



Università degli Studi di Napoli "Parthenope"

DIPARTIMENTO DI SCIENZE E TECNOLOGIE
CORSO DI PROGRAMMAZIONE 3 E LABORATORIO DI PROGRAMMAZIONE 3

MedTaxi

Arenella Samuel 012400/2529
Iommelli Raffaele 012400/2491

Progetto realizzato per lo svolgimento dell'esame di Programmazione 3 e
laboratorio di programmazione 3

Anno Accademico 2023/24

Indice

1	Introduzione	3
1.1	Scopo del Progetto	3
1.2	Funzionalità Principali per gli Utenti	3
1.3	Funzionalità Principali per le Aziende di Servizi Sanitari	3
2	Aspetti di Reti di Calcolatori	4
2.1	Architettura del Sistema	4
2.2	Comunicazione tra Componenti	4
2.3	Tracciamento	4
3	Integrazione con Ingegneria del Software e Interazione Uomo-Macchina	5
3.1	Material Design	5
3.2	Sviluppo Software Robusto	5
3.3	Interazione Intuitiva	5
3.4	Test e Validazione	5
4	Proposta di realizzazione	6
4.1	Modalità di sviluppo	6
5	Teoria	7
5.1	Breve descrizione dei requisiti del progetto	7
5.2	Design Pattern Utilizzati	7
5.2.1	Singleton	8
5.2.2	Factory Method	8
5.2.3	Observer	9
5.2.4	Command	9
5.2.5	State	10
5.3	Diagramma UML delle classi	12
6	Pratica	17
6.1	Pattern utilizzati	17
6.1.1	Singleton	17
6.1.2	Factory Method	17
6.1.3	Observer	19
6.1.4	Command	20
6.1.5	State	21
7	Conclusioni	25

8	Manuale utente con le istruzioni su compilazione ed esecuzione	26
8.1	Prerequisiti	26
8.2	Configurazione del Database	26
8.3	Download e Configurazione dei Progetti	27
8.3.1	ServerMedTaxi	27
8.3.2	MedTaxi	27
8.4	Esecuzione	28
8.4.1	Azienda	28
8.4.2	Utente	32

Capitolo 1

Introduzione

1.1 Scopo del Progetto

MedTaxi ha l'obiettivo di sviluppare un sistema completo per la prenotazione e il tracciamento di ambulanze private. Questo sistema sarà accessibile agli utenti per prenotare un'ambulanza e monitorarne la posizione durante il tragitto verso l'utente. Inoltre, verrà fornito un pannello amministrativo per le aziende di servizi sanitari al fine di gestire il proprio parco auto, aggiornare la disponibilità e monitorare le prenotazioni.

1.2 Funzionalità Principali per gli Utenti

- Prenotazione di Ambulanze
- Visualizzazione di Storico
- Annullare Prenotazioni
- Tracciamento dell'Ambulanza

1.3 Funzionalità Principali per le Aziende di Servizi Sanitari

- Gestione del Parco Auto
- Aggiunta/Rimozione Disponibilità
- Visualizzazione delle Prenotazioni
- Comunicazione Continua con i Clienti

Capitolo 2

Aspetti di Reti di Calcolatori

2.1 Architettura del Sistema

Il sistema utilizzerà un server centrale che fungerà da punto di coordinamento tra utenti e aziende in fase di prenotazione, il protocollo di trasporto alla base di quest'architettura è il TCP utilizzato dalle "Socket"

-Le aziende private verranno registrate da un operatore che ha accesso al server centrale a seguito di un colloquio con MedTaxi (per chiarire tariffe, autenticare azienda ecc...), gli utenti si registreranno direttamente all'interno del client.

2.2 Comunicazione tra Componenti

Ogni richiesta di prenotazione sarà mandata al server centrale che dedicherà un thread per la i-esima richiesta, raccolti i dati risponderà al client utente con una lista di aziende disponibili.

Una volta che l'utente seleziona un'azienda, comunica la sua scelta al server centrale. Quest'ultimo agisce quindi come client, inviando tutte le informazioni pertinenti all'azienda scelta, che in questo contesto funge da server perché attende le informazioni. Se l'azienda conferma la prenotazione, il server centrale procederà con la registrazione della prenotazione nel database.

2.3 Tracciamento

Il tracciamento avviene in modo autonomo senza il coinvolgimento del server centrale. Il protocollo alla base di questo processo è UDP (User Datagram Protocol), realizzata attraverso l'utilizzo di DatagramSocket. Questo meccanismo permette la trasmissione di datagrammi.

La peculiarità di questo sistema è la sua capacità di operare in tempo reale, inviando aggiornamenti sulla posizione ogni 3 secondi. Tale frequenza garantisce che le informazioni relative alla localizzazione dell'ambulanza siano costantemente aggiornate, permettendo un tracciamento preciso e affidabile del veicolo durante i suoi spostamenti.

Capitolo 3

Integrazione con Ingegneria del Software e Interazione Uomo-Macchina

3.1 Material Design

L'interfaccia utente del sistema seguirà i principi del Material Design per garantire un'esperienza utente coerente e piacevole.

3.2 Sviluppo Software Robusto

Il progetto metterà in evidenza principi di ingegneria del software per garantire un codice robusto e manutenibile.

3.3 Interazione Intuitiva

L'interfaccia utente sarà progettata per essere intuitiva e facile da usare sia per gli utenti che per le aziende.

3.4 Test e Validazione

Saranno condotti test rigorosi per garantire che il sistema funzioni correttamente e soddisfi i requisiti specificati.

Capitolo 4

Proposta di realizzazione

4.1 Modalità di sviluppo

É stato scelto come modalità di sviluppo un programma standalone con supporto grafico, utilizzando l'IDE IntelliJ del pacchetto software JetBrains e il programma grafico Scene Builder per generare file FXML e definire quindi l'interfaccia dell'applicazione separatamente dalla logica dell'applicazione(backend).

Per la realizzazione del database é stato utilizzato MySQL che fornisce un'interfaccia phpMyAdmin per la gestione.

Per l'utilizzo della mappa, le funzioni di geocoding(conversione indirizzo in lat e lon) e la realizzazione del percorso sono state utilizzate le api di Google Maps.

Capitolo 5

Teoria

5.1 Breve descrizione dei requisiti del progetto

Il nostro obiettivo principale è sviluppare un sistema completo per la prenotazione e il tracciamento di ambulanze private, che sia accessibile sia agli utenti che alle aziende di servizi sanitari.

I requisiti del progetto includono la possibilità per gli utenti di prenotare un'ambulanza attraverso un'interfaccia utente intuitiva. Gli utenti devono essere in grado di prenotare un'ambulanza, visualizzare il loro storico prenotazioni, annullare una prenotazione se necessario e monitorare l'ambulanza durante il trasporto.

D'altro canto, le aziende di servizi sanitari avranno accesso a un pannello amministrativo per gestire il proprio parco auto, aggiornare la disponibilità delle ambulanze e monitorare le prenotazioni effettuate dai clienti. Sarà fondamentale garantire una comunicazione continua tra l'azienda e il cliente quando un'ambulanza è disponibile per un'appuntamento imminente.

Inoltre, il progetto sarà integrato con il corso di reti di calcolatori, ingegneria del software e interazione uomo-macchina, seguendo le migliori pratiche e le più recenti linee guida di progettazione. L'interfaccia utente sarà progettata seguendo il Material Design per garantire un'esperienza utente coerente e piacevole.

