

# Experia Coffee - Reti di Calcolatori

Avitabile Luigi 012400/2627

26 Settembre 2024



**Università degli Studi di Napoli "Parthenope"**

Dipartimento di Scienze e Tecnologie

**Corso di Reti di Calcolatori**

Anno Accademico 2023/2024

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Prefazione . . . . .	3
1.2	Descrizione del progetto . . . . .	3
1.3	Composizione del progetto . . . . .	3
<b>2</b>	<b>Descrizione e schema dell'architettura</b>	<b>5</b>
2.1	Ticketing . . . . .	5
2.2	Gestione Ordini e rifornimento del Magazzino . . . . .	6
2.3	Schema architetturale del progetto . . . . .	7
2.4	Schema Classe singleton Database . . . . .	8
<b>3</b>	<b>Dettagli implementativi dei client/server con relativo codice</b>	<b>9</b>
3.1	Implementazione dei Client . . . . .	9
3.2	Implementazione dei Server . . . . .	9
3.2.1	Invio delle richieste fra Dipendente e TicketingServer . . . . .	9
3.2.2	Ricezione delle richiesta da Dipendente a TicketingServer . . . . .	16
3.2.3	Invio delle richieste da Produttore a DipendenteServer . . . . .	24
3.2.4	Ricezione delle richieste a DipendenteServer da Produttore . . . . .	31
3.2.5	Inoltro delle richieste da DipendenteServer a MagazzinoServer . . . . .	39
3.2.6	Inoltro delle richieste da DipendenteServer a OrdineServer . . . . .	45
<b>4</b>	<b>Manuale utente con istruzioni di compilazione ed esecuzione annesse</b>	<b>50</b>
4.1	Prerequisiti . . . . .	50
4.2	Download e Configurazione dei Progetti . . . . .	50
4.3	Esecuzione . . . . .	50
4.3.1	Compound Database . . . . .	51
4.3.2	Compound Avvio Server Cluster . . . . .	52
4.3.3	Compound Avvio Clients . . . . .	55
4.3.4	Simulazione aggiornamento stato ticket [Dipendente - Ticketing Server] . . . . .	57

# 1 Introduzione

## 1.1 Prefazione

Il progetto è stato realizzato per lo svolgimento dell'esame di Reti di Calcolatori, integrato in Programmazione III, Tecnologie Web, Basi di Dati e Ingegneria del Software e Interazione uomo-macchina.

## 1.2 Descrizione del progetto

Experia Coffee è un gestionale all'avanguardia progettato per migliorare e facilitare l'esperienza utente nell'acquisto di prodotti per il caffè.

Questa piattaforma intuitiva e facile da usare, offre agli utenti la possibilità di acquistare prodotti con marchio made in Italy ed esteri riconosciuti dalla Comunità Europea. Inoltre, permette di gestire e visualizzare i propri ordini in tempo reale, conoscendone l'esatto stato.

## 1.3 Composizione del progetto

Il sistema è composto dai seguenti componenti:

- **Dipendente Server:**

- Funge da intermediario tra il Produttore e i server di Magazzino e Ordine.
- La classe Produttore registra i propri servizi ed esigenze come client verso questo server.
- Il produttore effettua le dovute operazioni di gestione ordini e controllo dei prodotti disponibili nel magazzino attraverso il Dipendente Server.

- **Magazzino Server:**

- Gestisce le richieste in arrivo da parte di Dipendente Server inerenti alla gestione dei prodotti presenti nel magazzino.
- Dipendente Server registra i propri servizi e richieste come client verso questo server.
- Il produttore effettua una richiesta di controllo prodotti nel Magazzino attraverso il Magazzino Server. I prodotti, su richiesta, possono subire una variazione da parte del produttore per tenere traccia in tempo reale della quantità disponibile in magazzino.

- **Ordine Server:**

- Gestisce le richieste in arrivo da parte di Dipendente Server inerenti alla gestione degli ordini effettuati dai clienti di Experia Coffee.
- Dipendente Server registra i propri servizi e richieste come client verso questo server.
- Il produttore effettua una richiesta di controllo ordini disponibili attraverso Ordine Server. Gli ordini, su richiesta, possono subire una variazione da parte del produttore per tenere traccia in tempo reale dello stato degli ordini.

- **Ticketing Server:**

- Gestisce le richieste in arrivo da parte di Dipendente inerente alla gestione delle segnalazioni aperte dai clienti che usufruiscono della piattaforma.

- Dipendente registra i propri servizi e richieste come client verso questo server.
  - Il dipendente effettua una richiesta di visualizzazione ticket disponibili attraverso il Ticketing Server. Le segnalazioni, su richiesta, possono essere modificate per tenere traccia dello stato corrente.
- **Comunicazione tra Dipendente Server, Magazzino Server, Ordine Server e Produttore:**
    - Il server centrale (ovvero Dipendente Server) inoltra le richieste in base al codice identificativo verso i vari destinatari.
    - Ordine e Magazzino, agendo come server, gestiscono queste richieste e mantengono uno stato aggiornato su di essi.
  - **Comunicazione tra Ticketing Server e Dipendente:**
    - Il dipendente (il quale rappresenta il Client) inoltra i ticket generati dagli utenti di Experia Coffee alla sezione incaricata nella gestione delle segnalazioni (Ticketing Server).
    - Ticketing Server, agendo come server, gestisce queste segnalazioni e mantiene uno stato aggiornato su di esse.

Ecco quali sono le caratteristiche principali del progetto:

- **Gestione e monitoraggio delle segnalazioni in tempo reale:**
  - Una volta completato l'acquisto, il cliente può direttamente pagare anziché ricompilare il form da zero.
- **Gestione e monitoraggio degli ordini in tempo reale:**
  - A partire da dopo l'acquisto degli articoli, Experia Coffee dà la possibilità di poter tracciare sin da subito lo stato di spedizione di quest'ultimi.

Ecco invece i vantaggi che offre Experia Coffee:

- **Vantaggi:**
  - Riduzione nella perdita di tempo durante la fase di acquisto.
  - Ottimizzazione delle operazioni logistiche per le filiali.
  - Controllo completo del processo di acquisto e monitoraggio per gli utenti.

## 2 Descrizione e schema dell'architettura

### 2.1 Ticketing

- **Dipendente (Client):** Il dipendente nei confronti del Ticketing Server, può effettuare diverse azioni quali:

- **Visualizzazione lista di tickets:** Il dipendente può richiedere di visualizzare la lista dei ticket attualmente disponibili.

**TicketingServer:** Il server restituisce indietro una lista di ticket attualmente disponibili, con tutte le informazioni del caso.

- **Visualizzazione lista tickets filtrati per stato:** Il dipendente può richiedere di visualizzare la lista degli stati dei ticket attualmente disponibili.

**TicketingServer:** Il server restituisce indietro una lista di stati assegnati ai vari ticket attualmente disponibili.

- **Inserimento nuovo ticket:** Il dipendente può inserire una nuova segnalazione manualmente, seguendo le istruzioni mostrate a sistema.

**TicketingServer:** Il server restituisce un log informativo di successo nel caso l'inserimento sia andato a buon fine, altrimenti gestisce l'errore mostrandolo all'interno dell'apposito log.

- **Aggiornamento stato ticket:** Il dipendente può aggiornare lo stato di una segnalazione manualmente, seguendo le istruzioni mostrate a sistema.

**TicketingServer:** Il server restituisce un log informativo di successo, nel caso l'aggiornamento sia andato a buon fine, altrimenti gestisce l'errore mostrandolo all'interno dell'apposito log.

- **Rimozione ticket:** Il dipendente può rimuovere una segnalazione manualmente, seguendo le istruzioni mostrate a sistema.

**TicketingServer:** Il server restituisce un log informativo di successo, nel caso la rimozione sia avvenuta correttamente, altrimenti gestisce l'errore mostrandolo all'interno dell'apposito log.

## 2.2 Gestione Ordini e rifornimento del Magazzino

- **Produttore (Client):** Il Produttore nei confronti di Dipendente Server, può effettuare diverse azioni. Dipendente Server funge da server intermedio il quale accoglie le richieste di Produttore e le smista in base al server di appartenenza, Ordine Server o Magazzino Server. Di seguito le seguenti azioni che può effettuare:

- **Visualizzazione lista prodotti nel Magazzino:** Il produttore può richiedere di visualizzare la lista dei prodotti attualmente disponibili all'interno del Magazzino.

**MagazzinoServer:** Il server restituisce indietro una lista di prodotti attualmente disponibili, con tutte le informazioni del caso.

- **Inserimento prodotto nel Magazzino:** Il produttore può richiedere di inserire un nuovo prodotto all'interno del Magazzino, seguendo le richieste presenti a terminale.

**MagazzinoServer:** Il server restituisce indietro un log di tipo SUCCESS se il prodotto è stato inserito correttamente, altrimenti viene gestito l'errore attraverso un log di ERROR.

- **Aggiornamento quantità prodotto nel Magazzino:** Il produttore può richiedere di aggiornare la quantità di un prodotto all'interno del Magazzino, seguendo le richieste presenti a terminale.

**MagazzinoServer:** Il server restituisce indietro un log di tipo SUCCESS se il prodotto è stato aggiornato correttamente, altrimenti viene gestito l'errore attraverso un log di ERROR.

- **Rimozione prodotto nel Magazzino:** Il produttore può richiedere di rimuovere un prodotto all'interno del Magazzino, seguendo le richieste presenti a terminale.

**MagazzinoServer:** Il server restituisce indietro un log di tipo SUCCESS se il prodotto è stato rimosso correttamente, altrimenti viene gestito l'errore attraverso un log di ERROR.

- **Visualizzazione lista di ordini nel server Ordine:** Il produttore può richiedere di visualizzare la lista di ordini all'interno dell'Ordine Server.

**OrdineServer:** Il server restituisce indietro una lista di ordini attualmente disponibili.

- **Visualizzazione lista ordini filtrati per stato:** Il produttore può richiedere di visualizzare la lista degli ordini filtrati per stato, all'interno dell'Ordine Server.

**OrdineServer:** Il server restituisce indietro una lista di stati associati ai singoli ordini attualmente disponibili.

- **Aggiornamento stato di un ordine:** Il produttore può richiedere di aggiornare lo stato di un ordine, all'interno dell'Ordine Server, seguendo le istruzioni mostrate

a terminale.

**OrdineServer:** Il server restituisce indietro un log di tipo SUCCESS se l'ordine è stato aggiornato correttamente, altrimenti viene gestito l'errore attraverso un log di ERROR.

