

Università degli Studi di Napoli "Parthenope"

DIPARTIMENTO DI SCIENZE E TECNOLOGIE
CORSO DI RETI DI CALCOLATORI

MedTaxi

Arenella Samuel 012400/2529
Iommelli Raffaele 012400/2491

Progetto realizzato per lo svolgimento dell'esame di Reti di calcolatori,
integrato a Programmazione III ed Ingegneria del software ed Interazione
Uomo-Macchina

Anno Accademico 2023/24

Indice

1	Descrizione del progetto	2
2	Descrizione e schema dell'architettura	4
2.1	Prenotazione	4
2.2	Tracciamento in Tempo Reale	4
2.2.1	Architettura di Rete per il Tracciamento	4
2.2.2	Integrazione delle API di Google Maps	5
2.2.3	Flusso di Dati e Gestione degli Aggiornamenti	5
2.3	Schema riassuntivo	6
3	Dettagli implementativi dei client/server con relativo codice	7
3.1	Implementazione del Client	7
3.1.1	Invio delle Prenotazioni e ricezioni delle aziende disponibili	7
3.1.2	Inoltro della prenotazione da server ad azienda	9
3.1.3	Invio Coordinate	10
4	Manuale utente con le istruzioni su compilazione ed esecuzione	14
4.1	Prerequisiti	14
4.2	Configurazione del Database	14
4.3	Download e Configurazione dei Progetti	15
4.3.1	ServerMedTaxi	15
4.3.2	MedTaxi	15
4.4	Esecuzione	16
4.4.1	Azienda	16
4.4.2	Utente	20

Capitolo 1

Descrizione del progetto

MedTaxi è un innovativo sistema di gestione dei trasferimenti sanitari progettato per migliorare l'efficienza e la trasparenza dei trasporti medici. Il progetto propone una soluzione autonoma per affrontare le sfide legate alla gestione dei trasferimenti sanitari.

Il principale obiettivo di MedTaxi è ridurre le attese e l'incertezza nei trasferimenti sanitari, fornendo agli utenti e alle aziende di trasporto sanitario un sistema efficiente e intuitivo per prenotare, monitorare e gestire tali trasferimenti.

Il sistema si compone di:

Server Centrale:

- Funge da intermediario tra le aziende di trasporto sanitario e gli utenti.
- Le aziende private registrano i propri servizi e disponibilità come client verso questo server.
- Gli utenti fanno le richieste di prenotazione attraverso il server centrale.

Comunicazione tra Server e Aziende:

- Il server centrale inoltra le richieste di prenotazione alle aziende di trasporto sanitario.
- Le aziende, agendo come server, gestiscono queste prenotazioni e mantengono uno stato aggiornato dei trasferimenti.

Comunicazione Continua:

- Quando il mezzo è disponibile, ad esempio, 30 minuti prima dell'appuntamento, l'azienda di trasporto apre una comunicazione continua con il cliente.
- Questa comunicazione permette di aggiornare il cliente sulla posizione corrente del mezzo, offrendo un servizio in stile "tracking" per la massima tranquillità del paziente.

Caratteristiche Principali del progetto sono:

- Prenotazioni Semplici: Gli utenti possono inserire gli indirizzi di partenza e destinazione, selezionare la data dell'appuntamento e scegliere tra le aziende di trasporto sanitario disponibili in base a vicinanza o prezzo.
- Monitoraggio in Tempo Reale: A partire da 30 minuti prima dell'appuntamento, MedTaxi consente agli utenti di monitorare l'ambulanza in tempo reale, permettendo loro di prepararsi in anticipo.
- Registrazione Aziendale: Le aziende di trasporto sanitario possono registrarsi su MedTaxi per inserire le loro tariffe e gestire le prenotazioni, garantendo un servizio efficiente

e tempestivo.

Vantaggi:

- Riduzione delle attese e dell'incertezza per i pazienti.
- Ottimizzazione delle operazioni per le aziende di trasporto sanitario.
- Controllo completo del processo di trasferimento per gli utenti.

Capitolo 2

Descrizione e schema dell'architettura

2.1 Prenotazione

Utente (Client): Invia la richiesta di prenotazione al server centrale, includendo i dettagli necessari come la destinazione e l'orario di partenza.

Server Centrale: Riceve la richiesta e la elabora per identificare le aziende di trasporto sanitario disponibili. Invia poi queste informazioni all'utente.

Utente (Client): Sceglie un'azienda di trasporto sanitario dall'elenco fornito dal server.

Server Centrale: Invia la scelta dell'utente all'azienda di trasporto sanitario selezionata.

Azienda di Trasporto Sanitario (Client): Riceve i dettagli della prenotazione e, una volta disponibile il mezzo, inizia la comunicazione in tempo reale con l'utente per fornire aggiornamenti sulla posizione.

Questo schema rappresenta un'architettura client-server dove il server centrale svolge un ruolo chiave nell'intermediazione delle comunicazioni tra gli utenti e le aziende di trasporto sanitario.

La capacità di fornire aggiornamenti in tempo reale e la gestione efficiente delle prenotazioni sono aspetti fondamentali di questa architettura.

2.2 Tracciamento in Tempo Reale

Questo meccanismo si basa sull'utilizzo combinato del protocollo UDP per lo streaming dei dati di posizione e delle API di Google Maps per il calcolo dei percorsi e la geolocalizzazione.

2.2.1 Architettura di Rete per il Tracciamento

L'architettura di rete di MedTaxi per il tracciamento utilizza due componenti principali:

- **Server UDP (Azienda di Trasporto Sanitario):** Incaricato di inviare le coordinate geografiche dell'ambulanza agli utenti. Utilizza un `DatagramSocket` per trasmettere pacchetti UDP che contengono le coordinate di posizione dell'ambulanza a intervalli regolari.
- **Utente UDP (Cliente):** Rappresenta il lato client che riceve gli aggiornamenti di posizione. Anch'esso impiega un `DatagramSocket` per ascoltare i pacchetti UDP

inviati dal server. Implementa il pattern Observer per gestire gli aggiornamenti di posizione.