5.2 Design Pattern Utilizzati

I design pattern, o modelli di progettazione, rappresentano delle soluzioni consolidate e riutilizzabili a problemi comuni nel design e nello sviluppo del software. L'adozione di questi pattern aiuta a rendere il codice più modulare, flessibile.

5.2.1 Singleton

Il design pattern Singleton assicura che una classe abbia una sola istanza in tutto il programma, fornendo un punto di accesso globale a tale istanza. Questo pattern è stato utilizzato per la creazione delle classi "database", "utente", "azienda".

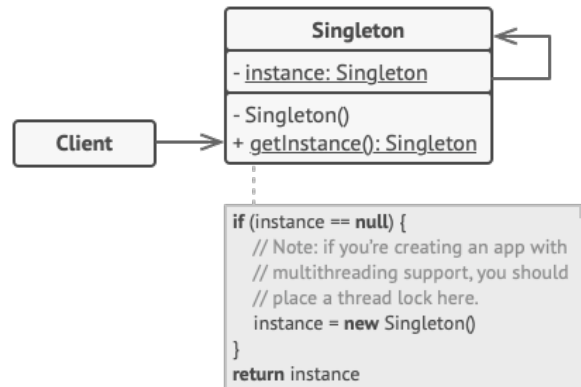


Figura 5.1: Esempio del pattern Singleton

5.2.2 Factory Method

Il Factory Method è un design pattern creazionale che fornisce un'interfaccia per creare oggetti in una superclasse, ma permette alle sottoclassi di alterare il tipo di oggetti che verranno creati. Questo pattern è stato utilizzato per la creazione della classe "prenotazione".

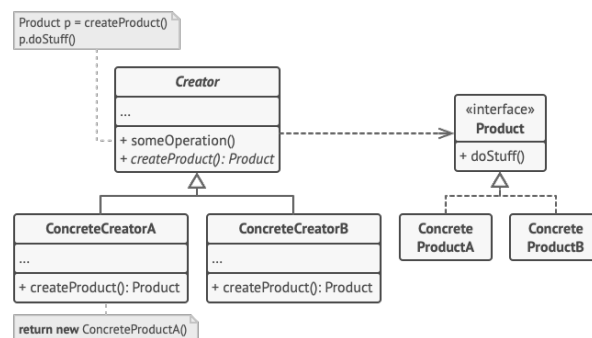


Figura 5.2: Esempio del pattern Factory Method

5.2.3 Observer

Il pattern Observer definisce una dipendenza uno-a-molti tra oggetti, in modo che quando un oggetto cambia stato, tutti i suoi dipendenti vengono notificati e aggiornati automaticamente. Questo pattern è stato utilizzato per le coordinate.

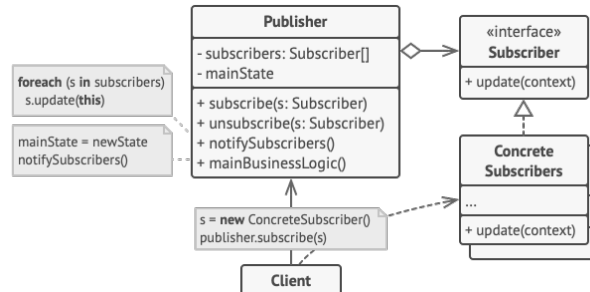


Figura 5.3: Esempio del pattern Observer

5.2.4 Command

Il Command è un pattern comportamentale che incapsula una richiesta come oggetto, consentendo di parametrizzare i clienti con diverse richieste, accodare le richieste, e implementare operazioni annullabili. Il pattern command è stato utilizzato per cambiare le scene.

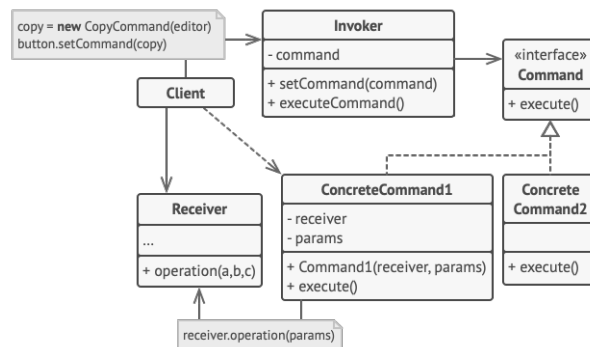


Figura 5.4: Esempio del pattern Command

5.2.5 State

Il design pattern State permette a un oggetto di modificare il suo comportamento quando il suo stato interno cambia. Questo si comporta come se l'oggetto cambiasse la sua classe. Il pattern State è stato utilizzato per per l'autenticazione del client come utente o azienda.