## 2.3 Schema architetturale del progetto

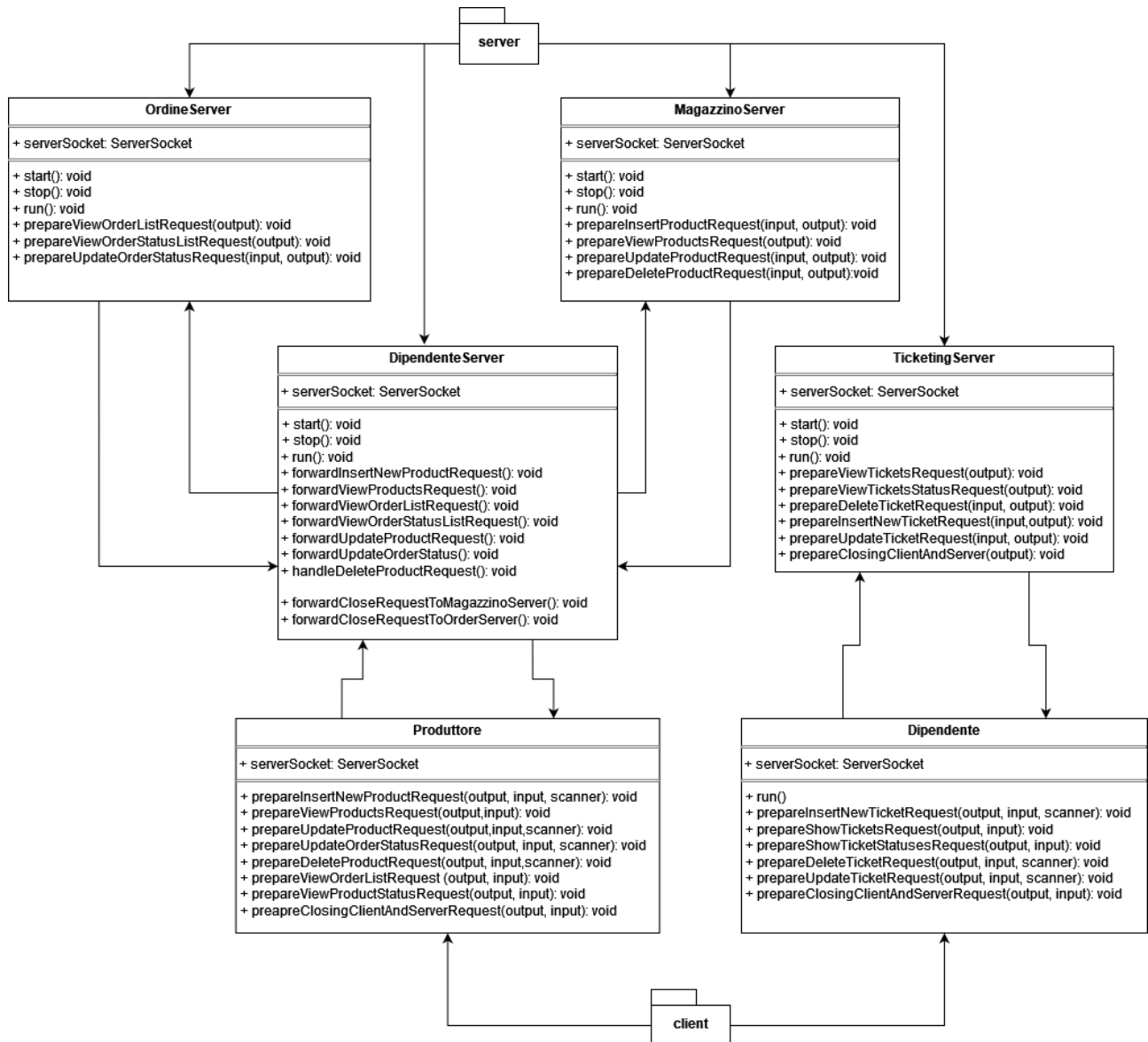


Figura 1: Schema architetturale del progetto

## 2.4 Schema Classe singleton Database

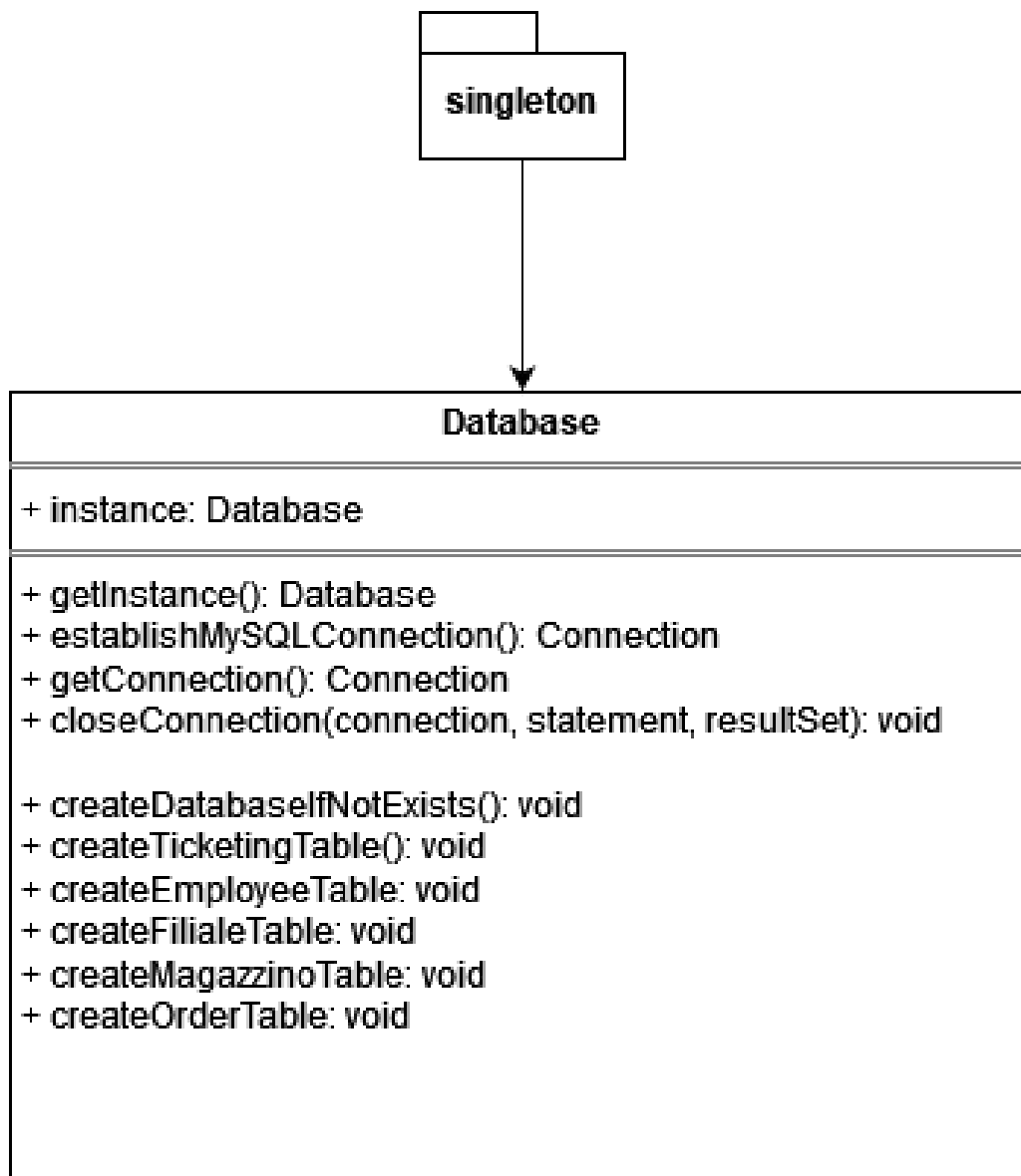


Figura 2: Schema architetturale della classe singleton Database



## 3 Dettagli implementativi dei client/server con relativo codice

Questo capitolo descrive in dettaglio l'implementazione dei componenti client e server nel sistema Experia Coffee, focalizzandosi su aspettative chiave come la comunicazione di rete, la gestione degli ordini, il monitoraggio degli articoli nei magazzini, la gestione delle segnalazioni inviate da parte degli utenti e molto altro.

### 3.1 Implementazione dei Client

I componenti client sono responsabili dell'invio delle richieste di visualizzazione/gestione ai server e della ricezione dei responsi tramite il protocollo TCP.

È implementato in Java e si interfaccia con il server centrale (Dipendente Server) per l'inoltro delle richieste e la ricezione delle risposte.

### 3.2 Implementazione dei Server

Le richieste di visualizzazione/gestione ordine, ticket o magazzino vengono inviate ai server di competenza utilizzando socket TCP. Il client crea una connessione socket verso il server di competenza, invia i dati della richiesta sotto forma di oggetto, e attende la risposta del server.

Di seguito viene mostrato il codice:

#### 3.2.1 Invio delle richieste fra Dipendente e TicketingServer

Listing 1: Dipendente.java

```
1 package client;
2
3 import logger.Log;
4 import utils.Constants;
5
6 import java.io.IOException;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9 import java.net.Socket;
10 import java.sql.Date;
11 import java.util.List;
12 import java.util.Scanner;
13
14 /**
15  * @description Classe referente al Client Dipdente
16  */
17 public class Dipendente {
18
19     private static final int TICKETING_SERVER_PORT = Constants.
TICKETING_SERVER_PORT;
20
21     public static void main(String[] args) {
22         startTicketingClient();
23     }
24 }
```

```

25  /**
26   * @description metodo always on per permettere all'utente di effettuare
    molteplici scelte, rispetto a quelle mostrate a terminale
27   */
28   private static void startTicketingClient() {
29       try (Socket socket = new Socket(Constants.HOSTNAME,
TICKETING_SERVER_PORT);
30           ObjectOutputStream output = new ObjectOutputStream(socket.
getOutputStream());
31           ObjectInputStream input = new ObjectInputStream(socket.
getInputStream());
32           Scanner scanner = new Scanner(System.in)) {
33
34           while (true) {
35               showChoices();
36
37               int choice = scanner.nextInt();
38               scanner.nextLine();
39
40               switch (choice) {
41                   case 1:
42                       prepareInsertNewTicketRequest(output, input, scanner
);
43                       break;
44                   case 2:
45                       prepareShowTicketsRequest(output, input);
46                       break;
47                   case 3:
48                       prepareShowTicketStatusesRequest(output, input);
49                       break;
50                   case 4:
51                       prepareDeleteTicketRequest(output, input, scanner);
52                       break;
53                   case 5:
54                       prepareUpdateTicketRequest(output, input, scanner);
55                       break;
56                   case 6:
57                       prepareClosingClientAndServerRequest(output, input);
58                       return;
59               }
60           }
61       } catch (IOException e) {
62           Log.error("Errore del client Ticketing: " + e.getMessage());
63       }
64   }
65
66   private static class ClientHandler implements Runnable {
67       private final Socket clientSocket;
68
69       public ClientHandler(Socket clientSocket) {
70           this.clientSocket = clientSocket;
71       }
72
73
74   /**

```

```

75      * @description metodo always on per permettere all'utente di
    effettuare molteplici scelte, rispetto a quelle mostrate a terminale
76      */
77      @Override
78      public void run() {
79          try (
80              Socket socket = new Socket(Constants.HOSTNAME, Constants
.DIPENDENTE_CLIENT_PORT);
81              ObjectOutputStream output = new ObjectOutputStream(
socket.getOutputStream());
82              ObjectInputStream input = new ObjectInputStream(socket.
getInputStream());
83              Scanner scanner = new Scanner(System.in)) {
84
85              while (true) {
86
87                  showChoices();
88
89                  int choice = scanner.nextInt();
90                  scanner.nextLine();
91
92                  switch (choice) {
93                      case 1:
94                          prepareInsertNewTicketRequest(output, input,
scanner);
95                          break;
96                      case 2:
97                          prepareShowTicketsRequest(output, input);
98                          break;
99                      case 3:
100                          prepareShowTicketStatusesRequest(output, input);
101                          break;
102                      case 4:
103                          prepareDeleteTicketRequest(output, input,
scanner);
104                          break;
105                      case 5:
106                          prepareUpdateTicketRequest(output, input,
scanner);
107                          break;
108                      case 6:
109                          prepareClosingClientAndServerRequest(output,
input);
110                          return;
111                  }
112              }
113          } catch (IOException e) {
114              Log.error(e.getMessage());
115          }
116      }
117  }
118
119  /**
120   * @description mostra le scelte che un dipendente puo' intraprendere
121   */
122  public static void showChoices() {

```

```

123     System.out.println(Constants.CHOOSE);
124     System.out.println(Constants.INSERT_NEW_TICKET);
125     System.out.println(Constants.SHOW_TICKETS);
126     System.out.println(Constants.SHOW_TICKETS_STATUSES);
127     System.out.println(Constants.DELETE_TICKET);
128     System.out.println(Constants.UPDATE_TICKET);
129     System.out.println(Constants.EXIT);
130     System.out.print(Constants.CHOICE);
131 }
132
133 /**
134  * @param output
135  * @param input
136  * @param scanner
137  * @description elaborare la richiesta di inserimento di un ticket
138  */
139 public static void prepareInsertNewTicketRequest(ObjectOutputStream
output, ObjectInputStream input, Scanner scanner) {
140     try {
141         System.out.print("Inserisci il titolo del ticket: ");
142         String ticketInsertTitle = scanner.nextLine();
143         System.out.print("Inserisci la descrizione del ticket: ");
144         String ticketInsertDescription = scanner.nextLine();
145         System.out.print("Inserisci lo stato del ticket: ");
146         String ticketInsertStatus = scanner.nextLine();
147
148         Date actualDate = new Date(System.currentTimeMillis());
149
150         // Invia al server la richiesta di cancellazione con l'ID del
ticket
151         output.writeObject(Constants.INSERT_NEW_TICKET);
152         output.writeObject(ticketInsertTitle);
153         output.writeObject(ticketInsertDescription);
154         output.writeObject(ticketInsertStatus);
155         output.writeObject(actualDate);
156
157         // Leggi la risposta dal server
158         String insertResponse = (String) input.readObject();
159
160         if (insertResponse.equals(Constants.SUCCESS)) {
161             Log.success("Il ticket e' stato inserito con successo.");
162         } else {
163             Log.error("Errore nell'inserimento del ticket");
164         }
165     } catch (IOException | ClassNotFoundException e) {
166         Log.error(e.getMessage());
167     }
168
169 }
170
171 /**
172  * @param output
173  * @param input
174  * @description elaborare la richiesta di visualizzazione di un ticket
175  */
176

```

```

177     public static void prepareShowTicketsRequest(ObjectOutputStream output,
ObjectInputStream input) {
178         try {
179             output.writeObject(Constants.VIEW_TICKETS);
180             List<String> tickets = (List<String>) input.readObject();
181             System.out.println(Constants.AVAILABLE_TICKETS);
182             for (String ticket : tickets) {
183                 System.out.println(ticket);
184             }
185         } catch (IOException | ClassNotFoundException e) {
186             Log.error(e.getMessage());
187         }
188     }
189
190     /**
191      * @param output
192      * @param input
193      * @description elaborata la richiesta di visualizzazione di un ticket in
base allo stato
194      */
195     public static void prepareShowTicketStatusesRequest(ObjectOutputStream
output, ObjectInputStream input) {
196         try {
197             output.writeObject(Constants.VIEW_TICKETS_STATUSES);
198             List<String> ticketsByStatuses = (List<String>) input.readObject
();
199             System.out.println(Constants.AVAILABLE_TICKETS);
200             for (String ticket : ticketsByStatuses) {
201                 System.out.println(ticket);
202             }
203         } catch (IOException | ClassNotFoundException e) {
204             Log.error(e.getMessage());
205         }
206     }
207
208     /**
209      * @param output
210      * @param input
211      * @param scanner
212      * @description elaborata la richiesta di cancellazione di un ticket
213      */
214     public static void prepareDeleteTicketRequest(ObjectOutputStream output,
ObjectInputStream input, Scanner scanner) {
215         try {
216             System.out.print("Inserisci l'ID del ticket da cancellare: ");
217             int ticketIdToDelete = scanner.nextInt();
218             scanner.nextLine(); // Consuma la nuova linea rimasta dopo
nextInt()
219
220             // Invia al server la richiesta di cancellazione con l'ID del
ticket
221             output.writeObject(Constants.DELETE_TICKET);
222             output.writeObject(ticketIdToDelete);
223
224             // Leggi la risposta dal server
225             String response = (String) input.readObject();

```

```

226         if (response.equals(Constants.SUCCESS)) {
227             Log.success(String.format("Il ticket con ID %d      stato
228 cancellato con successo.", ticketIdToDelete));
229         } else {
230             Log.error(String.format("Errore nella cancellazione del
231 ticket con ID %d.", ticketIdToDelete));
232         }
233     } catch (IOException | ClassNotFoundException e) {
234         Log.error(e.getMessage());
235     }
236
237     /**
238      * @param output
239      * @param input
240      * @param scanner
241      * @description elaborare la richiesta di aggiornamento di un ticket
242      */
243     public static void prepareUpdateTicketRequest(ObjectOutputStream output,
244 ObjectInputStream input, Scanner scanner) {
245         try {
246             System.out.print("Inserisci l'ID del ticket da aggiornare: ");
247             Integer ticketUpdateID = scanner.nextInt();
248             scanner.nextLine();
249
250             System.out.print("Inserisci lo stato da aggiornare: ");
251             String ticketUpdateStatus = scanner.nextLine();
252
253             output.writeObject(Constants.UPDATE_TICKET);
254             output.writeObject(ticketUpdateID);
255             output.writeObject(ticketUpdateStatus);
256
257             // Leggi la risposta dal server
258             String updateResponse = (String) input.readObject();
259
260             if (updateResponse.equals(Constants.SUCCESS)) {
261                 Log.success("Il ticket      stato aggiornato con successo.");
262             } else {
263                 Log.error("Errore nell'inserimento del ticket");
264             }
265         } catch (IOException | ClassNotFoundException e) {
266             Log.error(e.getMessage());
267         }
268     }
269
270     /**
271      * @param output
272      * @param input
273      * @description elaborare la richiesta di chiusura del client e server
274      */
275     public static void prepareClosingClientAndServerRequest(
276 ObjectOutputStream output, ObjectInputStream input) {
277         try {
278             System.out.println(Constants.ON_CLOSING_CLIENT_MESSAGE);

```

```

278         output.writeObject(Constants.EXIT);
279
280         // Leggi la conferma dal server prima di chiudere
281         String exitResponse = (String) input.readObject();
282         if (exitResponse.equals(Constants.SUCCESS)) {
283             Log.success("Client chiuso con successo.");
284         } else {
285             Log.error("Errore durante la chiusura del client.");
286         }
287     } catch (IOException | ClassNotFoundException e) {
288         Log.error(e.getMessage());
289     }
290 }
291
292 }

```

### 3.2.2 Ricezione delle richiesta da Dipendente a TicketingServer

Listing 2: TicketingServer.java

```
1 package server;
2
3 import logger.Log;
4 import singleton.Database;
5 import utils.Constants;
6 import utils.Queries;
7
8 import java.io.IOException;
9 import java.io.ObjectInputStream;
10 import java.io.ObjectOutputStream;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13 import java.sql.*;
14 import java.util.ArrayList;
15 import java.util.List;
16
17 /**
18  * @description classe inerente al TicketingServer, per la gestione dei
19  * ticket e delle segnalazioni da parte degli utenti di Experia Coffee
20  */
21 public class TicketingServer {
22
23     private ServerSocket serverSocket;
24     private boolean isRunning = true;
25
26     public TicketingServer(int port) throws IOException {
27         serverSocket = new ServerSocket(port);
28         Log.info(String.format(Constants.
29 SERVER_SOCKET_CONNECTION_ESTABLISHED, port));
30     }
31
32     public void start() throws IOException {
33         Log.info(Constants.SERVER_LISTENING);
34         while (isRunning) {
35             try {
36                 Socket clientSocket = serverSocket.accept();
37                 Log.info("Connessione stabilita con il client: " +
38 clientSocket.getInetAddress());
39                 new ClientHandler(this, clientSocket).start();
40             } catch (IOException e) {
41                 Log.error(e.getMessage());
42             }
43         }
44     }
45
46     public void stop() {
47         isRunning = false;
48         try {
49             if (serverSocket != null && !serverSocket.isClosed()) {
50                 serverSocket.close();
51             }
52             Log.info("Server chiuso.");
53         } catch (IOException e) {
```



```

51         Log.error("Errore durante la chiusura del server: " + e.
getMessage());
52     }
53 }
54
55 public static void main(String[] args) throws IOException {
56     TicketingServer server = new TicketingServer(Constants.
TICKETING_SERVER_PORT);
57     server.start();
58 }
59
60 private static class ClientHandler extends Thread {
61
62     private Socket clientSocket;
63     private TicketingServer server;
64
65     public ClientHandler(TicketingServer server, Socket socket) {
66         this.server = server;
67         this.clientSocket = socket;
68     }
69
70     private void prepareViewTicketsRequest(ObjectOutputStream output)
throws SQLException {
71         try {
72             List<String> tickets = viewTickets();
73             output.writeObject(tickets);
74
75         } catch (IOException e) {
76             Log.error(e.getMessage());
77         }
78     }
79
80     private void prepareViewTicketsStatusRequest(ObjectOutputStream
output) throws SQLException {
81         try {
82             List<String> ticketStatuses = viewTicketStatuses();
83             output.writeObject(ticketStatuses);
84         } catch (IOException e) {
85             Log.error(e.getMessage());
86         }
87     }
88
89     private void prepareDeleteTicketRequest(ObjectInputStream input,
ObjectOutputStream output) throws SQLException {
90         try {
91             int ticketId = (int) input.readObject();
92             boolean deleted = deleteTicket(ticketId);
93             if (deleted) {
94                 output.writeObject(Constants.SUCCESS);
95                 Log.info(String.format("Ticket con ID %d cancellato con
successo.", ticketId));
96             } else {
97                 output.writeObject(Constants.FAILURE);
98                 Log.warning(String.format("Cancellazione fallita per il
ticket con ID %d.", ticketId));
99             }
100         }