2.2.2 Integrazione delle API di Google Maps

Una componente cruciale dell'architettura di tracciamento è l'integrazione delle API di Google Maps, che permette di:

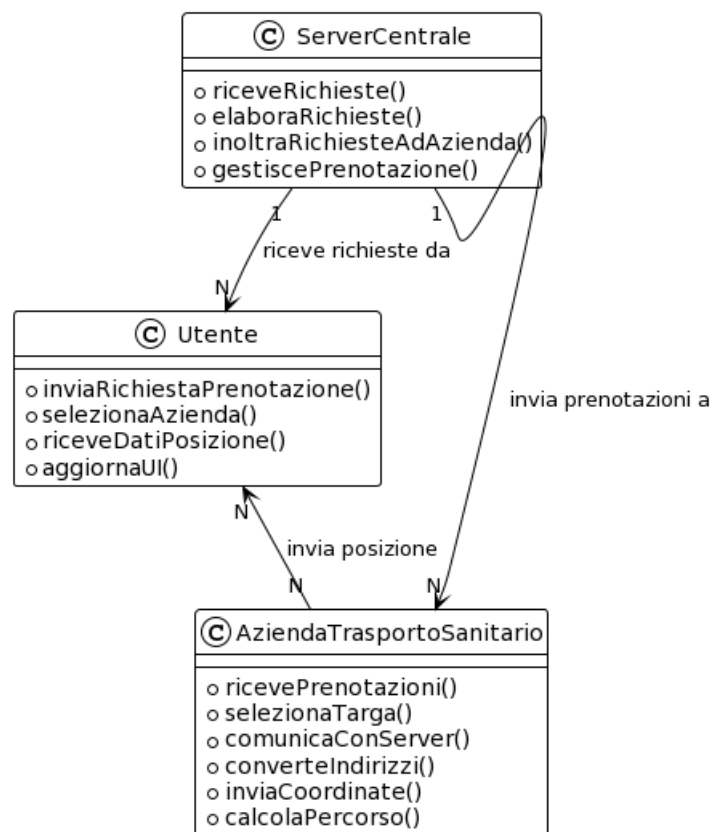
1. **Calcolare il Percorso:** All'inizio del tracciamento, il **ServerUDP** calcola il percorso tra l'indirizzo di partenza e quello di arrivo, utilizzando le API di Google Maps per ottenere le coordinate latitudinali e longitudinali degli indirizzi.
2. **Geocodifica:** Le API di Google Maps sono utilizzate per convertire gli indirizzi di partenza e arrivo in coordinate geografiche, facilitando così la definizione del percorso che l'ambulanza seguirà.
3. **Ottimizzazione del Percorso:** Le informazioni dettagliate sul percorso, incluse le coordinate di ogni punto del tragitto, vengono calcolate per simulare il movimento dell'ambulanza lungo il percorso.

Queste funzionalità sono essenziali per garantire che il sistema di tracciamento possa fornire aggiornamenti accurati e tempestivi sulla posizione dell'ambulanza, migliorando significativamente l'esperienza dell'utente finale.

2.2.3 Flusso di Dati e Gestione degli Aggiornamenti

Il server procede all'invio delle coordinate di posizione a intervalli regolari via pacchetti UDP. L'Utente UDP, ricevendo questi dati, notifica gli osservatori registrati degli aggiornamenti di posizione, consentendo loro di aggiornare l'interfaccia utente corrispondentemente.

2.3 Schema riassuntivo



Capitolo 3

Dettagli implementativi dei client/server con relativo codice

Questo capitolo descrive in dettaglio l'implementazione dei componenti client e server nel sistema MedTaxi, focalizzandosi su aspetti chiave come la comunicazione di rete, la gestione delle richieste, e l'integrazione con le API di Google Maps per il tracciamento in tempo reale.

3.1 Implementazione del Client

Il componente client è responsabile per l'invio delle richieste di prenotazione al server tramite protocollo TCP e la ricezione degli aggiornamenti di posizione tramite il protocollo UDP. È implementato in Java e si interfaccia con il server centrale per l'inoltro delle richieste e la ricezione delle risposte.

3.1.1 Invio delle Prenotazioni e ricezioni delle aziende disponibili

Le prenotazioni vengono inviate al server utilizzando socket TCP. Il client crea una connessione socket verso il server, invia i dati della prenotazione in formato JSON, e attende la risposta del server.

```
1  public void switchToNextScene(ActionEvent event) throws IOException {
2      Client client = new Client();
3      JSONObject prenotazioneJson = new JSONObject();
4
5      LocalDate dataSelezionata = data_trasporto.getValue();
6      prenotazioneJson.put("nome", nome_paziente.getText());
7      prenotazioneJson.put("cognome", cognome_paziente.getText());
8      prenotazioneJson.put("email", User.getInstance().getEmail());
9      prenotazioneJson.put("partenza", indirizzo_partenza.getText());
10     prenotazioneJson.put("arrivo", indirizzo_arrivo.getText());
11     prenotazioneJson.put("data", dataSelezionata.toString());
12     prenotazioneJson.put("fasciaOraria", fasceOrarieComboBox.getValue()
13 );
14     prenotazioneJson.put("cellulare", numero_cellulare.getText());
15     // Invio prenotazione e attesa risposta
```



```

16     String risposta = client.inviaPrenotazione(prenotazioneJson.
toString());
17     JSONObject Jrisposta = new JSONObject(risposta);
18
19     // Analisi della risposta per ottenere le aziende disponibili
20     JSONArray aziendeDisponibili = Jrisposta.getJSONArray("
aziendeDisponibili");
21     List<String> listaAziende = new ArrayList<>();
22     for (int i = 0; i < aziendeDisponibili.length(); i++) {
23         listaAziende.add(aziendeDisponibili.getString(i));
24     }
25     //presa del codice
26     this.codice = Jrisposta.getString("codice");
27
28
29     // Passaggio alla scena di selezione dell'ambulanza/azienda
30     FXMLLoader loader = new FXMLLoader(getClass().getResource("/com/
example/medtaxi/utente/prenota/seleziona_ambulanza.fxml"));
31     Parent root = loader.load();
32
33     //passiamo il tutto al controller successivo
34     SelezionaContr selezionaContr = loader.getController();
35     selezionaContr.setCodice(codice);
36     selezionaContr.setJSON(risposta);
37     selezionaContr.setAmbulanzeDisponibili(listaAziende);
38     selezionaContr.setClient(client);
39
40
41     stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
42     scene = new Scene(root);
43     stage.setScene(scene);
44     stage.show();
45 }
46
47
48
49
50
51     // Costruttore dove viene creata la socket
52     public Client() {
53         try {
54             socket = new Socket("localhost", 12346);
55             input = new BufferedReader(new InputStreamReader(socket
.getInputStream(), "UTF-8"));
56             output = new BufferedWriter(new OutputStreamWriter(
socket.getOutputStream(), "UTF-8"));
57         } catch (IOException e) {
58             e.printStackTrace();
59         }
60     }
61
62
63
64
65
66
67     // Metodo per l'invio
68     public String inviaPrenotazione(String prenotazione) {

