

WebRTC Music Learning

Elaborato di:
Applicazioni Telematiche

A cura di:
Colella Gianni M63/670
De Blasi Gabriele M63/686



Scuola Politecnica delle Scienze e di Base
Università degli studi di Napoli "Federico II"

Corso di Laurea Magistrale in Ingegneria Informatica

Prof. Simon Pietro Romano

Indice

1	Il problema	1
1.1	Descrizione del problema	1
1.2	Architettura	1
2	Sviluppo	3
2.1	Analisi del sistema	3
2.2	Utente: Professore	4
2.3	Utente: Allievo	6
3	WebRTC	12
3.1	Panoramica	12
3.2	Logica	13
3.2.1	Sequence Diagram	13
3.2.2	Codice	15
3.2.2.1	NodeServer.js	15
3.2.2.2	NodeClient.js	15
4	Demo Music Learning	17
4.1	Professore	17
4.2	Allievo	18
4.3	Connessione stabilita	20
4.4	Scenari alternativi	21
A	Codici	22
A.1	WebServer	22
A.1.1	Homepage	22
A.1.2	Controllo Form	25
A.1.3	Bacheca Room	26
A.1.3.1	Funzione asincrona per il caricamento della bacheca	27
A.1.4	WebRTC page	28
A.2	Servlet	29
A.2.1	Professore	29
A.2.2	Studente	32
A.3	Node Client	35
A.4	Node Server	40
A.5	Hibernate - MySQL	41

Capitolo 1

Il problema

1.1 Descrizione del problema

Si vuol progettare un'applicazione web che dia la possibilità di far comunicare professori ed allievi per effettuare videolezioni di musica per un apprendimento semplice, intuitivo e divertente.

L'applicazione permette agli utenti di effettuare video-chiamate peer-to-peer permettendo di avere delle stanze in cui sono presenti un professore ed un allievo.

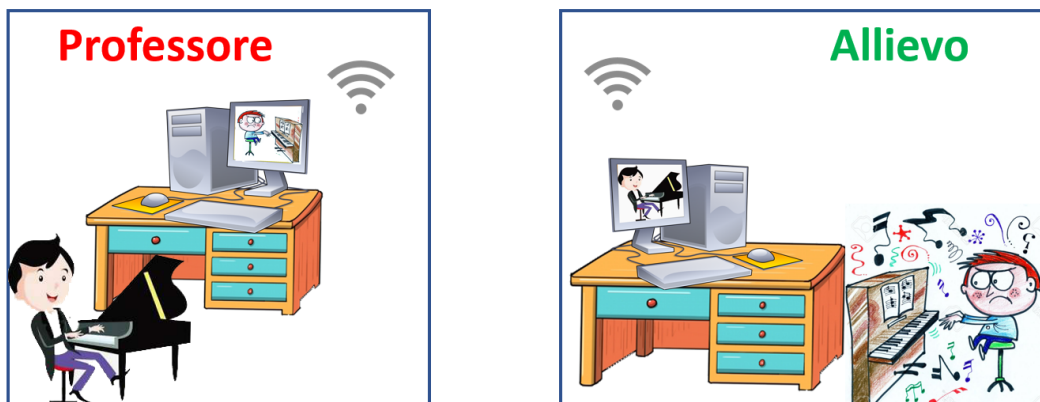


Figura 1.1: Esempio d'uso

1.2 Architettura

Per la realizzazione dell'applicazione sono stati utilizzati diversi componenti come si mostra in fig. 1.2:

- **HTML5 + Bootstrap** per l'interfaccia della Web-Application;
- per le interazioni asincrone tra il browser e il web-server mediante l'uso della libreria **jQuery**;
- **Apache Tomcat** come servlet container;
- **Java Servlet** per la realizzazione della logica del Server;
- **MySQL Database** per la memorizzazione dei dati;
- **WebRTC** per la gestione della comunicazione real-time multimediale basata su un server di segnalazione **NodeJS**;

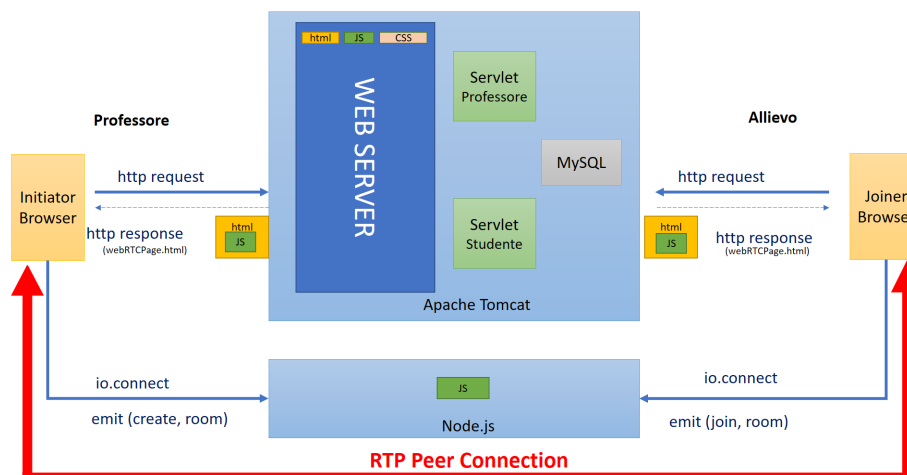


Figura 1.2: Architettura WebRTC Music Learning

Capitolo 2

Sviluppo

2.1 Analisi del sistema

Si individuano due attori distinti: **Professore** ed **Allievo**. Una volta collegatosi all'home page, un'utente deve preliminarmente selezionare se è un allievo oppure un professore.

Viene caricato un form diverso per la corrispondente tipologia di utente selezionato.

The figure displays two versions of a login page for a music learning application, both titled "Benvenuti Music Learning".

(a) Professore: This version features a header with the title and a musical staff graphic. Below the header, there are two buttons: "Professore" (highlighted in blue) and "Allievo". The form fields include: "Nome Professore" (with a sub-label "Inserisci Nome"), "Indirizzo email:" (with a sub-label "Inserisci email"), "Strumento" (a dropdown menu labeled "Seleziona..."), and "Difficoltà" (a dropdown menu labeled "Seleziona..."). A green "Login" button is at the bottom.

(b) Allievo: This version also has the same header. The buttons are "Professore" and "Allievo" (highlighted in blue). The form fields include: "Nome allievo" (with a sub-label "Inserisci Nome"), "Indirizzo email:" (with a sub-label "Inserisci indirizzo email"), and a checkbox labeled "Autorizzo il trattamento dei dati personali". A green "Login" button is at the bottom.

Figura 2.1: Si mostrano form diverse per i due utenti

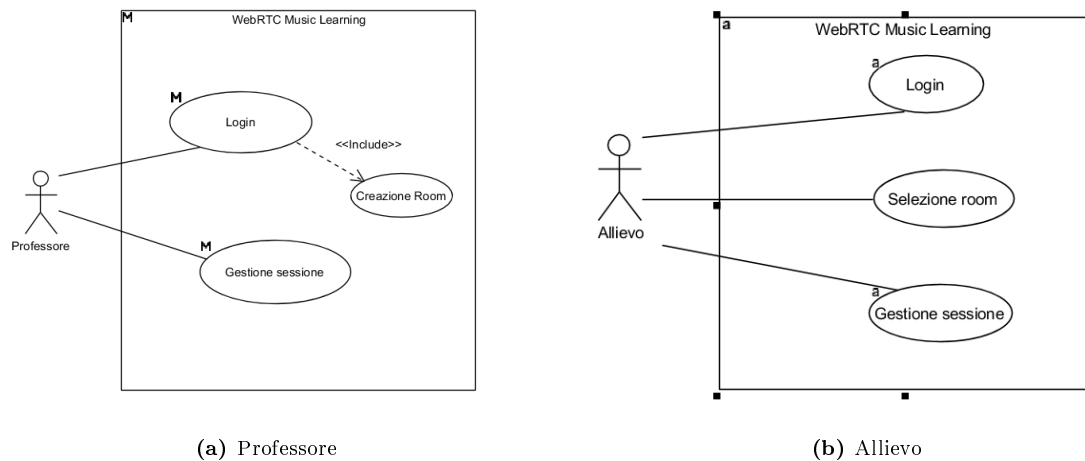


Figura 2.2: User Case Diagram

2.2 Utente: Professore

Per usufruire dell'applicazione web, un professore si collega all'indirizzo:

<http://192.168.1.199:8080/MusicLearning>

da cui scarica la homepage (`index.html`) e compila la relativa form.

Se il controllo sulla form ha avuto successo viene inviata una `http-request` (POST) verso il server, in particolare essendo esso un `servlet-container` si va ad invocare il metodo `doPost()` dalla `servlet Professor.java`.

Tale metodo preleva i dati (nome, mail, strumento e livello) dalla richiesta e, per poter creare una nuova stanza, produce un nuovo `roomID`.

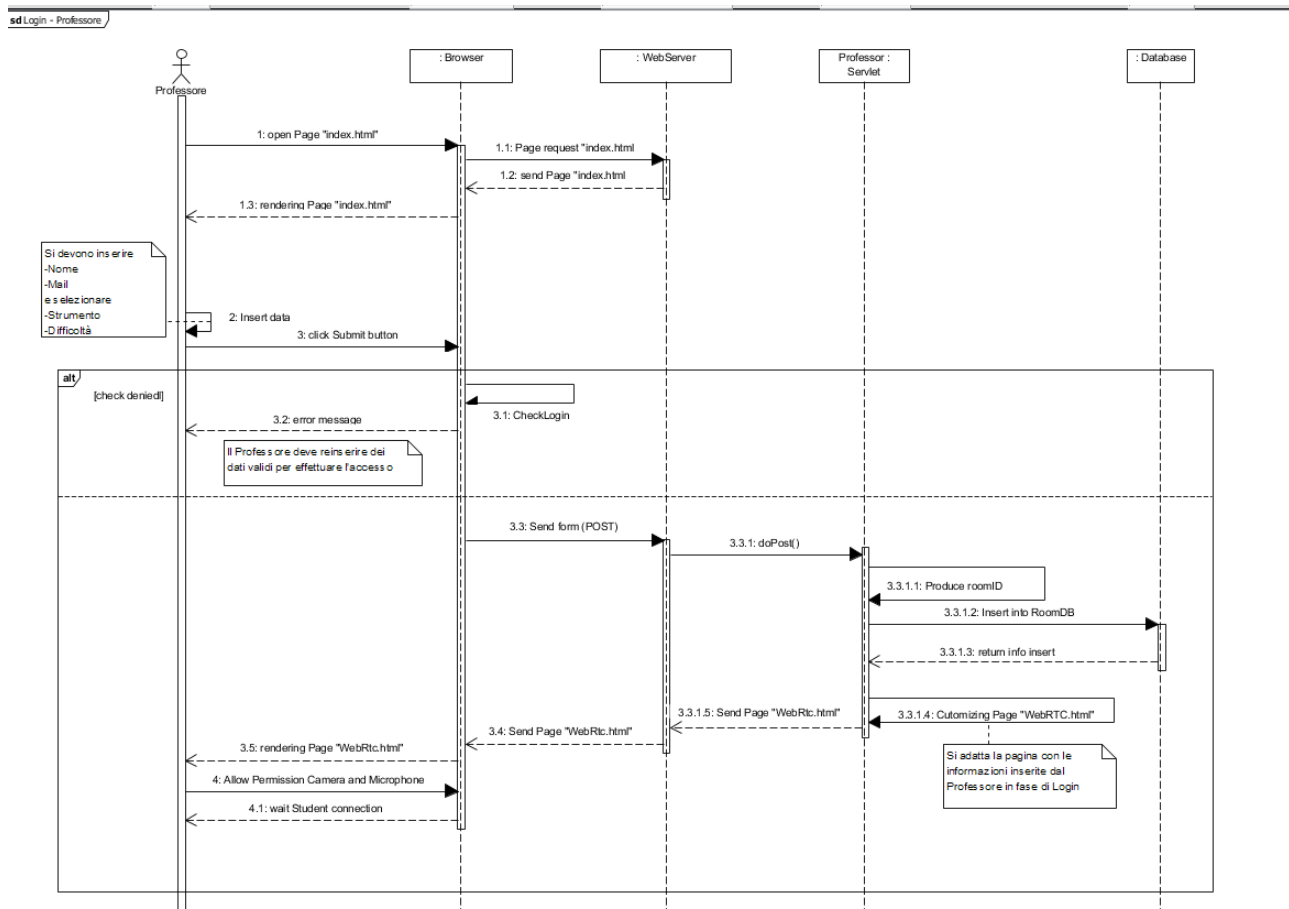


Figura 2.3: Sequence Diagram - Professore

Tale valore univoco è un numero intero ottenuto a partire dall' id delle room create in precedenza: in particolare si incrementa di un'unità il massimo valore di id memorizzato nel database utilizzando la funzione mutuamente esclusiva `incrementID()`.

```

31 public static synchronized void incrementID() {
32     lastID++;
33 }
  
```

Codice 2.1: "metodo incrementID()"

Creata l'id i dati vengono inseriti nella `roomDB`. La servlet infine prepara la pagina `WebRTC.html` personalizzata da inviare al browser del professore modificando il titolo e il tag relativo alla `roomID` (da riga 110 a 121).

