

Una volta generata la velocità casuale, questa viene aggiunta a `sommaVelocita`, che viene utilizzata per il calcolo della velocità media (infatti viene divisa per il numero di automobili rilevate).

³ Una velocità negativa non sarebbe realistica nel sistema reale

1.6 INTERFACCIA GRAFICA

L'interfaccia grafica è stata realizzata per agevolare l'interazione con gli utenti e con gli amministratori. In particolare, per ogni progetto (Centralina, Sistema Centrale e Applicazione) è stata realizzata un'interfaccia diversa.

L'interfaccia grafica è stata realizzata utilizzando Swing [VV_e] ed è stata integrata con le API fornite da OpenStreetMap [VV_b], distribuito con licenza GPL.

1.6.1 Utilizzo di Swing

Le interfacce grafiche sono state realizzate in modo simile. In particolare, si definisce **frame** il contenitore di tutti gli elementi visualizzati nell'interfaccia grafica.

All'interno di ogni frame possono essere posizionati più pannelli (**JPanel**), che vengono utilizzati come dei contenitori di altri elementi. Ad esempio, la GUI della centralina stradale ha tre pannelli, che contengono i dati di impostazione della centralina, l'impostazione (e visualizzazione) della velocità rilevata (per controllare il funzionamento del sistema complessivo) e i bottoni per confermare le scelte effettuate sulla schermata.

Gli elementi che possono essere utilizzati sono vari. Nel caso di questo progetto sono risultati molto utili gli **spinner** (per selezionare numeri in un dato intervallo), le **label**, i **textfield** (per brevi testi), **textpane** (per testi più lunghi), le **combobox** (per scegliere tra più opzioni, sotto forma di stringa), i **button** e gli **optionpane** (per visualizzare messaggi di errore e informazioni).

1.6.2 Login e registrazione



Figura 2: Schermata di scelta tra login e registrazione

Nonostante le interfacce grafiche utilizzate per il login e la registrazione di utenti e di amministratori siano uguali, i dati a cui si fa riferimento sono diversi a seconda della tipologia di accesso.

La schermata permette di scegliere il login, nel caso in cui l'utente (o l'amministratore) si sia già registrato durante la sessione in corso⁴,

⁴ La sessione inizia quando viene aperto il sistema centrale

oppure la registrazione, nel caso in cui la registrazione non sia già stata effettuata con il nome utente desiderato.

Nelle schermate successive sarà possibile fare il logout usando l'apposito tasto o, più semplicemente, chiudendo la finestra (per l'utente dell'applicazione mobile, per l'amministratore della mappa).

1.6.3 Sistema centrale

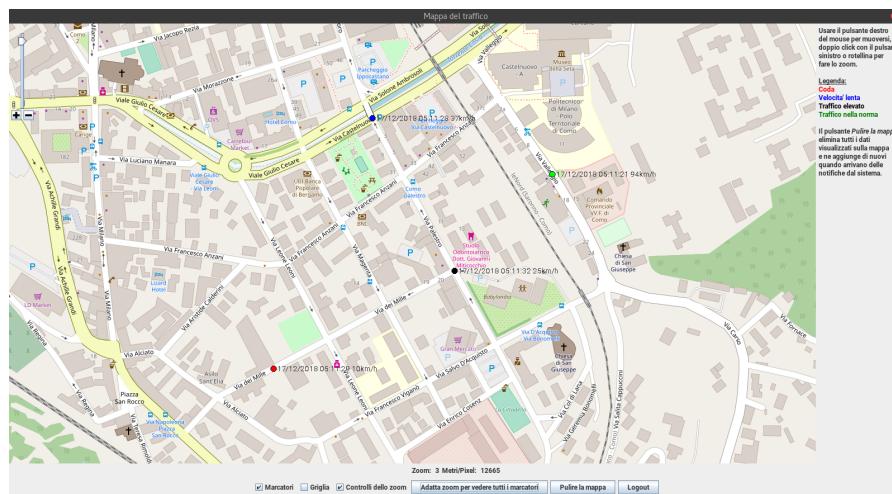


Figura 3: La mappa con alcuni marcatori posizionati durante una simulazione

L'accesso al sistema centrale viene permesso in seguito al login o alla registrazione di un nuovo amministratore. La prima schermata visualizzata, quindi, richiede di effettuare una di queste due operazioni.