```

```

69     try {
70         // Invia il JSON al server
71         output.write(prenotazione.toString());
72         System.out.println("JSON inviato al server: " + prenotazione);
73         output.newLine();
74         output.write("END");
75         output.newLine();
76         output.flush();
77
78         // Aspetta la risposta dal server con le aziende disponibili
79         System.out.println("Aspetto la risposta dal server...");
80         String response = input.readLine();
81
82         //presa del codice dal jsonObject
83         JSONObject jsonResponse = new JSONObject(response);
84
85         return response;
86     } catch (IOException e) {
87         e.printStackTrace();
88         return null;
89     }
90 }

```

Questo frammento di codice mostra come il client invii e riceva una stringa JSON contenente i dettagli della prenotazione al server e come gestisca la risposta ricevuta.

3.1.2 Inoltro della prenotazione da server ad azienda

```

1 public class AziendaHandler {
2     private String piva;
3     private Date giorno;
4     private int disponibilita_ambulanze;
5
6     public void immissione(String piva, Date giorno, int
7     disponibilita_ambulanze) {
8         this.piva = piva;
9         this.giorno = giorno;
10        this.disponibilita_ambulanze = disponibilita_ambulanze;
11    }
12
13
14
15
16
17 public void startServerTask() {
18     Task<Void> serverTask = new Task<Void>() {
19         @Override
20         protected Void call() {
21             try {
22                 serverSocket = new ServerSocket(54321);
23                 while (true) {
24                     final Socket clientSocket = serverSocket.accept();
25                     // Accetta una nuova connessione client
26                     new Thread(() -> handleClientSocket(clientSocket)).
27                     start(); // Gestisci ogni connessione client in un nuovo thread
28                 }
29             }
30         }
31     };
32 }

```

```

27         } catch (IOException e) {
28             e.printStackTrace();
29         }
30         return null;
31     }
32 };
33
34 Thread serverThread = new Thread(serverTask);
35 serverThread.setDaemon(true);
36 serverThread.start();
37 }
38
39
40
41
42
43
44
45 private void handleClientSocket(Socket clientSocket) {
46     try {
47         System.out.print("prova");
48         BufferedReader in = new BufferedReader(new InputStreamReader(
49 clientSocket.getInputStream()));
50         StringBuilder messaggioCompleto = new StringBuilder();
51         String linea;
52         while ((linea = in.readLine()) != null) {
53             if (linea.equals("——FINE——")) {
54                 break; // Interrompe la lettura quando trova il
55 delimitatore
56             }
57             messaggioCompleto.append(linea).append("\n");
58         }
59         String messaggioDalServer = messaggioCompleto.toString();
60         System.out.println("Messaggio ricevuto: " + messaggioDalServer);
61
62         Platform.runLater(() -> alertPrenotazione(messaggioDalServer,
63 clientSocket));
64     } catch (IOException e) {
65         System.err.println("Si verificato un problema con la socket
66 client: " + e.getMessage());
67         e.printStackTrace();
68     }
69 }

```

3.1.3 Invio Coordinate

```

1 package com.example.medtaxi.classi;
2
3 import com.example.medtaxi.design_patterns.singleton.Database;
4 import com.google.maps.GeoApiContext;
5 import com.google.maps.GeocodingApi;
6 import com.google.maps.errors.ApiException;
7 import com.google.maps.model.GeocodingResult;
8 import com.google.maps.model.LatLng;
9
10 import java.io.IOException;
11 import java.net.DatagramPacket;
12 import java.net.DatagramSocket;