Il professore avvia la sessione come **Initiator** e una volta forniti i permessi di accesso alla videocamera e al microfono si mette in attesa di un allievo.

```

76 /**
77  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
78  */
79 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
  
```

```

80 // TODO Auto-generated method stub
81
82 System.out.println("DEBUG: [ProfList_Servlet: doPost]");
83
84
85 String ProfName = request.getParameter("ProfName");
86 String ProfEmail = request.getParameter("ProfEmail");
87 String ProfInstrument = request.getParameter("Instrument");
88 String ProfLevel = request.getParameter("Level");
89
90
91
92
93 if((ProfName == null || ProfName.equals("")) || (ProfEmail == null || ProfEmail.equals(""))
94    || (ProfInstrument == null || ProfInstrument.equals("")) || (ProfLevel == null ||
95        ProfLevel.equals("")) ) {
96     System.out.println("DEBUG: [Professor_Servlet: doPost] -> No Content for Professor ");
97     response.sendError(HttpServletResponse.SC_NO_CONTENT);
98     return;
99 }
100 else{
101     incrementID();
102     RoomDB room= new RoomDB(lastID, ProfName, ProfEmail, ProfInstrument, ProfLevel,1);
103     // rooms.add(r); // Add to Local List of Rooms
104     room.SaveRoom();
105     System.out.println("DEBUG: [Professor_Servlet: doPost] -> Room saved into DB!");
106
107     File input = new File("/var/lib/tomcat8/webapps/MusicLearning WebRtcPage.html");
108
109
110     Document html = (Document) Jsoup.parse(input, "UTF-8");
111
112     Element tmp= html.getElementById("roomId");
113
114     tmp.text(Integer.toString(lastID));
115
116     tmp= html.getElementById("pageTitle");
117
118     tmp.text("VideoLezione RealTime di " + ProfInstrument + ". Difficolta': " + ProfLevel);
119     response.setContentType("text/html");
120     response.setHeader("Cache-Control", "no-cache");
121     response.getWriter().append(html.toString());
122 }
123
124 }

```

Codice 2.2: "metodo doPost() della Servlet Professore"

Per il codice completo della servlet relativa al Professore si rimanda a A.2.1.

2.3 Utente: Allievo

Per usufruire dell'applicazione web, un allievo si collega all'indirizzo:

http://192.168.1.199:8080/MusicLearning

e compila la corrispondente form. Così come avviene per il caso precedente, viene richiamato il metodo `doPost()` dalla servlet `Student.java`.

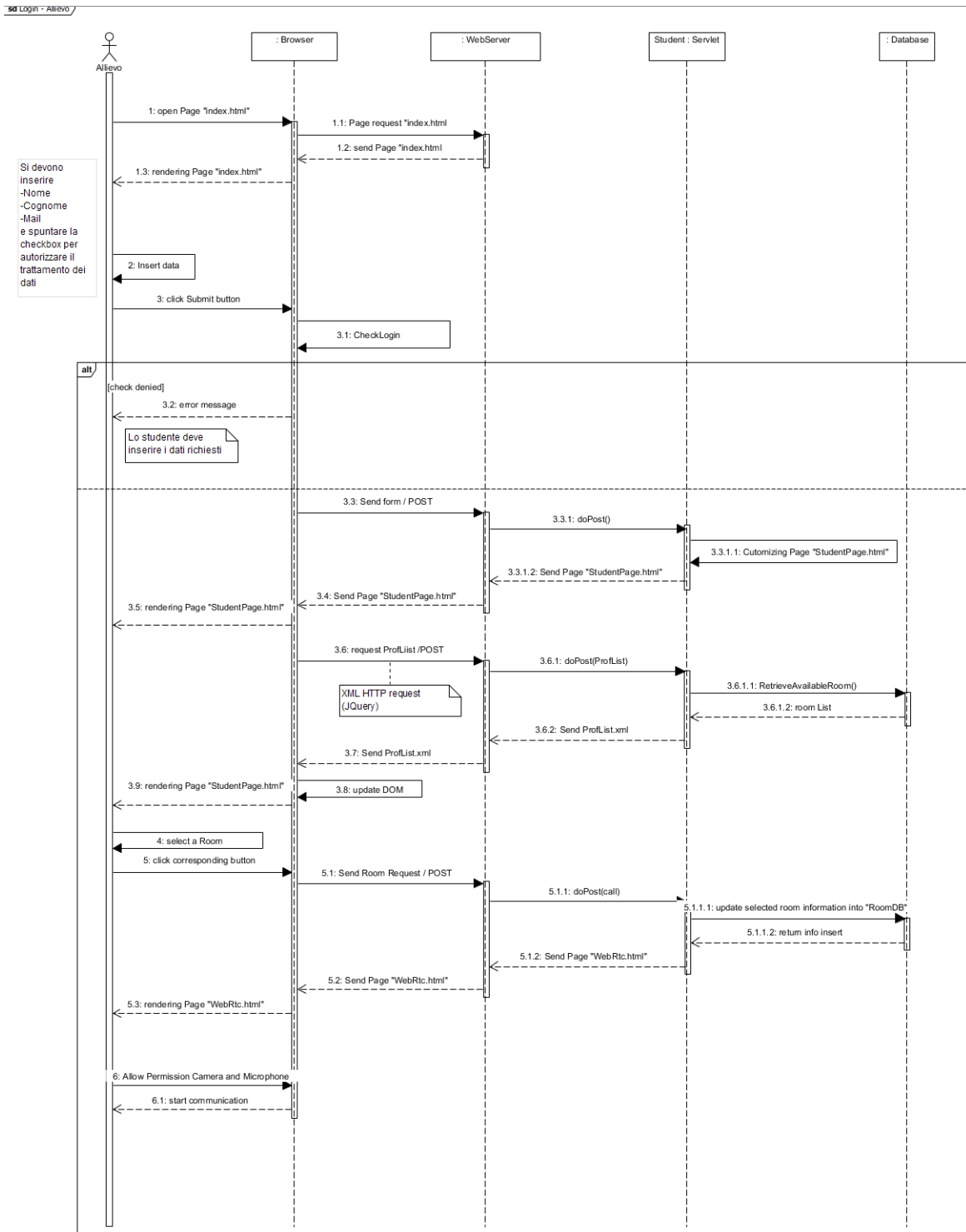


Figura 2.4: Sequence Diagram - Allievo

A differenza di quanto succede per il caso del professore, la servlet restituisce al browser dell'allievo la pagina `StudentPage.html` personalizzata contenente soltanto il titolo.

```

70 File input = new File("/var/lib/tomcat8/webapps/MusicLearning/StudentPage.html");
71 Document html = (Document) Jsoup.parse(input, "UTF-8");

```

```

72
73 Element r= html.getElementById("pageTitle");
74
75 r.text("Ciao "+ StudentName + "! Scegli un Professore");
76
77 response.setContentType("text/html");
78 response.setHeader("Cache-Control", "no-cache");
79 response.getWriter().append(html.toString());

```

Codice 2.3: "Customizing StudentPage.html"

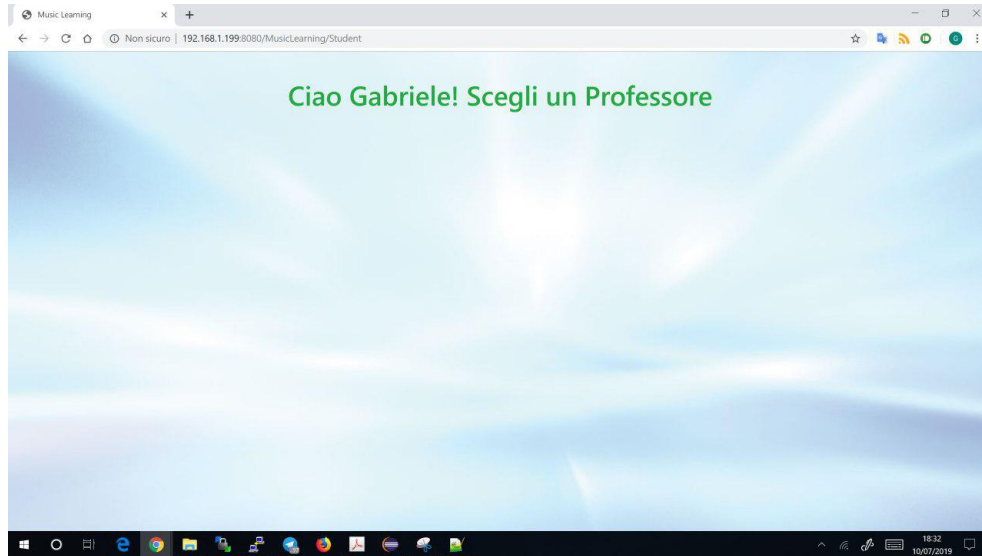


Figura 2.5: Rendering di StudentPage.html prima della richiesta asincrona

Successivamente, viene effettuata una richiesta asincrona con il codice JavaScript `Student.js` con cui si preleva la lista dei professori disponibili (coloro che hanno effettuato l'accesso e sono in attesa del collegamento di uno studente).

Per ognuno di essi viene creata una card in cui si caricano le informazioni inerenti (strumento musicale, nome del professore, livello di difficoltà e mail del professore) attraverso il seguente script.

```

1 $(document).ready(function() {
2
3   $.post("Student", {comand: "getProfList"},
4     function(xml) {
5
6       $(xml).find("prof").each(function() {
7
8         var name = $("<h3>").text($(this).find("name").text());
9         var level = $("<p>").text($(this).find("level").text());
10        var instr = $("<img>").attr("class", "img-fluid").attr("src", $(this).find("linkImage")
11          ).text());
12
13        var email= $("<span>").text($(this).find("email").text());
14        var room = $("<p>").attr("class", "btn btn-primary").attr({ "type": "_room" }).attr("room", $(this).find("room").text()).text("Accedi")
15          .on("click", function(){
16
17            var r= $(this).attr("room");
18            $.ajax({

```

```

18         type: 'POST',
19         url: 'Student',
20         data: {comand : 'call', room:r},
21         success: function(response) {
22             // re-writes the entire document
23             var newDoc = document.open("text/html", "replace");
24
25             newDoc.write(response);
26             newDoc.close();
27         }
28     });
29
30 });
31
32 var card=$( "<div>" ).attr("class", "col-lg-4")
33 .append(
34     $( "<div>" ).attr("class", "our-team-main")
35     .append(
36         $( "<div>" ).attr("class", "team-front")
37         .append(instr,name,level),
38
39         $( "<div>" ).attr("class", "team-back")
40         .append(email, "<br>", "<br>", $( "<div>" ).attr({ "align": "center", "type": "
         _room" })
41         .append(room)
42
43     )
44
45 )
46
47 );
48
49
50 $("#bacheca").append(card);
51
52 });
53
54 if($("p[type='_room']").length == 0){
55     alert("Professori non disponibili!");
56 }
57
58 }
59 );
60 );
61
62 });

```

Codice 2.4: "Student.js"

```

84 case "getProfList":
85     ArrayList<RoomDB> rooms= RoomDB.RetrieveAvailableRoom();
86
87
88     response.setContentType("text/xml");
89     response.setHeader("Cache-Control", "no-cache");
90     response.getWriter().append("<response>");
91
92
93     if( rooms != null){

```

```

94
95 for(RoomDB iter: rooms){
96
97     response.getWriter().append("<prof>");
98
99     response.getWriter().append("<name>");
100     response.getWriter().append(iter.getProfName());
101     response.getWriter().append("</name>");
102
103     response.getWriter().append("<level>");
104     response.getWriter().append(iter.getLevel());
105     response.getWriter().append("</level>");
106
107     response.getWriter().append("<email>");
108     response.getWriter().append(iter.getProfEmail());
109     response.getWriter().append("</email>");
110
111     response.getWriter().append("<linkImage>");
112     response.getWriter().append("img/" + iter.getMusicalInstrument() + ".png");
113     response.getWriter().append("</linkImage>");
114
115     response.getWriter().append("<room>");
116     response.getWriter().append(Integer.toString(iter.getID()));
117     response.getWriter().append("</room>");
118
119     response.getWriter().append("</prof>");
120 }
121
122 }
123
124 response.getWriter().append("</response>");

```

Codice 2.5: "getProfList di Student.java"

L'allievo per poter collegarsi ad un professore seleziona una card e clicca sul bottone Accedi; dunque viene invocato il metodo doPost() dalla servlet Student.java che esegue la Call e restituisce al browser dell'allievo la pagina WebRTCPage.html contenente il roomID per potersi collegare alla room selezionata.