```

```

100         }
101     } catch (IOException | ClassNotFoundException e) {
102         Log.error(e.getMessage());
103     }
104 }
105
106 private void prepareInsertNewTicketRequest(ObjectInputStream input,
107 ObjectOutputStream output) throws SQLException {
108     try {
109         String ticketTitle = (String) input.readObject();
110         String ticketDescription = (String) input.readObject();
111         String ticketStatus = (String) input.readObject();
112         Date ticketCreatedDate = (Date) input.readObject();
113         boolean inserted = insertNewTicket(ticketTitle,
114 ticketDescription, ticketStatus, ticketCreatedDate);
115         if (inserted) {
116             output.writeObject(Constants.SUCCESS);
117             Log.info("Ticket inserito con successo.");
118         } else {
119             output.writeObject(Constants.FAILURE);
120             Log.warning("Inserimento fallito del ticket.");
121         }
122     } catch (SQLException | RuntimeException |
123 ClassNotFoundException | IOException e) {
124         Log.error(e.getMessage());
125     }
126 }
127
128 private void prepareUpdateTicketRequest(ObjectInputStream input,
129 ObjectOutputStream output) throws SQLException {
130     try {
131         Integer ticketUpdateId = (Integer) input.readObject();
132         String ticketUpdateStatus = (String) input.readObject();
133         boolean updated = updateTicketStatus(ticketUpdateId,
134 ticketUpdateStatus);
135         if (updated) {
136             output.writeObject(Constants.SUCCESS);
137             Log.info("Ticket aggiornato con successo.");
138         } else {
139             output.writeObject(Constants.FAILURE);
140             Log.warning("Aggiornamento fallito del ticket.");
141         }
142     } catch (SQLException | ClassNotFoundException |
143 RuntimeException | IOException e) {
144         Log.error(e.getMessage());
145     }
146 }
147
148 private void prepareClosingClientAndServer(ObjectOutputStream output
149 ) throws SQLException {
150     try {
151         output.writeObject(Constants.SUCCESS);
152         server.stop();
153     } catch (IOException e) {
154         Log.error(e.getMessage());
155     }
156 }

```

```

149     }
150
151     }
152
153     public void run() {
154         try (
155             ObjectOutputStream output = new ObjectOutputStream(
156                 clientSocket.getOutputStream());
157             ObjectInputStream input = new ObjectInputStream(
158                 clientSocket.getInputStream())
159         ) {
160             while (true) {
161                 try {
162                     String request = (String) input.readObject();
163                     switch (request) {
164                         case Constants.VIEW_TICKETS:
165                             prepareViewTicketsRequest(output);
166                             break;
167                         case Constants.VIEW_TICKETS_STATUSES:
168                             prepareViewTicketsStatusRequest(output);
169                             break;
170                         case Constants.DELETE_TICKET:
171                             prepareDeleteTicketRequest(input, output);
172                             break;
173                         case Constants.INSERT_NEW_TICKET:
174                             prepareInsertNewTicketRequest(input, output);
175                             break;
176                         case Constants.UPDATE_TICKET:
177                             prepareUpdateTicketRequest(input, output);
178                             break;
179                         case Constants.EXIT:
180                             prepareClosingClientAndServer(output);
181                             break;
182                         default:
183                             output.writeObject(Constants.UNKNOWN_REQUEST);
184                             break;
185                     }
186                 } catch (ClassNotFoundException | SQLException e) {
187                     Log.error("Errore durante la gestione del client: "
188                         + e.getMessage());
189                     output.writeObject(Constants.FAILURE);
190                 }
191             }
192         } catch (IOException e) {
193             Log.error("Errore di I/O: " + e.getMessage());
194         } finally {
195             try {
196                 clientSocket.close();
197             } catch (IOException e) {
198                 Log.error("Errore durante la chiusura del socket: " + e.
199                     getMessage());
200             }
201             Log.info("Connessione chiusa con il client");
202         }
203     }

```

```

199     }
200
201
202     /**
203      * @return List<String> contenente la lista dei ticket trovati
204      * @throws SQLException
205      * @description funzione la quale mostra tutti i ticket disponibili
206      */
207     private List<String> viewTickets() throws SQLException {
208
209         Connection connection = null;
210         Statement statement = null;
211         ResultSet resultSet = null;
212
213         List<String> tickets = new ArrayList<>();
214         try {
215             connection = Database.getInstance().getConnection();
216             statement = connection.createStatement();
217             resultSet = statement.executeQuery(String.format(Queries.
218             GENERIC_QUERY_SELECT, Constants.TBL_TICKETING));
219
220             while (resultSet.next()) {
221                 int id = resultSet.getInt("ID");
222                 String titolo = resultSet.getString("TITOLO");
223                 String descrizione = resultSet.getString("DESCRIZIONE");
224                 String gestitoDa = resultSet.getString("GESTITO_DA");
225                 String creatoDa = resultSet.getString("CREATO_DA");
226                 Date dataCreazione = resultSet.getDate("DATA_CREAZIONE");
227
228                 ;
229
230                 String stato = resultSet.getString("STATO");
231
232                 // Costruisci una rappresentazione leggibile del ticket
233                 String ticket = String.format("ID: %d, Titolo: %s,
234                 Descrizione: %s, Gestito da: %s, Creato da: %s, Data creazione: %s, Stato
235                 : %s",
236
237                     id, titolo, descrizione, gestitoDa != null ?
238                     gestitoDa : "Non assegnato", creatoDa != null ? creatoDa : "Anonimo",
239                     dataCreazione, stato);
240
241                 tickets.add(ticket);
242             }
243         } finally {
244             Database.closeConnection(connection, statement, resultSet);
245         }
246         return tickets;
247     }
248
249     /**
250      * @return List<String> contenente gli stati dei ticket suddivisi
251      per ID
252      * @throws SQLException
253      * @description funzione la quale mostra gli id dei ticket con i
254      rispettivi stati
255      */
256     private List<String> viewTicketStatuses() throws SQLException {

```

```

247         Connection connection = null;
248         Statement statement = null;
249         ResultSet resultSet = null;
250
251         List<String> tickets = new ArrayList<>();
252         try {
253             connection = Database.getInstance().getConnection();
254             statement = connection.createStatement();
255             resultSet = statement.executeQuery(Queries.
TBL_TICKETING_SELECT_STATUSES_QUERY);
256
257             while (resultSet.next()) {
258                 int id = resultSet.getInt("ID");
259                 String stato = resultSet.getString("STATO");
260
261                 // Costruisci una rappresentazione leggibile del ticket
262                 String ticket = String.format("ID: %d, Stato: %s", id,
stato);
263
264                 tickets.add(ticket);
265             }
266         } catch (SQLException ex) {
267             Log.error(ex.getMessage());
268         } finally {
269             Database.closeConnection(connection, statement, resultSet);
270         }
271         return tickets;
272     }
273
274     /**
275      * @return true se l'operazione e' andata a buon fine, altrimenti
false
276      * @throws SQLException
277      * @description elimina ticket in base all'id fornito
278      */
279     private boolean deleteTicket(int ticketId) throws SQLException {
280         Connection connection = null;
281         PreparedStatement preparedStatement = null;
282
283         try {
284             connection = Database.getInstance().getConnection();
285             preparedStatement = connection.prepareStatement(Queries.
TBL_TICKETING_DELETE_TICKET_BY_ID_QUERY);
286
287             preparedStatement.setInt(1, ticketId);
288
289             int rowsAffected = preparedStatement.executeUpdate();
290
291             // Se viene cancellata almeno una riga, la cancellazione
avvenuta con successo
292             return rowsAffected > 0;
293         } finally {
294             Database.closeConnection(connection, preparedStatement, null
);
295         }
296     }

```

```

297     /**
298     * @param ticketTitle
299     * @param ticketDescription
300     * @param ticketStatus
301     * @param ticketCreationDate
302     * @return true se l'operazione e' andata a buon fine, altrimenti
false
303     * @throws SQLException
304     * @description inserisce un nuovo ticket
305     */
306     private boolean insertNewTicket(String ticketTitle, String
ticketDescription, String ticketStatus, Date ticketCreationDate) throws
SQLException {
307         Connection connection = null;
308         PreparedStatement preparedStatement = null;
309
310         try {
311             connection = Database.getInstance().getConnection();
312             preparedStatement = connection.prepareStatement(Queries.
TBL_TICKETING_INSERT_NEW_TICKET_BY_QUERY);
313             preparedStatement.setString(1, ticketTitle);
314             preparedStatement.setString(2, ticketDescription);
315             preparedStatement.setString(3, ticketStatus);
316             preparedStatement.setDate(4, ticketCreationDate);
317
318             int rowsAffected = preparedStatement.executeUpdate();
319
320             // Se viene cancellata almeno una riga, la cancellazione
avvenuta con successo
321             return rowsAffected > 0;
322
323         } finally {
324             Database.closeConnection(connection, preparedStatement, null
);
325         }
326     }
327
328     /**
329     * @param ticketId
330     * @param ticketStatus
331     * @return true se l'operazione e' andata a buon fine, altrimenti
false
332     * @throws SQLException
333     * @description aggiorna lo stato di un ticket
334     */
335     private boolean updateTicketStatus(Integer ticketId, String
ticketStatus) throws SQLException {
336         Connection connection = null;
337         PreparedStatement preparedStatement = null;
338
339         try {
340             connection = Database.getInstance().getConnection();
341             preparedStatement = connection.prepareStatement(Queries.
TBL_TICKETING_UPDATE_TICKET_STATUS_BY_QUERY);
342             preparedStatement.setString(1, ticketStatus);
343             preparedStatement.setInt(2, ticketId);

```

```

344
345         int rowsAffected = preparedStatement.executeUpdate();
346
347         // Se viene cancellata almeno una riga, la cancellazione
avvenuta con successo
348         return rowsAffected > 0;
349
350     } catch (SQLException e) {
351         Log.error(e.getMessage());
352     } finally {
353         Database.closeConnection(connection, preparedStatement, null
);
354     }
355
356     return false;
357 }
358
359 }
360
361 }

```

### 3.2.3 Invio delle richieste da Produttore a DipendenteServer

Listing 3: Produttore.java

```
1 package client;
2
3 import logger.Log;
4 import utils.Constants;
5 import utils.Utills;
6
7 import java.io.IOException;
8 import java.io.ObjectInputStream;
9 import java.io.ObjectOutputStream;
10 import java.net.Socket;
11 import java.util.List;
12 import java.util.Scanner;
13
14 public class Produttore {
15     package client;
16
17     import logger.Log;
18     import utils.Constants;
19     import utils.Utills;
20
21     import java.io.IOException;
22     import java.io.ObjectInputStream;
23     import java.io.ObjectOutputStream;
24     import java.net.Socket;
25     import java.util.List;
26     import java.util.Scanner;
27
28
29     /**
30      * @description Classe referente al Client Produttore
31      */
32     public class Produttore {
33
34         public static void main(String[] args) throws IOException {
35             try (
36                 Socket socket = new Socket(Constants.HOSTNAME, Constants.
37                     DIPENDENTE_SERVER_PORT);
38                 ObjectOutputStream output = new ObjectOutputStream(socket.
39                     getOutputStream());
40                 ObjectInputStream input = new ObjectInputStream(socket.
41                     getInputStream());
42                 Scanner scanner = new Scanner(System.in)) {
43
44                 while (true) {
45                     showChoices();
46
47                     int choice = scanner.nextInt();
48                     scanner.nextLine();
49
50                     switch (choice) {
51                         case 1:
52                             prepareInsertNewProductRequest(output, input,
53                                 scanner);
54                     }
55                 }
56             }
57         }
58     }
59 }
```



```

50         break;
51     case 2:
52         prepareViewProductsRequest(output, input);
53         break;
54     case 3:
55         prepareUpdateProductRequest(output, input, scanner);
56         break;
57     case 4:
58         prepareDeleteProductRequest(output, input, scanner);
59         break;
60     case 5:
61         prepareViewOrderListRequest(output, input);
62         break;
63     case 6:
64         prepareViewProductStatusRequest(output, input);
65         break;
66     case 7:
67         prepareUpdateOrderStatusRequest(output, input,
scanner);
68         break;
69     case 8:
70         prepareClosingClientAndServerRequest(output, input);
71         return;
72     }
73 }
74 } catch (IOException e) {
75     Log.error("Errore nella connessione: " + e.getMessage());
76 }
77 }
78
79 /**
80  * @description mostra le scelte che un produttore pu intraprendere
81  */
82 public static void showChoices() {
83     System.out.println("Scegli un'operazione:");
84     System.out.println("--- OPERAZIONI MAGAZZINO ---");
85     System.out.println("1 - Inserisci nuovo prodotto");
86     System.out.println("2 - Visualizza prodotti");
87     System.out.println("3 - Aggiorna quantit prodotto");
88     System.out.println("4 - Elimina prodotto");
89     System.out.println("--- OPERAZIONI ORDINI ---");
90     System.out.println("5 - Visualizza ordini");
91     System.out.println("6 - Visualizza stato degli ordini");
92     System.out.println("7 - Aggiorna lo stato di un ordine");
93     System.out.println("-----");
94     System.out.println("8 - Esci");
95
96     System.out.print("Scelta: ");
97 }
98
99 /**
100  * @param output
101  * @param input
102  * @param scanner
103  * @description elabora la richiesta di inserimento di un nuovo prodotto
104  */

```

```

105     public static void prepareInsertNewProductRequest(ObjectOutputStream
output, ObjectInputStream input, Scanner scanner) {
106         try {
107             // Parametri da inserire: CODICE_MAGAZZINO, ID_PRODOTTO,
QUANTITA_PRODOTTO, NOME_PRODOTTO, NOME_MAGAZZINO
108
109             System.out.print("Inserisci l'ID del prodotto: ");
110             String productId = scanner.nextLine();
111             System.out.print("Inserisci la quantit del prodotto: ");
112             int productQuantity = scanner.nextInt();
113             scanner.nextLine(); // Consumare newline
114
115             System.out.print("Inserisci il nome del prodotto: ");
116             String productName = scanner.nextLine();
117
118             System.out.print("Inserisci il nome del magazzino: ");
119             String nomeMagazzino = scanner.nextLine();
120
121             // Imposta CODICE_MAGAZZINO su "Z000"
122             String codiceMagazzino = "Z000";
123
124             // Invia al server la richiesta di inserimento prodotto
125             output.writeObject(Constants.MAGAZZINO_INSERT_NEW_PRODUCT);
126             output.writeObject(codiceMagazzino); // Aggiunto parametro
CODICE_MAGAZZINO
127             output.writeObject(productId);
128             output.writeObject(productQuantity);
129             output.writeObject(productName);
130             output.writeObject(nomeMagazzino);
131
132             // Leggi la risposta dal server
133             String insertResponse = (String) input.readObject();
134
135             if (insertResponse.equals(Constants.SUCCESS)) {
136                 Log.success(String.format("Prodotto %s inserito con successo
.", productName));
137             } else {
138                 Log.error(String.format("Errore nell'inserimento del
prodotto %s.", productName));
139             }
140             } catch (IOException | ClassNotFoundException e) {
141                 Log.error(e.getMessage());
142             }
143         }
144
145         /**
146          * @param output
147          * @param input
148          * @description elabora la richiesta di visualizzazione dei prodotti
149          */
150
151         public static void prepareViewProductsRequest(ObjectOutputStream output,
ObjectInputStream input) {
152             try {
153                 // Invia la richiesta a DipendenteServer
154                 output.writeObject(Constants.MAGAZZINO_VIEW_PRODUCTS);