Se il login o la registrazione vanno a buon fine viene visualizzata l'interfaccia grafica vera e propria del sistema centrale, costituita dalla mappa (i dati provengono da [VvC]), da alcuni controlli per spostarsi sulla mappa e da un pulsante che permette di effettuare il logout dal sistema centrale.

Quando la mappa riceve dei dati dagli altri sistemi (collegati grazie a RMI [VvD]), li visualizza utilizzando marcatori di colore diverso in base all'evento di traffico (la legenda⁵ sulla destra dell'interfaccia grafica spiega il significato dei diversi colori), riportando, per ogni marcitore, informazioni relative alla data e all'ora di rilevamento e alla velocità media calcolata (se disponibile). I marcatori vengono automaticamente posizionati in corrispondenza della posizione della segnalazione.

Nel caso in cui più marcatori si riferiscano alla stessa posizione viene mantenuta l'ultima informazione ricevuta dal database e i marca-

⁵ **Rosso:** coda; **Nero:** traffico elevato; **Blu:** velocità lenta; **Verde:** nessun evento di traffico.

tori relativi alla stessa posizione già presenti vengono eliminati dalla mappa.

1.6.4 Centralina stradale

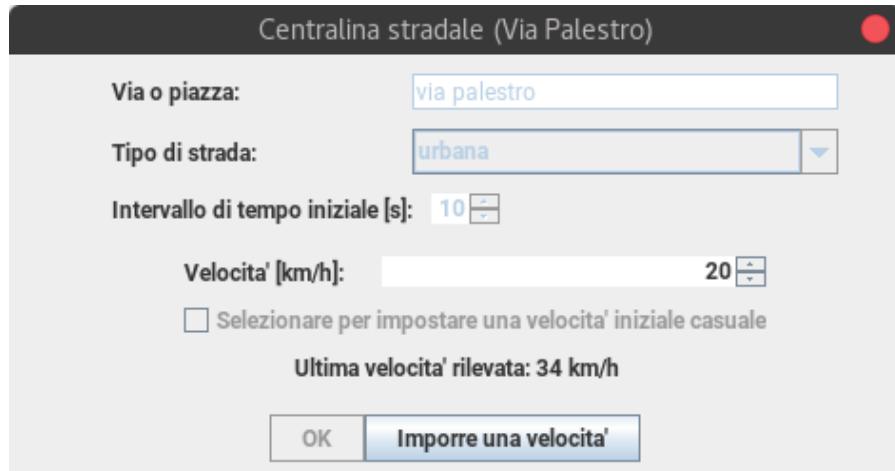


Figura 4: Schermata iniziale della centralina stradale

La centralina stradale viene impostata senza effettuare il login e la registrazione, infatti immaginando che il sistema venga effettivamente realizzato è ragionevole pensare che la centralina venga impostata, per esempio da un operatore, nel momento in cui viene installata e che la configurazione rimanga immutata fino ad un suo eventuale riavvio.

L’interfaccia grafica della centralina permette anche di imporre la velocità registrata in modo da testare il funzionamento dei sistemi anche in assenza di dati reali. In alternativa è possibile utilizzare un simulatore di velocità che genera dei valori casuali che oscillano intorno all’ultimo valore registrato.

Una volta effettuata l’impostazione iniziale, la posizione della centralina (che sarà necessariamente fissa) verrà indicata nella barra del titolo della finestra di impostazione. In ogni caso, sarà possibile osservare un marcitore, di colore variabile in funzione dell’evento di traffico registrato, in corrispondenza della posizione di installazione.

1.6.5 Applicazione mobile

L’applicazione mobile consente all’utente, che deve registrarsi o effettuare il login, di segnalare una coda nella posizione in cui si trova (questa apparirà sotto forma di marcitore sulla mappa, se in esecuzione) e di visualizzare le notifiche relative ad un’area circolare⁶ con centro nella posizione rilevata dall’applicazione e raggio di 500 m.

⁶ A cui è possibile riferirsi come **area critica**.



Figura 5: L'applicazione mobile con alcune notifiche.

Le notifiche più recenti appariranno nella parte superiore dell'area di notifica e saranno di colori diversi in base alla tipologia, seguendo le stesse convenzioni dei marcatori visualizzati sulla mappa. Non vengono visualizzate le segnalazioni di assenza di traffico perché non rilevanti.