L'allievo partecipa alla sessione come **Joiner** e quando dà i permessi di accesso alla videocamera e al microfono il canale multimediale è completamente instaurato.

```

84 case "call":
85     RoomDB temp= RoomDB.RetrieveRoomByID(Integer.parseInt(request.getParameter("room")));
86     if(temp.getNclient() == 1){
87         temp.setNclient(2);
88         temp.UpdateRoom();
89
90
91         File input = new File("/var/lib/tomcat8/webapps/MusicLearning/WebRtcPage.html");
92         Document html = (Document) Jsoup.parse(input, "UTF-8");
93
94         //response.addHeader("roomID", request.getParameter("room"));
95         //System.out.println(html.select("#roomID"));
96         Element tmp= html.getElementById("roomID");
97
98         tmp.text(request.getParameter("room"));
99
100         tmp= html.getElementById("pageTitle");
101
102         tmp.text("VideoLezione RealTime di " + temp.getMusicalInstrument() + ". Difficolta:
            "+ temp.getLevel());

```

```
103         response.setContentType("text/html");
104         response.setHeader("Cache-Control", "no-cache");
105         response.getWriter().append(html.toString());
106
107     }
108     else{
109
110         File input = new File("/var/lib/tomcat8/webapps/MusicLearning/StudentPage.html");
111         Document html = (Document) Jsoup.parse(input, "UTF-8");
112
113         Element r= html.getElementById("pageTitle");
114
115         r.text("Sessione avviata da un altro studente! Scegli un Professore");
116
117         response.setContentType("text/html");
118         response.setHeader("Cache-Control", "no-cache");
119         response.getWriter().append(html.toString());
120     }
121 }
```

Codice 2.6: "Call di Student.java"

Per il codice completo della servlet relativa all'allievo si rimanda a A.2.2.

Capitolo 3

WebRTC

3.1 Panoramica

WebRTC nasce con lo scopo di consentire la comunicazione real-time tramite il web. E' frutto del lavoro di standardizzazione congiunto che coinvolge l' IETF, per la definizione dei protocolli e formati da utilizzare, e il W3C per implementare la logica applicativa e quindi la definizione di API per poter accedere alle funzionalità sviluppate in ambito IETF. L'architettura di WebRTC si ispira al modello trapezoidale SIP anche se il modello più comune prevede una strutta a triangolo come si mostra in fig. 3.1: due browser che eseguono la stessa Web Application e un Server di Segnalazione.

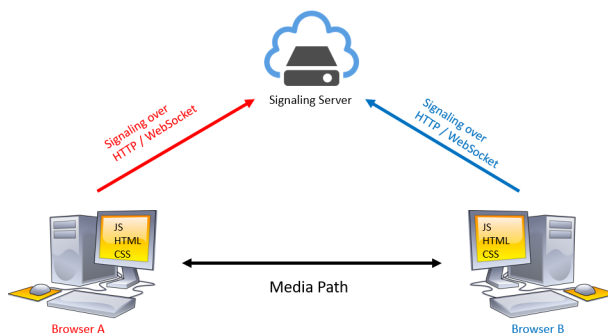


Figura 3.1: Architettura WebRTC

Il piano dati è standardizzato e instaurato direttamente tra i due Browser : la API `RTCPeerConnection` permette la comunicazione dei flussi di dati tra i due peer. La fase di segnalazione non è standardizzata, e avviene tramite lo scambio di messaggi sul protocollo HTTP o WebSocket. In questo progetto il Server di segnalazione è realizzato utilizzando le librerie JavaScript `socket.io`.

