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

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.

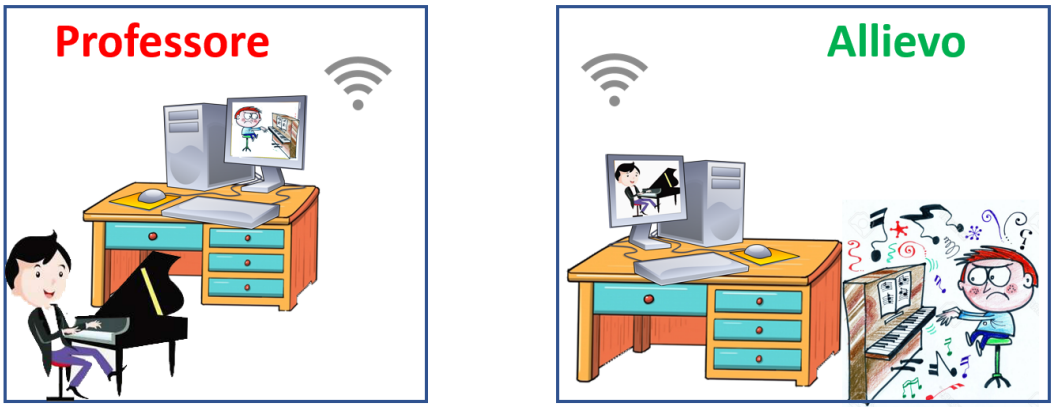


Figura 1.1: Esempio d'uso

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**;

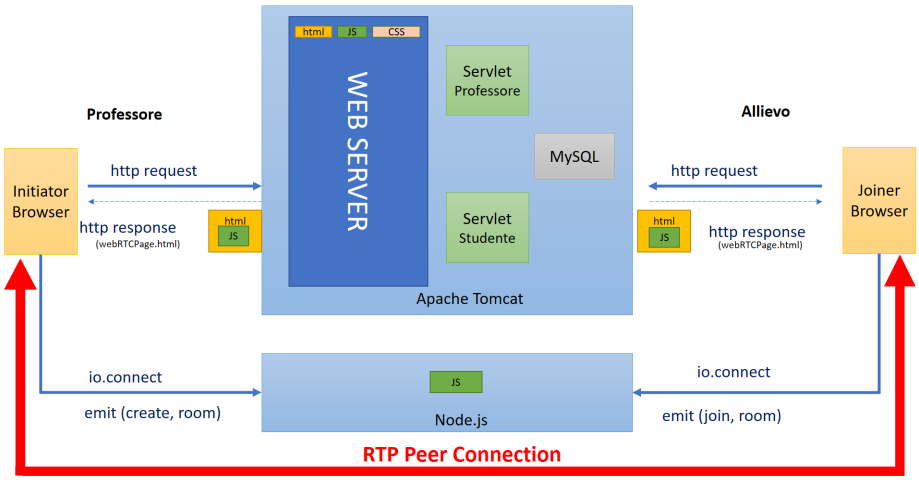


Figura 1.2: Architettura WebRTC Music Learning

Capitolo 2

Sviluppo

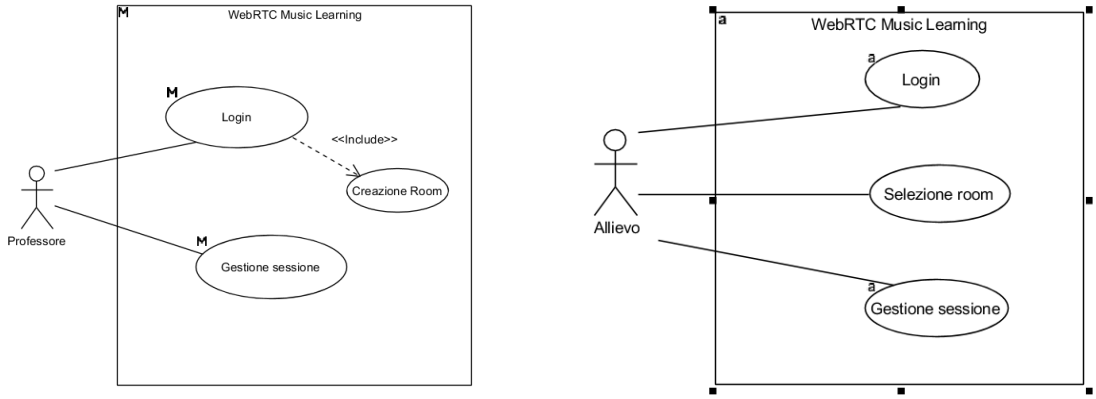
2.1 Analisi del sistema

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.

(a) Professore **(b)** Allievo

Figura 2.1: Si mostrano form diverse per i due utenti



(a) Professore **(b)** Allievo

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`.