Al fine di testare il sistema complessivo, è possibile richiedere che l'applicazione rimanga nella stessa posizione tra una rilevazione e l'altra.

1.7 FORMATO DELLE NOTIFICHE

Le notifiche sono fondamentali per garantire la comunicazione tra i diversi programmi. Per questa ragione le notifiche devono avere un formato standard che sia comprensibile da tutti i programmi.

In particolare, ogni notifica è caratterizzata da un tipo, che è una stringa. Il formato di questo attributo è il seguente: M oppure S⁷ + velocità + carattere '' + Tipo di evento.

Nel caso della segnalazione della coda da parte delle applicazioni mobili, la velocità non viene rilevata, quindi può essere inserita una qualsiasi stringa prima del carattere '' e questa verrà ignorata.

Un esempio di notifica può essere 'S10 Traffico elevato' e significa che una centralina stradale invia una notifica di traffico elevato registrando una velocità di 10 km/h.

Le notifiche verranno visualizzate sull'applicazione mobile se sono riferite a segnalazioni provenienti da altre applicazioni o centraline stradali nel raggio di 500 m.

Il formato di queste ultime notifiche, però, è più comprensibile per un utente umano. La notifica 'S10 Traffico elevato' proveniente da via Valleggio il giorno 17/12/2018 alle ore 11:11, per esempio, verrà visualizzata come 17/12/2018, 11:11 | Traffico elevato in via Valleggio ad una velocità media di 10 km/h e sarà di colore nero (vedere anche figura 5).

Le notifiche dell'applicazione mobile sono realizzate con un codice HTML, grazie alla libreria HTMLDocument[[VVa](#)].

1.8 PROTOCOLLO DI COMUNICAZIONE

Il sistema centrale, la centralina e l'applicazione vengono eseguiti su macchine differenti e quindi è necessario configurare una connessione tra questi. Il protocollo di comunicazione che è stato utilizzato è RMI. Sono state programmate tre differenti connessioni RMI.

Le centraline sono state sviluppate come client del GestoreCentraline che lavora da server. Il GestoreCentraline implementa l'interfaccia IGestoreCentraline attraverso la quale riceve i dati di traffico inviati dalle centraline.

Le applicazioni e il GestoreApplicazioni si comportano sia da server sia da client. Le applicazioni sono dei server quando devono ricevere delle notifiche dal GestoreApplicazioni, e a questo proposito implementano l'interfaccia IApplicazione. Il GestoreApplicazioni invece implementa l'interfaccia IGestoreApplicazioni per ricevere notifiche da parte delle applicazioni.

Le porte che sono state utilizzate sono:

⁷ M nel caso in cui il dato si origini dall'applicazione mobile, S nel caso in cui provenga da una centralina stradale

- 12344 per la connessione tra centraline e GestoreCentraline;
- 12345 per la connessione tra applicazioni e GestoreApplicazioni in cui è quest'ultimo a fare da server;
- da 12346 in poi, per la connessione in cui le applicazioni fanno da server e il numero di porta è ottenuto sommando a 12346 il numero di identificativo dell'applicazione.

1.9 CASI DI TEST JUNIT

1.9.1 Aggiunta e rimozione di utenti – AggiuntaUtentiTest

Prima del test vengono aggiunti tre utenti tramite il GestoreUtenti gest, quindi questi vengono inseriti all'interno della lista listaUtenti.

Successivamente, durante il test vero e proprio, gli utenti vengono rimossi da listaUtenti grazie alla funzione rimuoviUtente() di gest. ListaUtenti alla fine del test deve risultare vuota.

Il test fallirebbe nel caso in cui si dovesse chiamare rimuoviUtente() senza passare tutti gli username degli utenti aggiunti.

1.9.2 Calcolo dell'intervallo di aggiornamento delle centraline – CalcoloIntervalloTest

Il test serve per verificare che, in assenza di veicoli registrati da una centralina, l'intervallo di rilevazione raddoppi.

Viene creata una centralina stradale, la cui posizione è ininfluente ai fini del test, ha un intervallo iniziale di 10 s ed è posizionata su una strada urbana (anche questo dettaglio è ininfluente).

Quando viene chiamata la funzione calcolaIntervallo() si passa come argomento o, che è il numero di veicoli rilevati.