```

```

155         // Attendi la risposta (lista dei prodotti) da DipendenteServer
156         List<String> products = (List<String>) input.readObject();
157
158         // Mostra i prodotti
159         if (products != null && !products.isEmpty()) {
160             System.out.println("Prodotti disponibili:");
161             System.out.println(Utils.formatProductList(products));
162         } else {
163             System.out.println("Nessun prodotto trovato.");
164         }
165     } catch (IOException | ClassNotFoundException e) {
166         Log.error("Errore durante la visualizzazione dei prodotti: " + e
167             .getMessage());
168     }
169 }
170
171 /**
172  * @param output
173  * @param input
174  * @description elabora la richiesta di visualizzazione dello stato
175  * degli ordini
176  */
177 public static void prepareViewProductStatusRequest(ObjectOutputStream
178     output, ObjectInputStream input) {
179     try {
180         // Invia la richiesta a DipendenteServer
181         output.writeObject(Constants.ORDER_VIEW_STATUS_LIST);
182
183         // Attendi la risposta (lista dei prodotti) da DipendenteServer
184         List<String> status = (List<String>) input.readObject();
185
186         // Mostra i prodotti
187         if (status != null && !status.isEmpty()) {
188             System.out.println("Stato degli ordini disponibili:");
189             System.out.println(Utils.formatProductList(status));
190         } else {
191             System.out.println("Nessun prodotto trovato.");
192         }
193     } catch (IOException | ClassNotFoundException e) {
194         Log.error("Errore durante la visualizzazione dei prodotti: " + e
195             .getMessage());
196     }
197 }
198
199 /**
200  * @param output
201  * @param input
202  * @param scanner
203  * @description elabora la richiesta di aggiornamento della quantit  di
204  * un prodotto
205  */
206 public static void prepareUpdateProductRequest(ObjectOutputStream output
207     , ObjectInputStream input, Scanner scanner) {
208     try {
209         System.out.print("Inserisci l'ID del prodotto da aggiornare: ");

```

```

205     String productId = scanner.next();
206     scanner.nextLine();
207
208     System.out.print("Inserisci la nuova quantit : ");
209     int newQuantity = scanner.nextInt();
210     scanner.nextLine();
211
212     // Invia al server la richiesta di aggiornamento
213     output.writeObject(Constants.MAGAZZINO_UPDATE_PRODUCTS);
214     output.writeObject(productId);
215     output.writeObject(newQuantity);
216
217     // Leggi la risposta dal server
218     String updateResponse = (String) input.readObject();
219
220     if (updateResponse.equals(Constants.SUCCESS)) {
221         Log.success(String.format("Prodotto %s stato aggiornato
222 con successo. Quantit disponibile: %d", productId, newQuantity));
223     } else {
224         Log.error("Errore nell'aggiornamento del prodotto.");
225     }
226     } catch (IOException | ClassNotFoundException e) {
227         Log.error(e.getMessage());
228     }
229
230     /**
231     * @param output
232     * @param input
233     * @param scanner
234     * @description elabora la richiesta di eliminazione di un prodotto dal
235     magazzino
236     */
237     public static void prepareDeleteProductRequest(ObjectOutputStream output
238 , ObjectInputStream input, Scanner scanner) {
239         try {
240             System.out.print("Inserisci l'ID del prodotto da eliminare: ");
241             String productId = scanner.next();
242             scanner.nextLine();
243
244             // Invia al server la richiesta di aggiornamento
245             output.writeObject(Constants.MAGAZZINO_DELETE_PRODUCTS);
246             output.writeObject(productId);
247
248             // Leggi la risposta dal server
249             String updateResponse = (String) input.readObject();
250
251             if (updateResponse.equals(Constants.SUCCESS)) {
252                 Log.success(String.format("Prodotto %s rimosso con successo.
253 ", productId));
254             } else {
255                 Log.error(String.format("Errore nella rimozione del prodotto
256 %s.", productId));
257             }
258         } catch (IOException | ClassNotFoundException e) {
259             Log.error(e.getMessage());
260         }
261     }

```

```

256     }
257 }
258
259
260 /**
261  * @param output
262  * @param input
263  * @description elabora la richiesta di visualizzazione degli ordini
264  */
265 public static void prepareViewOrderListRequest(ObjectOutputStream output
, ObjectInputStream input) {
266     try {
267         // Invia la richiesta a DipendenteServer
268         output.writeObject(Constants.ORDER_VIEW_LIST);
269
270         // Attendi la risposta (lista dei prodotti) da DipendenteServer
271         List<String> orders = (List<String>) input.readObject();
272
273         // Mostra i prodotti
274         if (orders != null && !orders.isEmpty()) {
275             System.out.println("Prodotti disponibili:");
276             System.out.println(Utils.formatProductList(orders));
277         } else {
278             System.out.println("Nessun ordine trovato.");
279         }
280     } catch (IOException | ClassNotFoundException e) {
281         Log.error("Errore durante la visualizzazione dei prodotti: " + e
.getMessage());
282     }
283 }
284
285 /**
286  * @param output
287  * @param input
288  * @param scanner
289  * @description elabora la richiesta di aggiornamento della quantita di
un prodotto
290  */
291 public static void prepareUpdateOrderStatusRequest(ObjectOutputStream
output, ObjectInputStream input, Scanner scanner) {
292     try {
293         System.out.print("Inserisci il numero dell'ordine da aggiornare:
");
294
295         Integer orderID = scanner.nextInt();
296         scanner.nextLine();
297
298         System.out.print("Inserisci lo stato da aggiornare: ");
299         String orderStatus = scanner.nextLine();
300
301         // Invia al server la richiesta di aggiornamento
302         output.writeObject(Constants.ORDER_UPDATE_STATUS);
303         output.writeObject(orderID);
304         output.writeObject(orderStatus);
305
306         // Leggi la risposta dal server
307         String updateResponse = (String) input.readObject();

```

```

307
308         if (updateResponse.equals(Constants.SUCCESS)) {
309             Log.success(String.format("Stato dell'ordine %d      stato
aggiornato con successo. Stato attuale: %s", orderID, orderStatus));
310         } else {
311             Log.error("Errore nell'aggiornamento dell'ordine.");
312         }
313     } catch (IOException | ClassNotFoundException e) {
314         Log.error(e.getMessage());
315     }
316 }
317
318 /**
319  * @param output
320  * @param input
321  * @description elabora la richiesta di chiusura del client e server
322  */
323 public static void prepareClosingClientAndServerRequest(
ObjectOutputStream output, ObjectInputStream input) {
324     try {
325         System.out.println("Chiusura del client...");
326         output.writeObject(Constants.EXIT);
327
328         // Leggi la conferma dal server
329         String exitResponse = (String) input.readObject();
330         if (exitResponse.equals(Constants.SUCCESS)) {
331             Log.success("Client chiuso con successo.");
332         } else {
333             Log.error("Errore durante la chiusura del client.");
334         }
335     } catch (IOException | ClassNotFoundException e) {
336         Log.error(e.getMessage());
337     }
338 }
339 }

```

### 3.2.4 Ricezione delle richieste a DipendenteServer da Produttore

Listing 4: DipendenteServer.java

```
1 package server;
2
3 import logger.Log;
4 import utils.Constants;
5
6 import java.io.IOException;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9 import java.net.ServerSocket;
10 import java.net.Socket;
11 import java.sql.SQLException;
12 import java.util.List;
13
14
15 /**
16  * @description Classe inerente al server intermediario fra Produttore e
17  * MagazzinoServer e OrdineServer
18  */
19 public class DipendenteServer {
20
21     private ServerSocket serverSocket;
22     private boolean isRunning = true;
23
24     public DipendenteServer(int port) throws IOException {
25         serverSocket = new ServerSocket(port);
26         Log.info(String.format(Constants.
27 SERVER_SOCKET_CONNECTION_ESTABLISHED, port));
28     }
29
30     public void start() throws IOException {
31         Log.info(Constants.SERVER_LISTENING);
32         while (isRunning) {
33             try {
34                 Socket clientSocket = serverSocket.accept();
35                 Log.info("Connessione stabilita con il client: " +
36 clientSocket.getInetAddress());
37                 new DipendenteServer.ClientHandler(this, clientSocket).start
38 ();
39             } catch (IOException e) {
40                 Log.error(e.getMessage());
41             }
42         }
43     }
44
45     public void stop() {
46         isRunning = false;
47         try {
48             if (serverSocket != null && !serverSocket.isClosed()) {
49                 serverSocket.close();
50             }
51             Log.info("Server chiuso.");
52         } catch (IOException e) {
```

```

49         Log.error("Errore durante la chiusura del server: " + e.
getMessage());
50     }
51 }
52
53 public static void main(String[] args) throws IOException {
54     DipendenteServer server = new DipendenteServer(Constants.
DIPENDENTE_SERVER_PORT);
55     server.start();
56 }
57
58 private static class ClientHandler extends Thread {
59
60     private Socket clientSocket;
61     private DipendenteServer server;
62
63     public ClientHandler(DipendenteServer server, Socket socket) {
64         this.server = server;
65         this.clientSocket = socket;
66     }
67
68     public void run() {
69         try (
70             ObjectOutputStream output = new ObjectOutputStream(
clientSocket.getOutputStream());
71             ObjectInputStream input = new ObjectInputStream(
clientSocket.getInputStream())
72         ) {
73             // Ciclo che mantiene la connessione attiva
74             while (true) {
75                 try {
76                     String request = (String) input.readObject();
77                     switch (request) {
78                         case Constants.ORDER_VIEW_LIST:
79                             forwardViewOrderListRequest(output);
80                             break;
81                         case Constants.ORDER_VIEW_STATUS_LIST:
82                             forwardViewOrderStatusListRequest(output);
83                             break;
84                         case Constants.ORDER_UPDATE_STATUS:
85                             forwardUpdateOrderStatus(input, output);
86                             break;
87                         case Constants.MAGAZZINO_INSERT_NEW_PRODUCT:
88                             forwardInsertNewProductRequest(input, output
);
89                             break;
90                         case Constants.MAGAZZINO_UPDATE_PRODUCTS:
91                             forwardUpdateProductRequest(input, output);
92                             break;
93                         case Constants.MAGAZZINO_DELETE_PRODUCTS:
94                             handleDeleteProductRequest(input, output);
95                             break;
96                         case Constants.MAGAZZINO_VIEW_PRODUCTS:
97                             forwardViewProductsRequest(output);
98                             break;
99                         case Constants.EXIT:

```



```

100         forwardCloseRequestToMagazzinoServer();
101         forwardCloseRequestToOrderServer();
102         prepareClosingClientAndServer(output);
103         return;
104     default:
105         output.writeObject(Constants.UNKNOWN_REQUEST
106 );
107         break;
108     }
109     } catch (ClassNotFoundException | SQLException e) {
110         Log.error("Errore durante la gestione del client: "
111 + e.getMessage());
112         output.writeObject(Constants.FAILURE);
113     }
114     }
115     } catch (IOException e) {
116         Log.error("Errore di I/O: " + e.getMessage());
117     } finally {
118         try {
119             clientSocket.close();
120         } catch (IOException e) {
121             Log.error("Errore durante la chiusura del socket: " + e.
122 getMessage());
123         }
124     }
125     Log.info("Connessione chiusa con il client");
126 }
127
128 private void forwardInsertNewProductRequest(ObjectInputStream input,
129 ObjectOutputStream output) throws IOException, ClassNotFoundException {
130     try {
131         String codiceMagazzino = (String) input.readObject();
132         String productId = (String) input.readObject();
133         int productQuantity = (int) input.readObject();
134         String productName = (String) input.readObject();
135         String nomeMagazzino = (String) input.readObject();
136
137         // Inoltra la richiesta a MagazzinoServer
138         String response = forwardInsertNewProductToMagazzinoServer(
139 codiceMagazzino, productId, productQuantity, productName, nomeMagazzino);
140
141         // Invia la risposta a Produttore
142         output.writeObject(response);
143     } catch (IOException | ClassNotFoundException e) {
144         Log.error(e.getMessage());
145         output.writeObject(Constants.FAILURE);
146     }
147 }
148
149 private void forwardViewProductsRequest(ObjectOutputStream output)
150 throws SQLException {
151     try {
152         Socket magazzinoSocket = new Socket(Constants.HOSTNAME,
153 Constants.MAGAZZINO_SERVER_PORT);
154         ObjectOutputStream magazzinoOutput = new
155 ObjectOutputStream(magazzinoSocket.getOutputStream());

```

```

148         ObjectInputStream magazzinoInput = new ObjectInputStream
(magazzinoSocket.getInputStream())
149     ) {
150         // Invia la richiesta a MagazzinoServer
151         magazzinoOutput.writeObject(Constants.
MAGAZZINO_VIEW_PRODUCTS);
152
153         // Ricevi la lista di prodotti da MagazzinoServer
154         List<String> products = (List<String>) magazzinoInput.
readObject();
155
156         // Invia la lista di prodotti a Produttore
157         output.writeObject(products);
158     } catch (IOException | ClassNotFoundException e) {
159         Log.error("Errore durante l'inoltro della richiesta a
MagazzinoServer: " + e.getMessage());
160     }
161 }
162
163 private void forwardViewOrderListRequest(ObjectOutputStream output)
throws SQLException {
164     try (
165         Socket ordineSocket = new Socket(Constants.HOSTNAME,
Constants.ORDINE_SERVER_PORT);
166         ObjectOutputStream ordineaOutput = new
ObjectOutputStream(ordineSocket.getOutputStream());
167         ObjectInputStream ordineInput = new ObjectInputStream(
ordineSocket.getInputStream())
168     ) {
169         // Invia la richiesta a MagazzinoServer
170         ordineaOutput.writeObject(Constants.ORDER_VIEW_LIST);
171
172         // Ricevi la lista di prodotti da MagazzinoServer
173         List<String> orders = (List<String>) ordineInput.readObject
();
174
175         // Invia la lista di prodotti a Produttore
176         output.writeObject(orders);
177     } catch (IOException | ClassNotFoundException e) {
178         Log.error("Errore durante l'inoltro della richiesta a
MagazzinoServer: " + e.getMessage());
179     }
180 }
181
182 private void forwardViewOrderStatusListRequest(ObjectOutputStream
output) throws SQLException {
183     try (
184         Socket ordineSocket = new Socket(Constants.HOSTNAME,
Constants.ORDINE_SERVER_PORT);
185         ObjectOutputStream ordineaOutput = new
ObjectOutputStream(ordineSocket.getOutputStream());
186         ObjectInputStream ordineInput = new ObjectInputStream(
ordineSocket.getInputStream())
187     ) {
188         // Invia la richiesta a MagazzinoServer
189         ordineaOutput.writeObject(Constants.ORDER_VIEW_STATUS_LIST);