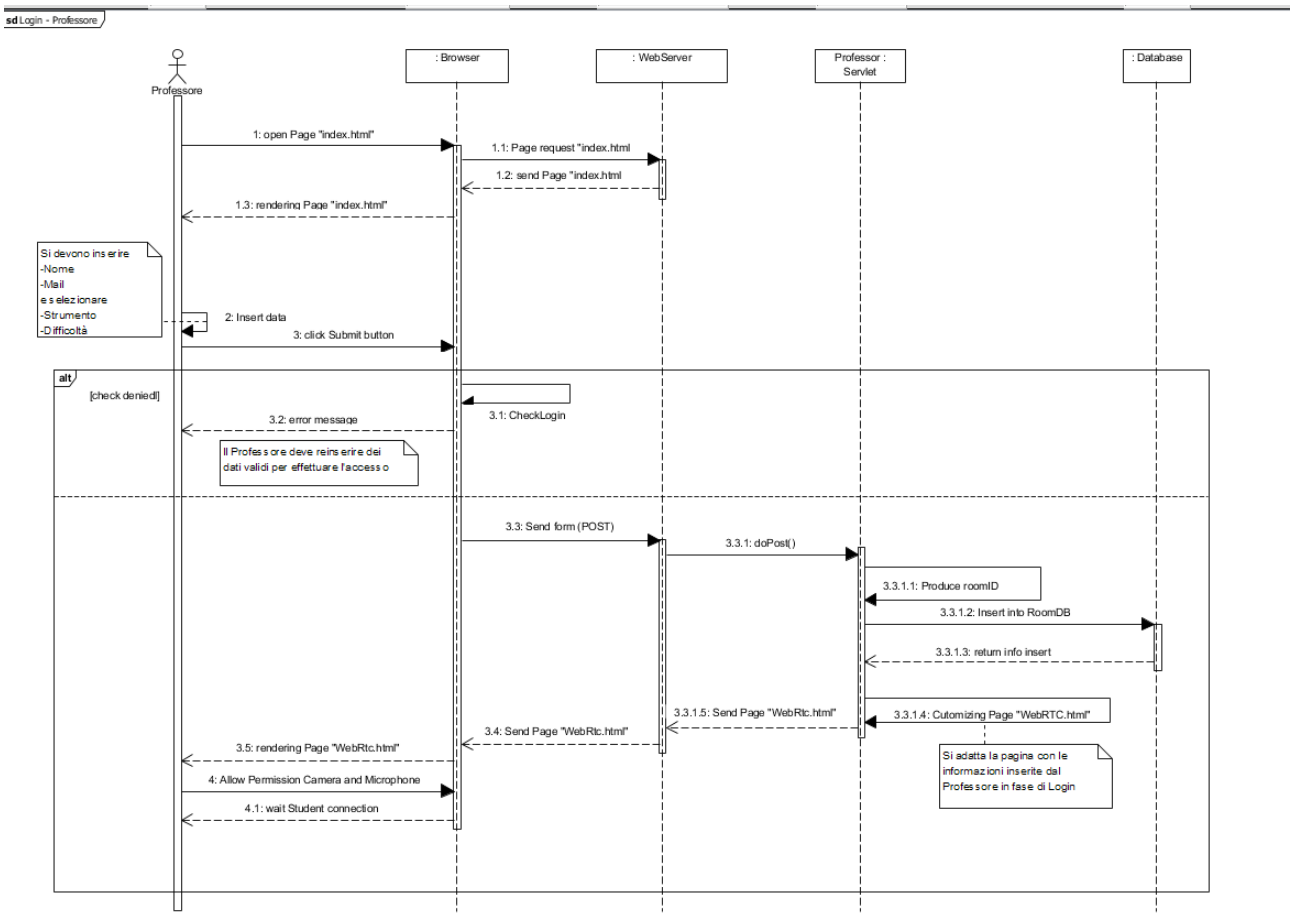


Figura 2.3: Sequence Diagram - Professore

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 masismo 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()"

Creato 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 {
```

```
  // TODO Auto-generated method stub

  System.out.println("DEBUG: [ProfList_Servlet: doPost]");


  String ProfName = request.getParameter("ProfName");
  String ProfEmail = request.getParameter("ProfEmail");
  String ProfInstrument = request.getParameter("Instrument");
  String ProfLevel = request.getParameter("Level");




  if((ProfName == null || ProfName.equals("")) || (ProfEmail == null || ProfEmail.equals("")
      )
    || (ProfInstrument == null || ProfInstrument.equals("")) || (ProfLevel == null ||
       ProfLevel.equals("")) ) {
      System.out.println("DEBUG: [Professor_Servlet: doPost] -> No Content for Professor ");
      response.sendError(HttpServletResponse.SC_NO_CONTENT);
      return;
  }
  else{
    incrementID();
    RoomDB room= new RoomDB(lastID, ProfName, ProfEmail, ProfInstrument, ProfLevel,1);
//      rooms.add(r); // Add to Local List of Rooms
    room.SaveRoom();
    System.out.println("DEBUG: [Professor_Servlet: doPost] -> Room saved into DB!");


  File input = new File("/var/lib/tomcat8/webapps/MusicLearning WebRtcPage.html");


    Document html = (Document) Jsoup.parse(input, "UTF-8");

    Element tmp= html.getElementById("roomID");

    tmp.text(Integer.toString(lastID));

    tmp= html.getElementById("pageTitle");

    tmp.text("VideoLezione RealTime di "+ ProfInstrument + ". Difficolta': "+ ProfLevel);
    response.setContentType("text/html");
    response.setHeader("Cache-Control", "no-cache");
    response.getWriter().append(html.toString());
  }

}
```

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`.

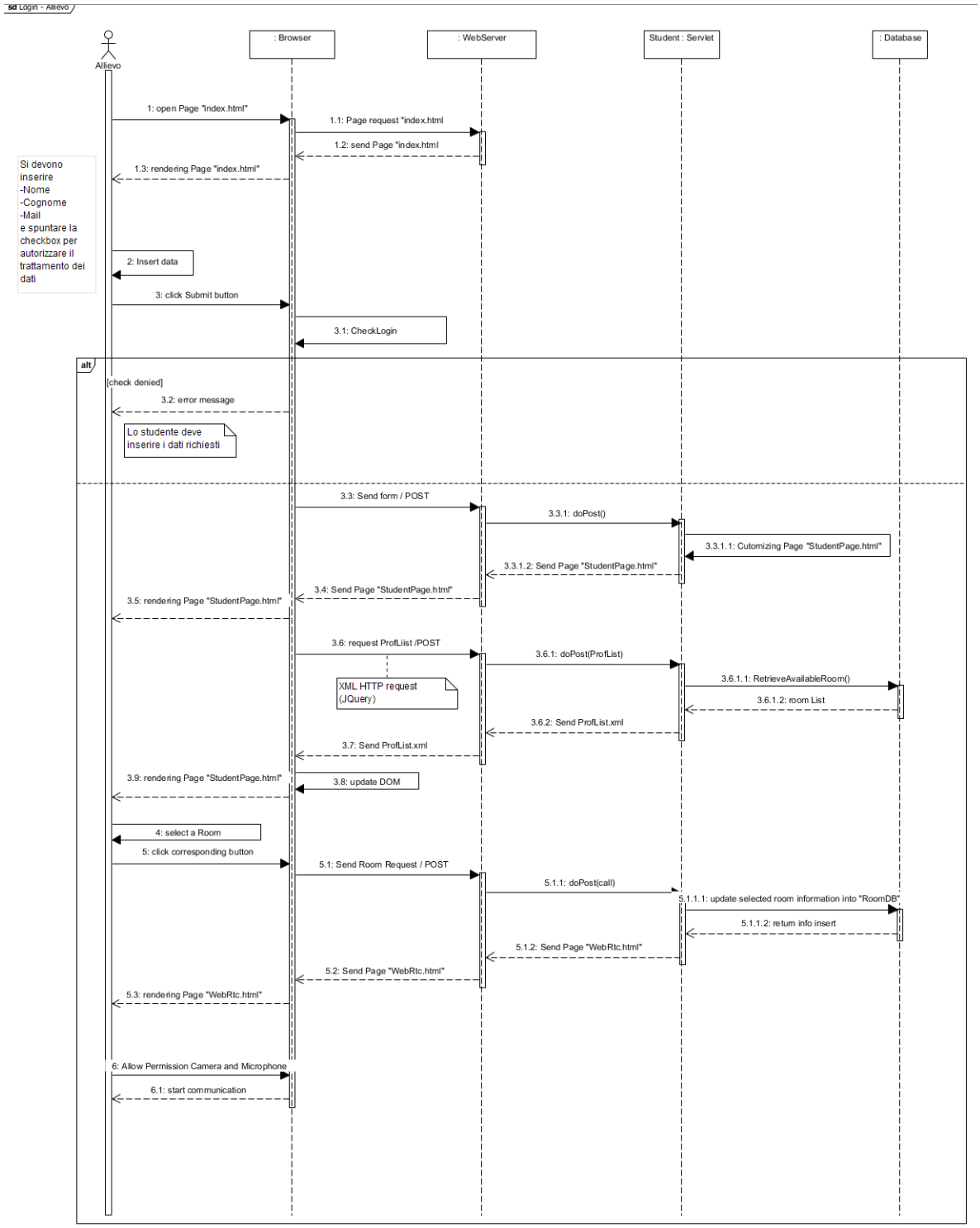


Figura 2.4: Sequence Diagram - Allievo

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.

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