3.2 Logica

3.2.1 Sequence Diagram

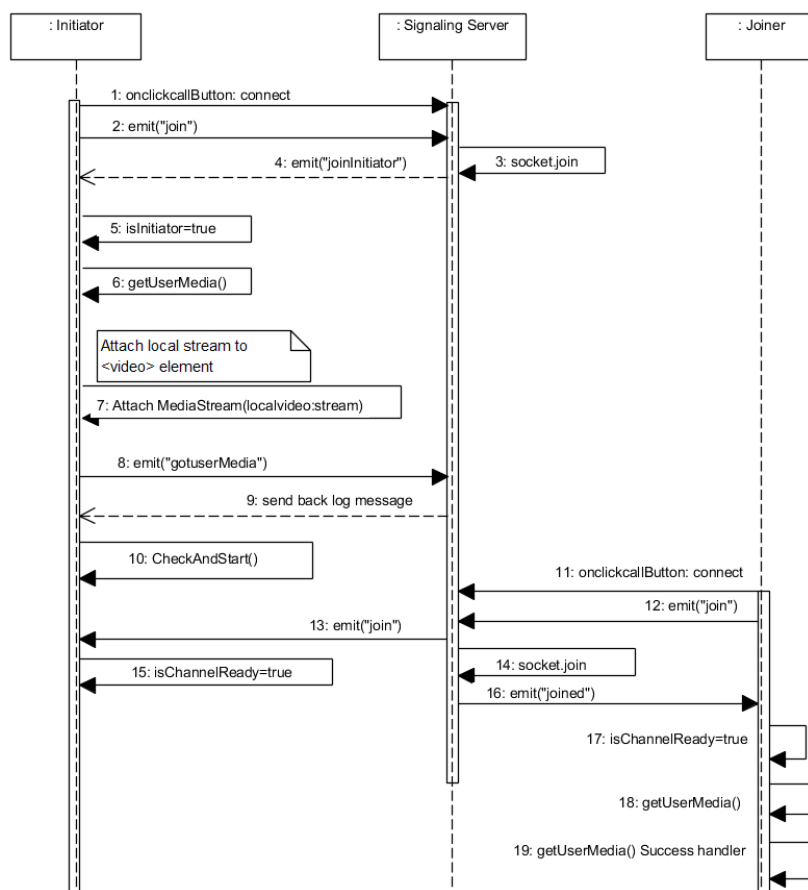


Figura 3.2: Sequence Diagram WebRTC parte 1

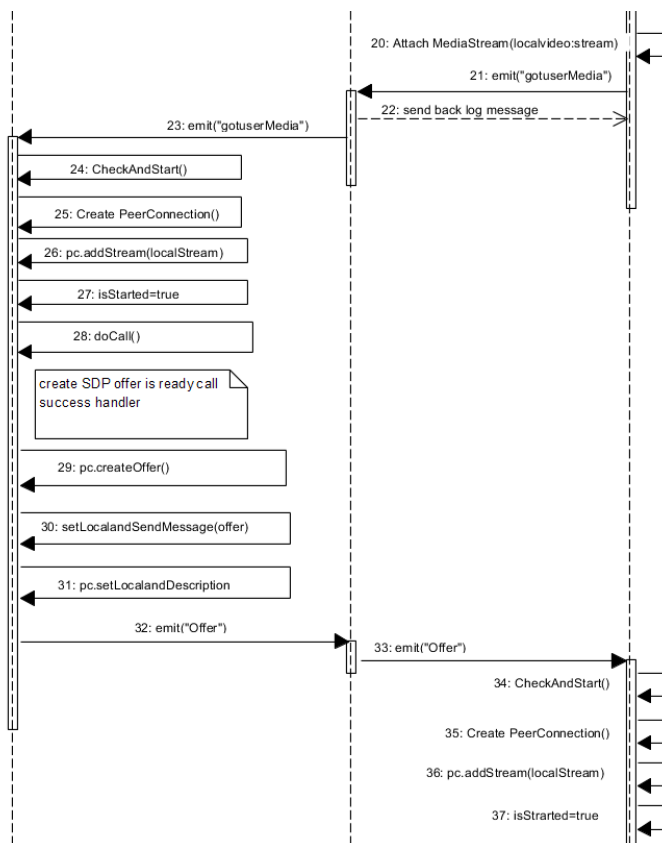


Figura 3.3: Sequence Diagram WebRTC parte 2

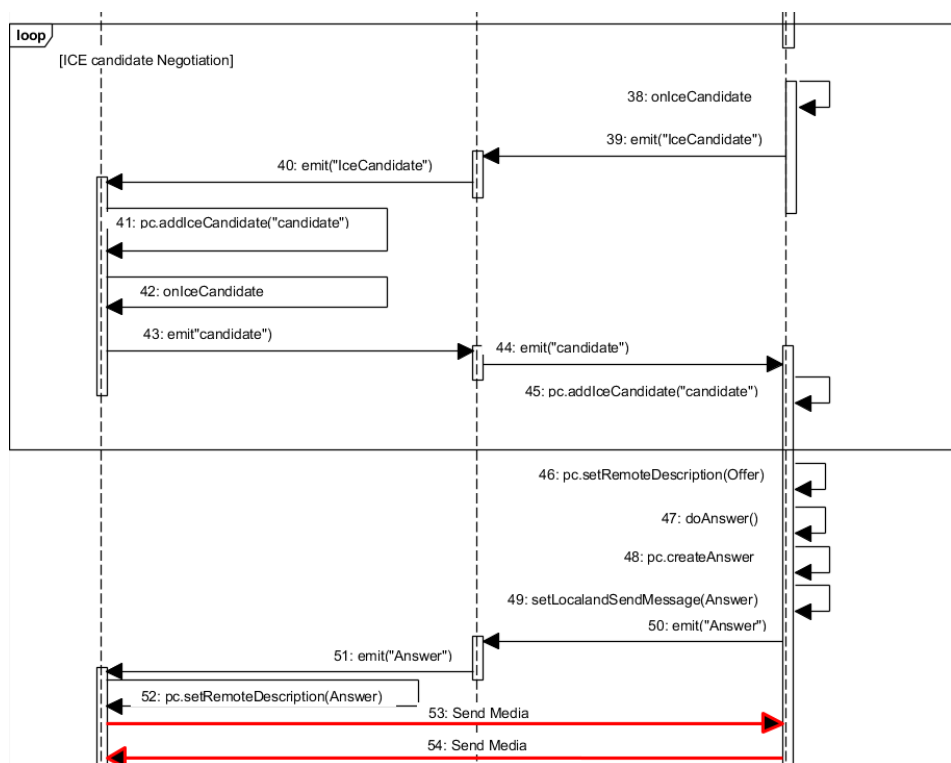


Figura 3.4: Sequence Diagram WebRTC parte 3

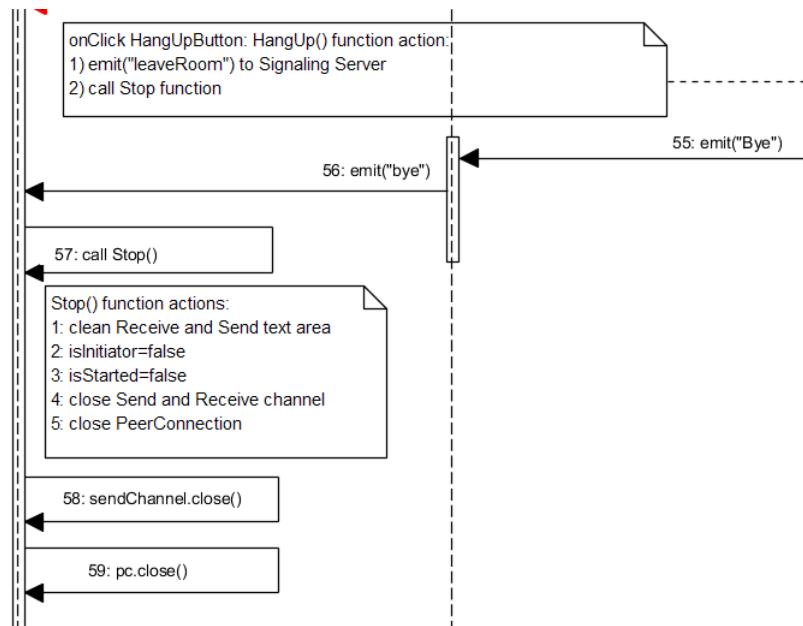


Figura 3.5: Sequence Diagram WebRTC parte 4

3.2.2 Codice

3.2.2.1 NodeServer.js

E' permesso l'accesso di un numero massimo di due utenti ad ogni room.

```

24 socket.on('create or join', function (room) {
25     var numClients = io.sockets.adapter.rooms[room]!==undefined ? Object.keys(io.sockets.
        adapter.rooms[room]).length:0;
26     log('S --> Room ' + room + ' has ' + numClients + ' client(s)');
27     log('S --> Request to create or join room', room);
28     console.log('Stanza: = ' + room + " Clienti:" + numClients);
29     // First client joining...
30     if (numClients == 0){
31         socket.join(room);
32         socket.emit('created', room);
33     }
34     else if (numClients == 1) {
35         // Second client joining...
36         io.sockets.in(room).emit('join', room);
37         socket.join(room);
38         socket.emit('joined', room);
39     }
40     else { // max two clients
41         socket.emit('full', room);
42     }
43 });
  
```

Codice 3.1: "funzione associata al messaggio 'create or join' "

3.2.2.2 NodeClient.js

Gestione delle Room E' stato automatizzato il processo di creazione ed accesso alle room. Quando un professore si connette al servizio MusicLearning, riempie correttamente la form e i dati raccolti sono successivamente utilizzati per poter creare una nuova entry nella tabella RoomDB di MySQL come mostrato in fig. 3.6.

```
mysql> select * from RoomDB;
```

id	level	musical_instrument	n_client	prof_email	prof_name
2	Intermedia	chitarra	2	gigi@hotmail.it	Gigi
3	Avanzata	tromba	2	ciro@ciro	Ciro da Grosseto
4	Intermedia	batteria	2	ceo.ceo.ceo@ceo.ceo	Daniele Lombardi
6	Intermedia	tromba	2	ciroguida@ceo.it	Ciro Guida
11	Base	pianoforte	1	antonio.russo@gmail.com	Antonio Russo
14	Base	chitarra	2	gdb@hotmail.it	Gabriele De Blasi
15	Avanzata	batteria	1	s.romano@unina.it	Simon Pietro Romano
18	Avanzata	tromba	1	gventre@unina.it	Giorgio Ventre
19	Intermedia	pianoforte	1	apescape@unina.it	Antonio Pescape

9 rows in set (0.00 sec)

Figura 3.6: RoomDB Table

La tabella è composta da tali campi (*id*, *level*, *musical – instrument*, *n – client*, *prof – email*, *prof – name*) in cui la **PRIMARY KEY** è *id* ed è un numero intero generato automaticamente per cui quest'ultimo viene utilizzato come identificativo della room associata alla sessione iniziata dal professore.

L' *id* della room viene inserito dal server nella pagina WebRTCPage.html all'interno di un tag <p> avente id roomID.

```
34 var room = document.getElementById("roomID").textContent;
```

Codice 3.2: "definizione della room"

Chiusura del canale Inoltre la terminazione della sessione da parte di uno dei due client è stata gestita utilizzando l'evento `onbeforeunload`.

```
4 // When I close the window, hangup() function starts.
5 window.onbeforeunload = function(e) {
6     hangup();
7 }
```

Codice 3.3: "windows.onbeforeunload"

```
285 // Clean-up functions
286 function hangup() {
287     console.log('Hanging up.');
```

```
288     stop();
289     sendMessage('bye');
```

```
290 }

291
292 function handleRemoteHangup() {
293     console.log('Session terminated.');
```

```
294     stop();
295     isInitiator = false;
```

```
296 }

297
298 function stop() {
299     isStarted = false;
```

```
300     if (sendChannel) sendChannel.close();
301     if (receiveChannel) receiveChannel.close();
302     if (pc) pc.close();
303     pc = null;
```

```
304     sendButton.disabled=true;
```

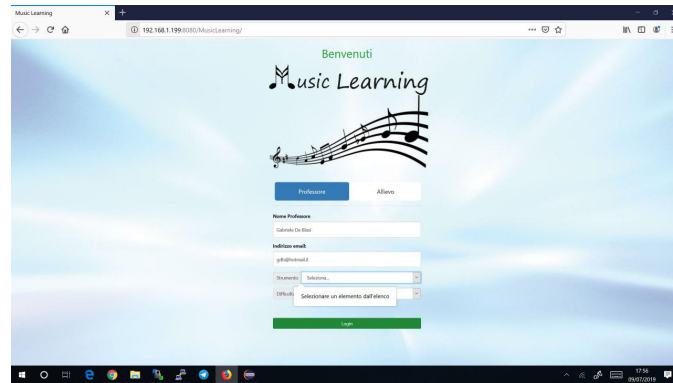
```
305 }
```

Codice 3.4: "Implementazione delle funzioni di chiusura del canale"

Capitolo 4

Demo Music Learning

4.1 Professore



The screenshot shows a web browser window with the URL `192.168.1.199:8080/MusicLearning/`. The page has a light blue background with a musical staff graphic. At the top, it says "Benvenuti" (Welcome) and "Music Learning". Below this, there are two buttons: "Professore" (Professor) and "Allievo" (Student). The "Professore" button is selected. Below the buttons, there is a form for the Professor login with the following fields: "Nome Professore" (Professor Name), "Cognome da Esat" (Last Name), "Indirizzo email" (Email Address), "password" (Password), "Documento" (Document) with a dropdown menu showing "Solenne", and "Difficoltà" (Difficulty) with a dropdown menu showing "Selezionare un elemento dall'elenco". A green "Login" button is at the bottom of the form.

Figura 4.1: Form Validation Professore

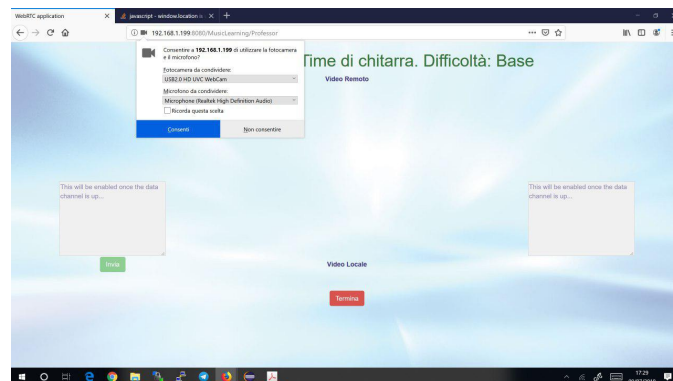


Figura 4.2: Il professore deve acconsentire l'utilizzo della videocamera e del microfono