```

```

13 import java.net.InetAddress;
14 import java.net.UnknownHostException;
15 import java.util.List;
16
17 public class ServerUDP {
18     DatagramSocket datagramSocket;
19     private List<LatLng> routePoints;
20     private int currentStep = 0;
21
22     public ServerUDP(String codice_track) throws Exception {
23         datagramSocket = new DatagramSocket(5001);
24
25         // Ottieni gli indirizzi di partenza e arrivo dal database
26         String indirizzoPartenza = Database.getInstance().
getIndirizzoPartenzaByCodeTrack(codice_track);
27         String indirizzoArrivo = Database.getInstance().
getIndirizzoArrivoByCodeTrack(codice_track);
28
29         // Ottieni le coordinate di latitudine e longitudine per l'origine
e la destinazione
30         LatLng origin = geocodeAddress(indirizzoPartenza);
31         LatLng destination = geocodeAddress(indirizzoArrivo);
32
33         routePoints = new RouteCalculator().getRoutePoints(origin ,
destination);
34     }
35
36
37     public void connetti() throws IOException {
38         new Thread(() -> {
39             while (currentStep < routePoints.size()) {
40                 byte[] buffer = ottieniPosizioneAmbulanza();
41                 InetAddress address = null;
42                 try {
43                     address = InetAddress.getByName("localhost");
44                 } catch (UnknownHostException e) {
45                     throw new RuntimeException(e);
46                 }
47                 DatagramPacket packet = new DatagramPacket(buffer , buffer.
length , address , 5002);
48                 try {
49                     datagramSocket.send(packet);
50                     // invio ogni 5 sec
51                     Thread.sleep(3000);
52                     currentStep++;
53                 } catch (IOException | InterruptedException e) {
54                     e.printStackTrace();
55                 }
56             }
57         }).start();
58     }
59
60     private byte[] ottieniPosizioneAmbulanza() {
61         if (currentStep < routePoints.size()) {
62             LatLng currentPoint = routePoints.get(currentStep);
63             String pos = currentPoint.lat + "," + currentPoint.lng;
64             return pos.getBytes();
65         } else {

```

```

66         return "end".getBytes();
67     }
68 }
69
70 public LatLng geocodeAddress(String address) throws ApiException,
InterruptedException, IOException {
71
72     String apiKey = "AIzaSyB-7VoL5g7xLox1cZA9KVYEAu6l34FZ-tQ";
73
74     GeoApiContext context = new GeoApiContext.Builder()
75         .apiKey(apiKey)
76         .build();
77
78     GeocodingResult[] results = GeocodingApi.newRequest(context)
79         .address(address)
80         .await();
81
82     if (results.length > 0) {
83         return new LatLng(results[0].geometry.location.lat, results[0].
84         geometry.location.lng);
85     }
86     return null;
87 }
88 }
89
90
91
92
93
94
95 package com.example.medtaxi.classi;
96
97 import com.example.medtaxi.design_patterns.observer.
98     CoordinateUpdateListener;
99 import com.example.medtaxi.design_patterns.observer.Subject;
100
101 import java.io.IOException;
102 import java.net.DatagramPacket;
103 import java.net.DatagramSocket;
104 import java.net.SocketException;
105 import java.util.ArrayList;
106 import java.util.List;
107
108 public class UtenteUDP implements Subject {
109     private DatagramSocket socket;
110     private byte[] buffer = new byte[256];
111     private List<CoordinateUpdateListener> observers = new ArrayList<>();
112
113     public UtenteUDP(int porta) throws SocketException {
114         this.socket = new DatagramSocket(porta);
115     }
116
117     @Override
118     public void addObserver(CoordinateUpdateListener o) {
119         if (!observers.contains(o)) {
120             observers.add(o);
121         }
122     }

```

```

121     }
122
123     @Override
124     public void removeObserver(CoordinateUpdateListener o) {
125         observers.remove(o);
126     }
127
128     @Override
129     public void notifyObservers(String coordinate) {
130         for (CoordinateUpdateListener observer : observers) {
131             observer.onCoordinateUpdate(coordinate);
132         }
133     }
134
135     public void ascolta() {
136         new Thread(() -> {
137             while (true) {
138                 try {
139                     DatagramPacket packet = new DatagramPacket(buffer,
140 buffer.length);
141                     socket.receive(packet);
142                     String ricevuto = new String(packet.getData(), 0,
143 packet.getLength());
144                     notifyObservers(ricevuto); // Notifica gli osservatori
145                 } catch (IOException e) {
146                     e.printStackTrace();
147                 }
148             }
149         }).start();
150     }

```

Capitolo 4

Manuale utente con le istruzioni su compilazione ed esecuzione

Questo manuale fornisce le istruzioni per la compilazione ed esecuzione del progetto MedTaxi e del suo server associato. Seguire attentamente le istruzioni per configurare e avviare correttamente l'applicazione.

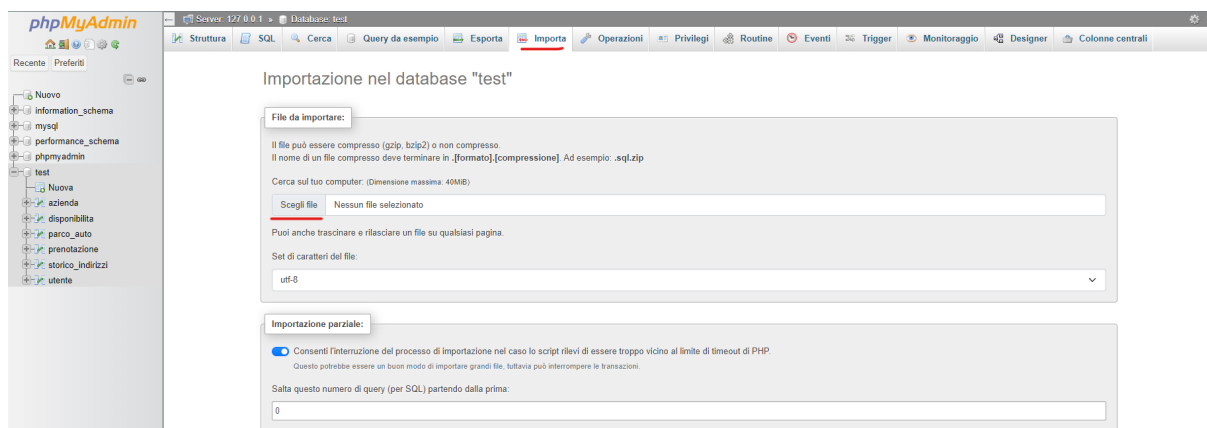
4.1 Prerequisiti

Prima di procedere con l'installazione e l'esecuzione del software, assicurarsi di avere installato sul proprio sistema:

- XAMPP: scaricabile da <https://www.apachefriends.org/it/index.html>. Utilizzato per gestire il database MySQL necessario per il progetto.
- IntelliJ IDEA: per l'importazione e l'esecuzione dei progetti Java.