In particolare, il test consiste nel verificare che il nuovo intervallo (centralina.getIntervallo()) sia pari a 20, ovvero il doppio dell'intervallo iniziale.

1.9.3 Verificare che la mappa, inizialmente vuota, venga mostrata – FunzionamentoSistemaCentraleTest

Il test viene utilizzato per verificare che la mappa venga effettivamente visualizzata nel momento in cui viene chiamata la funzione visualizzazioneMappaBase().

Si tratta di un test molto compatto dal momento che la funzione visualizzazioneMappaBase() ritorna mappa e che la funzione isVisible() è contenuta all'interno delle API di JMapViewer. Allo stesso

tempo è fondamentale che abbia successo per garantire che l'amministratore visualizzi le informazioni sul traffico.

1.9.4 Riconoscimento di un amministratore - GestoreAmministratoriTest

Il test serve per verificare che, una volta aggiunto un amministratore alla lista degli amministratori, questo venga riconosciuto nel momento in cui viene cercato.

La funzione centrale per il riconoscimento è `riconosciAmministratore()`, che richiede il nome utente e la password dell'amministratore e restituisce `true` o `false`.

1.9.5 Funzionamento di creaDatoGenerico - GestoreDatabaseTest

`DatoTrafico` è una specializzazione di `DatoGenerico`, di conseguenza alcune delle informazioni ottenute a partire da `DatoTrafico` potranno essere applicabili a `DatoGenerico` e le informazioni ottenute a partire da `DatoGenerico` saranno sicuramente applicabili a `DatoTrafico`.

Il test verifica che il `DatoGenerico` costruito a partire da alcune delle informazioni di `DatoTrafico` ed elaborato dal `GestoreDatabase` sia uguale al `DatoGenerico` che dovrebbe contenere gli stessi dati.

Il test, quindi, verifica indirettamente che `DatoTrafico` sia una specializzazione di `DatoGenerico`, anche se il suo scopo è la verifica del funzionamento corretto di `creaDatoGenerico()` in `GestoreDatabase`.

1.9.6 Test sulla rilevazione della posizione - PosizioneApplicazioneTest

Il test è suddiviso in due test e vuole verificare il corretto funzionamento dell'aggiornamento della posizione di un'applicazione.

Il primo test crea un'applicazione e aggiorna la sua posizione con `getPosizione()`. Successivamente viene imposto all'applicazione di fissare la sua posizione, e quindi si prova a aggiornare la posizione. L'asserzione verifica che la posizione non sia cambiata.

Il secondo test invece svolge gli stessi passaggi del test precedente. A differenza del primo però, dopo aver aggiornato la posizione, toglie il vincolo che fissa la posizione e chiama nuovamente `getPosizione()`. In questo test si verifica quindi che la posizione è cambiata rispetto alla posizione iniziale.

1.9.7 Test sulla gestione delle notifiche ricevute dall'applicazione - SegnalaApplicazioneTest

Anche questo test è diviso in due parti e vuole verificare la corretta gestione della lista delle notifiche che riceve l'applicazione.

Nel primo test l'applicazione riceve tre notifiche e le aggiunge alla lista notifiche. Per verificare che ciò venga fatto, si fa un controllo sul numero di notifiche all'interno dell'ArrayList.

Il secondo test aggiunge al test precedente l'istruzione che svuota completamente la lista notifiche. In questo caso la dimensione dell'ArrayList deve tornare a zero perché tutte le notifiche vengono eliminate.

1.9.8 Test sulla ricezione di notifiche applicazione da parte del sistema centrale - SegnalaDatabaseTest

Il test ancora una volta è diviso in due. In questo caso si vuole verificare il corretto funzionamento della gestione dei dati che riceve in ingresso il sistema centrale.

Nel primo test vengono create due notifiche applicazione in posizioni differenti e si vuole verificare che il GestoreDatabase le tramuti correttamente in DatoGenerico e le aggiunga alla tabella di traffico.

Nel secondo test si prova a inviare una notifica nella stessa posizione di una notifica precedente e si verifica che il sistema confronti nel modo corretto le posizioni e elimini la notifica meno recente per aggiungere quella nuova.

1.9.9 Test sulla ricezione di dati di traffico - SegnalaDatabaseSTest

Questo test è molto simile al precedente, l'unica differenza è che al posto delle notifiche applicazione, il test viene effettuato su dati di traffico in arrivo dalle centraline.