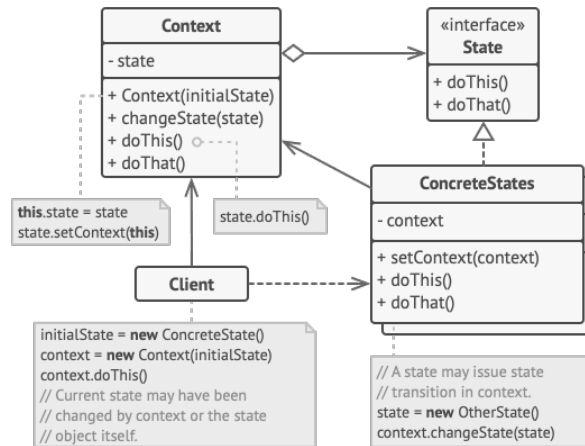
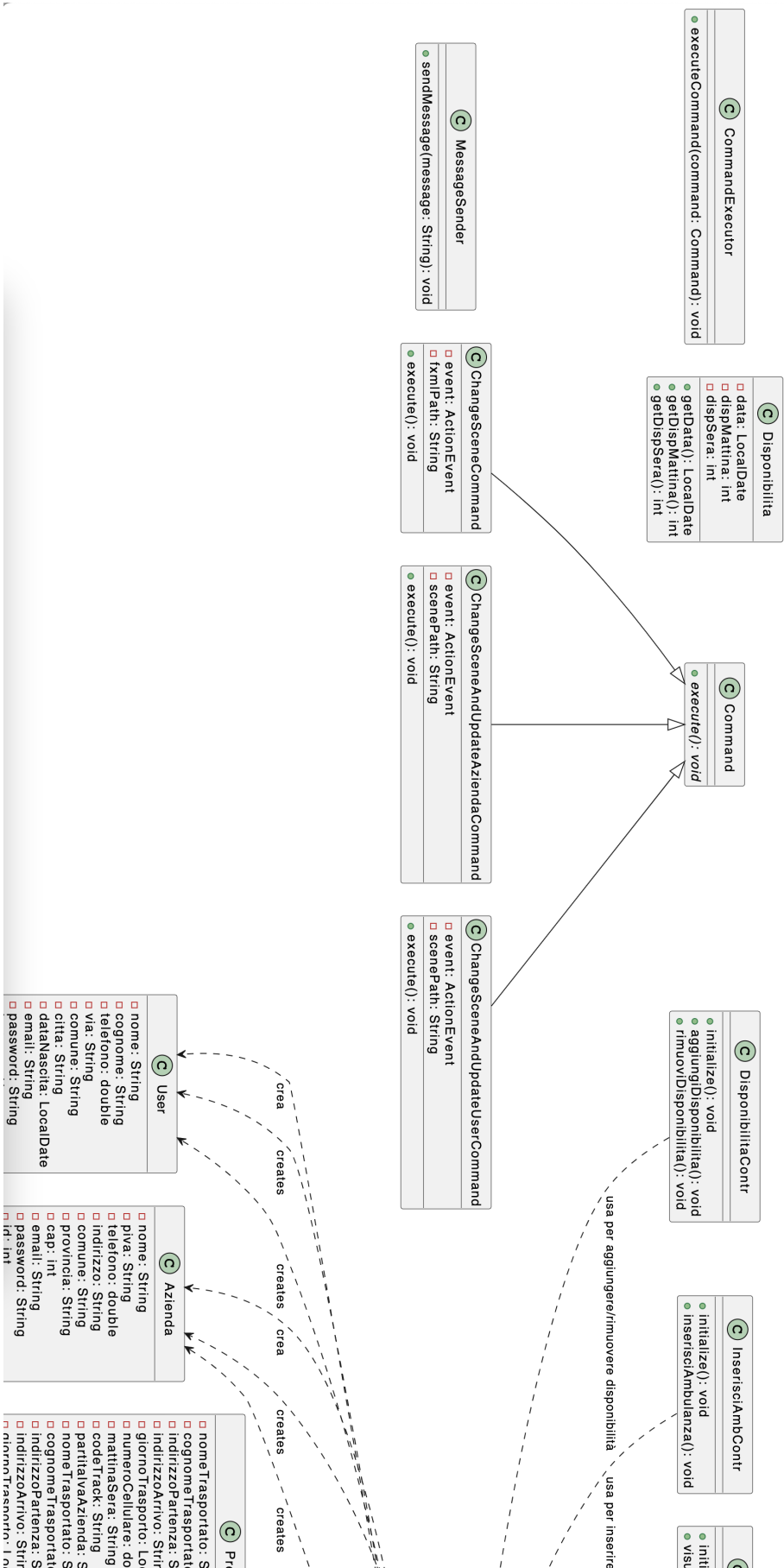
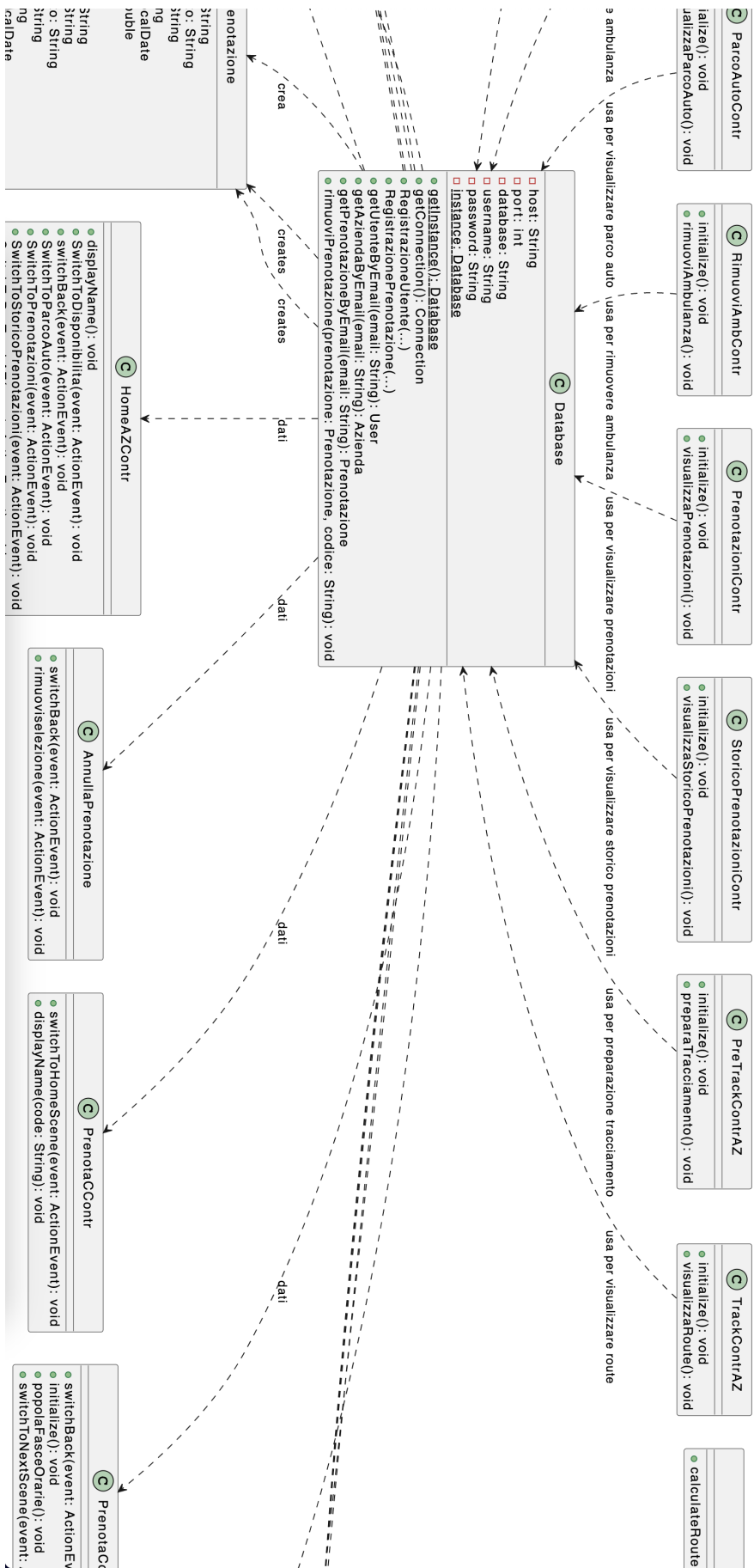
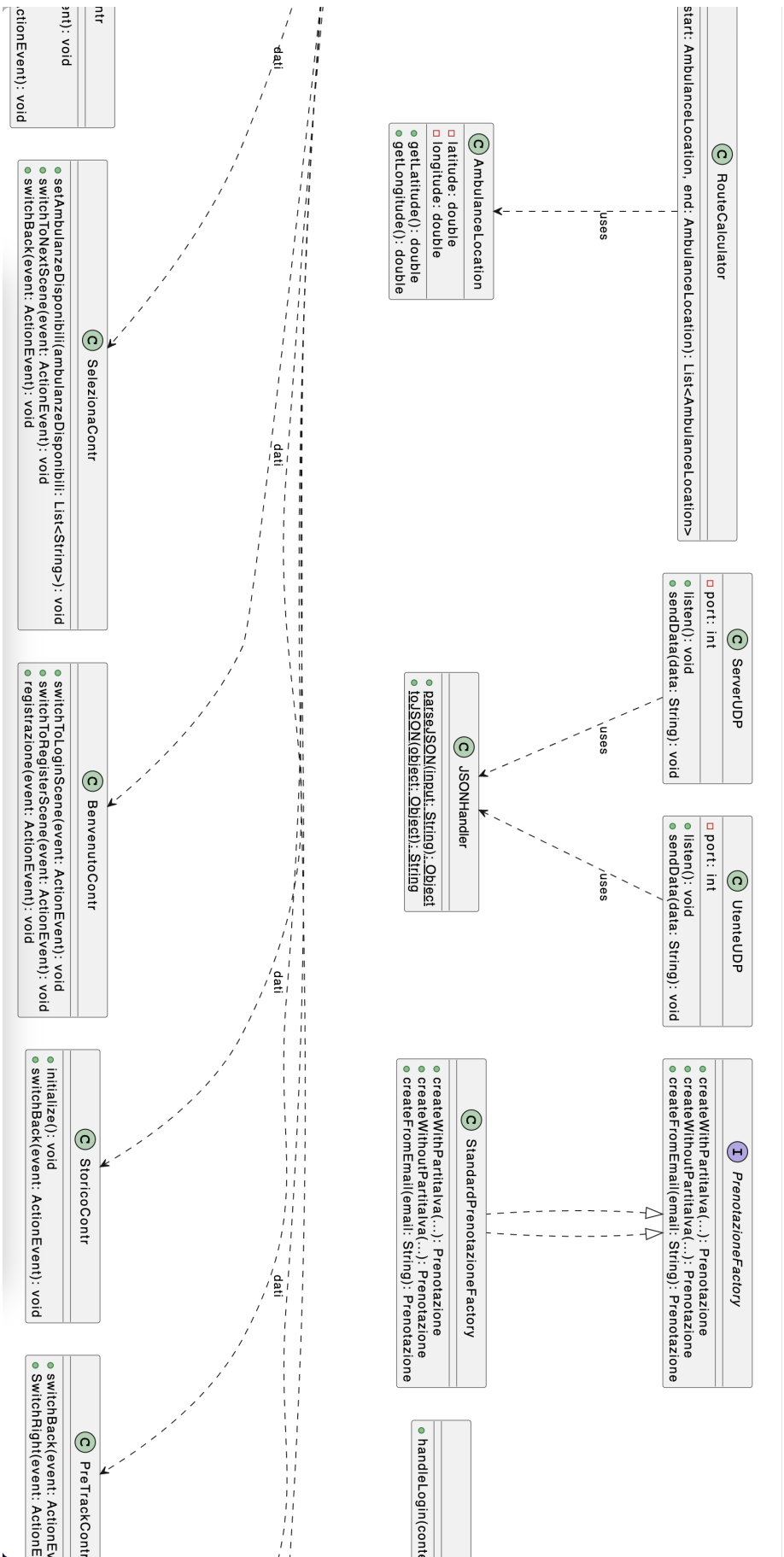