4.2 Configurazione del Database

1. Avviare XAMPP e avviare i moduli Apache e MySQL.
2. Accedere a phpMyAdmin tramite <http://localhost/phpmyadmin/> e importare il file del database fornito con il progetto.



4.3 Download e Configurazione dei Progetti

I progetti MedTaxi e ServerMedTaxi sono disponibili su GitHub. Seguire i link per scaricare entrambi i progetti.

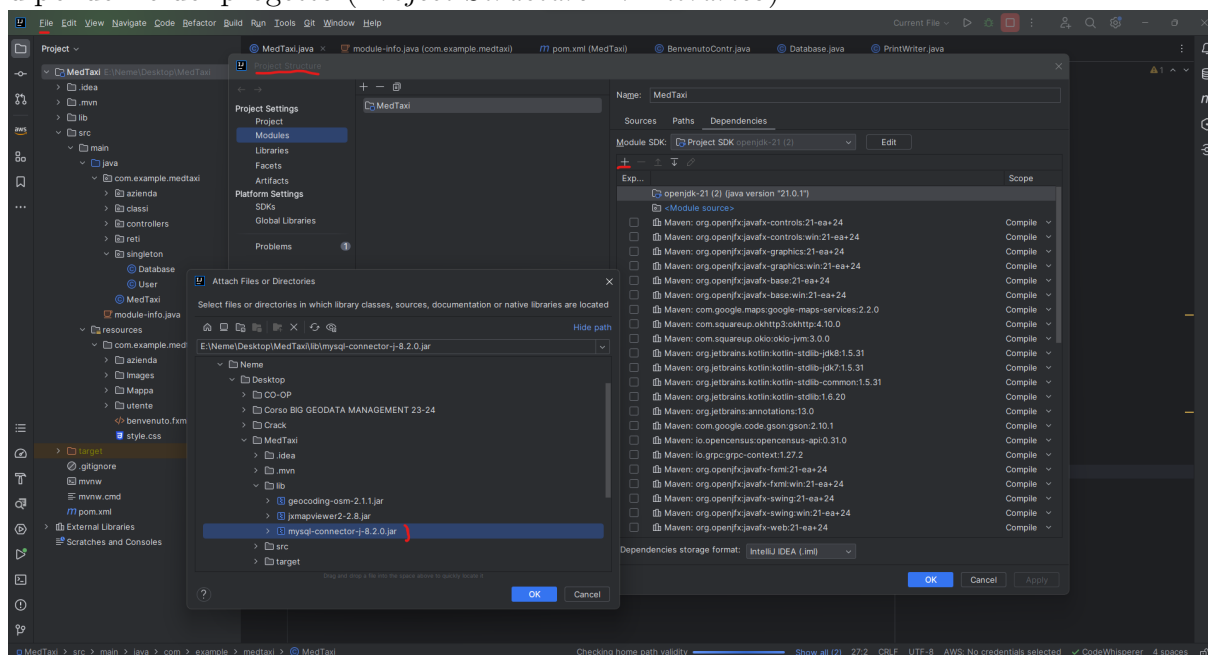
MedTaxi <https://github.com/TheNemesis47/MedTaxi>

Se si è su Windows, effettuare il checkout del branch "Master". Se si è su MacOS effettuare il checkout del branch "Mac" a causa di problemi di compatibilità con la WebView di JavaFX.

Server <https://github.com/TheNemesis47/ServerMedTaxi>

4.3.1 ServerMedTaxi

1. Importare il progetto in IntelliJ IDEA come progetto Maven.
2. Verificare che il file JAR nella cartella lib sia importato correttamente nelle dipendenze del progetto (*Project Structure* → *Libraries*).



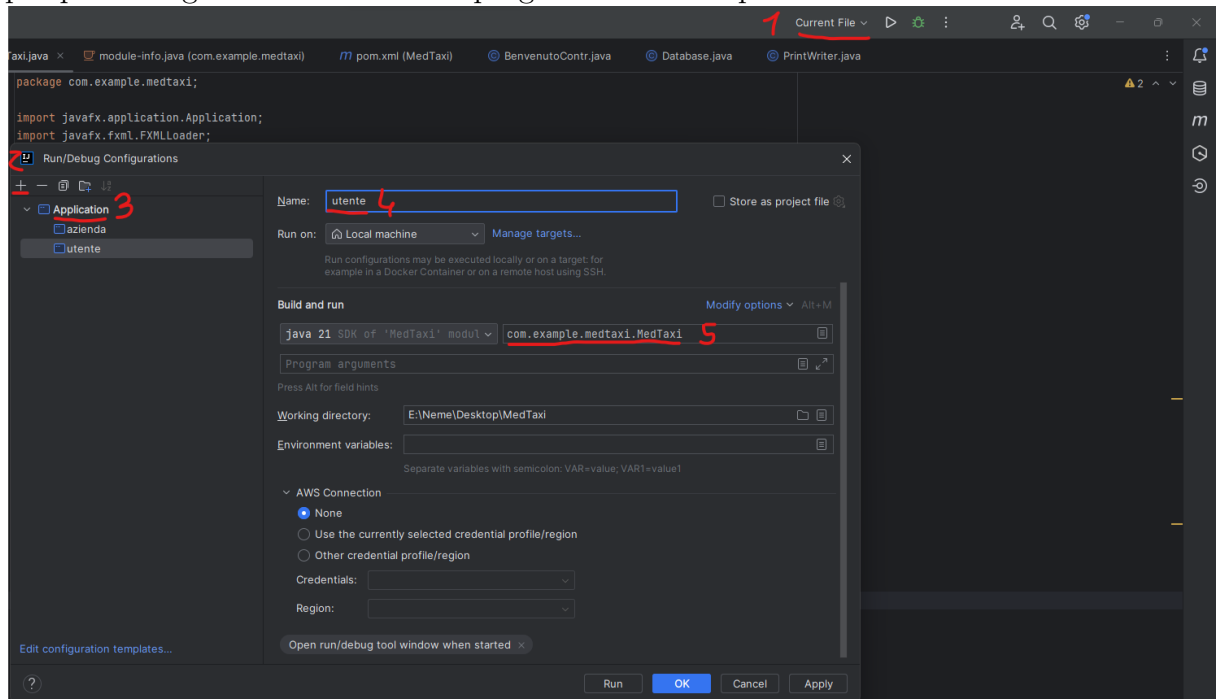
3. Avviare il server eseguendo la classe principale. Utilizzare i comandi nel terminale per gestire il server:

```
Inserire:  
[1] Per avviare il server  
[2] Per fermare il server  
[3] Per inserire azienda
```