1.9.10 Verifica dell'accesso e dell'esistenza di un utente usando la classe GestoreApplicazioni - RegistrazioneUtenteTest

Nel momento in cui un utente tenta di registrarsi, è necessario che l'username scelto sia diverso rispetto a quello di tutti gli altri utenti. Inoltre, nel momento in cui un utente vuole fare il login, in seguito ad una precedente registrazione, bisogna controllare che l'utente esista e, in questo caso, che la password inserita sia corretta.

Per questa ragione sono stati realizzati due metodi `verificaAccesso` ed entrambi vengono testati. Viene testato, quindi, sia il riconoscimento di un utente effettivamente esistente, sia il mancato riconoscimento di un utente non ancora registrato.

Entrambe le versioni (riconoscimento di utenti esistenti e blocco di utenti non ancora registrati) sono molto importanti, infatti verificano che non si possa accedere al sistema se non autorizzati.

1.9.11 Verifica dell'accesso e dell'esistenza di un utente usando la classe GestoreUtenti - RiconoscimentoUtenteTest

Questo test è molto simile al precedente, ma sfrutta la classe `GestoreUtenti` piuttosto di `GestoreApplicazioni`.

Lo scopo del test è quindi quello di verificare che il sistema funzioni anche chiamando `riconosciUtente` al posto di `verificaAccesso`.

1.9.12 Test sulla creazione di DatoTraffico - CreaDatoTrafficoTest

Il test in analisi è suddiviso in tre verifiche distinte che controllano la correttezza della creazione di dati di traffico da parte delle centraline.

Nella sezione `BeforeEach` vengono create tre centraline, ognuna con un tipo di strada differente. In tutte e tre le centraline viene settata la velocità media pari a 45 km/h, e viene creato un dato di traffico per ciascuna.

I tre test verificano che il dato creato abbia il tipo di evento di traffico corretto, che varia a seconda della velocità media per ogni tipologia di strada.

1.10 SIMULAZIONE

Per simulare al meglio il progetto è consigliato seguire i seguenti passi:

1. Nel progetto "SistemaCentrale" aprire il package "prog" e eseguire "FunzionamentoSistemaCentrale".
2. Completare la registrazione dell'amministratore (username e password non possono essere vuoti altrimenti viene visualizzato un messaggio di errore). Se la registrazione andrà a buon fine verrà aperta la mappa (verificare di essere connessi a una rete internet).
3. Può essere eseguito solamente un "FunzionamentoSistemaCentrale" alla volta, se un altro venisse eseguito verrebbe mostrato un messaggio di errore e terminerebbe.
4. Nel progetto "Centralina" aprire il package "prog" e eseguire "FunzionamentoCentralinaS". Se "FunzionamentoCentralinaS" venisse eseguito prima di "FunzionamentoSistemaCentrale" verrebbe mostrato un messaggio di errore e terminerebbe.

5. Scrivere il nome di una via di Como (anteponendo "via"), se non dovesse essere presente nella lista delle vie possibili verrà chiesto di inserirne un'altra. Per essere sicuri di inserire vie esistenti, **aprire il file vie3.xls presente in tutti e tre i progetti e selezionare una tra le vie presenti.**
6. Compilare gli altri campi e nel caso si volesse scegliere una velocità iniziale casuale selezionare la casella apposita.
7. Nel progetto "Applicazione" aprire il package "prog" e eseguire "FunzionamentoApplicazione". Se "FunzionamentoApplicazione" venisse eseguito prima di "FunzionamentoSistemaCentrale" verrebbe mostrato un messaggio di errore e terminerebbe. Non è invece obbligatorio eseguire prima la centralina dell'applicazione.
8. Completare la registrazione dell'applicazione (username e password non possono essere vuoti altrimenti viene visualizzato un messaggio di errore). Se la registrazione andrà a buon fine verrà aperta l'interfaccia grafica dell'applicazione.
9. Possono essere eseguiti contemporaneamente quante applicazioni e centraline si voglia.