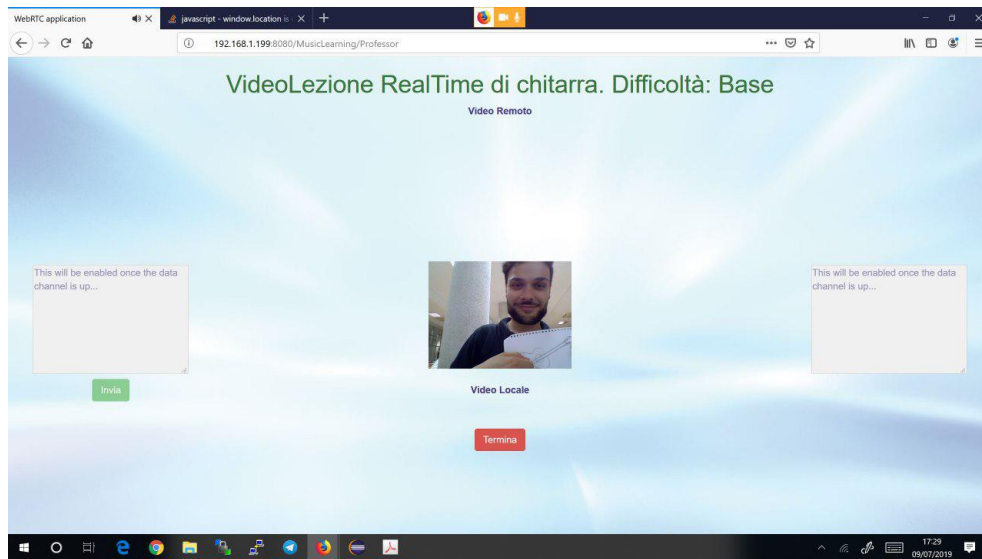


Figura 4.3: Il professore attende un nuovo allievo

4.2 Allievo

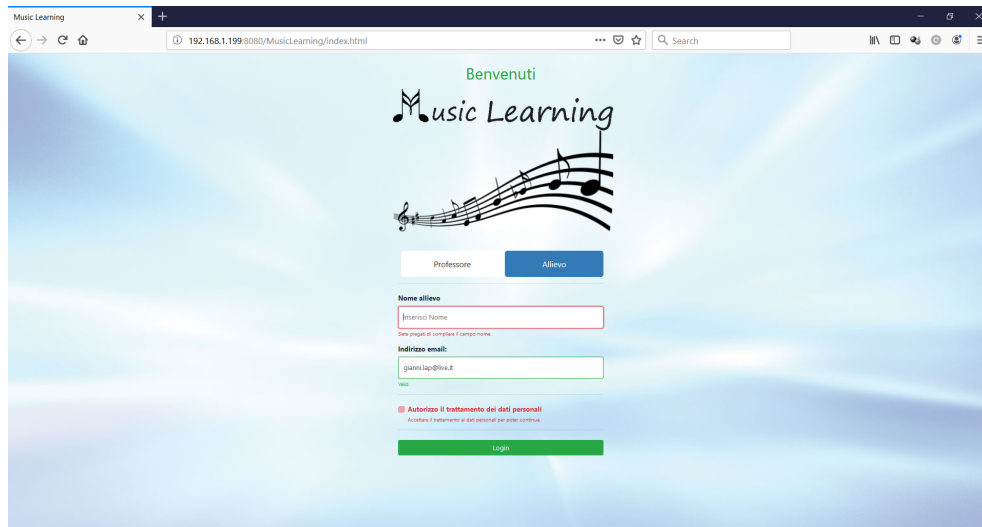


Figura 4.4: Form Validation allievo

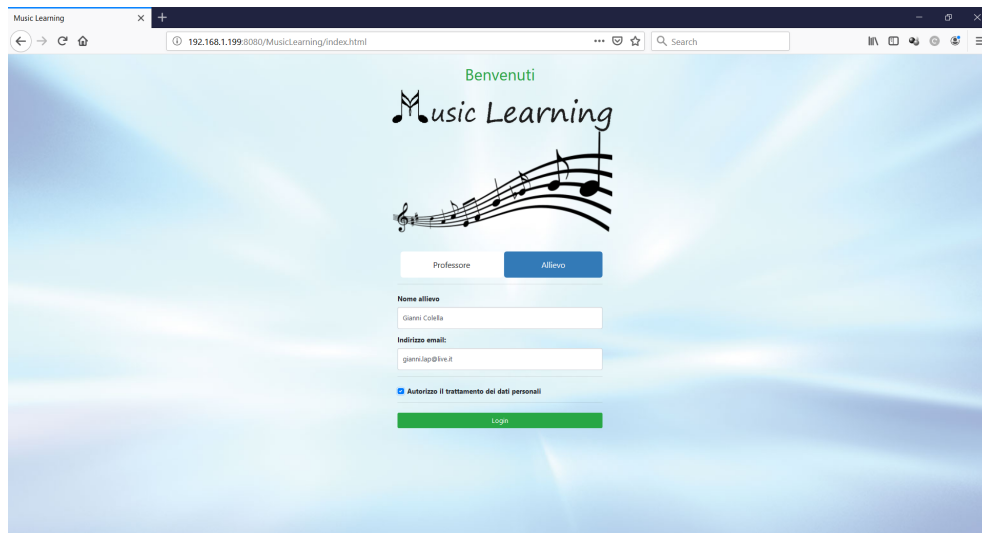


Figura 4.5: L'allievo è pronto a cliccare sul bottone Login

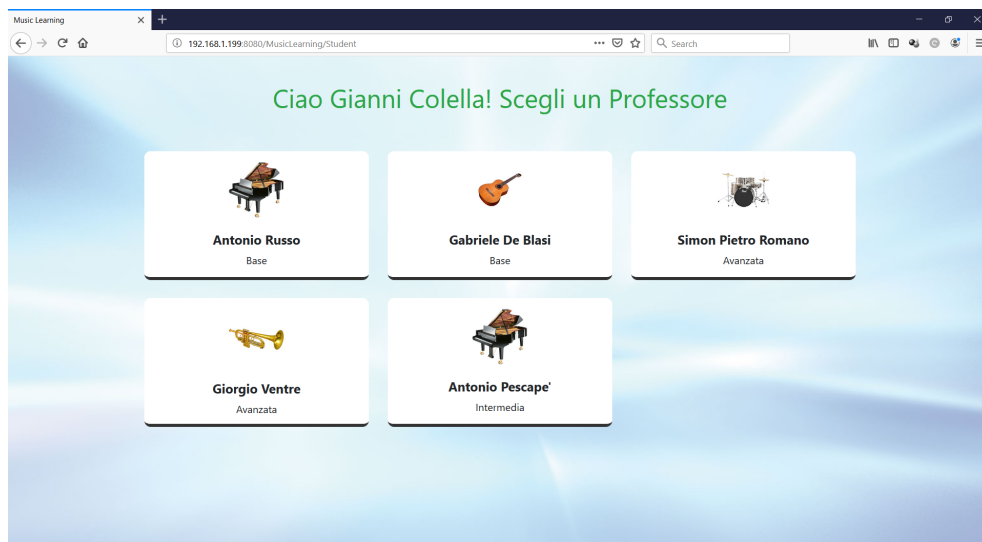


Figura 4.6: L'allievo è reindirizzato alla risorsa Student

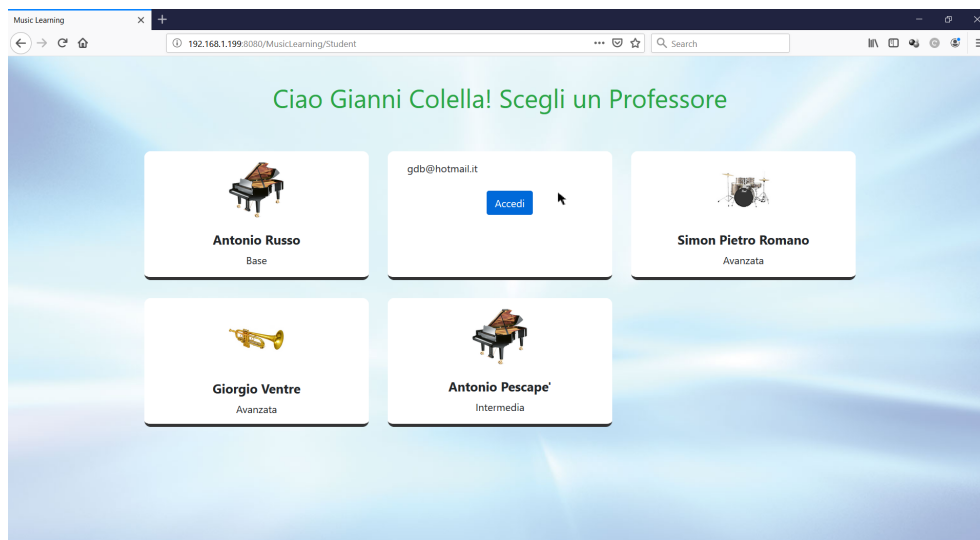
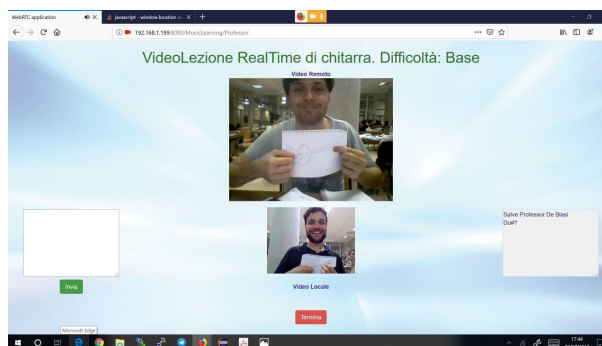


Figura 4.7: L'allievo seleziona il professore desiderato

4.3 Connessione stabilita



(a) Vista Allievo



(b) Vista Professore

Figura 4.8: La videoLezione può cominciare

4.4 Scenari alternativi

Nessun professore presente

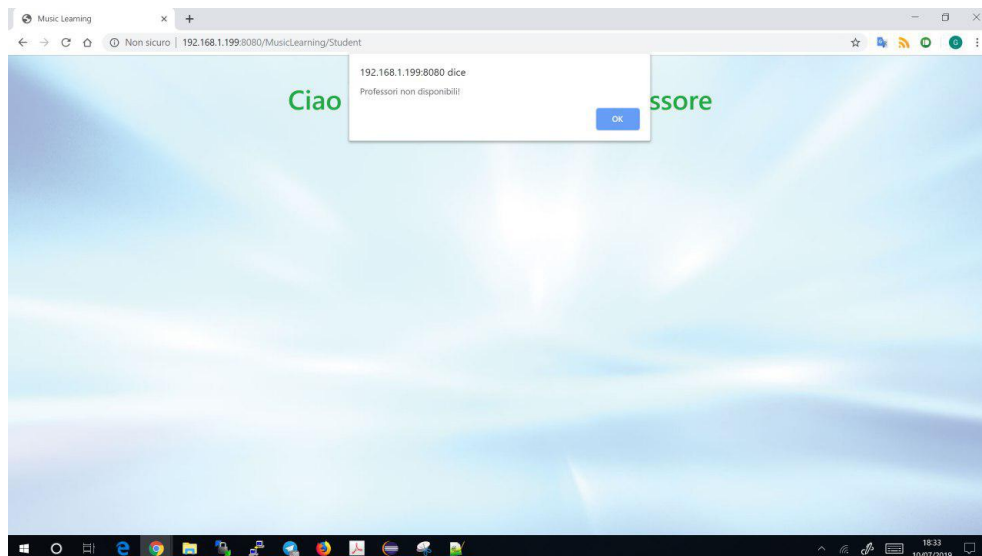


Figura 4.9: Un alert espone il problema all'allievo

Room selezionata occupata

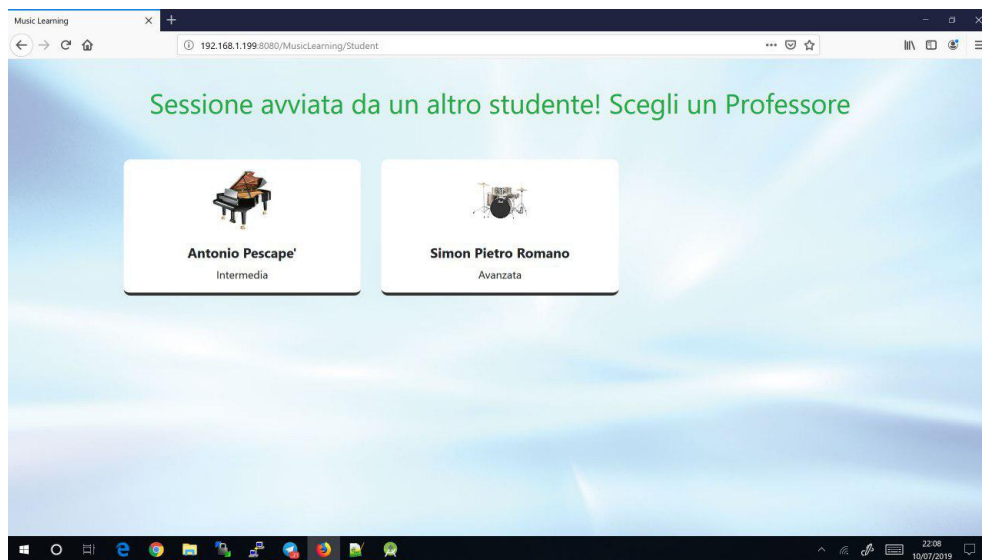


Figura 4.10: L'allievo deve scegliere un'altra room

Appendice A

Codici

In seguito sono riportati tutti i codici utilizzati:

A.1 WebServer

A.1.1 Homepage

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <title>Music Learning</title>
5   <meta charset="utf-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap
8     .min.css">
9   <!-- script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></
10     script-->
11   <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
12   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script
13     >
14
15 <!-- Documentation extras -->
16
17 <link href="https://cdn.jsdelivr.net/npm/docsearch.js@2/dist/cdn/docsearch.min.css" rel="
18   stylesheet">
19
20 <!-- codice jQuery -->
21 <script type="text/javascript" src="CheckLogin.js"></script>
22
23 <!-- Bootstrap core CSS -->
24 <link href="index.css" rel="stylesheet">
25
26 </head>
27
28
29 <body background="img/background.jpg">
30
31 <div class="container"><!-- style="margin-top: 5%;" -->
32   <div class="row">
33     <div class="col-sm-4"> </div>
```



```

34 <div class="col-md-4">
35   <h1 class="text-center text-success"> Benvenuti </h1>
36
37   
38
39   
41
42   <div class="col-sm-12">
43
44     <ul class="nav nav-pills" >
45       <br/>
46
47       <hr class="mb-4">
48
49       <li class="active" style="width:50%"><a class="btn btn-lg btn-default" data-toggle
50         ="tab" href="#menuProfessore">Professore</a></li>
51
52       <li class="" style="width:48%"><a class=" btn btn-lg btn-default" data-toggle="tab
53         " href="#menuAllievo">Allievo</a></li>
54
55     </ul>
56
57     <hr class="mb-4">
58
59     <div class="tab-content" name="tab">
60       <div id="menuProfessore" class="tab-pane fade in active">
61
62       <form method="post" action="Professor" name="formProf">
63
64
65       <div class="form-group">
66         <label for="username">Nome Professore</label>
67         <input type="text" class="form-control" id="ProfName" placeholder="Inserisci Nome"
68           name="ProfName" required>
69         <div class="valid-feedback">Valid.</div>
70         <div class="invalid-feedback"> Siete pregati di compilare il campo nome.</div>
71       </div>
72
73
74       <div class="form-group">
75         <label for="email">Indirizzo email:</label>
76         <input type="email" class="form-control" placeholder="Inserisci email" id="email"
77           name="ProfEmail" required>
78         <div class="invalid-feedback">
79           E' richiesto un indirizzo email valido.
80         </div>
81       </div>
82
83       <div class="input-group mb-3">
84         <div class="input-group-prepend">
85           <label class="input-group-text" for="instrument">Strumento</label>
86         </div>
87         <select class="form-control" id="instrument" required name="Instrument">
88

```