```

```

190         // Ricevi la lista di prodotti da MagazzinoServer
191         List<String> orderStatuses = (List<String>) ordineInput.
192         readObject();
193
194         // Invia la lista di prodotti a Produttore
195         output.writeObject(orderStatuses);
196     } catch (IOException | ClassNotFoundException e) {
197         Log.error("Errore durante l'inoltro della richiesta a
198         MagazzinoServer: " + e.getMessage());
199     }
200
201     private void forwardUpdateOrderStatus(ObjectInputStream input,
202     ObjectOutputStream output) throws IOException, ClassNotFoundException,
203     SQLException {
204         try {
205             Integer orderId = (Integer) input.readObject();
206             String orderStatus = (String) input.readObject();
207
208             // Inoltra la richiesta a MagazzinoServer
209             String response = forwardUpdateOrderStatusToOrdineServer(
210             orderId, orderStatus);
211
212             // Invia la risposta a Produttore
213             output.writeObject(response);
214         } catch (IOException | ClassNotFoundException e) {
215             Log.error(e.getMessage());
216             output.writeObject(Constants.FAILURE);
217         }
218     }
219
220     private String forwardUpdateOrderStatusToOrdineServer(Integer
221     orderId, String orderStatus) throws IOException, ClassNotFoundException {
222         try {
223             Socket orderSocket = new Socket(Constants.HOSTNAME,
224             Constants.ORDINE_SERVER_PORT);
225             ObjectOutputStream orderOutput = new ObjectOutputStream(
226             orderSocket.getOutputStream());
227             ObjectInputStream orderInput = new ObjectInputStream(
228             orderSocket.getInputStream())
229         ) {
230             // Invia richiesta di aggiornamento a MagazzinoServer
231             orderOutput.writeObject(Constants.ORDER_UPDATE_STATUS);
232             orderOutput.writeObject(orderId);
233             orderOutput.writeObject(orderStatus);
234
235             // Leggi la risposta da MagazzinoServer
236             return (String) orderInput.readObject();
237         }
238     }
239
240     private void handleDeleteProductRequest(ObjectInputStream input,
241     ObjectOutputStream output) {
242         try {
243             // Riceve l'ID del prodotto dal Produttore

```

```

236         String productId = (String) input.readObject();
237
238         // Inoltra la richiesta al MagazzinoServer
239         String response = forwardDeleteProduct(productId);
240
241         // Invia la risposta al Produttore
242         output.writeObject(response);
243     } catch (IOException | ClassNotFoundException e) {
244         Log.error(e.getMessage());
245         try {
246             output.writeObject(Constants.FAILURE);
247         } catch (IOException ex) {
248             Log.error(ex.getMessage());
249         }
250     }
251 }
252
253 private String forwardInsertNewProductToMagazzinoServer(String
codiceMagazzino, String productId, int productQuantity, String
productName, String nomeMagazzino) throws IOException,
ClassNotFoundException {
254     try (
255         Socket magazzinoSocket = new Socket(Constants.HOSTNAME,
Constants.MAGAZZINO_SERVER_PORT);
256         ObjectOutputStream magazzinoOutput = new
ObjectOutputStream(magazzinoSocket.getOutputStream());
257         ObjectInputStream magazzinoInput = new ObjectInputStream
(magazzinoSocket.getInputStream())
258     ) {
259         // Invia richiesta di inserimento a MagazzinoServer
260         magazzinoOutput.writeObject(Constants.
MAGAZZINO_INSERT_NEW_PRODUCT);
261         magazzinoOutput.writeObject(codiceMagazzino);
262         magazzinoOutput.writeObject(productId);
263         magazzinoOutput.writeObject(productQuantity);
264         magazzinoOutput.writeObject(productName);
265         magazzinoOutput.writeObject(nomeMagazzino);
266
267         // Leggi la risposta da MagazzinoServer
268         return (String) magazzinoInput.readObject();
269     }
270 }
271
272 private String forwardUpdateProductToMagazzinoServer(String
productId, int newQuantity) throws IOException, ClassNotFoundException {
273     try (
274         Socket magazzinoSocket = new Socket(Constants.HOSTNAME,
Constants.MAGAZZINO_SERVER_PORT);
275         ObjectOutputStream magazzinoOutput = new
ObjectOutputStream(magazzinoSocket.getOutputStream());
276         ObjectInputStream magazzinoInput = new ObjectInputStream
(magazzinoSocket.getInputStream())
277     ) {
278         // Invia richiesta di aggiornamento a MagazzinoServer
279         magazzinoOutput.writeObject(Constants.
MAGAZZINO_UPDATE_PRODUCTS);

```

```

280         magazzinoOutput.writeObject(productId);
281         magazzinoOutput.writeObject(newQuantity);
282
283         // Leggi la risposta da MagazzinoServer
284         return (String) magazzinoInput.readObject();
285     }
286 }
287
288 // Metodo che inoltra la richiesta di eliminazione al
289 MagazzinoServer
290 private String forwardDeleteProduct(String productId) {
291     try (
292         Socket socket = new Socket(Constants.HOSTNAME, Constants
293         .MAGAZZINO_SERVER_PORT);
294         ObjectOutputStream magazzinoOutput = new
295         ObjectOutputStream(socket.getOutputStream());
296         ObjectInputStream magazzinoInput = new ObjectInputStream
297         (socket.getInputStream())
298     ) {
299         // Invia la richiesta al MagazzinoServer
300         magazzinoOutput.writeObject(Constants.
301         MAGAZZINO_DELETE_PRODUCTS);
302         magazzinoOutput.writeObject(productId);
303
304         // Ricevi la risposta dal MagazzinoServer
305         return (String) magazzinoInput.readObject();
306     } catch (IOException | ClassNotFoundException e) {
307         Log.error(e.getMessage());
308         return Constants.FAILURE;
309     }
310 }
311
312 private void forwardCloseRequestToMagazzinoServer() {
313     try (
314         Socket magazzinoSocket = new Socket(Constants.HOSTNAME,
315         Constants.MAGAZZINO_SERVER_PORT);
316         ObjectOutputStream magazzinoOutput = new
317         ObjectOutputStream(magazzinoSocket.getOutputStream());
318         ObjectInputStream magazzinoInput = new ObjectInputStream
319         (magazzinoSocket.getInputStream());
320     ) {
321         // Invia richiesta di chiusura a MagazzinoServer
322         magazzinoOutput.writeObject(Constants.EXIT);
323
324         // Attende conferma della chiusura
325         String response = (String) magazzinoInput.readObject();
326         if (Constants.SUCCESS.equals(response)) {
327             Log.info("Chiusura confermata da MagazzinoServer.");
328         }
329     } catch (IOException | ClassNotFoundException e) {
330         Log.error("Errore durante la chiusura di MagazzinoServer: "
331         + e.getMessage());
332     }
333 }
334
335 private void forwardCloseRequestToOrderServer() {

```

```

327         try (
328             Socket orderSocket = new Socket(Constants.HOSTNAME,
Constants.ORDINE_SERVER_PORT);
329             ObjectOutputStream orderOutput = new ObjectOutputStream(
orderSocket.getOutputStream());
330             ObjectInputStream orderInput = new ObjectInputStream(
orderSocket.getInputStream());
331         ) {
332             // Invia richiesta di chiusura a OrdineServer
333             orderOutput.writeObject(Constants.EXIT);
334
335             // Attende conferma della chiusura
336             String response = (String) orderInput.readObject();
337             if (Constants.SUCCESS.equals(response)) {
338                 Log.info("Chiusura confermata da OrdineServer.");
339             }
340         } catch (IOException | ClassNotFoundException e) {
341             Log.error("Errore durante la chiusura di OrdineServer: " + e
.getMessage());
342         }
343     }
344
345     private void forwardUpdateProductRequest(ObjectInputStream input,
ObjectOutputStream output) throws IOException, ClassNotFoundException,
SQLException {
346         try {
347             String productId = (String) input.readObject();
348             int newQuantity = (int) input.readObject();
349
350             // Inoltra la richiesta a MagazzinoServer
351             String response = forwardUpdateProductToMagazzinoServer(
productId, newQuantity);
352
353             // Invia la risposta a Produttore
354             output.writeObject(response);
355         } catch (IOException | ClassNotFoundException e) {
356             Log.error(e.getMessage());
357             output.writeObject(Constants.FAILURE);
358         }
359     }
360
361     private void prepareClosingClientAndServer(ObjectOutputStream output
) throws SQLException {
362         try {
363             output.writeObject(Constants.SUCCESS);
364             server.stop();
365         } catch (IOException e) {
366             Log.error(e.getMessage());
367         }
368     }
369 }
370
371 }
372 }

```

### 3.2.5 Inoltro delle richieste da DipendenteServer a MagazzinoServer

Listing 5: MagazzinoServer.java

```
1 package server;
2
3 import logger.Log;
4 import singleton.Database;
5 import utils.Constants;
6 import utils.Queries;
7
8 import java.io.IOException;
9 import java.io.ObjectInputStream;
10 import java.io.ObjectOutputStream;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13 import java.sql.*;
14 import java.util.ArrayList;
15 import java.util.List;
16
17 /**
18  * @description classe inerente al server MagazzinoServer per la gestione
19  * dei prodotti presenti nel magazzino
20  */
21 public class MagazzinoServer {
22
23     private ServerSocket serverSocket;
24     private boolean isRunning = true;
25
26     public MagazzinoServer(int port) throws IOException {
27         serverSocket = new ServerSocket(port);
28         Log.info(String.format(Constants.
29 SERVER_SOCKET_CONNECTION_ESTABLISHED, port));
30     }
31
32     public void start() throws IOException {
33         Log.info(Constants.SERVER_LISTENING);
34         while (isRunning) {
35             try {
36                 Socket clientSocket = serverSocket.accept();
37                 Log.info("Connessione stabilita con il client: " +
38 clientSocket.getInetAddress());
39                 new ClientHandler(this, clientSocket).start();
40             } catch (IOException e) {
41                 Log.error(e.getMessage());
42             }
43         }
44     }
45
46     public void stop() {
47         isRunning = false;
48         try {
49             if (serverSocket != null && !serverSocket.isClosed()) {
50                 serverSocket.close();
51             }
52             Log.info("Server chiuso.");
53         } catch (IOException e) {
```

```

51         Log.error("Errore durante la chiusura del server: " + e.
getMessage());
52     }
53 }
54
55 public static void main(String[] args) throws IOException {
56     MagazzinoServer server = new MagazzinoServer(Constants.
MAGAZZINO_SERVER_PORT);
57     server.start();
58 }
59
60 private static class ClientHandler extends Thread {
61
62     private Socket clientSocket;
63     private MagazzinoServer server;
64
65     public ClientHandler(MagazzinoServer server, Socket socket) {
66         this.server = server;
67         this.clientSocket = socket;
68     }
69
70     public void run() {
71         try (
72             ObjectOutputStream output = new ObjectOutputStream(
clientSocket.getOutputStream());
73             ObjectInputStream input = new ObjectInputStream(
clientSocket.getInputStream())
74         ) {
75             while (true) {
76                 try {
77                     String request = (String) input.readObject();
78                     switch (request) {
79                         case Constants.MAGAZZINO_INSERT_NEW_PRODUCT:
80                             prepareInsertProductRequest(input, output);
81                             break;
82                         case Constants.MAGAZZINO_UPDATE_PRODUCTS:
83                             prepareUpdateProductRequest(input, output);
84                             break;
85                         case Constants.MAGAZZINO_VIEW_PRODUCTS:
86                             prepareViewProductsRequest(output);
87                             break;
88                         case Constants.MAGAZZINO_DELETE_PRODUCTS:
89                             prepareDeleteProductRequest(input, output);
90                             break;
91                         case Constants.EXIT:
92                             output.writeObject(Constants.SUCCESS);
93                             server.stop();
94                             return;
95                         default:
96                             output.writeObject(Constants.UNKNOWN_REQUEST
);
97                             break;
98                     }
99                 } catch (ClassNotFoundException | SQLException e) {
100                     Log.error("Errore durante la gestione del client: "
+ e.getMessage());

```



```

101         output.writeObject(Constants.FAILURE);
102     }
103 }
104 } catch (IOException e) {
105     Log.error("Errore di I/O: " + e.getMessage());
106 } finally {
107     try {
108         clientSocket.close();
109     } catch (IOException e) {
110         Log.error("Errore durante la chiusura del socket: " + e.
getMessage());
111     }
112     Log.info("Connessione chiusa con il client");
113 }
114 }
115
116 private void prepareInsertProductRequest(ObjectInputStream input,
ObjectOutputStream output) throws SQLException, IOException {
117     try {
118         // Ricevi i parametri dal DipendenteServer
119         String codiceMagazzino = (String) input.readObject();
120         String productId = (String) input.readObject();
121         int productQuantity = (int) input.readObject();
122         String productName = (String) input.readObject();
123         String nomeMagazzino = (String) input.readObject();
124
125         // Inserisci il prodotto nel magazzino
126         String response = insertProduct(codiceMagazzino, productId,
productQuantity, productName, nomeMagazzino);
127
128         // Invia la risposta al DipendenteServer
129         output.writeObject(response);
130     } catch (IOException | ClassNotFoundException e) {
131         Log.error(e.getMessage());
132         output.writeObject(Constants.FAILURE);
133     }
134 }
135
136 private void prepareViewProductsRequest(ObjectOutputStream output)
throws SQLException {
137     try {
138         // Ottieni la lista dei prodotti
139         List<String> products = viewProducts();
140
141         // Invia la lista di prodotti indietro a DipendenteServer
142         output.writeObject(products);
143     } catch (IOException e) {
144         Log.error("Errore durante l'invio della lista di prodotti: "
+ e.getMessage());
145     }
146 }
147
148 private void prepareUpdateProductRequest(ObjectInputStream input,
ObjectOutputStream output) throws SQLException, IOException {
149     try {
150         // Ricevi l'ID del prodotto e la nuova quantita

```

```

151         String productId = (String) input.readObject();
152         int newQuantity = (int) input.readObject();
153
154         // Effettua l'aggiornamento del prodotto
155         String response = updateProduct(productId, newQuantity);
156
157         // Invia la risposta a DipendenteServer
158         output.writeObject(response);
159
160     } catch (IOException | ClassNotFoundException e) {
161         Log.error(e.getMessage());
162         output.writeObject(Constants.FAILURE);
163     }
164 }
165
166 private void prepareDeleteProductRequest(ObjectInputStream input,
ObjectOutputStream output) {
167     try {
168         // Ricevi l'ID del prodotto dal DipendenteServer
169         String productId = (String) input.readObject();
170
171         // Esegui l'eliminazione del prodotto dal database
172         String response = deleteProduct(productId);
173
174         // Invia la risposta al DipendenteServer
175         output.writeObject(response);
176     } catch (IOException | ClassNotFoundException | SQLException e)
{
177         Log.error(e.getMessage());
178         try {
179             output.writeObject(Constants.FAILURE);
180         } catch (IOException ex) {
181             Log.error(ex.getMessage());
182         }
183     }
184 }
185
186
187 // Metodo che esegue la cancellazione del prodotto dal database
188 private String deleteProduct(String productId) throws SQLException {
189     Connection connection = null;
190     PreparedStatement statement = null;
191
192     try {
193         connection = Database.getInstance().getConnection();
194
195         String query = Queries.TBL_MAGAZZINO_DELETE_PRODUCT_QUERY;
196         statement = connection.prepareStatement(query);
197         statement.setString(1, productId);
198
199         int rowsAffected = statement.executeUpdate();
200
201         if (rowsAffected > 0) {
202             return Constants.SUCCESS;
203         } else {
204             return Constants.FAILURE;