A questo punto si può scegliere di testare tutti i comandi a disposizione, alcuni esempi sono:

- Effettuare il logout e login dell'applicazione.
- Segnalare la coda con l'apposito pulsante dell'applicazione.
- Pulire la lista delle notifiche.
- Fissare la posizione di qualsiasi applicazione.
- Impostare una velocità media alla centralina stradale per impostare un evento di traffico.
- Pulire manualmente la mappa o attendere che venga pulita in automatico (ogni punto rimane sulla mappa per tre minuti, poi viene eliminato).
- Adattare lo zoom in modo che tutti i punti vengano visualizzati al meglio sulla mappa premendo l'apposito pulsante sulla mappa.

A | LISTA DI INDIRIZZI POSSIBILI

Quello che segue è l'elenco degli indirizzi che è possibile inserire nel campo "Via o piazza" della centralina stradale, presenti anche nel foglio di calcolo. Il campo non è case sensitive ed è necessario inserire il nome completo della via (ad es. via Valleggio, e non Valleggio).

Via Abba	Via Fiammenghino	Via Piave
Via Acquanera	Via Fiume	Via Picasso
Via Adamello	Via Florio	Via Picchi
Via Agliati	Via Fogazzaro	Via della Pila
Via Airoldi	Via Fontana	Piazzetta Pinchetti
Via Alberina	Via Fontanella	Via Pio XI
Via Albertolli	Via Fornace	Via Plinio
Via Albricci	Via Foscolo	Via Pola
Via Alciato	Via Fossati	Via Polano
Via Alebbio	Via Fra Silvestro da Siena	Via Ponte Nuovo
Via Ambrosoli	Via Franscini	Piazza del Popolo
Piazza Amendola	Via Frigerio	Via Porro
Via Amoretti	Via Frisia	Via Porta
Via Annunciata	Via Friuli	Via Prato Pasquee
Via Antelami	Via Fulda	Via Prestino
Via Appiani	Via Fumagalli	Via Pretorio
Via Arcioni	Via Gaggi	Via Primo Maggio
Via Artaria	Via Gagini	Via Prinetti
Via Asiago	Via Galilei	Via Priva
Via Auguadri	Via Galli	Via Privata Perlasca
Via Badone	Via Gallio	Via Prudenziiana
Via Bainsizza	Via Galvani	Viale Puecher
Via Balbiani	Via Garibaldi	Via Quadrio
Via Balestra	Via Campo Garibaldi	Via Quaglio
Via Ballarini	Via Garovaglio	Via Quarcino
Via Baracca	Via Gasparotto	Via Quasimodo
Via Baraggia	Via Gattoni	Via Raffaello
Via Baragiola	Viale Geno	Via Raimondi
Via Baravalle	Piazzale Gerbetto	Via Rampoldi
Via Barberini	Via Ghislanzoni	Via Raschi
Via Barelli	Via Giacosa	Via Recchi
Via Bari	Via Gioa	Via Regazzoni
Via Barsanti	Via Giorgione	Via Regina Teodolinda
Via Barzaghi	Piazzale Giotto	Via Repubblica Romana
Via Baserga	Via Giovio	Via Rezzonico
Via al Bassone	Via Giudici	Via Rho
Via della Bastiglia	Via Giulio Cesare	Via Ricci
Viale Battisti	Via Giustizia e Liberta	Via Rienti
Via Bazzoni	Via Gobbi	Via Rienza
Via Beccaria	Via Gobetti	Via Righi
Via Bellini	Via Golasecca	Viale Rimembranza
Via Bellinzona	Via Gorio	Via Rimoldi