```
Element r= html.getElementById("pageTitle");

r.text("Ciao "+ StudentName +"!  Scegli un Professore");

response.setContentType("text/html");
response.setHeader("Cache-Control", "no-cache");
response.getWriter().append(html.toString());
```

Codice 2.3: "Customizing StudentPage.html"

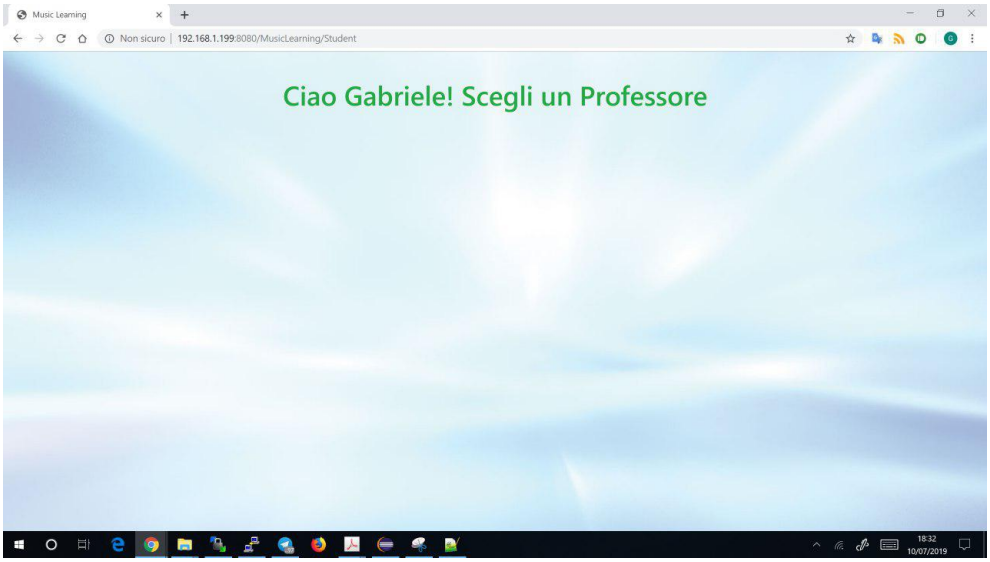


Figura 2.5: Rendering di `StudentPage.html` prima della richiesta asincrona

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.

```
$(document).ready(function() {

$.post("Student",{comand:"getProfList"},
    function(xml) {

      $(xml).find("prof").each(function(){

        var name = $("<h3>").text($(this).find("name").text());
        var level =  $("<p>").text($(this).find("level").text());
        var instr = $("<img>").attr("class","img-fluid").attr("src",$(this).find("linkImage"
            ).text());

        var email= $("<span>").text($(this).find("email").text());
        var room = $("<p>").attr("class","btn btn-primary").attr({"type":"_room"}).attr("
            room",$(this).find("room").text()).text("Accedi")
           .on("click", function(){

              var r= $(this).attr("room");
              $.ajax({
```

```
                    type: 'POST',
                    url: 'Student',
                    data: {comand : 'call', room:r},
                    success: function(response) {
                      // re-writes the entire document
                      var newDoc = document.open("text/html", "replace");

                      newDoc.write(response);
                      newDoc.close();
                    }
                  });

              });

        var card=$("<div>").attr("class","col-lg-4")
            .append(
                $("<div>").attr("class","our-team-main")
                .append(
                    $("<div>").attr("class","team-front")
                    .append(instr,name,level),

                    $("<div>").attr("class","team-back")
                    .append(email,"<br>","<br>",$("<div>").attr({"align":"center","type":"
                       _room"})
                                .append(room)

                        )

                    )

                );


        $("#bacheca").append(card);

      });

      if($("p[type='_room']").length == 0){
        alert("Professori non disponibili!");
      }


    }
);

});
```

Codice 2.4: "Student.js"

```
case "getProfList":
ArrayList<RoomDB> rooms= RoomDB.RetrieveAvailableRoom();


response.setContentType("text/xml");
response.setHeader("Cache-Control", "no-cache");
response.getWriter().append("<response>");


if( rooms != null){
```

```
  for(RoomDB iter: rooms){

    response.getWriter().append("<prof>");

      response.getWriter().append("<name>");
        response.getWriter().append(iter.getProfName());
      response.getWriter().append("</name>");

      response.getWriter().append("<level>");
        response.getWriter().append(iter.getLevel());
      response.getWriter().append("</level>");

      response.getWriter().append("<email>");
        response.getWriter().append(iter.getProfEmail());
      response.getWriter().append("</email>");

      response.getWriter().append("<linkImage>");
        response.getWriter().append("img/"+ iter.getMusicalInstrument( +".png");
      response.getWriter().append("</linkImage>");

      response.getWriter().append("<room>");
        response.getWriter().append(Integer.toString(iter.getID()));
      response.getWriter().append("</room>");

    response.getWriter().append("</prof>");
  }

}

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.

```
case "call":
    RoomDB temp= RoomDB.RetrieveRoomByID(Integer.parseInt(request.getParameter("room")));
        if(temp.getNclient() == 1){
          temp.setNclient(2);
          temp.UpdateRoom();


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

          //response.addHeader("roomID", request.getParameter("room"));
          //System.out.println(html.select("#roomID"));
          Element tmp= html.getElementById("roomID");

          tmp.text(request.getParameter("room"));

          tmp= html.getElementById("pageTitle");

          tmp.text("VideoLezione RealTime di "+ temp.getMusicalInstrument() + ". Difficolta:
              "+ temp.getLevel());
```

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

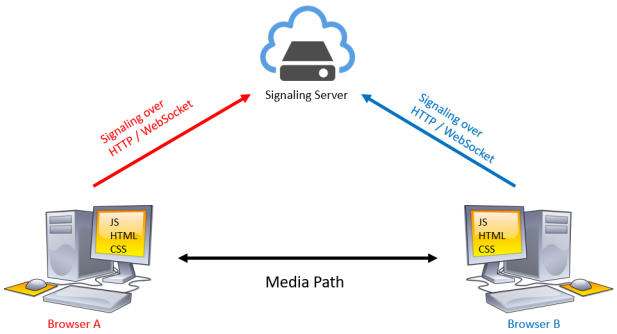


Figura 3.1: Architettura WebRTC

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 libreria JavaScript `socket.io`.