```

89         <option value>Seleziona...</option>
90         <option value="tromba">Tromba</option>
91         <option value="pianoforte">Pianoforte</option>
92         <option value="batteria">Batteria</option>
93         <option value="chitarra">Chitarra</option>
94     </select>
95 </div>
96
97 <div class="input-group mb-3">
98     <div class="input-group-prepend">
99         <label class="input-group-text" for="difficoult">Difficoltà</label>
100     </div>
101     <select class="form-control" id="difficoult" required name="Level">
102
103         <option value>Seleziona...</option>
104         <option value="Base">Base</option>
105         <option value="Intermedia">Intermedia</option>
106         <option value="Avanzata">Avanzata</option>
107     </select>
108 </div>
109
110 <br/>
111
112 <br/>
113
114
115
116
117
118     <button type="submit" class="btn btn-block btn-success">Login</button>
119
120 </form>
121 <br/>
122
123
124
125 </div>
126
127     <div id="menuAllievo" class="tab-pane fade">
128
129 <form method="post" action="Student" class="needs-validation" novalidate>
130
131     <div class="form-group">
132         <label for="name">Nome allievo</label>
133         <input type="text" class="form-control" id="name" placeholder="Inserisci Nome" name="
134             StudentName" required>
135         <div class="valid-feedback">Valid.</div>
136         <div class="invalid-feedback"> Siete pregati di compilare il campo nome.</div>
137     </div>
138 <!--     <div class="form-group">
139         <label for="surname">Cognome allievo</label>
140         <input type="text" class="form-control" id="surname" placeholder="Inserisci cognome"
141             name="urname" required>
142         <div class="valid-feedback">Valid.</div>
143         <div class="invalid-feedback"> Siete pregati di compilare il campo cognome.</div>
144     </div> -->
145
146     <div class="form-group">

```

```

147     <label for="email">Indirizzo email:</label>
148     <input type="email" class="form-control" id="email" placeholder="Inserisci indirizzo
149         email" name="StudentEmail" required>
149 <div class="valid-feedback">Valid.</div>
150 <div class="invalid-feedback"> Inserire una mail valida</div>
151
152 </div>
153
154
155
156 <hr class="mb-4">
157
158
159 <div class="custom-control custom-checkbox">
160     <input type="checkbox" class="custom-control-input" id="accept-info" required >
161     <label class="custom-control-label" for="accept-info">Autorizzo il trattamento dei dati
162         personali</label>
162     <div class="invalid-feedback">Accettare il trattamento ai dati personali per poter
163         continue.</div>
163 </div>
164 <hr class="mb-4">
165
166
167     <button type="submit" class="btn btn-block btn-success">Login</button>
168
169 </form>
170 <br/>
171
172
173
174
175 </div>
176
177
178 </div>
179 </div>
180 </div>
181 </div>
182 </div>
183
184 </body>
185 </html>

```

Codice A.1: "index.html"

A.1.2 Controllo Form

```

1  /**
2   * Check Login JavaScript
3   */
4
5  // Disable form submissions if there are invalid fields
6  (function() {
7      'use strict';
8      window.addEventListener('load', function() {
9          // Get the forms we want to add validation styles to
10         var forms = document.getElementsByClassName('needs-validation');
11         // Loop over them and prevent submission
12         var validation = Array.prototype.filter.call(forms, function(form) {

```

```

13     form.addEventListener('submit', function(event) {
14         if (form.checkValidity() === false) {
15             event.preventDefault();
16             event.stopPropagation();
17         }
18         form.classList.add('was-validated');
19     }, false);
20 });
21 }, false);
22 }) ();

```

Codice A.2: "CheckLogin.js"

A.1.3 Bacheca Room

```

1 <!doctype html>
2 <html lang="en">
3 <head>
4     <!-- Required meta tags -->
5     <meta charset="utf-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
7
8     <!-- jQuery first, then Popper.js, then Bootstrap JS -->
9     <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
10    <!--script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.
11        js" integrity="sha384-
12        UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="
13        anonymous"></script-->
14    <script src="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"></script>
15
16    <script type="text/javascript" src="StudentPage.js"></script>
17
18    <!-- Bootstrap CSS -->
19    <link href="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" rel="
20        stylesheet" id="bootstrap-css">
21    <!-- Bootstrap core CSS -->
22    <link href="StudentPage.css" rel="stylesheet">
23
24    <title>Music Learning</title>
25 </head>
26
27 <body background="img/background.jpg">
28
29    <h1 id="pageTitle" class="text-center"></h1>
30
31    <div class="container">
32        <div id="bacheca" class="row">
33
34
35        </div>
36    </div>
37 </body>
38 </html>

```

Codice A.3: "StudentPage.html"

A.1.3.1 Funzione asincrona per il caricamento della bacheca

```

1 $(document).ready(function() {
2
3
4     $.post("Student", {comand: "getProfList"},
5         function(xml) {
6
7             $(xml).find("prof").each(function() {
8
9                 var name = $("<h3>").text($(this).find("name").text());
10                var level = $("<p>").text($(this).find("level").text());
11                var instr = $("<img>").attr("class", "img-fluid").attr("src", $(this).find("
12                    linkImage").text());
13
14                var email = $("<span>").text($(this).find("email").text());
15                var room = $("<p>").attr("class", "btn btn-primary").attr({ "type": "_room" }).attr(
16                    "room", $(this).find("room").text()).text("Accedi")
17                    .on("click", function() {
18
19                        var r = $(this).attr("room");
20                        $.ajax({
21                            type: 'POST',
22                            url: 'Student',
23                            data: {comand : 'call', room:r},
24                            success: function(response) {
25                                // re-writes the entire document
26                                var newDoc = document.open("text/html", "replace");
27
28                                newDoc.write(response);
29                                newDoc.close();
30                            }
31                        });
32
33                var card = $("<div>").attr("class", "col-lg-4")
34                    .append(
35                        $("<div>").attr("class", "our-team-main")
36                            .append(
37                                $("<div>").attr("class", "team-front")
38                                    .append(instr, name, level),
39
40                                $("<div>").attr("class", "team-back")
41                                    .append(email, "<br>", "<br>", $("<div>").attr({ "align": "center", "type"
42                                        : "_room" })
43                                        .append(room)
44
45                                )
46
47                            )
48
49                    );
50
51                $("#bacheca").append(card);
52
53            });
54
55            if ($("p[type='_room']").length == 0) {

```

```
56         alert("Professori non disponibili!");
57     }
58
59
60     }
61 );
62
63
64
65
66
67
68 });
```

Codice A.4: "Student.js"

A.1.4 WebRTC page

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>WebRTC application</title>
5      <style>
6          .button {
7              background-color: #DAA520;
8              border: none;
9              color: white;
10             padding: 15px 32px;
11             text-align: center;
12             text-decoration: none;
13             display: inline-block;
14             font-size: 16px;
15             margin: 10px 100px;
16             cursor: pointer;
17         }
18     </style>
19
20 </head>
21
22 <body background="img/background.jpg">
23
24     <h1 id="pageTitle" class="text-center text-success"></h1>
25
26     <div id='mainDiv'>
27         <table border="0" width="100%" style="color: #483d8b">
28             <tr>
29                 <td></td>
30                 <!-- td align="center" colspan="3" style="color: #00ff00">Video Remoto</td -->
31                 <td align="center" style="color: #483d8b">
32                     <strong>Video Remoto</strong>
33                 </td>
34                 <td></td>
35             </tr>
36             <tr>
37                 <td align="center" colspan="3" >
38                 <!-- <video width="450px" height="450" id="remoteVideo" autoplay> </video> -->
39                     <p align="center"><video width="27%" id="remoteVideo" autoplay> </video><p>
40                 </td>
41             </tr>
```

```

42
43     <tr>
44         <td align="center">
45             <textarea rows="7" cols="30" id="dataChannelSend" disabled placeholder="This will
46                 be enabled once the data channel is up..."></textarea>
47         </td>
48         <td>
49             <p align="center"> <video width="25%" id="localVideo" autoplay></video> </p>
50         </td>
51         <td align="center">
52             <textarea rows="7" cols="30" id="dataChannelReceive" disabled placeholder="This
53                 will be enabled once the data channel is up..."></textarea>
54         </td>
55     </tr>
56
57     <tr>
58         <td align="center">
59             <button id="sendButton" class="btn btn-success" disabled>Invia</button>
60         </td>
61         <td align="center" style="color: #483d8b">
62             <strong>Video Locale</strong>
63         </td>
64     </tr>
65 </table>
66
67 </div>
68 <div id="roomID" hidden></div>
69
70 <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
71 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></
72     script>
73
74 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap
75     .min.css">
76
77 <script src='socket.io/socket.io.js'></script>
78 <script src='js/lib/adapters.js'></script>
79 <script src='js/NodeClient.js'></script>
80 <br/><br/>
81 <center><a href="index.html" class="btn btn-danger">Termina</a></center>
82
83 </body>
84 </html>

```

Codice A.5: "WebRTCPage.html"

A.2 Servlet

A.2.1 Professore

```

1 package it.musicLearning;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.util.ArrayList;
6
7 import javax.servlet.RequestDispatcher;

```

```
8 import javax.servlet.ServletException;
9 import javax.servlet.annotation.WebServlet;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 import org.jsoup.Jsoup;
15 import org.jsoup.nodes.Document;
16 import org.jsoup.nodes.Element;
17
18 /**
19  * Servlet implementation class ProfList
20  */
21 @WebServlet("/Professor")
22 public class Professor extends HttpServlet {
23     private static final long serialVersionUID = 1L;
24
25
26     private ArrayList<RoomDB> rooms;
27     private static int lastID;
28
29
30
31     public static synchronized void incrementID() {
32         lastID++;
33     }
34
35
36     /**
37      * @see HttpServlet#HttpServlet()
38      */
39     public Professor() {
40         super();
41         // TODO Auto-generated constructor stub
42         try{
43             if (RoomDB.getLastID() != null)
44                 lastID = RoomDB.getLastID().getID();
45             else{
46                 lastID = 0;
47                 //System.out.println("Else");
48             }
49         }
50         catch (Exception e) {
51             // TODO: handle exception
52             lastID = 0;
53             System.out.println(e);
54         }
55
56         System.out.println("Servlet Professore istanziata");
57     }
58
59
60     /**
61      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
62      */
63     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
64         ServletException, IOException {
65         // TODO Auto-generated method stub
66         doPost(request, response);
67         // DEBUG
```



```
67     System.out.println(request.getQueryString());
68     System.out.println("DEBUG: [Professor_Servlet: doGet] -> doPost");
69
70
71 }
72
73 /**
74  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
75  */
76 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
77     // TODO Auto-generated method stub
78
79     System.out.println("DEBUG: [ProfList_Servlet: doPost]");
80
81
82     String ProfName = request.getParameter("ProfName");
83     String ProfEmail = request.getParameter("ProfEmail");
84     String ProfInstrument = request.getParameter("Instrument");
85     String ProfLevel = request.getParameter("Level");
86
87
88
89
90     if((ProfName == null || ProfName.equals("")) || (ProfEmail == null || ProfEmail.equals("
91         "")) || (ProfInstrument == null || ProfInstrument.equals("")) || (ProfLevel == null ||
92         ProfLevel.equals("")) ) {
93         System.out.println("DEBUG: [Professor_Servlet: doPost] -> No Content for Professor "
94             );
95         response.sendError(HttpServletResponse.SC_NO_CONTENT);
96         return;
97     }
98     else{
99         incrementID();
100         RoomDB room= new RoomDB(lastID, ProfName, ProfEmail, ProfInstrument, ProfLevel,1);
101         room.SaveRoom();
102         System.out.println("DEBUG: [Professor_Servlet: doPost] -> Room saved into DB!");
103
104
105         File input = new File("/var/lib/tomcat8/webapps/MusicLearning/WebRtcPage.html");
106
107         Document html = (Document) Jsoup.parse(input, "UTF-8");
108
109         Element tmp= html.getElementById("roomId");
110
111         tmp.text(Integer.toString(lastID));
112
113         tmp= html.getElementById("pageTitle");
114
115         tmp.text("VideoLezione RealTime di " + ProfInstrument + ". Difficolta': " + ProfLevel);
116
117         response.setContentType("text/html");
118         response.setHeader("Cache-Control", "no-cache");
119         response.getWriter().append(html.toString());
120
121     }
122 }
```

```
123  
124 }
```

Codice A.6: "Servlet Professore"