Via Belvedere	Via Gorizia	Via Ripamonti
Via Benzi	Via Gramsci	Via Rismondo
Via Bernasconi	Via Grandi	Via Risorgimento
Via Bernina	Via Grassi	Via Ristori
Via Bertacchi	Via Grilloni	Via Riviera
Via Bertinelli	Via Guaita	Via Roasio
Via Bertolone	Via Guanella	Via al Roccolo
Via Bianchi Giovini	Via Imbonati	Via Rodari
Via Bignanico	Via Indipendenza	Piazza Roma
Via Binda	Viale Innocenzo XI	Via Roncate
Via Bixio	Via Interna	Via Ronchetto
Via Boccioni	Via Isonzo	Via al Ronco
Via Boldoni	Via Italia Libera	Via Rosales
Via Bonanomi	Piazza IV Novembre	Via Rosati
Via Bonardi	Via Juvara	Via Roscio
Via Bonola	Via Kolbe	Viale F.Lli Rosselli
Via Bonomelli	Via Lambertenghi	Via Rossini
Via Bontempelli	Via Landriani	Via Rota
Via Borgo Vico	Via del Lavoro	Via Rovelli
Via Borsieri	Via Lazzago	Via Rubini
Via Bosatta	Viale Lecco	Via Rusconi
Via Boselli	Strada Statale per Lecco	Via Rutschi
Via Bossi	Via Lega Insurrezionale	Via Sabotino
Via Botticelli	Largo Leopardi	Via Sacchi
Via Brambilla	Via Liberta	Via Sacco
Via Brenna	Via Linati	Via Sagnino
Via Brennero	Via Lissi	Via Sala
Via Brenta	Via Longhena	Via Salvadonica
Via Breva	Via Longoni	Piazza San Fedele
Via Briantea	Via di Lora	Via San Felice
Via Brogeda	Via Konrad Lorenz	Viale San Fermo della Battaglia
Via per Bronno	Via Lucini	Via San Francesco d'Assisi
Via Bronzetti	Via Luini	Via San Giacomo
Via per Brunate	Via Macchi	Via San Giovanni da Meda
Via B. Buozzi	Via Maderno	Piazzale San Gottardo
Via Buschi	Via Madruzza	Via San Martino
Piazza Cacciatori delle Alpi	Via Maestri Campionesi	Via San Michele del Carso
Via Cadorna	Via Maestri Comacini	Piazzale San Rocchetto
Via Caduti Albatesi	Via Magistretti	Piazzale San Rocco
Via Cairoli	Via Majnoni	Via San Zenone
Via Calderini	Via Maloja	Via Sant'Abbondio
Piazza Camerlata	Via Malvito	Via Sant'Antonino
Via Camozzi	Via Manara	Via Sant'Arialdo da Cucciago
Via Campagna	Via Mantegna	Via Sant'Elia
Via Campari	Via Marchesi	Via Sant'Eutichio
Via Campora	Viale Marconi	Via Santa Caterina
Via Canonica	Via Mariani	Via Santa Chiara
Via Canova	Via Martinelli	Via Santa Maria in Cristino
Via Cantoni	Via Masaccio	Piazzale Santa Teresa
Via Cantoniga	Via Mascherpa	Via Saporiti
Via Cantore	Viale Masia	Via N. Sauro
Via Cantu	Via Massardi	Via Scalabrinii
Via Canturina	Piazza Matteotti	Via Scalini
Via Capiaghi	Viale G. Matteotti	Via Scarabota
Via Cappelletti	Piazza Mazzini	Via Schiavio
Via Caprani	Piazza Medaglie d'Oro	Via Scolari
Via Caracciolo	Via Mentana	Via Sebenico