3.2 Logica

3.2.1 Sequence Diagram

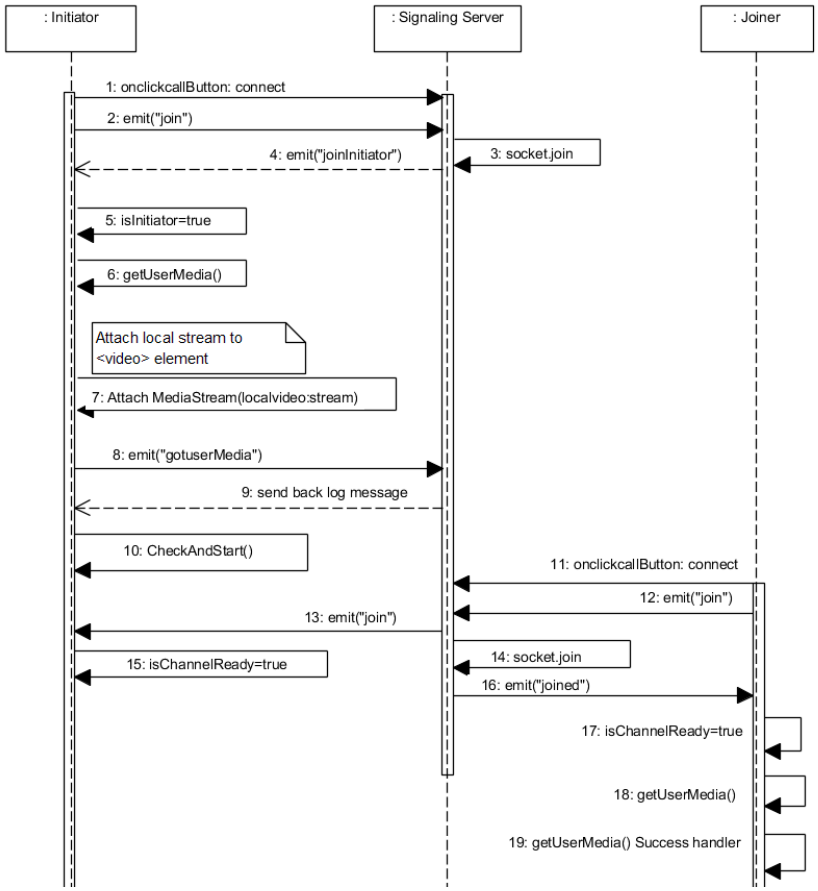


Figura 3.2: Sequence Diagram WebRTC parte 1

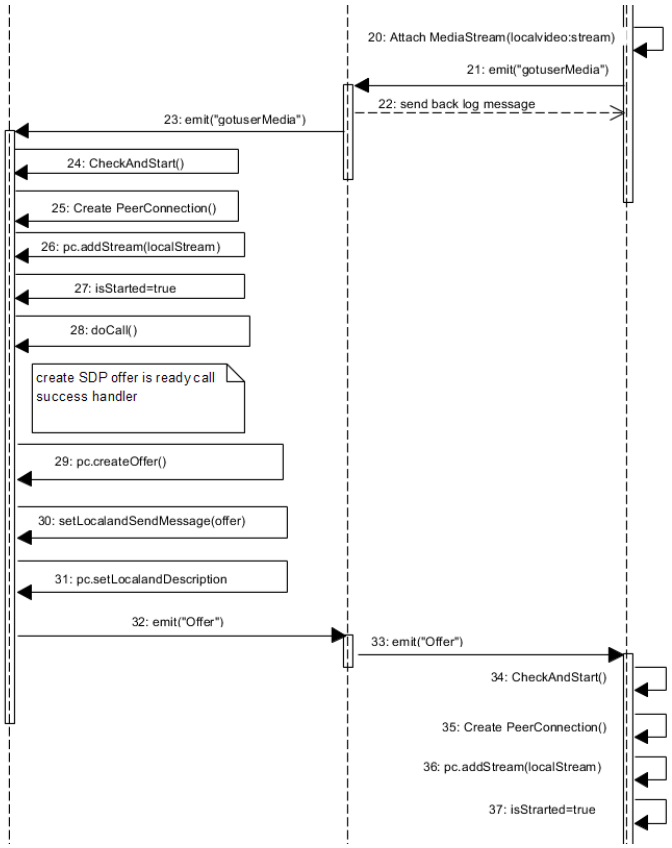


Figura 3.3: Sequence Diagram WebRTC parte 2

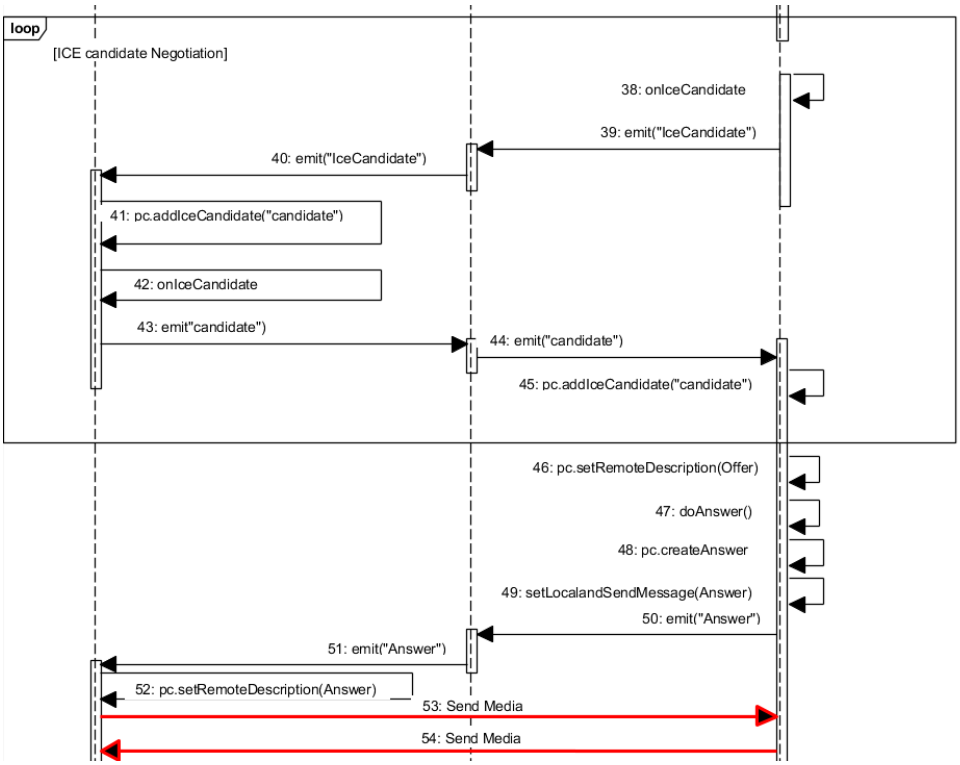


Figura 3.4: Sequence Diagram WebRTC parte 3

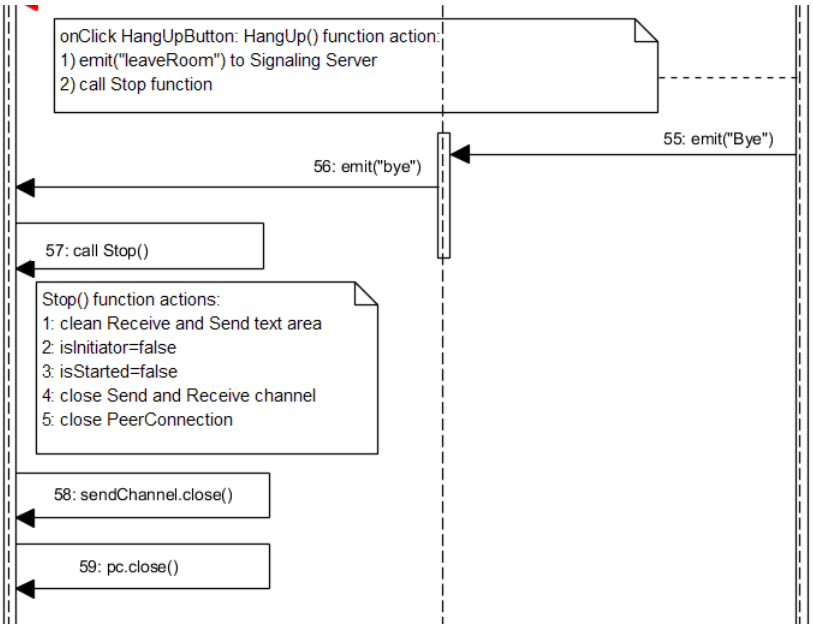


Figura 3.5: Sequence Diagram WebRTC parte 4

3.2.2 Codice

3.2.2.1 NodeServer.js

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

```
socket.on('create or join', function (room) {
  var numClients = io.sockets.adapter.rooms[room]!=undefined ? Object.keys(io.sockets.
      adapter.rooms[room]).length:0;
  log('S --> Room ' + room + ' has ' + numClients + ' client(s)');
  log('S --> Request to create or join room', room);
  console.log('Stanza: = ' + room + " Clienti:" + numClients);
  // First client joining...
  if (numClients == 0){
    socket.join(room);
    socket.emit('created', room);
  }
  else if (numClients == 1) {
    // Second client joining...
    io.sockets.in(room).emit('join', room);
    socket.join(room);
    socket.emit('joined', room);
  }
  else { // max two clients
    socket.emit('full', room);
  }
});
```

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`.

```
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`.

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

Codice 3.3: "windows.onbeforeunload"

```
// Clean-up functions
function hangup() {
  console.log('Hanging up.');
  stop();
  sendMessage('bye');
}

function handleRemoteHangup() {
  console.log('Session terminated.');
  stop();
  isInitiator = false;
}