```

```

205     }
206     } finally {
207         Database.closeConnection(connection, statement, null);
208     }
209 }
210
211 private String insertProduct(String codiceMagazzino, String
productId, int productQuantity, String productName, String nomeMagazzino)
throws SQLException {
212     Connection connection = null;
213     PreparedStatement statement = null;
214
215     try {
216         connection = Database.getInstance().getConnection();
217         String query = Queries.
TBL_MAGAZZINO_INSERT_NEW_PRODUCT_QUERY;
218         statement = connection.prepareStatement(query);
219         statement.setString(1, codiceMagazzino);
220         statement.setString(2, productId);
221         statement.setInt(3, productQuantity);
222         statement.setString(4, productName);
223         statement.setString(5, nomeMagazzino);
224
225         int rowsAffected = statement.executeUpdate();
226
227         if (rowsAffected > 0) {
228             return Constants.SUCCESS;
229         } else {
230             return Constants.FAILURE;
231         }
232     } finally {
233         Database.closeConnection(connection, statement, null);
234     }
235 }
236
237 /**
238  * @param productId
239  * @param newQuantity
240  * @return true se l'aggiornamento    avvenuto con successo,
altrimenti false
241  * @throws SQLException
242  */
243 private String updateProduct(String productId, int newQuantity)
throws SQLException {
244     Connection connection = null;
245     PreparedStatement preparedStatement = null;
246
247     try {
248         connection = Database.getInstance().getConnection();
249         String updateQuery = Queries.
TBL_MAGAZZINO_UPDATE_PRODUCT_QUANTITY_QUERY;
250         preparedStatement = connection.prepareStatement(updateQuery)
;
251         preparedStatement.setInt(1, newQuantity);
252         preparedStatement.setString(2, productId);
253

```

```

254         int rowsUpdated = preparedStatement.executeUpdate();
255         if (rowsUpdated > 0) {
256             Log.info(String.format("Prodotto %s aggiornato con
successo a quantita : %d", productId, newQuantity));
257             return Constants.SUCCESS;
258         } else {
259             Log.error("Nessun prodotto trovato con ID: " + productId
);
260             return Constants.FAILURE;
261         }
262     } finally {
263         Database.closeConnection(connection, preparedStatement, null
);
264     }
265 }
266
267 private List<String> viewProducts() throws SQLException {
268     Connection connection = null;
269     Statement statement = null;
270     ResultSet resultSet = null;
271
272     List<String> products = new ArrayList<>();
273     try {
274         connection = Database.getInstance().getConnection();
275         statement = connection.createStatement();
276         resultSet = statement.executeQuery(String.format(Queries.
GENERIC_QUERY_SELECT, Constants.TBL_MAGAZZINO));
277
278         while (resultSet.next()) {
279             String codiceMagazzino = resultSet.getString("
CODICE_MAGAZZINO");
280             String id = resultSet.getString("ID_PRODOTTO");
281             int quantita = resultSet.getInt("QUANTITA_PRODOTTO");
282             String nome = resultSet.getString("NOME_PRODOTTO");
283             String nomeMagazzino = resultSet.getString("
NOME_MAGAZZINO");
284
285             String product = String.format("ID: %s, Nome: %s,
Quantita : %d, Codice magazzino: %s, Nome magazzino: %s", id, nome,
quantita, codiceMagazzino, nomeMagazzino);
286             products.add(product);
287         }
288     } finally {
289         Database.closeConnection(connection, statement, resultSet);
290     }
291     return products;
292 }
293 }
294 }

```

### 3.2.6 Inoltro delle richieste da DipendenteServer a OrdineServer

Listing 6: OrdineServer.java

```
1 package server;
2
3 import logger.Log;
4 import singleton.Database;
5 import utils.Constants;
6 import utils.Queries;
7
8 import java.io.IOException;
9 import java.io.ObjectInputStream;
10 import java.io.ObjectOutputStream;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13 import java.sql.*;
14 import java.util.ArrayList;
15 import java.util.List;
16
17 /**
18  * @description classe inerente al server OrdineServer, per la gestione
19  * degli ordini da parte del produttore
20  */
21 public class OrdineServer {
22
23     private ServerSocket serverSocket;
24     private boolean isRunning = true;
25
26     public OrdineServer(int port) throws IOException {
27         serverSocket = new ServerSocket(port);
28         Log.info(String.format(Constants.
29 SERVER_SOCKET_CONNECTION_ESTABLISHED, port));
30     }
31
32     public void start() throws IOException {
33         Log.info(Constants.SERVER_LISTENING);
34         while (isRunning) {
35             try {
36                 Socket clientSocket = serverSocket.accept();
37                 Log.info("Connessione stabilita con il client: " +
38 clientSocket.getInetAddress());
39                 new ClientHandler(this, clientSocket).start();
40             } catch (IOException e) {
41                 Log.error(e.getMessage());
42             }
43         }
44     }
45
46     public void stop() {
47         isRunning = false;
48         try {
49             if (serverSocket != null && !serverSocket.isClosed()) {
50                 serverSocket.close();
51             }
52             Log.info("Server chiuso.");
53         } catch (IOException e) {
```

```

51         Log.error("Errore durante la chiusura del server: " + e.
getMessage());
52     }
53 }
54
55 public static void main(String[] args) throws IOException {
56     OrdineServer server = new OrdineServer(Constants.ORDINE_SERVER_PORT)
;
57     server.start();
58 }
59
60 private static class ClientHandler extends Thread {
61
62     private Socket clientSocket;
63     private OrdineServer server;
64
65     public ClientHandler(OrdineServer server, Socket socket) {
66         this.server = server;
67         this.clientSocket = socket;
68     }
69
70     public void run() {
71         try (
72             ObjectOutputStream output = new ObjectOutputStream(
clientSocket.getOutputStream());
73             ObjectInputStream input = new ObjectInputStream(
clientSocket.getInputStream())
74         ) {
75             while (true) {
76                 try {
77                     String request = (String) input.readObject();
78                     switch (request) {
79                         case Constants.ORDER_VIEW_LIST:
80                             prepareViewOrderListRequest(output);
81                             break;
82                         case Constants.ORDER_VIEW_STATUS_LIST:
83                             prepareViewOrderStatusListRequest(output);
84                             break;
85                         case Constants.ORDER_UPDATE_STATUS:
86                             prepareUpdateOrderStatusRequest(input,
output);
87
88                         case Constants.EXIT:
89                             output.writeObject(Constants.SUCCESS);
90                             server.stop();
91                             return;
92                         default:
93                             output.writeObject(Constants.UNKNOWN_REQUEST
);
94
95                             break;
96                     }
97                 } catch (ClassNotFoundException | SQLException e) {
98                     Log.error("Errore durante la gestione del client: "
+ e.getMessage());
99                     output.writeObject(Constants.FAILURE);
100                 }
101             }
102         }
103     }
104 }

```

```

100         } catch (IOException e) {
101             Log.error("Errore di I/O: " + e.getMessage());
102         } finally {
103             try {
104                 clientSocket.close();
105             } catch (IOException e) {
106                 Log.error("Errore durante la chiusura del socket: " + e.
getMessage());
107             }
108             Log.info("Connessione chiusa con il client");
109         }
110     }
111
112     private void prepareViewOrderListRequest(ObjectOutputStream output)
throws SQLException {
113         try {
114             // Ottieni la lista dei prodotti
115             List<String> orders = viewOrders();
116
117             // Invia la lista di prodotti indietro a DipendenteServer
118             output.writeObject(orders);
119         } catch (IOException e) {
120             Log.error("Errore durante l'invio della lista di prodotti: "
+ e.getMessage());
121         }
122     }
123
124     private void prepareViewOrderStatusListRequest(ObjectOutputStream
output) throws SQLException {
125         try {
126             // Ottieni la lista dei prodotti
127             List<String> status = viewOrderStatus();
128
129             // Invia la lista di prodotti indietro a DipendenteServer
130             output.writeObject(status);
131         } catch (IOException e) {
132             Log.error("Errore durante l'invio della lista di prodotti: "
+ e.getMessage());
133         }
134     }
135
136     private void prepareUpdateOrderStatusRequest(ObjectInputStream input
, ObjectOutputStream output) throws SQLException {
137         try {
138             Integer orderId = (Integer) input.readObject();
139             String orderStatus = (String) input.readObject();
140             boolean updated = updateOrderStatus(orderId, orderStatus);
141             if (updated) {
142                 output.writeObject(Constants.SUCCESS);
143                 Log.info("Stato ordine aggiornato con successo.");
144             } else {
145                 output.writeObject(Constants.FAILURE);
146                 Log.warning("Aggiornamento dell'ordine fallito.");
147             }
148         } catch (SQLException | ClassNotFoundException |
RuntimeException | IOException e) {

```

```

149         Log.error(e.getMessage());
150     }
151 }
152
153 private List<String> viewOrders() throws SQLException {
154     Connection connection = null;
155     Statement statement = null;
156     ResultSet resultSet = null;
157
158     List<String> orders = new ArrayList<>();
159     try {
160         connection = Database.getInstance().getConnection();
161         statement = connection.createStatement();
162         resultSet = statement.executeQuery(String.format(Queries.
163         GENERIC_QUERY_SELECT, Constants.TBL_ORDINE));
164
165         while (resultSet.next()) {
166             Integer id = resultSet.getInt("ID");
167             String fattura = resultSet.getString("FATTURA");
168             Integer numeroOrdine = resultSet.getInt("NUMERO_ORDINE")
169 ;
170             Integer idCarrello = resultSet.getInt("ID_CARRELLO");
171             String indirizzoSpedizione = resultSet.getString("
172             INDIRIZZO_SPEDIZIONE");
173             String statoOrdine = resultSet.getString("STATO_ORDINE")
174 ;
175
176             String order = String.format("ID: %d, Fattura: %s,
177             Numero ordine: %d, ID carrello: %d, Indirizzo spedizione: %s, Stato
178             ordine: %s", id, fattura, numeroOrdine, idCarrello, indirizzoSpedizione,
179             statoOrdine);
180
181             orders.add(order);
182         }
183     } finally {
184         Database.closeConnection(connection, statement, resultSet);
185     }
186     return orders;
187 }
188
189 private List<String> viewOrderStatus() throws SQLException {
190     Connection connection = null;
191     Statement statement = null;
192     ResultSet resultSet = null;
193
194     List<String> orderStatus = new ArrayList<>();
195     try {
196         connection = Database.getInstance().getConnection();
197         statement = connection.createStatement();
198         resultSet = statement.executeQuery(String.format(Queries.
199         GENERIC_QUERY_SELECT, Constants.TBL_ORDINE));
200
201         while (resultSet.next()) {
202             int id = resultSet.getInt("ID");
203             String statoOrdine = resultSet.getString("STATO_ORDINE")
204 ;

```



```

196         String status = String.format("ID: %s, Stato ordine: %s"
, id, statoOrdine);
197         orderStatus.add(status);
198     }
199     } finally {
200         Database.closeConnection(connection, statement, resultSet);
201     }
202     return orderStatus;
203 }
204
205 /**
206  * @param orderNumber
207  * @param orderStatus
208  * @return true se l'operazione e' andata a buon fine, altrimenti
false
209  * @throws SQLException
210  * @description aggiorna lo stato di un ordine
211  */
212 private boolean updateOrderStatus(Integer orderId, String
orderStatus) throws SQLException {
213     Connection connection = null;
214     PreparedStatement preparedStatement = null;
215
216     try {
217         connection = Database.getInstance().getConnection();
218         preparedStatement = connection.prepareStatement(Queries.
TBL_ORDER_UPDATE_ORDER);
219         preparedStatement.setString(1, orderStatus);
220         preparedStatement.setInt(2, orderId);
221
222
223         int rowsAffected = preparedStatement.executeUpdate();
224
225         return rowsAffected > 0;
226
227     } catch (SQLException e) {
228         Log.error(e.getMessage());
229     } finally {
230         Database.closeConnection(connection, preparedStatement, null
);
231     }
232
233     return false;
234 }
235
236 }
237
238 }

```

## 4 Manuale utente con istruzioni di compilazione ed esecuzione annesse

Questo manuale fornisce le istruzioni per la compilazione ed esecuzione del progetto Experia Coffee e dei server associati. 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:

- IntelliJ IDEA: per l'importazione e l'esecuzione dei progetti in Java.
- Avere le seguenti porte libere: 8080, 8081, 13000, 6000

### 4.2 Download e Configurazione dei Progetti

Il progetto Experia Coffee e Experia-Coffee-Reti sono disponibili su GitHub. Seguire i link per scaricare entrambi i progetti. (opzionale).

Una volta approdati sulla pagina di GitHub interessata, è possibile seguire la procedura analoga descritta all'interno del file **README.md**.

**Experia Coffee** <https://github.com/Daevel/Experia-Coffe-Prog3-Lab>.

**Experia Coffee Reti** <https://github.com/Daevel/Experia-Coffee-Reti>.

### 4.3 Esecuzione

Una volta scaricato il progetto verrà già stanziato con le relative configurazioni eseguibili. In IntelliJ basterà aprire la sezione delle configurazioni ed eseguirle nel seguente ordine:

- Eseguire il compound: **Database Sync**
- Eseguire il compound: **Avvio Server Cluster**
- Eseguire il compound: **Avvio Clients**

Per interagire con il programma è necessario sfruttare la console di Produttore o Dipendente.