Figura 5.5: Esempio del pattern State

5.3 Diagramma UML delle classi









Come mostrato nella figura seguente, l'immagine è disponibile solo parzialmente a causa delle limitazioni di spazio. Tuttavia, è possibile visualizzare l'immagine completa cliccando sul seguente link.

Capitolo 6

Pratica

6.1 Pattern utilizzati

6.1.1 Singleton

```
1 //SINGLETON
2 private static Database instance;
3
4 //Costruttore.
5 private Database() {
6 }
7
8 //Metodo sincronizzato per ottenere l'istanza del Singleton da pi
9 Thread.
10 public static synchronized Database getInstance() {
11     if (instance == null) {
12         instance = new Database();
13     }
14     return instance;
15 }
16
17 //CLIENT
18 Database db = Database.getInstance();
```

6.1.2 Factory Method

```
1 //Interfaccia PRODUCT
2 public interface PrenotazioneFactory {
3     Prenotazione createWithPartitaIva(String nomeTrasportato, String
4     cognomeTrasportato, String indirizzoPartenza, String indirizzoArrivo,
5     LocalDate giornoTrasporto, double numeroCellulare, String mattinaSera,
6     String codeTrack, String pIva);
7
8     Prenotazione createWithoutPartitaIva(String nomeTrasportato, String
9     cognomeTrasportato, String indirizzoPartenza, String indirizzoArrivo,
10    LocalDate giornoTrasporto, double numeroCellulare, String mattinaSera,
11    String codeTrack);
12 }
```

```

9
10
11     Prenotazione createFromEmail(String email);
12 }
13
14
15 // CONCRETE PRODUCT
16 public class StandardPrenotazioneFactory implements PrenotazioneFactory {
17     // Metodo per creare una Prenotazione con la Partita IVA dell'azienda
18     @Override
19     public Prenotazione createWithPartitaIva(String nomeTrasportato, String
        cognomeTrasportato, String indirizzoPartenza, String indirizzoArrivo,
        LocalDate giornoTrasporto, double numeroCellulare, String mattinaSera,
        String codeTrack, String pIva) {
20         return new Prenotazione(nomeTrasportato, cognomeTrasportato,
            indirizzoPartenza, indirizzoArrivo, giornoTrasporto, numeroCellulare,
            mattinaSera, codeTrack, pIva);
21     }
22
23
24
25     // Metodo per creare una Prenotazione senza la Partita IVA dell'azienda
26     @Override
27     public Prenotazione createWithoutPartitaIva(String nomeTrasportato,
        String cognomeTrasportato, String indirizzoPartenza, String
        indirizzoArrivo, LocalDate giornoTrasporto, double numeroCellulare,
        String mattinaSera, String codeTrack) {
28         return new Prenotazione(nomeTrasportato, cognomeTrasportato,
            indirizzoPartenza, indirizzoArrivo, giornoTrasporto, numeroCellulare,
            mattinaSera, codeTrack);
29     }
30
31
32
33     // Metodo per creare una Prenotazione basata su un'email
34     @Override
35     public Prenotazione createFromEmail(String email) {
36         return new Prenotazione(email);
37     }
38 }
39
40
41 //CONCRETE CREATOR
42 public Prenotazione getPrenotazioneByEmail(String email) throws
    SQLException {
43     Connection connection = getConnection();
44     PrenotazioneFactory prenotazioneFactory = new
        StandardPrenotazioneFactory();
45     Prenotazione prenotazione = null;
46
47     String sql = "SELECT * FROM prenotazione WHERE email = ?";
48     try (PreparedStatement statement = connection.prepareStatement(sql)
        ) {
49         statement.setString(1, email);
50
51         try (ResultSet resultSet = statement.executeQuery()) {
52             if (resultSet.next()) {
53                 // Recupera i dati dal ResultSet

```

```

54         String nomeTrasportato = resultSet.getString("
nome_trasportato");
55         String cognomeTrasportato = resultSet.getString("
cognome_trasportato");
56         String indirizzoPartenza = resultSet.getString("
indirizzo_partenza");
57         String indirizzoArrivo = resultSet.getString("
indirizzo_arrivo");
58         LocalDate giornoTrasporto = resultSet.getDate("
giorno_trasporto").toLocalDate();
59         double numeroCellulare = resultSet.getDouble("
numero_cellulare");
60         String mattinaSera = resultSet.getString("mattina_sera"
);
61         String codeTrack = resultSet.getString("code_track");
62         String pIva = resultSet.getString("p_iva");
63
64         // Usa la factory method per creare l'oggetto
Prenotazione
65         prenotazione = prenotazioneFactory.createWithPartitaIva
(nomeTrasportato, cognomeTrasportato, indirizzoPartenza, indirizzoArrivo
, giornoTrasporto, numeroCellulare, mattinaSera, codeTrack, pIva);
66     }
67 }
68 } finally {
69     connection.close();
70 }
71
72     return prenotazione;
73 }
74
75 // CREATOR
76 Prenotazione prenotazioneFromDB = Database.getInstance().
getPrenotazioneByEmail(email);

```

6.1.3 Observer

```

1 //OBSERVER
2 public interface CoordinateUpdateListener {
3     void onCoordinateUpdate(String coordinate);
4 }
5
6
7 //CONCRETE OBSERVABLE A     classe UtenteUDP
8 @Override
9     public void notifyObservers(String coordinate) {
10         for (CoordinateUpdateListener observer : observers) {
11             // Notifica ciascun osservatore chiamando il metodo appropriato
12             observer.onCoordinateUpdate(coordinate);
13         }
14     }
15
16
17 //CONCRETE OBSERVABLE classe trackContr
18     public void initialize() {
19         webEngine = mappa.getEngine();