function stop() {
  isStarted = false;
  if (sendChannel) sendChannel.close();
  if (receiveChannel) receiveChannel.close();
  if (pc) pc.close();
  pc = null;
  sendButton.disabled=true;
}
```

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

Capitolo 4

Demo Music Learning

4.1 Professore

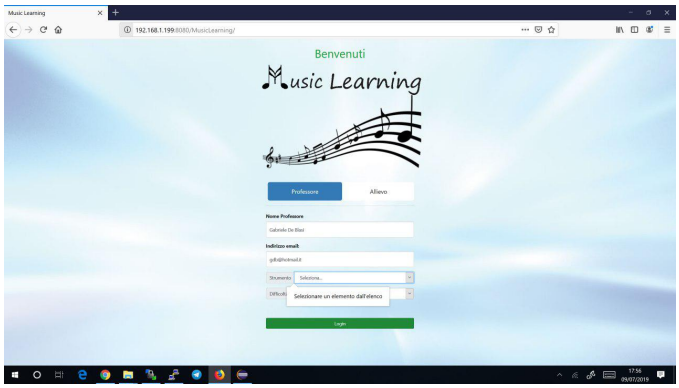


Figura 4.1: Form Validation Professore

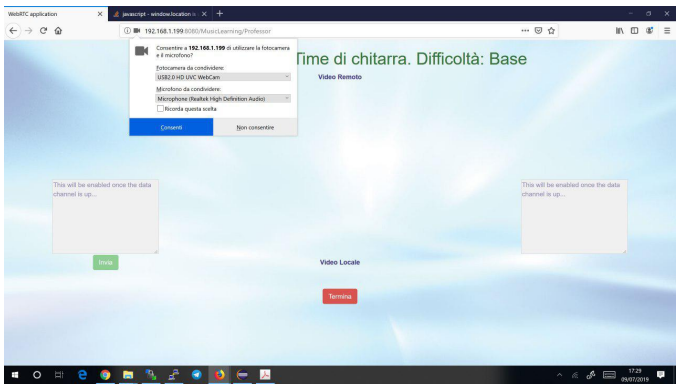


Figura 4.2: Il professore deve acconsentire l'utilizzo della videocamera e del microfono

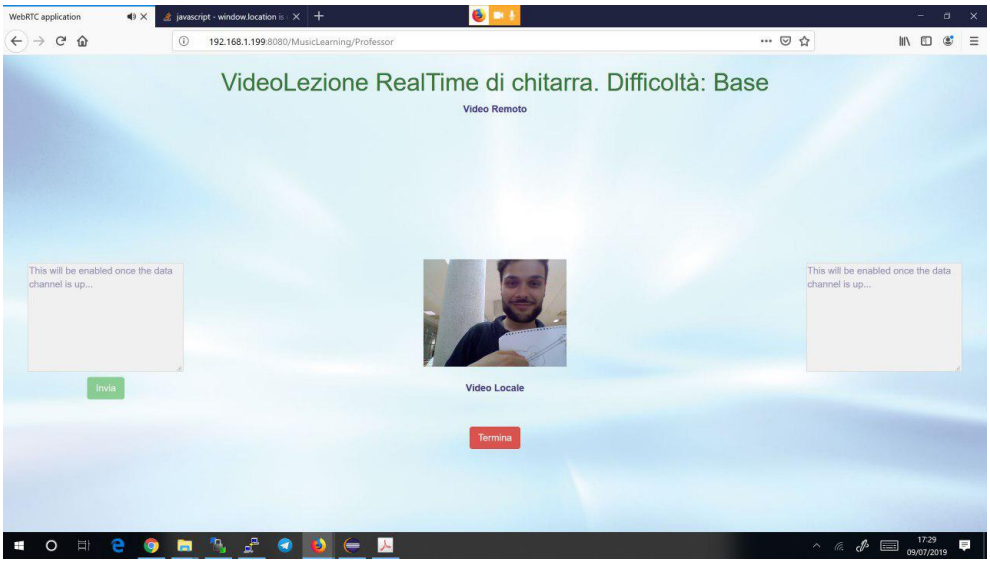


Figura 4.3: Il professore attende un nuovo allievo

4.2 Allievo

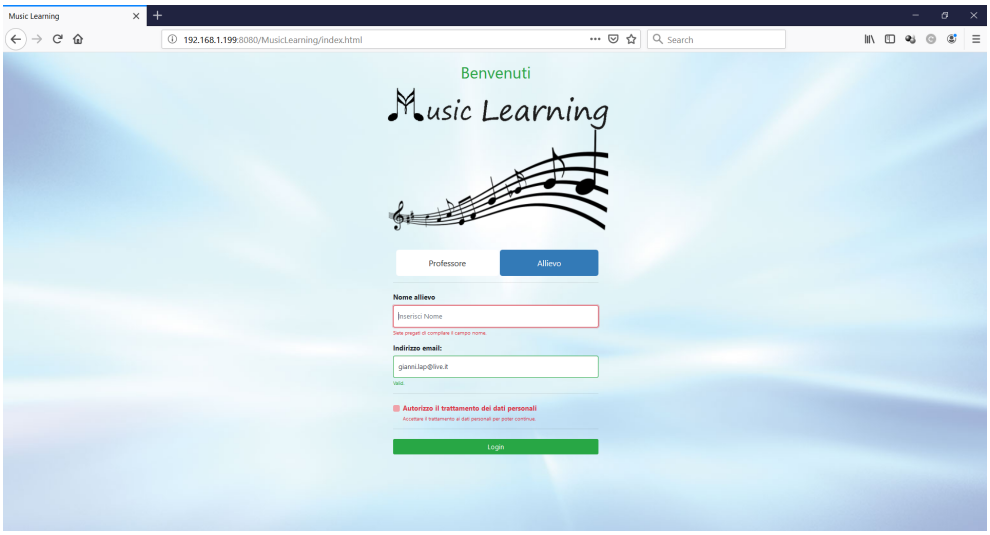


Figura 4.4: Form Validation allievo

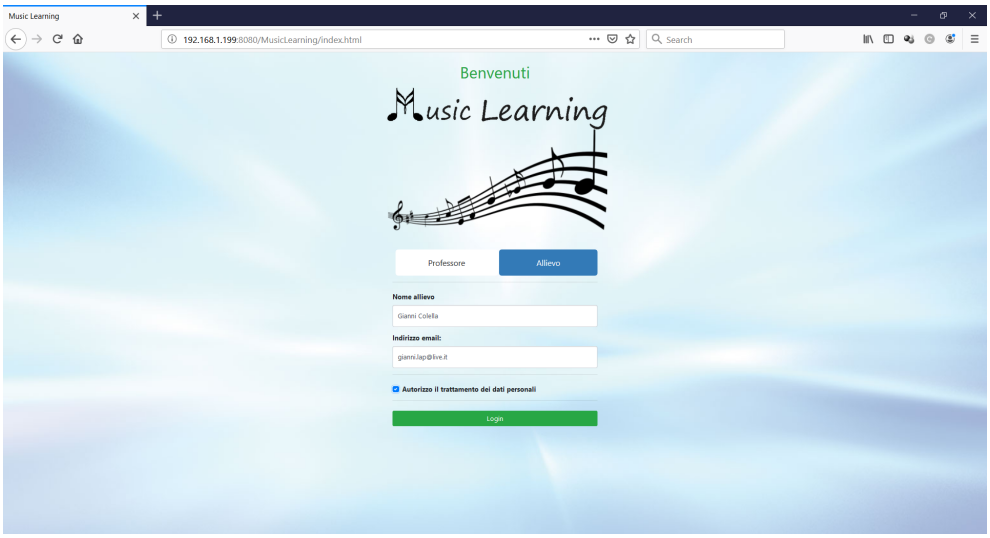


Figura 4.5: L'allievo è pronto a cliccare sul bottone Login

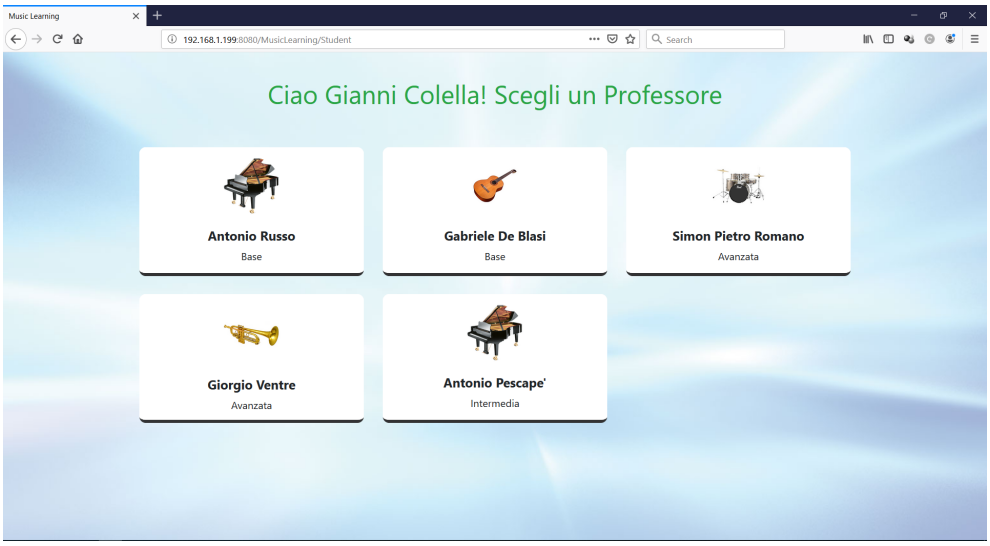


Figura 4.6: L'allievo è reinderizzato alla risorsa Student

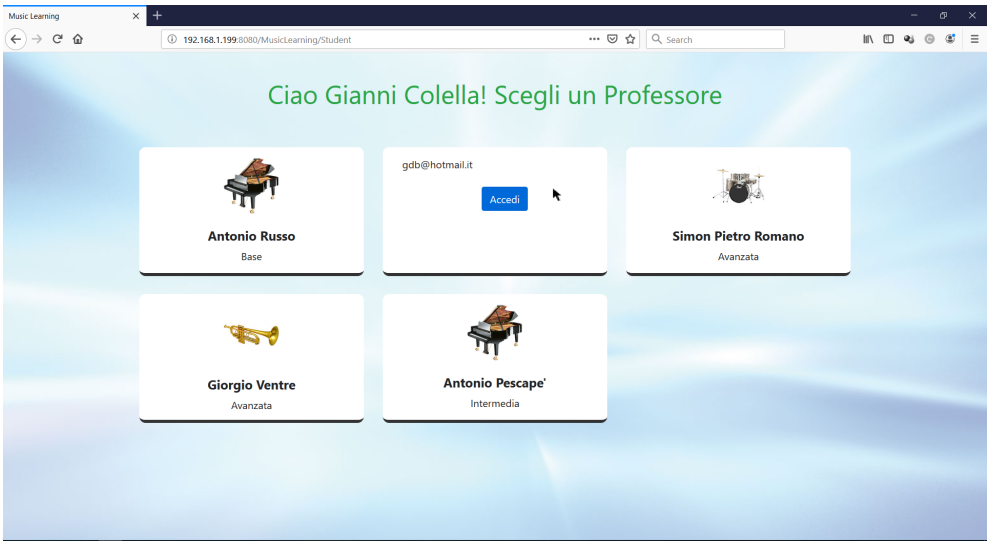
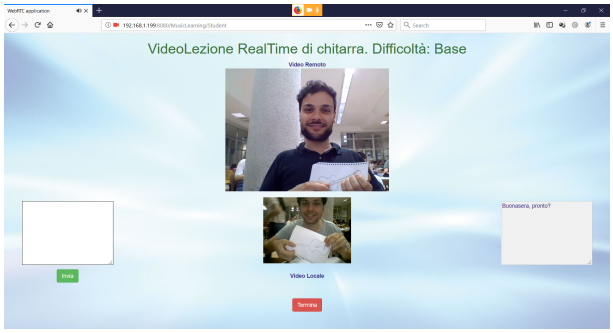
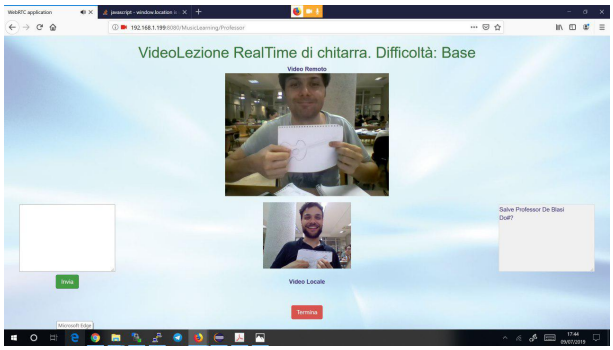


Figura 4.7: L'allievo seleziona il professore desiderato

4.3 Connessione stabilita



(a) Vista Allievo



(b) Vista Professore

Figura 4.8: La videoLezione può cominciare

4.4 Scenari alternativi

Nessun professore presente

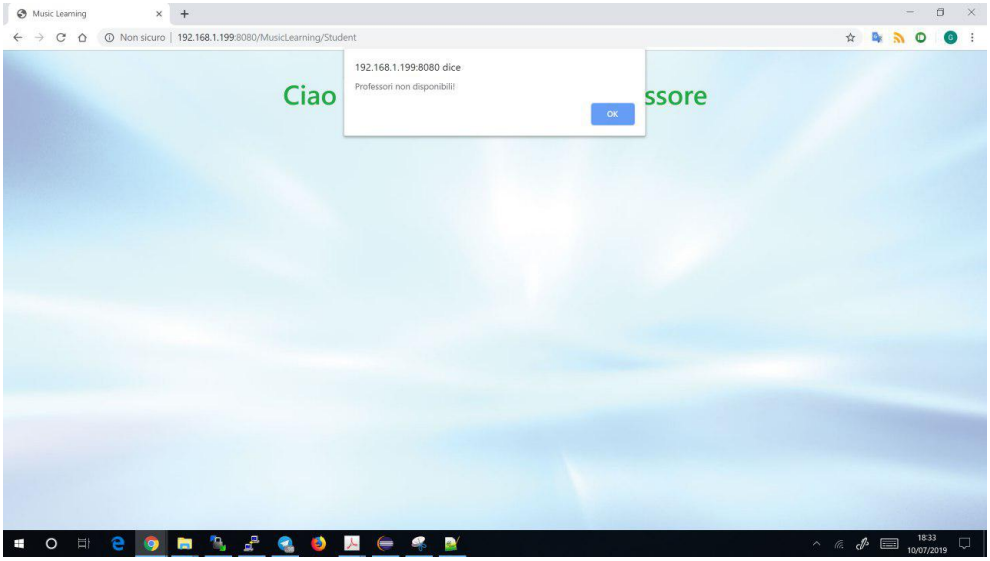


Figura 4.9: Un allert espone il problema all'allievo

Room selezionata occupata

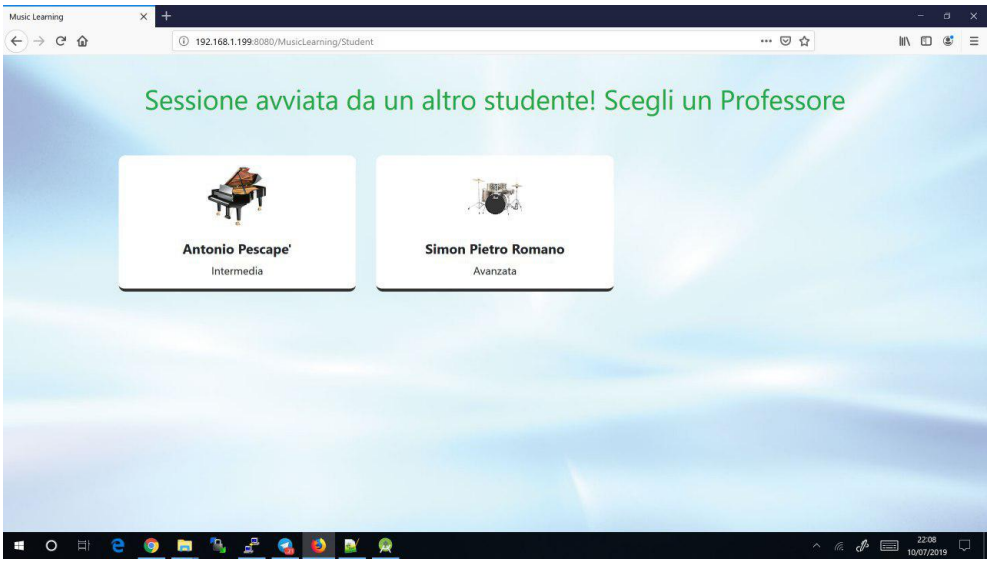


Figura 4.10: L'allievo deve scegliere un altra room

Appendice A

Codici

In seguito sono riportati tutti i codici utilizzati:

A.1 WebServer

A.1.1 Homepage

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Music Learning</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap
      .min.css">
  <!-- script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></
      script-->
  <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script
      >


<!-- Documentation extras -->

<link href="https://cdn.jsdelivr.net/npm/docsearch.js@2/dist/cdn/docsearch.min.css" rel="
    stylesheet">

  <!-- codice jQuery -->