A.2.2 Studente

```
1 package it.musicLearning;  
2  
3 import java.io.File;  
4 import java.io.IOException;  
5 import java.util.ArrayList;  
6  
7 import javax.servlet.RequestDispatcher;  
8 import javax.servlet.ServletException;  
9 import javax.servlet.annotation.WebServlet;  
10 import javax.servlet.http.HttpServlet;  
11 import javax.servlet.http.HttpServletRequest;  
12 import javax.servlet.http.HttpServletResponse;  
13  
14 import org.jsoup.Jsoup;  
15 import org.jsoup.nodes.Document;  
16 import org.jsoup.nodes.Element;  
17  
18  
19 /**  
20  * Servlet implementation class Student  
21  */  
22 @WebServlet("/Student")  
23 public class Student extends HttpServlet {  
24     private static final long serialVersionUID = 1L;  
25  
26     /**  
27      * @see HttpServlet#HttpServlet()  
28      */  
29     public Student() {  
30         super();  
31         // TODO Auto-generated constructor stub  
32     }  
33  
34     /**  
35      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  
36      */  
37     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
38         ServletException, IOException {  
39         // TODO Auto-generated method stub  
40         doPost(request, response);  
41         // DEBUG  
42         //System.out.println(request.getQueryString());  
43         System.out.println("DEBUG: [Student_Servlet: doGet] -> doPost");  
44     }  
45  
46     /**  
47      * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  
48      */  
49     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
50         ServletException, IOException {  
51         // TODO Auto-generated method stub  
52         System.out.println("DEBUG: [Student_Servlet: doPost]");  
53     }  
54 }
```

```
51 String comand= request.getParameter("comand"); //true or false
52
53
54 String StudentName = request.getParameter("StudentName");
55 String StudentEmail = request.getParameter("StudentEmail");
56
57
58 if(comand == null || comand.equals("")){
59
60     if((StudentName == null || StudentName.equals("")) || (StudentEmail == null ||
61         StudentEmail.equals("")) ) {
62         System.out.println("DEBUG: [Student_Servlet: doPost] -> No Content for Student ");
63         response.sendError(HttpServletResponse.SC_NO_CONTENT);
64         return;
65     }
66
67     File input = new File("/var/lib/tomcat8/webapps/MusicLearning/StudentPage.html");
68     Document html = (Document) Jsoup.parse(input, "UTF-8");
69
70     Element r= html.getElementById("pageTitle");
71
72     r.text("Ciao " + StudentName + "! Scegli un Professore");
73
74     response.setContentType("text/html");
75     response.setHeader("Cache-Control", "no-cache");
76     response.getWriter().append(html.toString());
77 }
78 else{
79     switch (comand) {
80         case "getProfList":
81             ArrayList<RoomDB> rooms= RoomDB.RetrieveAvailableRoom();
82
83             response.setContentType("text/xml");
84             response.setHeader("Cache-Control", "no-cache");
85             response.getWriter().append("<response>");
86
87
88             if( rooms != null){
89
90                 for(RoomDB iter: rooms){
91
92                     response.getWriter().append("<prof>");
93
94                     response.getWriter().append("<name>");
95                     response.getWriter().append(iter.getProfName());
96                     response.getWriter().append("</name>");
97
98                     response.getWriter().append("<level>");
99                     response.getWriter().append(iter.getLevel());
100                     response.getWriter().append("</level>");
101
102                     response.getWriter().append("<email>");
103                     response.getWriter().append(iter.getProfEmail());
104                     response.getWriter().append("</email>");
105
106                     response.getWriter().append("<linkImage>");
107                     response.getWriter().append("img/" + iter.getMusicalInstrument() + ".png");
108                     response.getWriter().append("</linkImage>");
109 }
```

```
110         response.getWriter().append("<room>");
111         response.getWriter().append(Integer.toString(iter.getID()));
112         response.getWriter().append("</room>");
113
114         response.getWriter().append("</prof>");
115     }
116
117 }
118
119 response.getWriter().append("</response>");
120
121 break;
122
123 case "call":
124
125     RoomDB temp= RoomDB.RetrieveRoomByID(Integer.parseInt(request.getParameter("room")))
126     ;
127     if(temp.getNclient() == 1){
128         temp.setNclient(2);
129         temp.UpdateRoom();
130
131         File input = new File("/var/lib/tomcat8/webapps/MusicLearning/WebRtcPage.html");
132         Document html = (Document) Jsoup.parse(input, "UTF-8");
133
134         Element tmp= html.getElementById("roomId");
135
136         tmp.text(request.getParameter("room"));
137
138         tmp= html.getElementById("pageTitle");
139
140         tmp.text("VideoLezione RealTime di " + temp.getMusicalInstrument() + ". Difficolta
141             ' : " + temp.getLevel());
142
143         response.setContentType("text/html");
144         response.setHeader("Cache-Control", "no-cache");
145         response.getWriter().append(html.toString());
146
147     }
148     else{
149
150         File input = new File("/var/lib/tomcat8/webapps/MusicLearning/StudentPage.html");
151         Document html = (Document) Jsoup.parse(input, "UTF-8");
152
153         Element r= html.getElementById("pageTitle");
154
155         r.text("Sessione avviata da un altro studente! Scegli un Professore");
156
157         response.setContentType("text/html");
158         response.setHeader("Cache-Control", "no-cache");
159         response.getWriter().append(html.toString());
160
161     }
162
163
164
165
166 break;
167
```

```

168     default:
169         System.out.println("ERROR: [Student_Servlet: doPost] -> Get Professor List");
170         response.sendError(HttpServletResponse.SC_NO_CONTENT);
171         break;
172     }
173
174
175
176     }
177 }
178
179 }

```

Codice A.7: "Servlet Studente"

A.3 Node Client

```

1  'use strict';
2
3  // When I close the window, hangup() function starts.
4  window.onbeforeunload = function(e) {
5      hangup();
6  }
7
8  // Send channel and receive channel
9  var sendChannel, receiveChannel;
10 // Variables associated with HTML5 elements
11 var sendButton = document.getElementById("sendButton");
12 var sendTextarea = document.getElementById("dataChannelSend");
13 var receiveTextarea = document.getElementById("dataChannelReceive");
14 var localVideo = document.querySelector('#localVideo');
15 var remoteVideo = document.querySelector('#remoteVideo');
16 // Handler associated with 'Send' button
17 sendButton.onclick = sendData;
18 // Flags
19 var isChannelReady;
20 var isInitiator;
21 var isStarted;
22 // WebRTC local and remote stream
23 var localStream;
24 var remoteStream;
25 // Peer Connection
26 var pc;
27 // Peer Connection ICE protocol configuration and constraints
28 var pc_config = {'iceServers': [{'urls': 'stun:stun.l.google.com:19302'}]};
29 var pc_constraints = null;
30 var sdpConstraints = {'mandatory': {'OfferToReceiveAudio': true, 'OfferToReceiveVideo': true
31     }};
32
33 var room = document.getElementById("roomID").textContent;
34 // Connect to signaling server
35 var socket = io.connect(window.location.hostname+' :8181');
36 // Create or join' message to signaling server
37 if (room !== '') {
38     console.log('Create or join room', room);
39     socket.emit('create or join', room);
40 }
41 //GetUserMedia()
42 var constraints = {video: true, audio: true};

```

```

42 navigator.mediaDevices.getUserMedia(constraints)
43 .then(handleSuccess)
44 .catch( function(error) {
45 console.log("navigator.mediaDevices.getUserMedia error: ", error);
46 });
47 console.log('Getting user media with constraints', constraints);
48 // GetUserMedia() handler succes
49 function handleSuccess(stream) {
50     localStream = stream;
51     localVideo.srcObject=stream;
52     console.log('Adding local stream. ');
53     sendMessage('got user media');
54     if (isInitiator) {
55         checkAndStart();
56         console.log('Initiator');
57     }
58 }
59 //-----
60 // 1. SERVER ---> CLIENT
61 // Handle 'created' message coming back from server to initiator peer
62 socket.on('created', function (room){
63     console.log('Created room ' + room);
64     isInitiator = true;
65 });
66
67 // Handle 'full' message coming back from server
68 socket.on('full', function (room){
69     console.log('Room ' + room + ' is full');
70 });
71 // Handle 'join' message coming back from server to initiator peer
72 socket.on('join', function (room){
73     console.log('Another peer made a request to join room ' + room);
74     console.log('This peer is the initiator of room ' + room + '!');
75     isChannelReady = true;
76 });
77
78 // Handle 'joined' message coming back from server to joiner peer
79 socket.on('joined', function (room){
80     console.log('This peer has joined room ' + room);
81     isChannelReady = true;
82 });
83
84 // Handle 'log' message coming back from server to "console peer"
85 socket.on('log', function (array){
86     console.log.apply(console, array);
87 });
88
89 // Receive message from the other peer via the signaling server
90 socket.on('message', function (message){
91     console.log('Received message:', message);
92     if (message === 'got user media') {
93         checkAndStart();
94     }
95     else if (message.type === 'offer') {
96         if (!isInitiator && !isStarted) {
97             checkAndStart();
98         }
99         pc.setRemoteDescription(new RTCSessionDescription(message));
100         doAnswer();
101     }

```

```

102 else if (message.type === 'answer' && isStarted) {
103     pc.setRemoteDescription(new RTCSessionDescription(message));
104 }
105 else if (message.type === 'candidate' && isStarted) {
106     var candidate = new RTCIceCandidate({sdpMLineIndex:message.label,
107     candidate:message.candidate});
108     pc.addIceCandidate(candidate);
109 }
110 else if (message === 'bye' && isStarted) {
111     handleRemoteHangup();
112 }
113 });
114 //-----
115 // 2. CLIENT ---> SERVER
116 // Send message to the other peer via the signaling server
117 function sendMessage(message){
118     console.log('Sending message: ', message);
119     socket.emit('message', message);
120 }
121
122 // Check and Start
123 function checkAndStart() {
124     if (!isStarted && typeof localStream !== 'undefined' && isChannelReady) {
125         createPeerConnection();
126         pc.addStream(localStream);
127         isStarted = true;
128         if (isInitiator) {
129             doCall();
130         }
131     }
132 }
133
134 //-----
135 // Peer connection and create data channel
136 function createPeerConnection() {
137     console.log('call Function- createPeerConnection');
138     try {
139         pc = new RTCPeerConnection(pc_config, pc_constraints);
140         pc.onicecandidate = handleIceCandidate;
141         console.log('Created RTCPeerConnection with:\n' +
142         ' config: \'' + JSON.stringify(pc_config) + '\';\n' +
143         ' constraints: \'' + JSON.stringify(pc_constraints) + '\'.');
144     }
145     catch (e) {
146         console.log('Failed to create PeerConnection, exception: ' + e.message);
147         alert('Cannot create RTCPeerConnection object. ');
148         return;
149     }
150     pc.onaddstream = handleRemoteStreamAdded;
151     pc.onremovestream = handleRemoteStreamRemoved;
152     if (isInitiator) {
153         try {
154             // Create a reliable data channel
155             sendChannel = pc.createDataChannel("sendDataChannel",
156             {reliable:true});
157             console.log('Created send data channel');
158             console.log('readystate:' +sendChannel.readyState);
159         }
160         catch (e) {
161             alert('Failed to create data channel. ');

```

```

162     }
163     sendChannel.onopen = handleSendChannelStateChange;
164     sendChannel.onmessage = handleMessage;
165     sendChannel.onclose = handleSendChannelStateChange;
166     }
167     else { // Joiner
168         pc.ondatachannel = gotReceiveChannel;
169     }
170 }
171 // Send data from a peer to the other one
172 function sendData() {
173     var data = sendTextarea.value;
174     sendTextarea.value='';
175     if(isInitiator) sendChannel.send(data);
176     else receiveChannel.send(data);
177     console.log('Sent data: ' + data);
178 }
179
180 //-----
181 // Handlers
182 function gotReceiveChannel(event) {
183     console.log('Receive Channel Callback');
184     receiveChannel = event.channel;
185     receiveChannel.onmessage = handleMessage;
186     receiveChannel.onopen = handleReceiveChannelStateChange;
187     receiveChannel.onclose = handleReceiveChannelStateChange;
188 }
189
190 function handleMessage(event) {
191     console.log('Received message: ' + event.data);
192     receiveTextarea.value += event.data + '\n';
193 }
194
195 function handleSendChannelStateChange() {
196     var readyState = sendChannel.readyState;
197     console.log('Send channel state is: ' + readyState);
198     // If channel ready, enable user's input
199     if (readyState == "open") {
200         dataChannelSend.disabled = false;
201         dataChannelSend.focus();
202         dataChannelSend.placeholder = "";
203         sendButton.disabled = false;
204     }
205     else {
206         dataChannelSend.disabled = true;
207         sendButton.disabled = true;
208     }
209 }
210
211 function handleReceiveChannelStateChange() {
212     var readyState = receiveChannel.readyState;
213     console.log('Receive channel state is: ' + readyState);
214
215     // If channel ready, enable user's input
216     if (readyState == "open") {
217         dataChannelSend.disabled = false;
218         dataChannelSend.focus();
219         dataChannelSend.placeholder = "";
220         sendButton.disabled = false;
221     } else {