```

```

20     webEngine.load(getClass().getResource("/com/example/medtaxi/Mappa/
Mappa.html").toExternalForm());
21     try {
22         UtenteUDP udpClient = new UtenteUDP(5002);
23         udpClient.addObserver(this);
24         udpClient.ascolta();
25     } catch (SocketException e) {
26         throw new RuntimeException(e);
27     }
28 }
29
30
31 //OBSERVABLE
32 public interface Subject {
33     void addObserver(CoordinateUpdateListener o);
34
35
36
37     void removeObserver(CoordinateUpdateListener o);
38
39
40
41     void notifyObservers(String coordinate);
42 }

```

6.1.4 Command

```

1 //RECIVER
2 public class CommandExecutor {
3     // Metodo per eseguire un comando
4     public static void executeCommand(Command command) {
5         try {
6             // Esegue il comando chiamando il metodo execute() sull'oggetto
Command
7             command.execute();
8         } catch (Exception e) {
9             // Gestisce eventuali eccezioni che possono verificarsi durante
l'esecuzione del comando
10             handleException(e);
11         }
12     }
13
14
15
16     // Metodo per gestire un'eccezione
17     private static void handleException(Exception e) {
18         // Stampa un messaggio di errore sulla console
19         System.err.println("Si verificato un errore: " + e.getMessage());
20
21         // Stampa lo stack trace dell'eccezione per scopi di debug
22         e.printStackTrace();
23     }
24
25 //CONCRETE COMMAND
26 public class ChangeSceneCommand implements Command {
27     private ActionEvent event;

```

```

28     private String fxmlPath;
29
30
31
32     // Costruttore della classe
33     public ChangeSceneCommand(ActionEvent event, String fxmlPath) {
34         this.event = event;
35         this.fxmlPath = fxmlPath;
36     }
37
38
39
40     // Metodo execute() definito dall'interfaccia Command
41     @Override
42     public void execute() throws IOException {
43         // Carica il file FXML della nuova scena
44         Parent root = FXMLLoader.load(getClass().getResource(fxmlPath));
45         // Ottiene la finestra attuale
46         Stage stage = (Stage) ((Node) event.getSource()).getScene().
getWindow();
47         // Crea una nuova scena con la radice caricata dal file FXML
48         Scene scene = new Scene(root);
49         // Imposta la nuova scena sulla finestra attuale
50         stage.setScene(scene);
51         // Mostra la nuova finestra
52         stage.show();
53     }
54 }
55
56
57 //COMMAND
58 public interface Command {
59     void execute() throws Exception;
60 }
61
62
63
64 //INVOKER     classe HomeContr
65 //Torna alla schermata di login
66 @FXML
67 public void switchBack(ActionEvent event) throws IOException {
68     User.getInstance().disconnect();
69     // Crea un oggetto ChangeSceneCommand per cambiare la scena a "
login.fxml"
70     Command command = new ChangeSceneCommand(event, "/com/example/
medtaxi/utente/registrazione_e_login/login.fxml");
71     // Esegue il comando
72     CommandExecutor.executeCommand(command);
73 }

```

6.1.5 State

```

1 //interfaccia STATE
2 public interface UserState {
3     void handleLogin(BenvenutoContr context, ActionEvent event, String
email, String password) throws IOException, SQLException;
4 }

```

```

5
6
7 //CONCRETE STATE
8 public class AziendaAutenticata implements UserState {
9
10
11
12     @Override
13     public void handleLogin(BenvenutoContr context, ActionEvent event,
14 String email, String password) {
15         try {
16             context.switchToAziendaHomeScene(event);
17         } catch (IOException e) {
18             e.printStackTrace();
19         }
20 }
21
22
23 //CONTEXT
24 public class UtenteNonAutenticato implements UserState {
25     private static final String CLIENTE_AZIENDALE = "2";
26     private Parent root;
27
28
29
30     @Override
31     public void handleLogin(BenvenutoContr context, ActionEvent event,
32 String emailValue, String passwordValue) throws IOException,
33 SQLException {
34         Database connectNow = Database.getInstance();
35         Connection connectDB = connectNow.getConnection();
36
37         String verifyType = "SELECT client_type FROM utente WHERE email = ?
38 AND psw = ?";
39         String verifyLogin = "SELECT count(1) FROM utente WHERE email = ?
40 AND psw = ?";
41         String fxmlPathHome = "/com/example/medtaxi/utente/home.fxml";
42         String fxmlPathAzienda = "/com/example/medtaxi/azienda/homeAz.fxml"
43 ;
44         String errorMessage = "Login errato, riprova.";
45
46         try (PreparedStatement typeStatement = connectDB.prepareStatement(
47 verifyType)) {
48             typeStatement.setString(1, emailValue);
49             typeStatement.setString(2, passwordValue);
50
51             try (ResultSet typeResult = typeStatement.executeQuery()) {
52                 if (typeResult.next()) {
53                     String tipou = typeResult.getString("client_type");
54
55                     try (PreparedStatement loginStatement = connectDB.
56 prepareStatement(verifyLogin)) {
57                         loginStatement.setString(1, emailValue);
58                         loginStatement.setString(2, passwordValue);
59
60                         try (ResultSet loginResult = loginStatement.
61 executeQuery()) {

```

```

54         if (loginResult.next() && loginResult.getInt(1)
55             == 1) {
56             context.showError(""); // Pulisce messaggi
57             di errore precedenti
58             FXMLLoader loader = new FXMLLoader();
59             loader.setLocation(getClass().getResource("
60             2".equals(tipou) ? fxmlPathAzienda : fxmlPathHome));
61             Parent root = loader.load();
62             if (CLIENTE_AZIENDALE.equals(tipou)) {
63                 Azienda.initInstanceWithEmail(
64                 emailValue);
65                 Azienda azienda = Azienda.getInstance()
66                 ;
67                 HomeAZContr homeAzContr = loader.
68                 getController();
69                 homeAzContr.displayName();
70                 homeAzContr.startServerTask();
71                 context.setState(new AziendaAutenticata
72                 ());
73             } else {
74                 User.initInstance(emailValue);
75                 User utente = User.getInstance();
76                 HomeContr homeController = loader.
77                 getController();
78                 homeController.displayName();
79                 context.setState(new UtenteAutenticato
80                 ());
81             }
82             Stage stage = (Stage) ((Node) event.
83             getSource()).getScene().getWindow();
84             Scene scene = new Scene(root);
85             stage.setScene(scene);
86             stage.show();
87             } else {
88                 context.showError(errorMessage);
89             }
90         }
91     }
92 }
93
94 //CLIENT classe benvenutoContr
95 public void loginButtonOnAction(ActionEvent event) {
96     String emailValue = remail.getText();
97     String passwordValue = rpsw.getText();
98     try {
99         this.currentState.handleLogin(this, event, emailValue,
100         passwordValue);
101     } catch (IOException | SQLException e) {

```



```
101         e.printStackTrace();
102         showError("Errore di sistema. Per favore, riprova più tardi.")
103     ;
104 }
```

Capitolo 7

Conclusioni

Durante lo sviluppo del sistema di prenotazione e tracciamento di ambulanze, abbiamo affrontato con successo diversi obiettivi chiave.

Abbiamo implementato con successo un'interfaccia utente intuitiva che consente agli utenti di prenotare un'ambulanza e monitorarne la posizione durante il trasporto. Le aziende di servizi sanitari hanno a disposizione un pannello amministrativo completo per gestire il proprio parco auto, la disponibilità delle ambulanze e le prenotazioni dei clienti.