  <script type="text/javascript" src="CheckLogin.js"></script>

    <!-- Bootstrap core CSS -->
    <link href="index.css" rel="stylesheet">



</head>



<body background="img/background.jpg">

<div class="container"><!-- style="margin-top: 5%;">-->
  <div class="row">
    <div class="col-sm-4"> </div>
```

```
34    <div class="col-md-4">
35      <h1 class="text-center text-success"> Benvenuti </h1>
36
37        <img class="img-fluid" src="img/title.png" alt="Title" width="460" height="92">
38
39        <img class="img-fluid" src="img/music_note.png" alt="NotaMusicale" width="460" height=
            "345">
40
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
              ="tab" href="#menuProfessore">Professore</a></li>
50
51            <li class="" style="width:48%"><a class=" btn btn-lg btn-default" data-toggle="tab
              " href="#menuAllievo">Allievo</a></li>
52
53          </ul>
54
55        <hr class="mb-4">
56
57
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"
                name="ProfName" required>
68          <div class="valid-feedback">Valid.</div>
69          <div class="invalid-feedback"> Siete pregati di compilare il campo nome.</div>
70        </div>
71
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"
                name="ProfEmail" required>
77          <div class="invalid-feedback">
78                E' richiesto un indirizzo email valido.
79             </div>
80         </div>
81
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
```

```
          <option value>Seleziona...</option>
          <option value="tromba">Tromba</option>
          <option value="pianoforte">Pianoforte</option>
          <option value="batteria">Batteria</option>
          <option value="chitarra">Chitarra</option>
        </select>
      </div>

      <div class="input-group mb-3">
        <div class="input-group-prepend">
          <label class="input-group-text" for="difficoult">Difficoltà</label>
        </div>
        <select class="form-control" id="difficoult" required name="Level">