4.3.2 MedTaxi

1. Importare anche questo progetto in IntelliJ IDEA come progetto Maven.
2. Assicurarsi che il file JAR nella cartella lib sia stato aggiunto correttamente alle dipendenze.

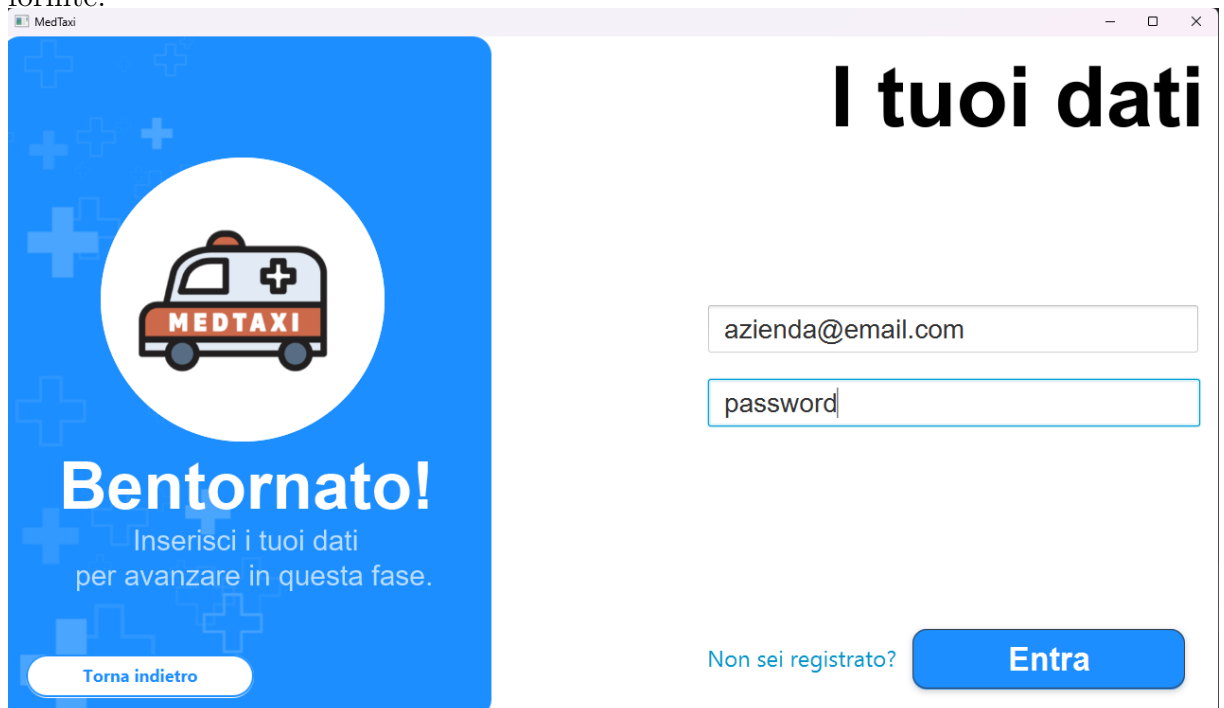
3. Configurare una *Application Run Configuration* con la classe principale `MedTaxi` per poter eseguire due istanze del programma contemporaneamente.



4.4 Esecuzione

4.4.1 Azienda

1. Mantenere aperta l'istanza dell'applicazione MedTaxi dedicata all'azienda nella schermata home nel momento della prenotazione da parte dell'utente.
2. L'accesso è riservato alle aziende registrate. Effettuare il login con le credenziali fornite.



Modifica disponibilità

MedTaxi

MODIFICA DISPONIBILITÀ

Data	Disponibilità Mattina	Disponibilità Sera
2024-02-01	1	1
2024-02-02	1	1
2024-02-03	1	1
2024-02-04	1	1
2024-02-05	1	1
2024-02-06	1	1
2024-02-07	1	1
2024-02-08	1	1
2024-02-09	1	1
2024-02-10	1	1
2024-02-11	1	1
2024-02-12	1	1
2024-02-13	1	1
2024-02-14	1	1
2024-02-15	1	1

15/02/2024

(+) Aggiungi disponibilità

(-) Rimuovi disponibilità

Torna indietro

A sinistra è possibile visualizzare la schermata con tutte le disponibilità dell'azienda fino al 31 dicembre dell'anno corrente. E' possibile selezionare una data e aggiungere o rimuovere disponibilità in quella data specifica.

Parco auto



Targhe già inserite

aa111aa

Inserisci ambulanza

Rimuovi ambulanza

Torna indietro



Inserisci la targa dell'ambulanza

ch011vr

Torna indietro

Inserisci

tuale e successive. Le storiche potranno essere visualizzate in "Storico prenotazioni"

[illegible]

Traccia ambulanza

Inserisci il codice di track per avviare il tracciamento.



The screenshot shows a web browser window with the title bar 'MedTaxi'. The main heading is 'Tracciamento' in large white text on a blue background. Below this, the instruction 'Inserisci il codice di track' is displayed in bold black text. A text input field contains the code 'IZ2PC'. At the bottom, there are two buttons: 'Torna indietro' (light blue) and 'Traccia' (dark blue).