Largo Caradocco	Via Meroni	Via Sebregondi
Via Carcano	Via Merzario	Via Selva Fiorita
Via Cardano	Via Meucci	Via Silva
Via Cardina	Via Miani	Viale Sinigaglia
Via Carducci	Via Michelangelo	Via Sirtori
Via Carloni	Via Milano	Via Soave
Via Carluccio	Via dei Mille	Via Solari
Via Caronti	Via Mincio	Piazzale Somaini
Via Carpani	Via Mirabello	Via Somigliana
Via Carso	Via Mocchetti	Via Spalato
Via Casartelli	Via Mognano	Via Spallanzani
Via Casati	Via Mola	Via Spallino
Via Cascina Viola	Via Monte Bianco	Via Spartaco
Via Castel Carnasino	Via Monte Caprino	Via Spazzi
Via Castellini	Via Monte Croce	Largo Spluga
Via Castelnuovo	Via Monte Goi	Viale Sport
Via Catenazzi	Via Montegrappa	Via Sportivi Comaschi
Via Catone	Via Montello	Via Stampa
Viale Cattaneo	Via Montelungo	Via Stazzi
Via Catullo	Via Montenero	Via Stoppani
Via alla Cava	Via Monterotondo	Via Strabone
Viale Cavallotti	Via Monticelli	Via Tagliamento
Piazza Cavour	Via Moraglia	Via Tatti
Via Cecilio	Via Morazzone	Via Tentorio
Largo Ceresio	Via Aldo Moro	Via Terlizza
Via Ceresola	Via Muggio	Via Tettamanti
Via Cermenati	Via Mugiasca	Via Tibaldi
Via per Cernobbio	Via dei Mulini	Via Ticozzi
Via Ceruti	Via Muralto	Via Timavo
Via Cetti	Via Musa	Via Tintoretto
Via Ciapparelli	Via Museo Giovio	Via Tiziano
Via Cigalini	Via Muttoni	Via Tobagi
Via Cinque Giornate	Via Napoleona	Via Tofane
Via Clemente XIII	Via Natta	Viale Tokamachi
Via Clerici	Via Navedano	Via Torno
Via Col di Lana	Via Negretti	Via Torriani
Via Colli	Via Nicolodi	Via Trau
Via Collina	Via I. Nievo	Via Trecallo
Via C. Colombo	Via Ninguarda	Via Trento
Via Colonna	Via Nulli	Via Lungo Lario Trento
Via Comerio	Via Odescalchi	Via Tridi
Via Comolli	Via Oldelli	Via Lungo Lario Trieste
Via Conciliazione	Via Olgati	Via Urbano II
Via Conconi	Via Olginati	Via Valbasca
Piazza Concordia	Via Oltrecolle	Via Valera
Via Confalonieri	Via Orazio	Via Valeria
Via Coni Zugna	Via Orelli	Via Valleggio
Via Coretta	Via Oriani	Via Valorsa
Via Corridoni	Via Ortelli	Via Varese
Via dei Cortili	Via Ortigara	Via Varesina
Via Cossoni	Via dell' Orto	Via Vassena
Via Crespi	Via Osjava	Via Vela
Via Cressoni	Via Ostinelli	Via Velzi
Via Crispi	Via Ovidio	Via Venini
Via Croce Rossa	Via Pacinotti	Viale Venturino
Via Crotto del Sergente	Via Pagani	Via Venusti
Via Cuzzi	Via Palestro	Piazza Verdi

Via D'Acquisto	Via Paluda	Via Verga
Via D'Azeglio	Via Pannilani	Via Vigano
Via D'Oggiono	Via Pantera	Via Villa Giovio
Via Pietro Da Breggia	Via Paradiso	Villa Olmo
Via Leonardo Da Vinci	Via Parini	Via dei Villini
Via Dante	Via Parrocchiale	Via Virgilio
Via De Cristoforis	Via Partigiani	Via Vitani
Via G. Di Vittorio	Via Pascoli	Piazza Vittoria
Via A. Diaz	Via Passeri	Via Vittorio Emanuele II
Via Don Bosco	Via Pastonchi	Via Vittorio Veneto
Via Don Gnocchi	Via Pastrengo	Via Vodice
Via Don Minzoni	Via Pasubio	Via Volpati
Via Don Monza	Via Pedemonte	Via Volpi
Via Donatori di Sangue	Salita Peltlera	Piazza Volta
Via del Dos	Via Perego	Via A. Volta
Via Dottesio	Via Perlasca	Passeggiata Voltiana
Via Dotti	Via del Pero	Via XX Settembre
Via Druso	Via Perpenti	Piazza XXIV Maggio
Piazza Duomo	Via Pertì	Via XXVII Maggio
Vicolo Duomo	Via Perugino	Via Zamenhof
Via Durini	Via Pessina	Via Zampiero
Via El Alamein	Via Petrarca	Via Zanella
Via Falciola	Via Petrolo	Via Zara
Via G. Ferrari	Via Piadeni	Via Zezio
Via Ferrata	Via al Piano	Via alla Zocca

BIBLIOGRAFIA

- [VVa] AA. VV. *Class HTMLDocument - javax.swing.text.html.HTMLDocument*. URL: <https://docs.oracle.com/javase/10/docs/api/javax/swing/text/html/HTMLDocument.html>.
- [VVb] AA. VV. *JMapView*. URL: <https://wiki.openstreetmap.org/wiki/JMapView>.
- [VVc] AA. VV. *OpenStreetMap*. URL: <https://www.openstreetmap.org/>.
- [VVd] AA. VV. *Package java.rmi*. URL: <https://docs.oracle.com/javase/10/docs/api/java/rmi/package-summary.html>.
- [VVe] AA. VV. *Package javax.swing*. URL: <https://docs.oracle.com/javase/10/docs/api/javax/swing/package-summary.html>.