          <option value>Seleziona...</option>
          <option value="Base">Base</option>
          <option value="Intermedia">Intermedia</option>
          <option value="Avanzata">Avanzata</option>
        </select>
      </div>

<br/>

<br/>




  <button type="submit" class="btn btn-block btn-success">Login</button>

</form>
<br/>



    </div>

    <div id="menuAllievo" class="tab-pane fade">

<form method="post" action="Student" class="needs-validation" novalidate>

  <div class="form-group">
    <label for="name">Nome allievo</label>
    <input type="text" class="form-control" id="name" placeholder="Inserisci Nome" name="
        StudentName" required>
  <div class="valid-feedback">Valid.</div>
  <div class="invalid-feedback"> Siete pregati di compilare il campo nome.</div>
  </div>

<!--    <div class="form-group">
    <label for="surname">Cognome allievo</label>
    <input type="text" class="form-control" id="surname" placeholder="Inserisci cognome"
        name="uname" required>
  <div class="valid-feedback">Valid.</div>
  <div class="invalid-feedback">  Siete pregati di compilare il campo cognome.</div>
  </div> -->


  <div class="form-group">
```

```
    <label for="email">Indirizzo email:</label>
    <input type="email" class="form-control" id="email" placeholder="Inserisci indirizzo email" name="StudentEmail" required>
  <div class="valid-feedback">Valid.</div>
  <div class="invalid-feedback"> Inserire una mail valida</div>

  </div>



<hr class="mb-4">


  <div class="custom-control custom-checkbox">
    <input type="checkbox" class="custom-control-input" id="accept-info" required >
    <label class="custom-control-label" for="accept-info">Autorizzo il trattamento dei dati personali</label>
    <div class="invalid-feedback">Accettare il trattamento ai dati personali per poter continue.</div>
  </div>
<hr class="mb-4">


  <button type="submit" class="btn btn-block btn-success">Login</button>

</form>
<br/>




    </div>


   </div>
  </div>
</div>
</div>
</div>

</body>
</html>
```