MedTaxi

Tracciamento

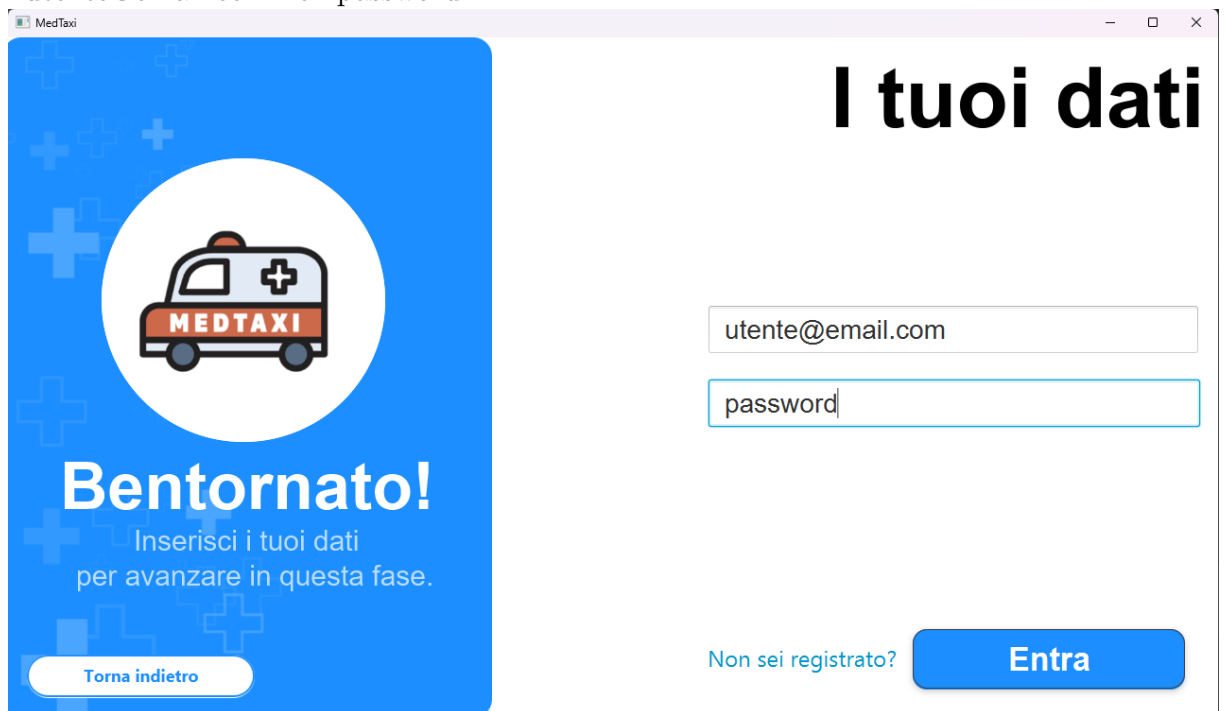
Inserisci il codice di track

Torna indietro Traccia

4.4.2 Utente

1. Eseguire l'istanza dell'applicazione MedTaxi per l'utente.

2. Effettuare il login o registrarsi con i propri dati, in caso di prova usare "utente@email.com" e "password".



The image shows a web browser window titled "MedTaxi". On the left, there is a blue sidebar with a white circle containing a red and white ambulance icon with "MEDTAXI" written on its side. Below the icon, the text "Bentornato!" is displayed in large white font, followed by "Inserisci i tuoi dati per avanzare in questa fase." in smaller white font. At the bottom of the sidebar is a white button with the text "Torna indietro". On the right, the main area has the heading "I tuoi dati" in large black font. Below it are two input fields: the first contains "utente@email.com" and the second contains "password". At the bottom right, there is a link "Non sei registrato?" and a blue button labeled "Entra".

3. Seguire le istruzioni nell'interfaccia utente per prenotare un trasporto, annullare una prenotazione, visualizzare lo storico degli indirizzi o tracciare l'ambulanza.



The image shows a web browser window titled "MedTaxi". On the left, there is a blue sidebar with a white circle containing a red and white ambulance icon with "MEDTAXI" written on its side. Below the icon, the text "Ciao Raffaele" is displayed in large white font, followed by "cosa desideri fare?" in smaller white font. At the bottom of the sidebar is a white button with the text "Logout". On the right, the main area has a list of four white buttons with blue text: "Prenota", "Traccia ambulanza", "Annulla prenotazione", and "Storico indirizzi".

MedTaxi

-□×

PRENOTA

raffaele

iommelli

via genova 8, frattamaggi

ospedale monaldi, napoli

20/02/2024

10:00

3921342840

Torna indietro

Seleziona ambulanza

MedTaxi

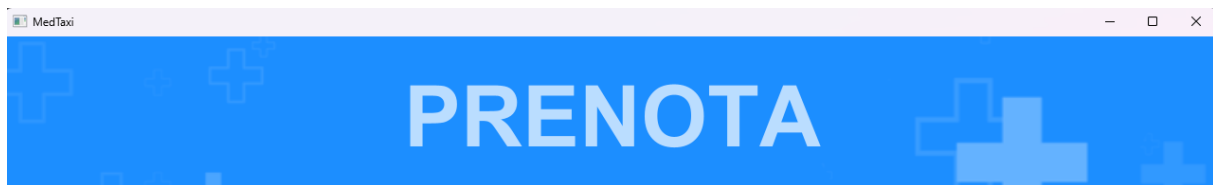
-□×

AMBULANZE DISPONIBILI

Azienda: San Leonardo SRL - P.IVA: 12435678901 - Costo per Km: 1,50€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 8,86 km - Costo totale tragitto: 11,79€