Nel contesto del corso di reti di calcolatori, abbiamo sviluppato un sistema basato su una comunicazione client-server, garantendo una comunicazione efficace tra utenti e aziende private. La comunicazione continua tra l'azienda e il cliente durante il trasporto è stata implementata con successo.

Il progetto è stato integrato con successo con il corso di ingegneria del software e interazione uomo-macchina, seguendo le migliori pratiche di sviluppo software e l'approccio del Material Design per l'interfaccia utente.

In conclusione, MedTaxi rappresenta una soluzione completa per la prenotazione e il tracciamento di ambulanze, con una particolare attenzione sulla facilità d'uso per gli utenti e un efficace sistema di gestione per le aziende di servizi sanitari.

Capitolo 8

Manuale utente con le istruzioni su compilazione ed esecuzione

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

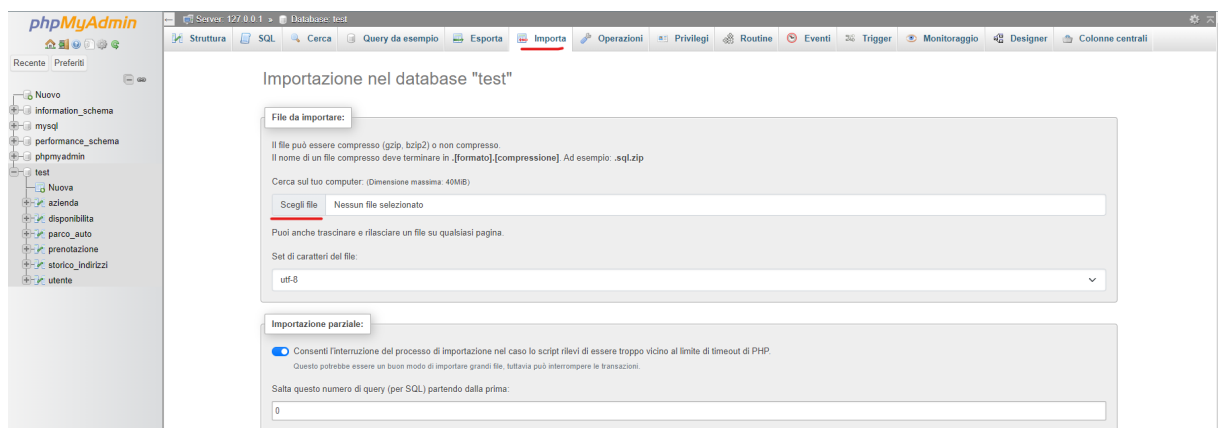
8.1 Prerequisiti

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.

8.2 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.



8.3 Download e Configurazione dei Progetti

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

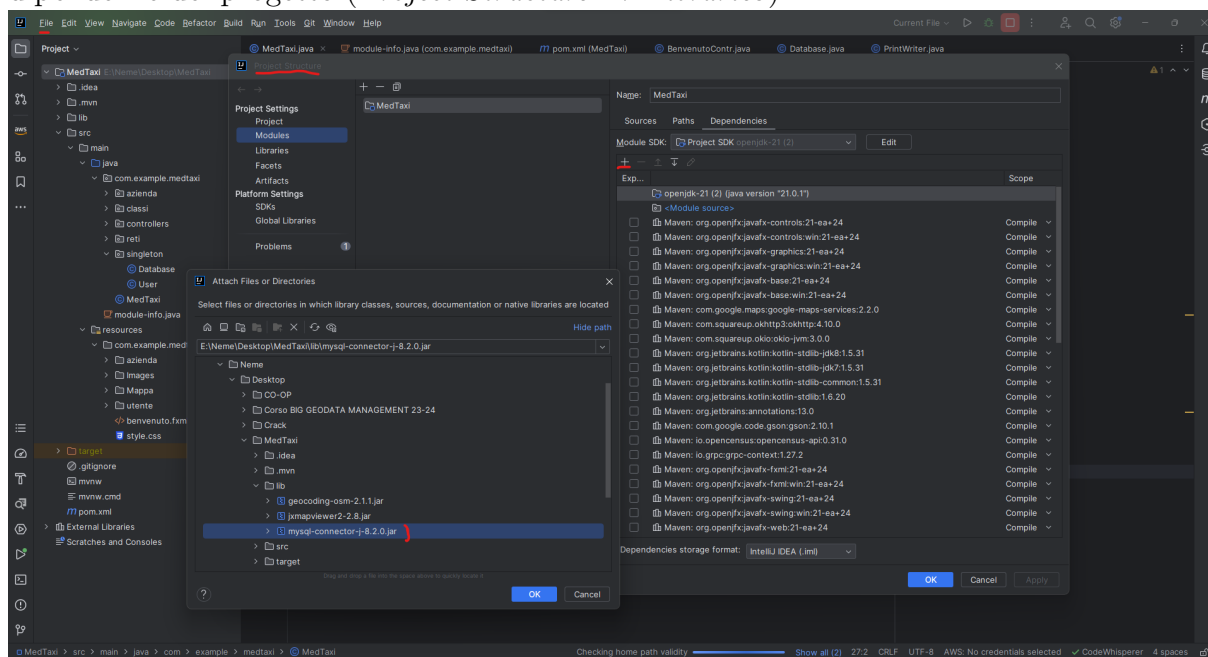
MedTaxi <https://github.com/TheNemesis47/MedTaxi>

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>

8.3.1 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*).



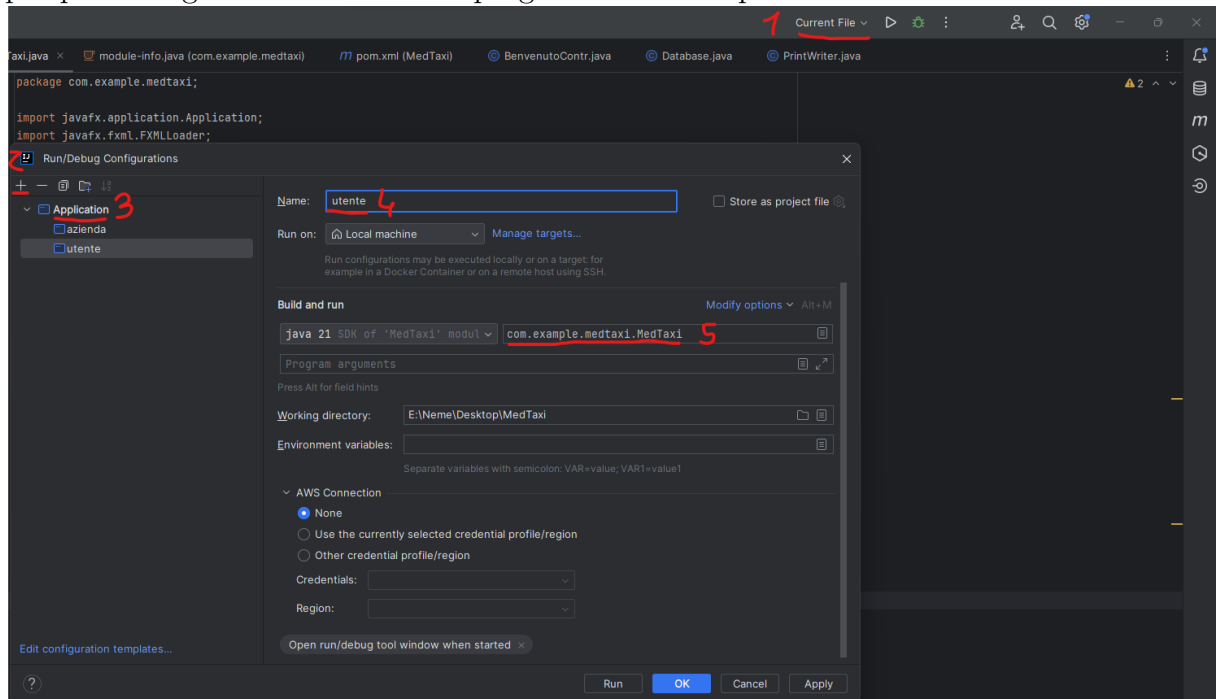
3. Avviare il server eseguendo la classe principale. Utilizzare i comandi nel terminale per gestire il server:

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

8.3.2 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.