### 4.3.1 Compound Database

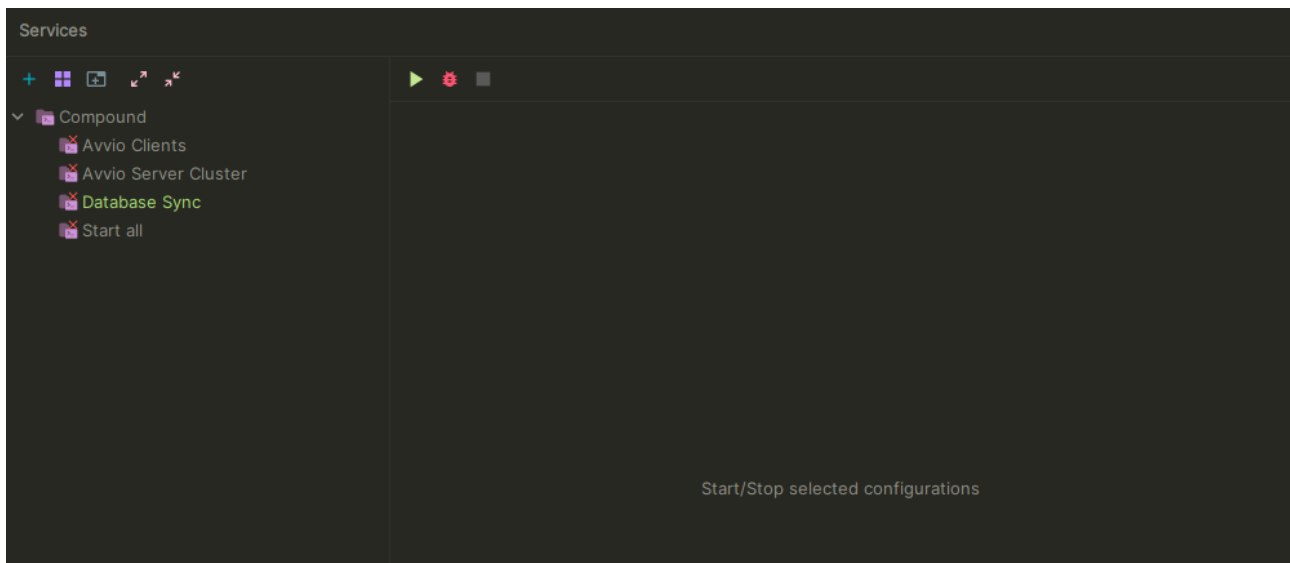


Figura 3: Lista dei compound: selezione di Database Sync

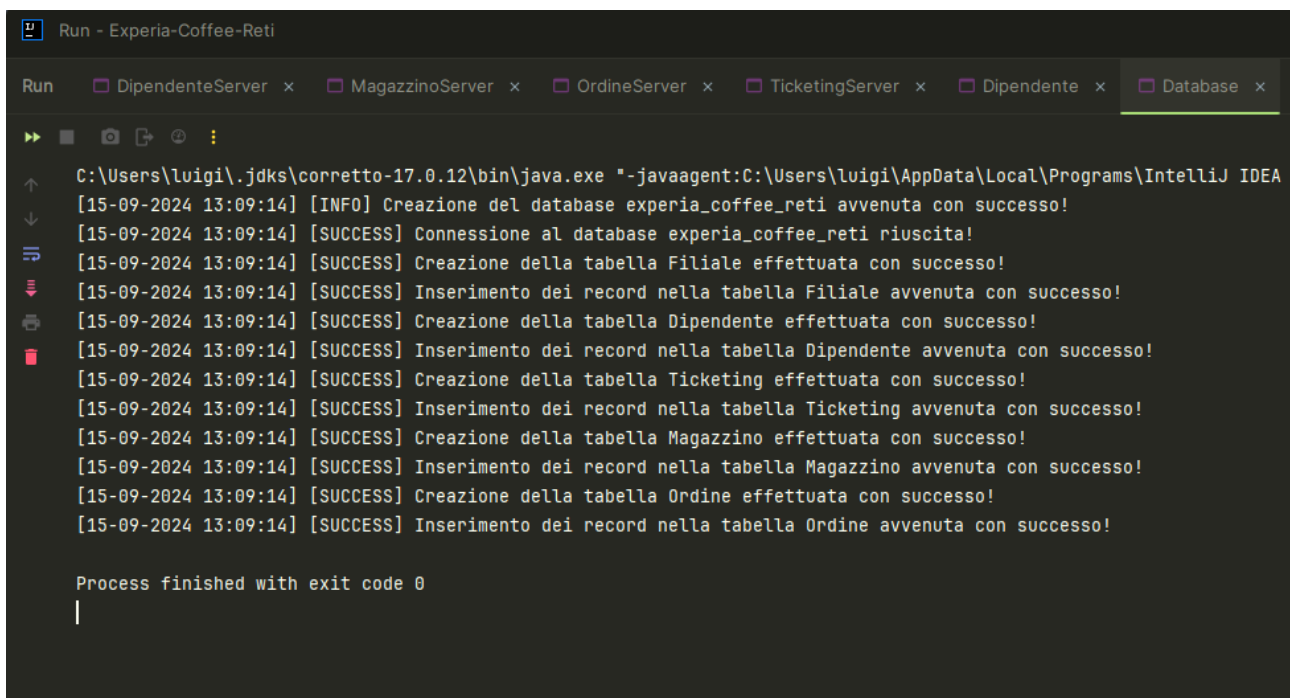


Figura 4: Log avvio della classe Database.java

### 4.3.2 Compound Avvio Server Cluster

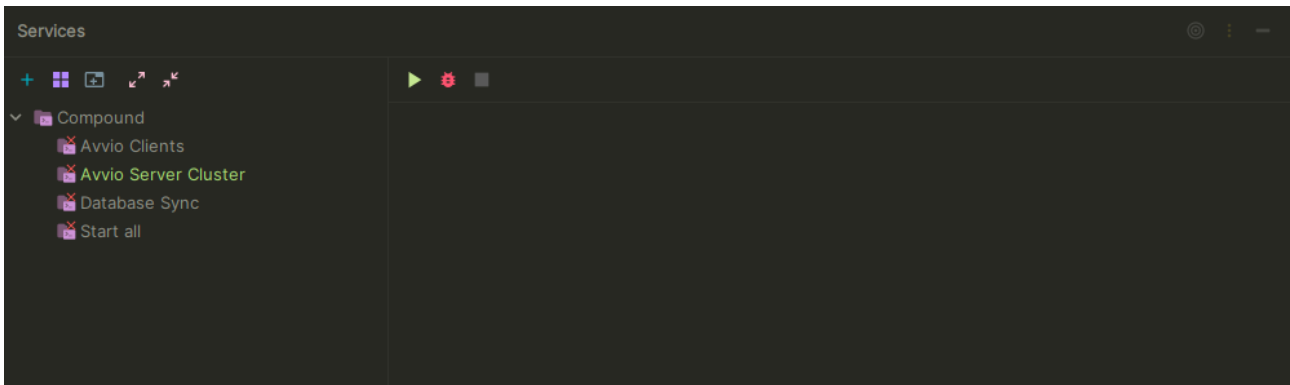


Figura 5: Lista dei compound: selezione di Avvio Server Cluster

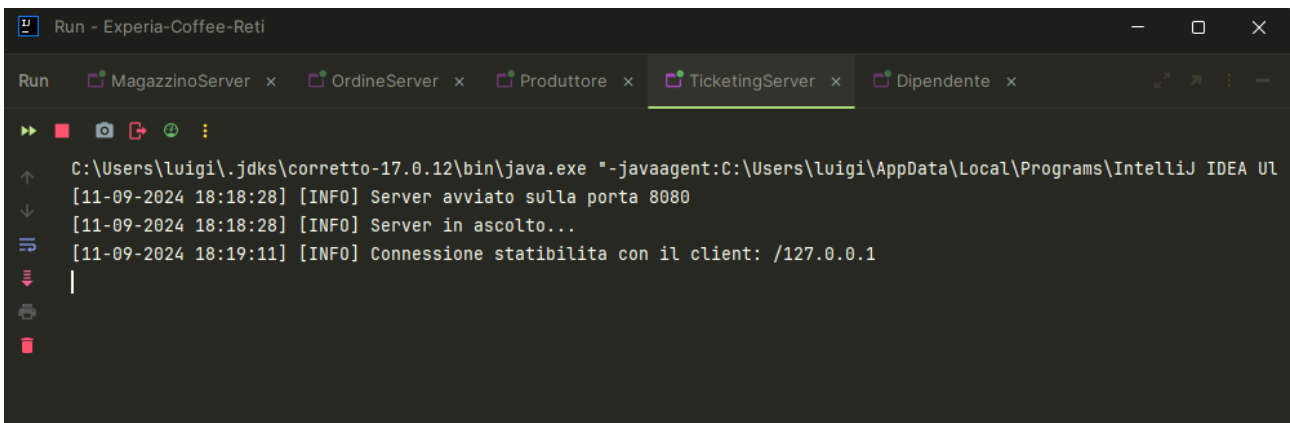
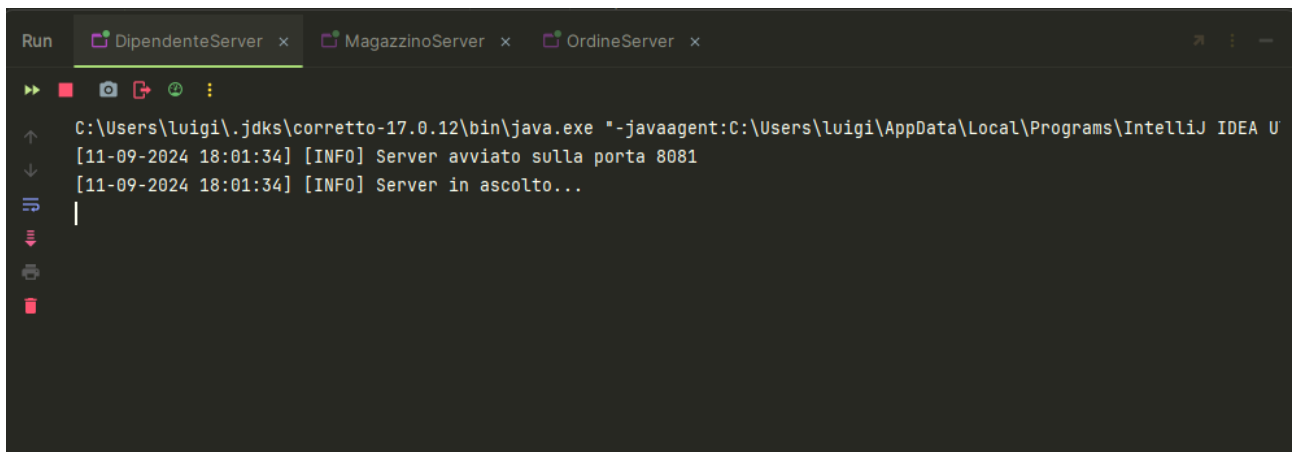


Figura 6: Log avvio della classe TicketingServer.java

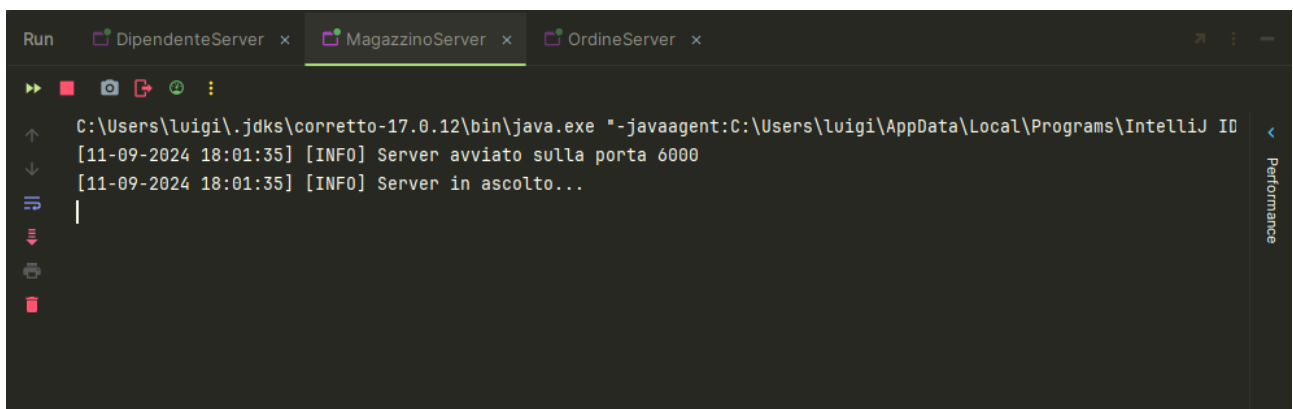


The screenshot shows the IntelliJ IDEA Run console with three tabs: DipendenteServer, MagazzinoServer, and OrdineServer. The DipendenteServer tab is active. The console output shows the Java command and the server startup logs for DipendenteServer.

```
Run  DipendenteServer x  MagazzinoServer x  OrdineServer x

C:\Users\luigi\jdk\corretto-17.0.12\bin\java.exe "-javaagent:C:\Users\luigi\AppData\Local\Programs\IntelliJ IDEA U
[11-09-2024 18:01:34] [INFO] Server avviato sulla porta 8081
[11-09-2024 18:01:34] [INFO] Server in ascolto...
```

Figura 7: Log avvio della classe DipendenteServer.java

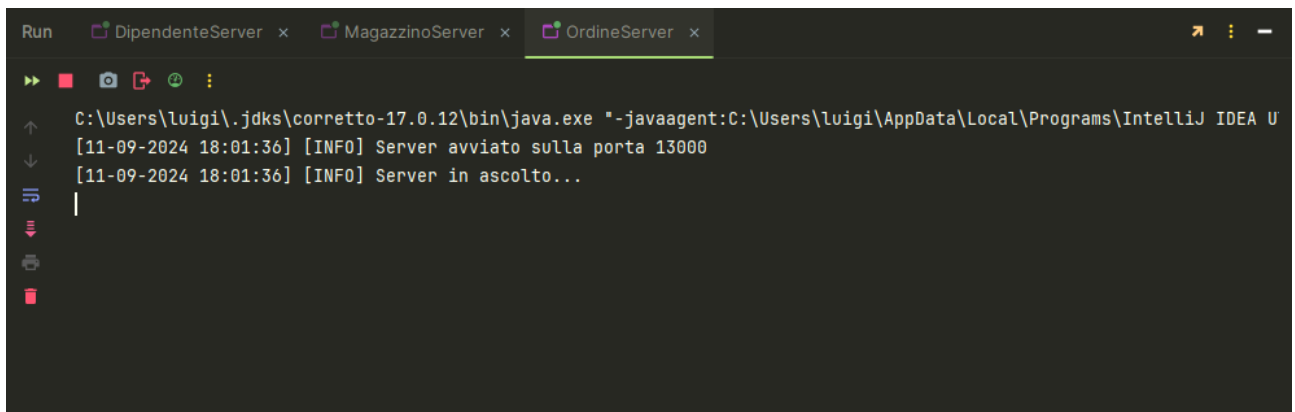


The screenshot shows the IntelliJ IDEA Run console with three tabs: DipendenteServer, MagazzinoServer, and OrdineServer. The MagazzinoServer tab is active. The console output shows the Java command and the server startup logs for MagazzinoServer.

```
Run  DipendenteServer x  MagazzinoServer x  OrdineServer x

C:\Users\luigi\jdk\corretto-17.0.12\bin\java.exe "-javaagent:C:\Users\luigi\AppData\Local\Programs\IntelliJ ID
[11-09-2024 18:01:35] [INFO] Server avviato sulla porta 6000
[11-09-2024 18:01:35] [INFO] Server in ascolto...
```

Figura 8: Log avvio della classe MagazzinoServer.java



```
Run  DipendenteServer x  MagazzinoServer x  OrdineServer x
>> ■ 📷 🔄 🕒 ⋮
C:\Users\luigi\.jdk\corretto-17.0.12\bin\java.exe "-javaagent:C:\Users\luigi\AppData\Local\Programs\IntelliJ IDEA U
[11-09-2024 18:01:36] [INFO] Server avviato sulla porta 13000
[11-09-2024 18:01:36] [INFO] Server in ascolto...
|
```

Figura 9: Log avvio della classe OrdineServer.java

### 4.3.3 Compound Avvio Clients

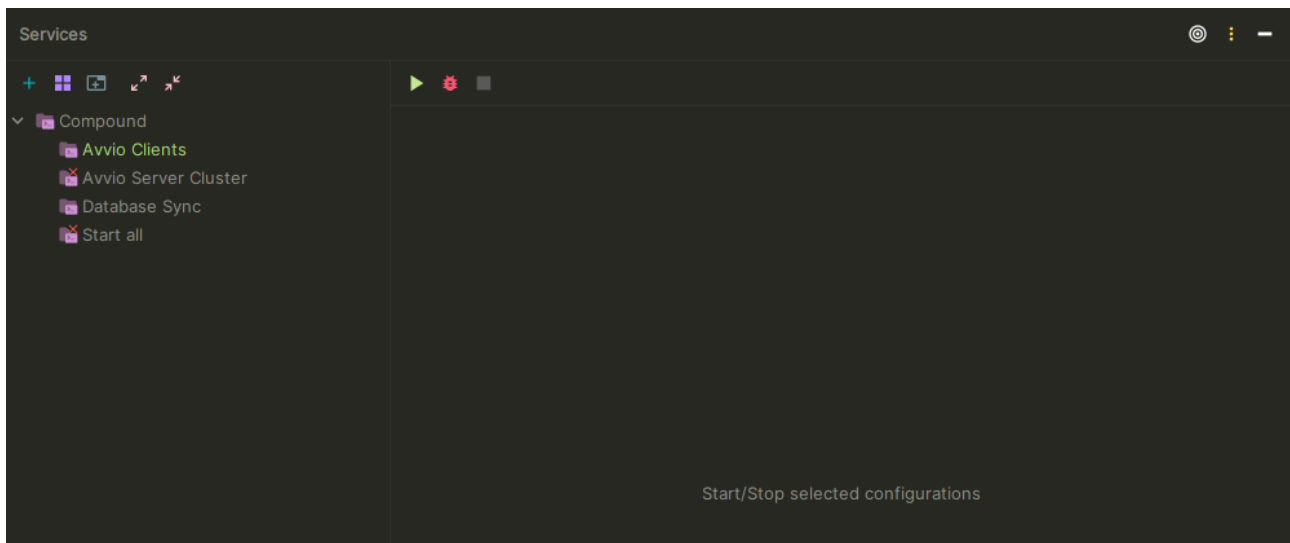
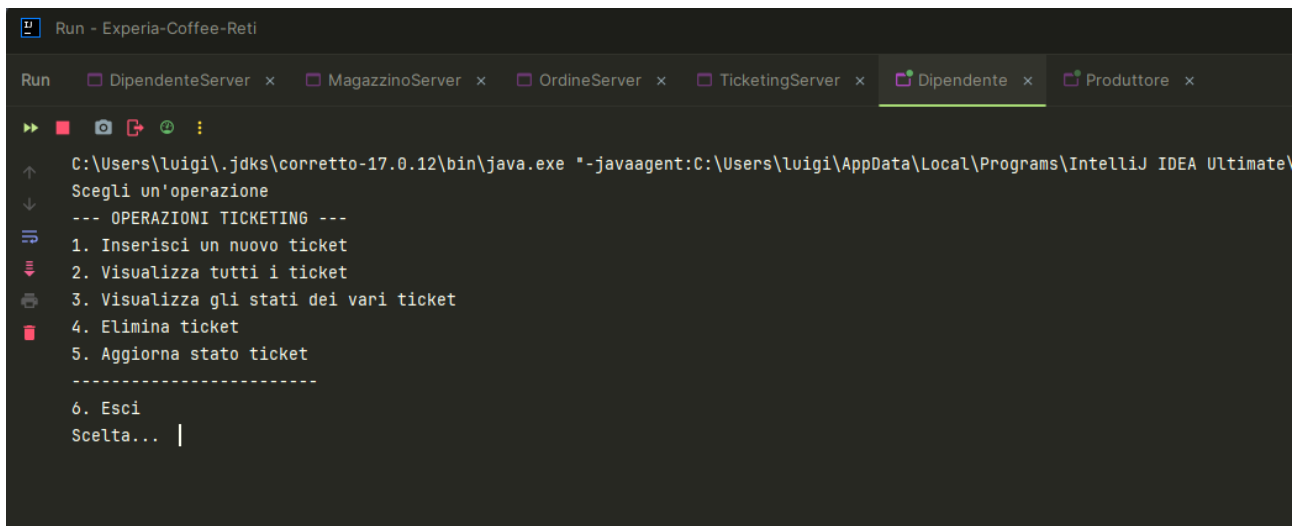