Torna indietro

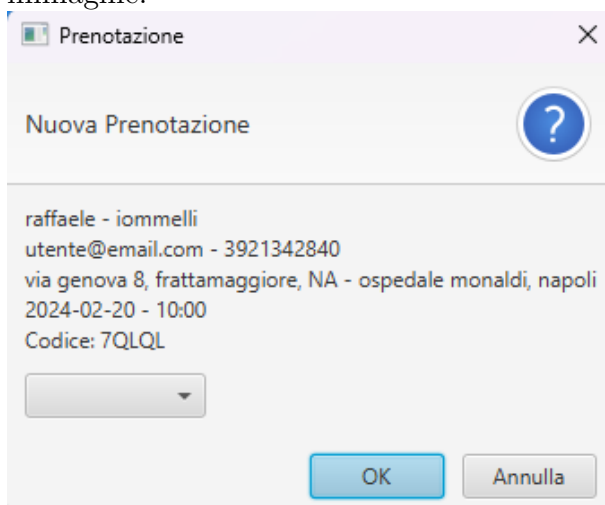
Prenota trasferimento



**Il codice dell'ambulanza è
7QLQL**

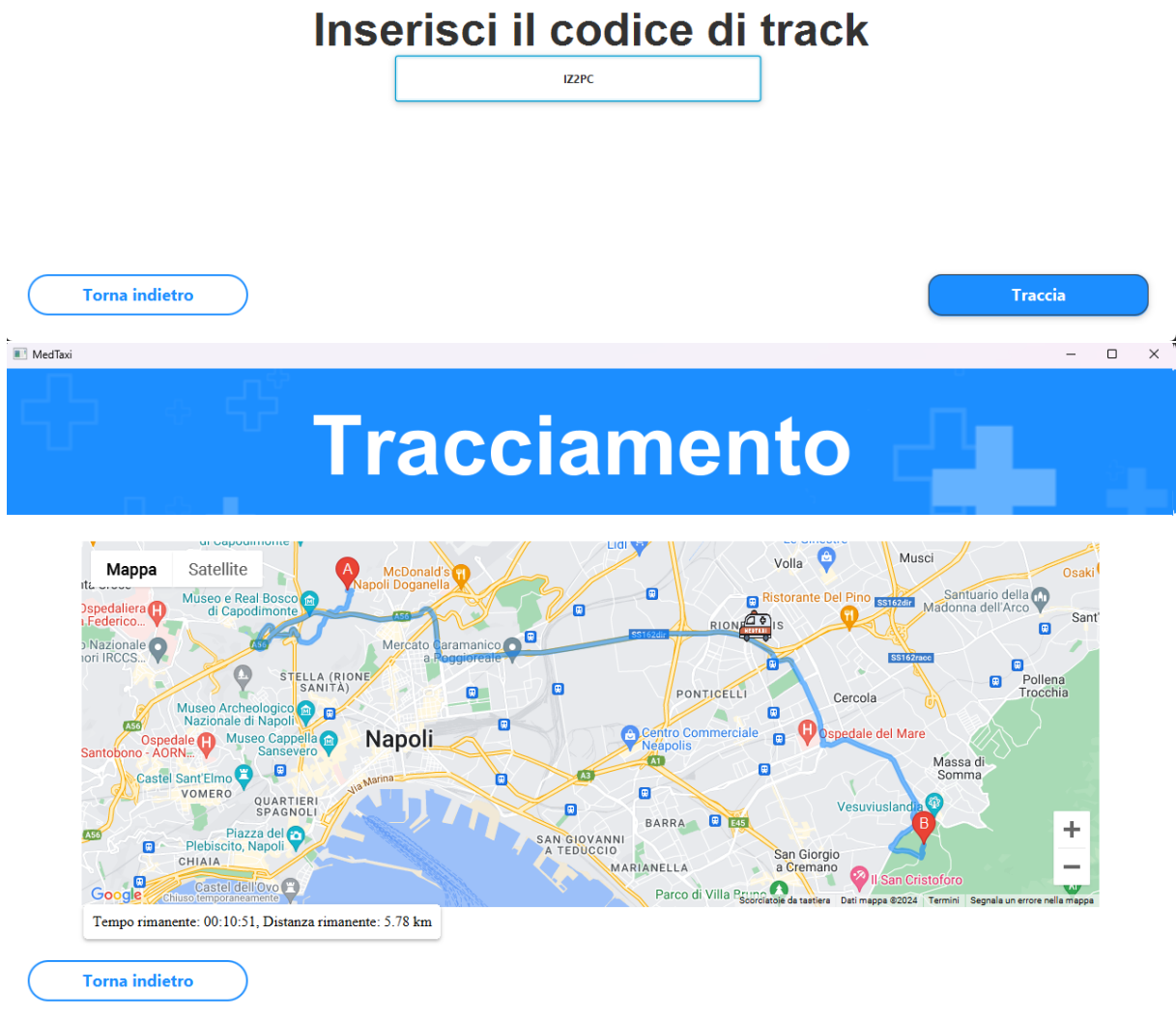
Torna alla Home

Inserisci tutti i dati e clicca su "Seleziona ambulanza". Verrà mostrata una lista di ambulanze. Scegli quella più conveniente per te e clicca su "Prenota". Verà fornito un codice di tracking. Quando l'utente effettua una prenotazione, all'azienda uscirà questa immagine.



L'azienda quindi assegnerà alla prenotazione una targa, altrimenti non verrà registrata nel database.

Tracciamento



Inserisci il codice di track fornito all'atto della prenotazione e clicca su "Traccia". Ogni 5 secondi verrà effettuato un movimento dell'icona dell'ambulanza e verranno indicate distanza e tempo rimanenti. Se non viene effettuato alcun movimento dell'ambulanza, effettuare la stessa operazione prima dal client Azienda.

Annula prenotazione

[illegible]

Seleziona sulla prenotazione che vuoi annullare e clicca su "Annulla prenotazione".

Storico indirizzi

In questa sezione verranno visualizzati tutti gli storici degli indirizzi delle prenotazioni effettuate dall'utente loggato.