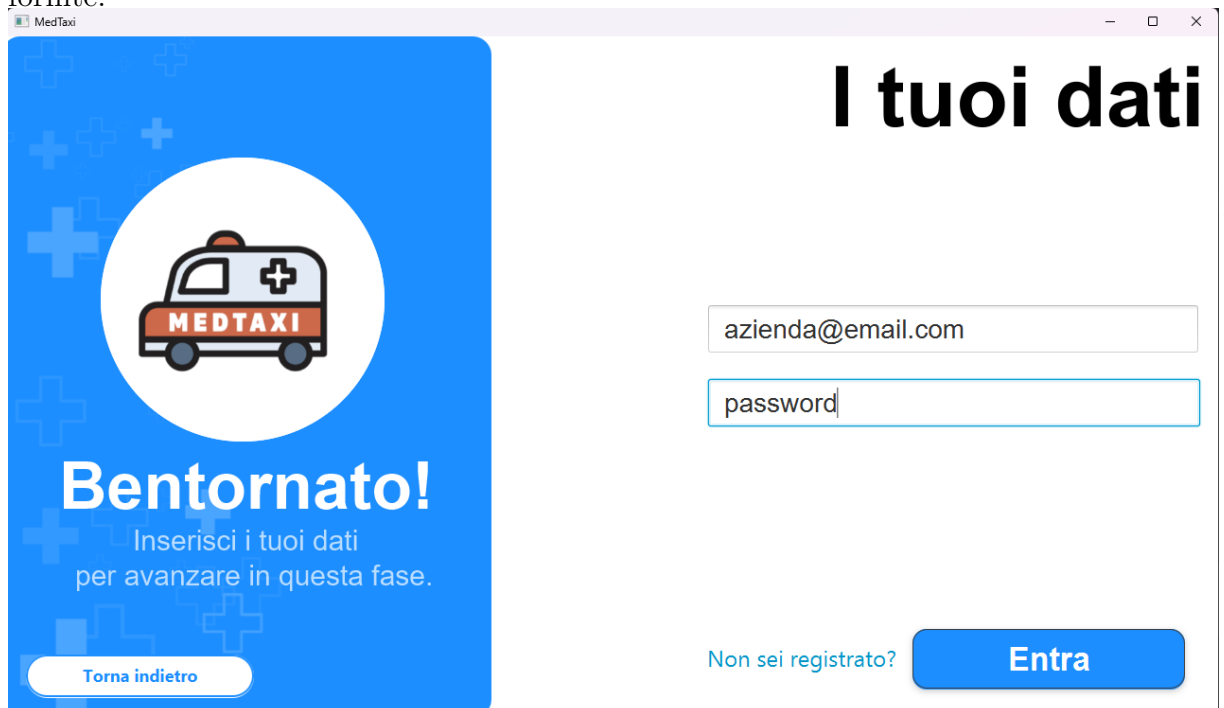
3. Configurare una *Application Run Configuration* con la classe principale `MedTaxi` per poter eseguire due istanze del programma contemporaneamente.



8.4 Esecuzione

8.4.1 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

MODIFICA DISPONIBILITÀ

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

16/02/2024

(+) Aggiungi disponibilità

(-) Rimuovi disponibilità

Rimosso!

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

AB123CD
EF456GH
IJ789KL
MNO12OP
QR345ST
UV678WX
YZ901AB
CD234EF
GH567IJ
KL890MN
aa000aa

Inserisci ambulanza

Rimuovi ambulanza

Torna indietro

INSERISCI AMBULANZA

Inserisci la targa dell'ambulanza

Torna indietro

Inserisci

RIMUOVI AMBULANZA

Seleziona la targa dell'ambulanza da rimuovere

AB123CD
EF456GH
IJ789KL
MN012OP
QR345ST
UV678WX
YZ901AB
CD234EF
GH567IJ
KL890MN
aa000aa

[Torna indietro](#)

[Rimuovi](#)

E' possibile inserire o rimuovere ambulanze e visualizzare a sinistra la lista delle targhe delle ambulanze già inserite. Nel caso dell'inserimento dell'ambulanza basterà inserire la targa dell'ambulanza. Nella rimozione invece, basterà selezionare la targa dell'ambulanza.

Prenotazioni

PRENOTAZIONI FUTURE

Nome	Cognome	Indirizzo Partenza	Indirizzo Arrivo	Giorno Trasporto	Numero Cellulare	Mattina/Sera	Code Track	
Raffaele	Iommelli	Via Genova 8 frattamaggiore	CTO Napoli	2024-02-22	3.92134284E9	04:00	70ZTV	
Raffaele	Iommelli	Via della resistenza, 74 villaricca	CTO Napoli	2024-03-01	3.92134284E9	03:00	2023S	
raffaele	iommelli	via genova 8, frattamaggiore, NA	ospedale monaldi, napoli	2024-02-20	3.92134284E9	10:00	7QLQL	

[Torna indietro](#)

Cliccando su "Prenota" è possibile visionare tutte le prenotazioni programmate per la giornata attuale e successive. Le storiche potranno essere visualizzate in "Storico prenotazioni"

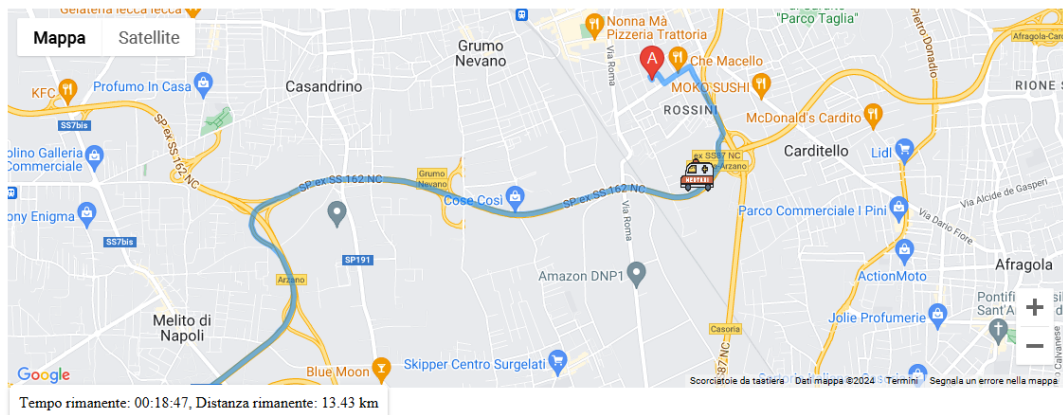
STORICO PRENOTAZIONI

Nome	Cognome	Indirizzo Partenza	Indirizzo Arrivo	Giorno Trasporto	Numero Cellulare	Mattina/Sera	Code Track
Nessun contenuto nella tabella							

[Torna indietro](#)

Traccia ambulanza

Inserisci il codice di track per avviare il tracciamento.