```



```
222   dataChannelSend.disabled = true;
223   sendButton.disabled = true;
224 }
225 }
226 // ICE candidates management
227 function handleIceCandidate(event) {
228   console.log('handleIceCandidate event: ', event);
229   if (event.candidate) {
230     sendMessage({
231       type: 'candidate',
232       label: event.candidate.sdpMLineIndex,
233       id: event.candidate.sdpMid,
234       candidate: event.candidate.candidate});
235   } else {
236     console.log('End of candidates.');
```

```
237   }
238 }
239
240 // Create Offer
241 function doCall() {
242   console.log('Creating Offer...');
243   pc.createOffer(sdpConstraints)
244     .then(setLocalAndSendMessage)
245     .catch(function(error) {
246       console.log('Failed to create signaling message : ' + error.name);
247     });
248 }
249
250 // Create Answer
251 function doAnswer() {
252   console.log('Sending answer to peer.');
```

```
253   pc.createAnswer(sdpConstraints)
254     .then(setLocalAndSendMessage)
255     .catch(function(error) {
256       console.log('Failed to create signaling message : ' + error.name);
257     });
258 }
259
260 // Handler success for Offer and Answer
261 function setLocalAndSendMessage(sessionDescription) {
262   pc.setLocalDescription(sessionDescription);
263   sendMessage(sessionDescription);
264 }
265
266
267 // -----
268 // Remote stream handlers
269 function handleRemoteStreamAdded(event) {
270   console.log('Remote stream added.');
```

```
271   remoteVideo.srcObject=event.stream;
272   remoteStream = event.stream;
273 }
274
275 function handleRemoteStreamRemoved(event) {
276   console.log('Remote stream removed. Event: ', event);
277 }
278
279
280 //-----
281 // Clean-up functions
```

```

282 function hangup() {
283     console.log('Hanging up. ');
284     stop();
285     sendMessage('bye');
286 }
287
288 function handleRemoteHangup() {
289     console.log('Session terminated. ');
290     stop();
291     isInitiator = false;
292 }
293
294 function stop() {
295     isStarted = false;
296     if (sendChannel) sendChannel.close();
297     if (receiveChannel) receiveChannel.close();
298     if (pc) pc.close();
299     pc = null;
300     sendButton.disabled=true;
301 }

```

Codice A.8: "NodeClient.js"

A.4 Node Server

```

1  var static = require('node-static');
2  var http = require('http');
3  // Create a node-static server instance
4  var file = new(static.Server)();
5  // We use the http module createServer function and rely on our instance of node-static to
   // serve the files
6  var app = http.createServer(function (req, res) {
7      file.serve(req, res);
8  }).listen(8181);
9  console.log('Listening on ' + app.address().port);
10 // Use socket.io JavaScript library for real-time web applications
11 var io = require('socket.io').listen(app);
12 // Connection
13 io.sockets.on('connection', function (socket){
14
15     // Handle 'message' messages
16     socket.on('message', function (message) {
17         log('S --> got message: ', message);
18         // channel-only broadcast...
19         socket.broadcast.emit('message', message);
20     });
21
22     // Handle 'create or join' messages
23     socket.on('create or join', function (room) {
24         var numClients = io.sockets.adapter.rooms[room]!==undefined ? Object.keys(io.sockets.
           adapter.rooms[room]).length:0;
25         log('S --> Room ' + room + ' has ' + numClients + ' client(s)');
26         log('S --> Request to create or join room', room);
27         console.log('Stanza: ' + room + " Clienti:" + numClients);
28         // First client joining...
29         if (numClients == 0){
30             socket.join(room);
31             socket.emit('created', room);
32         }

```

```

33     else if (numClients == 1) {
34         // Second client joining...
35         io.sockets.in(room).emit('join', room);
36         socket.join(room);
37         socket.emit('joined', room);
38     }
39     else { // max two clients
40         socket.emit('full', room);
41     }
42 });
43
44 function log(){
45     var array = [">>> "];
46     for (var i = 0; i < arguments.length; i++) {
47         array.push(arguments[i]);
48     }
49     socket.emit('log', array);
50 }
51 });

```

Codice A.9: "NodeServer.js"

A.5 Hibernate - MySQL

```

1  package it.musicLearning;
2
3  import java.util.ArrayList;
4
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.GeneratedValue;
8  import javax.persistence.Id;
9
10 import org.hibernate.Query;
11 import org.hibernate.Session;
12
13 @Entity
14 public class RoomDB {
15
16     @Id
17     // @GeneratedValue
18     @Column(name="id",unique=true,nullable=false)
19     private int ID;
20
21
22     @Column(name="prof_name",unique=false,nullable=false)
23     private String ProfName;
24
25     @Column(name="prof_email",unique=false,nullable=false)
26     private String ProfEmail;
27
28     @Column(name="musical_instrument",unique=false,nullable=false)
29     private String MusicalInstrument;
30
31     @Column(name="level",unique=false,nullable=false)
32     private String Level;
33
34     @Column(name="n_client",unique=false,nullable=false)
35     private int Nclient;

```

```
36
37 // ----- Constructors -----
38 public RoomDB() {}
39
40 public RoomDB(int id, String profName, String profEmail, String musicalInstrument, String
    level,int nClient){
41     this.ID = id;
42     this.ProfName = profName;
43     this.ProfEmail = profEmail;
44     this.MusicalInstrument = musicalInstrument;
45     this.Level = level;
46     this.Nclient = nClient;
47
48 }
49
50 // ----- Set & Get -----
51 public int getID(){
52     return ID;
53 }
54
55 public void setID(int id){
56     ID = id;
57 }
58
59 public int getNclient(){
60     return Nclient;
61 }
62
63 public void setNclient(int nClient){
64     Nclient = nClient;
65 }
66
67 public String getProfName() {
68     return ProfName;
69 }
70
71 public void setProfName(String profName) {
72     ProfName = profName;
73 }
74
75 public String getProfEmail() {
76     return ProfEmail;
77 }
78
79 public void setProfEmail(String profEmail) {
80     ProfEmail = profEmail;
81 }
82
83 public String getMusicalInstrument() {
84     return MusicalInstrument;
85 }
86
87 public void setMusicalInstrument(String musicalInstrument) {
88     MusicalInstrument = musicalInstrument;
89 }
90
91 public String getLevel() {
92     return Level;
93 }
94
```

```
95 public void setLevel(String level) {
96     Level = level;
97 }
98
99
100 //----- DATABASE ACCESS OPERATION -----
101
102 // Save
103 public void SaveRoom() {
104     Session session = HibernateUtil.getSessionFactory().openSession();
105     try {
106
107         session.getTransaction().setTimeout(2);
108         session.beginTransaction();
109
110         //Save object to DB
111         session.save(this);
112
113
114         session.getTransaction().commit();
115
116     }
117     catch (RuntimeException e) {
118         System.out.println("\n\nException: Room Saving!\n\n");
119         session.getTransaction().rollback();
120         throw e;
121     }
122     finally {
123         session.close();
124     }
125 }
126
127
128 // Update
129 public void UpdateRoom() {
130     Session session = HibernateUtil.getSessionFactory().openSession();
131     try {
132
133         session.getTransaction().setTimeout(2);
134         session.beginTransaction();
135
136         //Update object to DB
137         session.update(this);
138
139
140         session.getTransaction().commit();
141     }
142     catch (RuntimeException e) {
143         System.out.println("\n\nException: Room Updating!\n\n");
144         session.getTransaction().rollback();
145         throw e;
146     }
147     finally {
148         session.close();
149     }
150 }
151
152 // Delete
153 public void DeleteRoom() {
154     Session session = HibernateUtil.getSessionFactory().openSession();
```

```
155     try {
156         session.getTransaction().setTimeout(2);
157         session.beginTransaction();
158
159         //Delete object from DB
160         session.delete(this);
161
162         session.getTransaction().commit();
163     }
164     catch (RuntimeException e) {
165         System.out.println("\n\nException: Room Deleting!\n\n");
166         session.getTransaction().rollback();
167         throw e;
168     }
169     finally {
170         session.close();
171     }
172 }
173
174 //Query1 - Recupera tutte le ROOM
175 @SuppressWarnings("unchecked")
176 public static ArrayList<RoomDB> RetrieveRooms() {
177     Session session = HibernateUtil.getSessionFactory().openSession();
178     try {
179         Query query;
180         session.getTransaction().setTimeout(2);
181         session.beginTransaction();
182
183         //Query
184         query=session.createQuery("from RoomDB");
185
186
187         ArrayList <RoomDB> result = (ArrayList<RoomDB>) query.list();
188
189         session.getTransaction().commit();
190
191         if(result.isEmpty())
192             return null;
193         else
194             return result; //restituisce tutte le Room
195     }
196     catch (RuntimeException e) {
197         System.out.println("\n\nEccezione: Query1 Room!\n\n");
198         session.getTransaction().rollback();
199         throw e;
200     }
201     finally {
202         session.close();
203     }
204 }
205 }
206
207
208 @SuppressWarnings("unchecked")
209 //Query2 - Recupera Stanza attraverso ID
210 public static RoomDB RetrieveRoomByID(int ID) {
211     Session session = HibernateUtil.getSessionFactory().openSession();
212     try {
213         Query query;
214         session.getTransaction().setTimeout(2);
```

```
215         session.beginTransaction();
216
217         //Query
218         query=session.createQuery("from RoomDB WHERE id = :room_id");
219         query.setParameter("room_id", ID);
220
221
222         ArrayList <RoomDB> result = (ArrayList<RoomDB>) query.list();
223
224         session.getTransaction().commit();
225
226         if(result.isEmpty())
227             return null;
228         else
229             return result.get(0); //restituisce la prima room della lista
230
231     }
232     catch (RuntimeException e) {
233         System.out.println("\n\nEccezione: Query2 Room!\n\n");
234         session.getTransaction().rollback();
235         throw e;
236     }
237     finally {
238         session.close();
239     }
240 }
241
242 //Query3 - Recupera tutte le ROOM
243 @SuppressWarnings("unchecked")
244 public static RoomDB getLastID() {
245     Session session = HibernateUtil.getSessionFactory().openSession();
246     try {
247         Query query;
248         session.getTransaction().setTimeout(6);
249         session.beginTransaction();
250
251         //Query
252         query=session.createQuery("from RoomDB WHERE id = (SELECT MAX(id) FROM RoomDB)");
253
254         ArrayList <RoomDB> result = (ArrayList<RoomDB>) query.list();
255
256
257         session.getTransaction().commit();
258
259         if(result.isEmpty())
260             return null;
261         else
262             return result.get(0); //restituisce la prima room della lista
263
264
265     }
266     catch (RuntimeException e) {
267         System.out.println("\n\nEccezione: Query3 Room!\n\n" + e);
268         session.getTransaction().rollback();
269         throw e;
270     }
271     finally {
272         session.close();
273     }
274 }
```

```
275 }
276
277 @SuppressWarnings("unchecked")
278 //Query4 - Recupera Stanza attraverso ID
279 public static ArrayList<RoomDB> RetrieveAvailableRoom() {
280     Session session = HibernateUtil.getSessionFactory().openSession();
281     try {
282         Query query;
283         session.getTransaction().setTimeout(2);
284         session.beginTransaction();
285
286         //Query
287         query=session.createQuery("from RoomDB WHERE n_client = 1");
288
289         ArrayList <RoomDB> result = (ArrayList<RoomDB>) query.list();
290
291         session.getTransaction().commit();
292
293         if(result.isEmpty())
294             return null;
295         else
296             return result; //restituisce la prima room della lista
297     }
298     catch (RuntimeException e) {
299         System.out.println("\n\nEccezione: Query4 Room!\n\n");
300         session.getTransaction().rollback();
301         throw e;
302     }
303     finally {
304         session.close();
305     }
306 }
307 }
308 }
```

Codice A.10: "RoomDB"