Figura 10: Lista dei compound: selezione di Avvio Clients



Figura 11: Log avvio della classe Produttore.java



```
Run - Experia-Coffee-Reti
Run  DipendenteServer x  MagazzinoServer x  OrdineServer x  TicketingServer x  Dipendente x  Produttore x

C:\Users\luigi\.jdk\corretto-17.0.12\bin\java.exe "-javaagent:C:\Users\luigi\AppData\Local\Programs\IntelliJ IDEA Ultimate\
Scegli un'operazione
--- OPERAZIONI TICKETING ---
1. Inserisci un nuovo ticket
2. Visualizza tutti i ticket
3. Visualizza gli stati dei vari ticket
4. Elimina ticket
5. Aggiorna stato ticket
-----
6. Esci
Scelta... |
```

Figura 12: Log avvio della classe Dipendente.java



### 4.3.4 Simulazione aggiornamento stato ticket [Dipendente - Ticketing Server]

```
Run - Experia-Coffee-Net
Run  DipendenteServer x  MagazzinoServer x  OrdineServer x  TicketingServer x  Dipendente x  Produttore x

C:\Users\luigi\jdk\corretto-17.0.12\bin\java.exe "-javaagent:C:\Users\luigi\AppData\Local\Programs\IntelliJ IDEA Ultimate\lib\idea_rt.jar=62586:C:\Users\luigi\AppData\Local\Programs\IntelliJ IDEA Ultimate\bin" -Dfile.encoding=UTF-8
Scegli un'operazione
--- OPERAZIONI TICKETING ---
1. Inserisci un nuovo ticket
2. Visualizza tutti i ticket
3. Visualizza gli stati dei vari ticket
4. Elimina ticket
5. Aggiorna stato ticket
-----
6. Esci
Sceglia... 2
Tickets disponibili:
ID: 1, Titolo: Richiesta assistenza macchina caffè, Descrizione: La macchina non si accende., Gestito da: m.rossi@experiacoffee.it, Creato da: cliente1@gmail.com, Data creazione: 2024-09-01, Stato: Non gestito
ID: 2, Titolo: Manutenzione urgente, Descrizione: Richiesta di manutenzione per macchina difettosa., Gestito da: f.blanchi@experiacoffee.it, Creato da: cliente2@yahoo.com, Data creazione: 2024-09-02, Stato: Gestito
ID: 3, Titolo: Aggiornamento software, Descrizione: Errore durante aggiornamento del software., Gestito da: l.verdi@experiacoffee.it, Creato da: cliente3@hotmail.com, Data creazione: 2024-09-03, Stato: Non gestito
ID: 4, Titolo: Ritiro prodotto, Descrizione: Il prodotto non è stato consegnato., Gestito da: p.neri@experiacoffee.it, Creato da: cliente4@libero.it, Data creazione: 2024-09-04, Stato: In lavorazione
ID: 5, Titolo: Problema cialde, Descrizione: Le cialde non si incastrano nella macchina., Gestito da: v.gialli@experiacoffee.it, Creato da: cliente5@live.com, Data creazione: 2024-09-05, Stato: Non gestito
ID: 6, Titolo: Supporto tecnico, Descrizione: Richiesta di supporto tecnico per nuovo modello., Gestito da: a.ferri@experiacoffee.it, Creato da: cliente6@outlook.com, Data creazione: 2024-09-06, Stato: Gestito
ID: 7, Titolo: Richiesta di sostituzione, Descrizione: La macchina è difettosa e richiede sostituzione., Gestito da: m.silveri@experiacoffee.it, Creato da: cliente7@gmail.com, Data creazione: 2024-09-07, Stato: Non gestito
ID: 8, Titolo: Assistenza capsule, Descrizione: Problema con la capsula che non viene perforata., Gestito da: d.marroni@experiacoffee.it, Creato da: cliente8@yahoo.com, Data creazione: 2024-09-08, Stato: In lavorazione
ID: 9, Titolo: Surriscaldamento, Descrizione: La macchina si surriscalda dopo 10 minuti., Gestito da: g.blanchi@experiacoffee.it, Creato da: cliente9@libero.it, Data creazione: 2024-09-09, Stato: Non gestito
ID: 10, Titolo: Problema di connessione, Descrizione: Impossibile connettere la macchina al Wi-Fi., Gestito da: r.grigi@experiacoffee.it, Creato da: cliente10@hotmail.com, Data creazione: 2024-09-10, Stato: Gestito
ID: 11, Titolo: Riparazione urgente, Descrizione: La macchina si è bloccata completamente., Gestito da: s.brunetti@experiacoffee.it, Creato da: cliente11@gmail.com, Data creazione: 2024-09-11, Stato: Non gestito
ID: 12, Titolo: Errore nel programma di pulizia, Descrizione: Il programma di pulizia non parte., Gestito da: n.lupi@experiacoffee.it, Creato da: cliente12@yahoo.com, Data creazione: 2024-09-12, Stato: In lavorazione
ID: 13, Titolo: Problema di rumorosità, Descrizione: La macchina è molto rumorosa durante l'uso., Gestito da: b.volpi@experiacoffee.it, Creato da: cliente13@libero.it, Data creazione: 2024-09-13, Stato: Non gestito
ID: 14, Titolo: Richiesta di rimborso, Descrizione: Non sono soddisfatto del prodotto, richiedo rimborso., Gestito da: e.guerrieri@experiacoffee.it, Creato da: cliente14@live.com, Data creazione: 2024-09-14, Stato: Gestito
ID: 15, Titolo: Macchina lenta, Descrizione: Richiesta di assistenza per lentezza di funzionamento., Gestito da: m.tosti@experiacoffee.it, Creato da: cliente15@outlook.com, Data creazione: 2024-09-15, Stato: Non gestito
ID: 16, Titolo: Problema con erogazione, Descrizione: La macchina non eroga correttamente il caffè., Gestito da: a.vitali@experiacoffee.it, Creato da: cliente16@gmail.com, Data creazione: 2024-09-16, Stato: In lavorazione
ID: 17, Titolo: Impossibile accendere la macchina, Descrizione: Richiesta urgente di assistenza per blocco., Gestito da: v.rubini@experiacoffee.it, Creato da: cliente17@yahoo.com, Data creazione: 2024-09-17, Stato: Non gestito
ID: 18, Titolo: Caffè freddo, Descrizione: La macchina non riscalda l'acqua a sufficienza., Gestito da: l.brunelli@experiacoffee.it, Creato da: cliente18@hotmail.com, Data creazione: 2024-09-18, Stato: Gestito
ID: 19, Titolo: Problema con il vapore, Descrizione: Non esce vapore per schiumare il latte., Gestito da: g.marchi@experiacoffee.it, Creato da: cliente19@gmail.com, Data creazione: 2024-09-19, Stato: Non gestito
ID: 20, Titolo: Manutenzione programmata, Descrizione: Richiesta di manutenzione programmata., Gestito da: c.naldi@experiacoffee.it, Creato da: cliente20@yahoo.com, Data creazione: 2024-09-20, Stato: In lavorazione
Scegli un'operazione
--- OPERAZIONI TICKETING ---
1. Inserisci un nuovo ticket
2. Visualizza tutti i ticket
3. Visualizza gli stati dei vari ticket
4. Elimina ticket
5. Aggiorna stato ticket
-----
6. Esci
Sceglia... |
```

Figura 13: ID: 19, Stato: "Non gestito"

```
--- OPERAZIONI TICKETING ---
1. Inserisci un nuovo ticket
2. Visualizza tutti i ticket
3. Visualizza gli stati dei vari ticket
4. Elimina ticket
5. Aggiorna stato ticket
-----
6. Esci
Scelta... 5
Inserisci l'ID del ticket da aggiornare: 19
Inserisci lo stato da aggiornare: In lavorazione
[15-09-2024 13:12:57] [SUCCESS] Il ticket è stato aggiornato con successo.
Scegli un'operazione
--- OPERAZIONI TICKETING ---
1. Inserisci un nuovo ticket
2. Visualizza tutti i ticket
3. Visualizza gli stati dei vari ticket
4. Elimina ticket
5. Aggiorna stato ticket
-----
6. Esci
Scelta... 2
Ticket disponibili:
ID: 1, Titolo: Richiesta assistenza macchina caffè, Descrizione: La macchina non si accende., Gestito da: m.rossi@experiaccoffee.it, Creato da: cliente1@gmail.com, Data creazione: 2024-09-01, Stato: Non gestito
ID: 2, Titolo: Manutenzione urgente, Descrizione: Richiesta di manutenzione per macchina difettosa., Gestito da: f.bianchi@experiaccoffee.it, Creato da: cliente2@yahoo.com, Data creazione: 2024-09-02, Stato: Gestito
ID: 3, Titolo: Aggiornamento software, Descrizione: Errore durante aggiornamento del software., Gestito da: l.verdi@experiaccoffee.it, Creato da: cliente3@hotmail.com, Data creazione: 2024-09-03, Stato: Non gestito
ID: 4, Titolo: Ritiro prodotto, Descrizione: Il prodotto non è stato consegnato., Gestito da: p.neri@experiaccoffee.it, Creato da: cliente4@libero.it, Data creazione: 2024-09-04, Stato: In lavorazione
ID: 5, Titolo: Problema cialde, Descrizione: Le cialde non si incastrano nella macchina., Gestito da: v.gialli@experiaccoffee.it, Creato da: cliente5@live.com, Data creazione: 2024-09-05, Stato: Non gestito
ID: 6, Titolo: Supporto tecnico, Descrizione: Richiesta di supporto tecnico per nuovo modello., Gestito da: a.ferri@experiaccoffee.it, Creato da: cliente6@outlook.com, Data creazione: 2024-09-06, Stato: Gestito
ID: 7, Titolo: Richiesta di sostituzione, Descrizione: La macchina è difettosa e richiede sostituzione., Gestito da: m.silveri@experiaccoffee.it, Creato da: cliente7@gmail.com, Data creazione: 2024-09-07, Stato: Non gestito
ID: 8, Titolo: Assistenza capsule, Descrizione: Problema con la capsula che non viene perforata., Gestito da: d.marroni@experiaccoffee.it, Creato da: cliente8@yahoo.com, Data creazione: 2024-09-08, Stato: In lavorazione
ID: 9, Titolo: Surriscaldamento, Descrizione: La macchina si surriscalda dopo 10 minuti., Gestito da: g.bianchi@experiaccoffee.it, Creato da: cliente9@libero.it, Data creazione: 2024-09-09, Stato: Non gestito
ID: 10, Titolo: Problema di connessione, Descrizione: Impossibile connettere la macchina al Wi-Fi., Gestito da: r.grigi@experiaccoffee.it, Creato da: cliente10@hotmail.com, Data creazione: 2024-09-10, Stato: Gestito
ID: 11, Titolo: Riparazione urgente, Descrizione: La macchina si è bloccata completamente., Gestito da: s.brunetti@experiaccoffee.it, Creato da: cliente11@gmail.com, Data creazione: 2024-09-11, Stato: Non gestito
ID: 12, Titolo: Errore nel programma di pulizia, Descrizione: Il programma di pulizia non parte., Gestito da: n.lupi@experiaccoffee.it, Creato da: cliente12@yahoo.com, Data creazione: 2024-09-12, Stato: In lavorazione
ID: 13, Titolo: Problema di rumorosità, Descrizione: La macchina è molto rumorosa durante l'uso., Gestito da: b.volpi@experiaccoffee.it, Creato da: cliente13@libero.it, Data creazione: 2024-09-13, Stato: Non gestito
ID: 14, Titolo: Richiesta di rimborso, Descrizione: Non sono soddisfatto del prodotto, richiedo rimborso., Gestito da: e.guerrieri@experiaccoffee.it, Creato da: cliente14@live.com, Data creazione: 2024-09-14, Stato: Gestito
ID: 15, Titolo: Macchina lenta, Descrizione: Richiesta di assistenza per lentezza di funzionamento., Gestito da: m.tosti@experiaccoffee.it, Creato da: cliente15@outlook.com, Data creazione: 2024-09-15, Stato: Non gestito
ID: 16, Titolo: Problema con erogazione, Descrizione: La macchina non eroga correttamente il caffè., Gestito da: a.vitali@experiaccoffee.it, Creato da: cliente16@gmail.com, Data creazione: 2024-09-16, Stato: In lavorazione
ID: 17, Titolo: Impossibile accendere la macchina, Descrizione: Richiesta urgente di assistenza per blocco., Gestito da: v.rubini@experiaccoffee.it, Creato da: cliente17@yahoo.com, Data creazione: 2024-09-17, Stato: Non gestito
ID: 18, Titolo: Caffè freddo, Descrizione: La macchina non riscalda l'acqua a sufficienza., Gestito da: l.brunelli@experiaccoffee.it, Creato da: cliente18@hotmail.com, Data creazione: 2024-09-18, Stato: Gestito
ID: 19, Titolo: Problema con il vapore, Descrizione: Non esce vapore per schiumare il latte., Gestito da: g.marchi@experiaccoffee.it, Creato da: cliente19@gmail.com, Data creazione: 2024-09-19, Stato: In lavorazione
ID: 20, Titolo: Manutenzione programmata, Descrizione: Richiesta di manutenzione programmata., Gestito da: c.naldi@experiaccoffee.it, Creato da: cliente20@yahoo.com, Data creazione: 2024-09-20, Stato: In lavorazione
Scegli un'operazione
--- OPERAZIONI TICKETING ---
1. Inserisci un nuovo ticket
2. Visualizza tutti i ticket
3. Visualizza gli stati dei vari ticket
4. Elimina ticket
5. Aggiorna stato ticket
-----
6. Esci
Scelta... |
```

Figura 14: ID: 19, Stato: "In lavorazione"