[Torna indietro](#)

8.4.2 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".



Bentornato!

Inserisci i tuoi dati
per avanzare in questa fase.

[Torna indietro](#)

I tuoi dati

[Non sei registrato?](#) [Entra](#)

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

MedTaxi

Ciao Raffaele

cosa desideri fare?

[Prenota](#)
[Traccia ambulanza](#)
[Annulla prenotazione](#)
[Storico indirizzi](#)
[Logout](#)



Prenotazione

MedTaxi

PRENOTA

raffaele

via genova 8, frattamaggi

20/02/2024

3921342840

iommelli

ospedale monaldi, napoli

10:00

Torna indietro

Seleziona ambulanza

AMBULANZE DISPONIBILI

Azienda: MediAmbulance S.r.l. - P.IVA: 12345678901 - Costo per Km: 10,50€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 10,78 km - Costo totale tragitto: 102,65€

Azienda: Emergenza Salute S.p.A. - P.IVA: 23456789012 - Costo per Km: 11,25€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 10,78 km - Costo totale tragitto: 109,98€

Azienda: SOS Trasporti Sanitari Spa - P.IVA: 34567890123 - Costo per Km: 9,75€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 10,78 km - Costo totale tragitto: 95,32€

Azienda: Ambulanza Veloce S.r.l. - P.IVA: 45678901234 - Costo per Km: 12,00€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 10,78 km - Costo totale tragitto: 117,31€

Azienda: ProntoSoccorso Trasporti S.p.A. - P.IVA: 56789012345 - Costo per Km: 10,75€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 10,78 km - Costo totale tragitto: 105,09€

Azienda: Ambulanza 24 Ore S.p.A. - P.IVA: 90123456789 - Costo per Km: 10,90€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 10,78 km - Costo totale tragitto: 106,56€

Azienda: San Leonardo SRL - P.IVA: 12435678901 - Costo per Km: 1,50€ - Dist azienda-partenza: -1,00 km - Dist partenza-arrivo: 10,78 km - Costo totale tragitto: 14,66€

Torna indietro

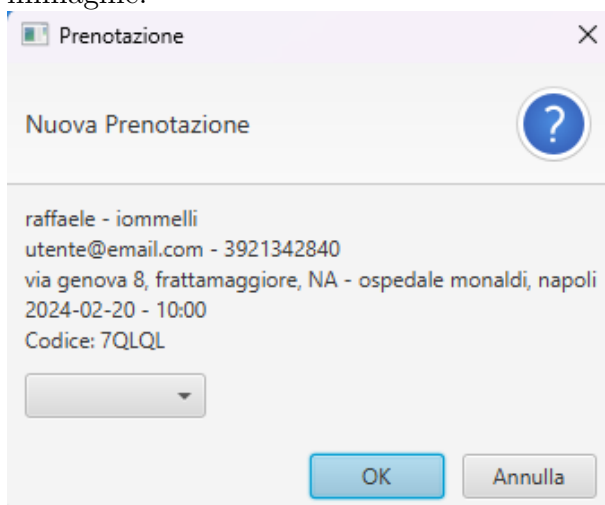
Prenota trasferimento



**Il codice dell'ambulanza è
7QLQL**

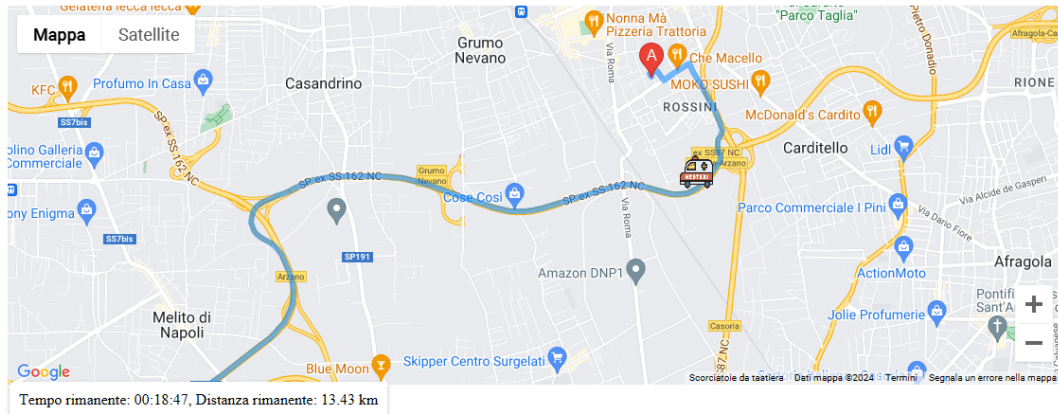
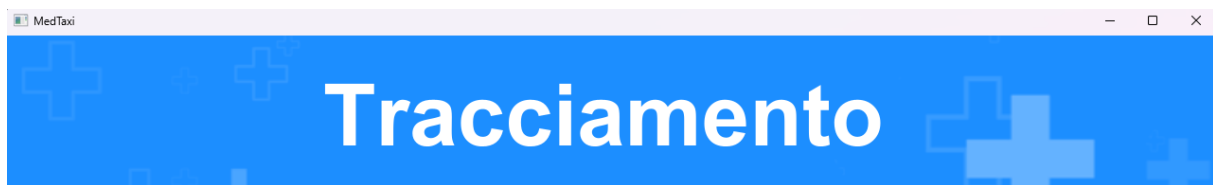
Torna alla Home

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". Verrà fornito un codice di tracking. Quando l'utente effettua una prenotazione, all'azienda uscirà questa immagine.

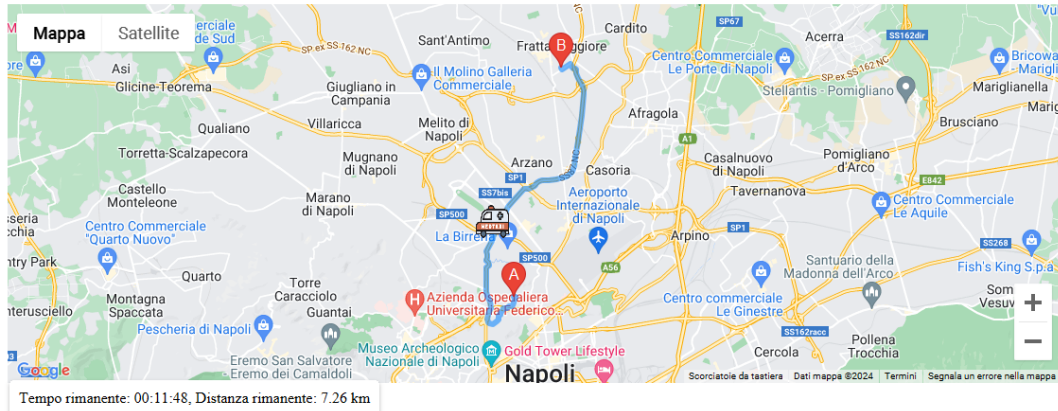


L'azienda quindi assegnerà alla prenotazione una targa, altrimenti non verrà registrata nel database.

Tracciamento



[Torna indietro](#)



[Torna indietro](#)

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

ANNULLA PRENOTAZIONE

[illegible]

[Torna indietro](#)

Annula prenotazione

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

Storico indirizzi

STORICO INDIRIZZI

Via Genova 8 frattammaggiore
Via della resistenza, 74 villaricca
via genova 8, frattammaggiore, NA

[Torna indietro](#)

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