Codice A.1: "index.html"

A.1.2 Controllo Form

```
/**
 *  Check Login JavaScript
 */

// Disable form submissions if there are invalid fields
(function() {
  'use strict';
  window.addEventListener('load', function() {
    // Get the forms we want to add validation styles to
    var forms = document.getElementsByClassName('needs-validation');
    // Loop over them and prevent submission
    var validation = Array.prototype.filter.call(forms, function(form) {
```

```
      form.addEventListener('submit', function(event) {
        if (form.checkValidity() === false) {
          event.preventDefault();
          event.stopPropagation();
        }
        form.classList.add('was-validated');
      }, false);
    });
  }, false);
})();
```

Codice A.2: "CheckLogin.js"

A.1.3 Bacheca Room

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- jQuery first, then Popper.js, then Bootstrap JS -->
  <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
    <!--script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.
        js" integrity="sha384-
        UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="
        anonymous"></script-->
  <script src="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"></script>

  <script type="text/javascript" src="StudentPage.js"></script>


    <!-- Bootstrap CSS -->
  <link href="//maxcdn.bootstrapcdn.com/bootstrap/4.1.1/css/bootstrap.min.css" rel="
      stylesheet" id="bootstrap-css">
  <!-- Bootstrap core CSS -->
    <link href="StudentPage.css" rel="stylesheet">


    <title>Music Learning</title>
  </head>

  <body background="img/background.jpg">


    <h1 id="pageTitle" class="text-center"></h1>


  <div class="container">
    <div id="bacheca" class="row">



    </div>
  </div>
  </body>
</html>
```

Codice A.3: "StudentPage.html"

A.1.3.1 Funzione asincrona per il caricamento della bacheca

```
$(document).ready(function() {


    $.post("Student",{comand:"getProfList"},
        function(xml) {

          $(xml).find("prof").each(function(){

            var name = $("<h3>").text($(this).find("name").text());
            var level =  $("<p>").text($(this).find("level").text());
            var instr = $("<img>").attr("class","img-fluid").attr("src",$(this).find("
                linkImage").text());

            var email= $("<span>").text($(this).find("email").text());
            var room = $("<p>").attr("class","btn btn-primary").attr({"type":"_room"}).attr(
                "room",$(this).find("room").text()).text("Accedi")
              .on("click", function(){

                var r= $(this).attr("room");
                $.ajax({
                    type: 'POST',
                    url: 'Student',
                    data: {comand : 'call', room:r},
                    success: function(response) {
                      // re-writes the entire document
                      var newDoc = document.open("text/html", "replace");

                      newDoc.write(response);
                      newDoc.close();
                    }
                  });

                });

            var card=$("<div>").attr("class","col-lg-4")
                .append(
                    $("<div>").attr("class","our-team-main")
                    .append(
                        $("<div>").attr("class","team-front")
                        .append(instr,name,level),

                        $("<div>").attr("class","team-back")
                        .append(email,"<br>","<br>",$("<div>").attr({"align":"center","type"
                            :"_room"})
                                    .append(room)

                            )

                        )

                    );


            $("#bacheca").append(card);

          });

          if($("p[type='_room']").length == 0){
```

```
            alert("Professori non disponibili!");
          }


        }
    );




});
```

Codice A.4: "Student.js"

A.1.4 WebRTC page

```
<!DOCTYPE html>
<html>
<head>
  <title>WebRTC application</title>
  <style>
    .button {
    background-color: #DAA520;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 10px 100px;
    cursor: pointer;
    }
  </style>

</head>

<body background="img/background.jpg">

  <h1 id="pageTitle" class="text-center text-success"></h1>

  <div id='mainDiv'>
    <table border="0" width="100%" style="color: #483d8b">
      <tr>
        <td></td>
        <!-- td align="center" colspan="3" style="color: #00ff00">Video Remoto</td -->
        <td align="center" style="color: #483d8b">
          <strong>Video Remoto</strong>
        </td>
        <td></td>
      </tr>
      <tr>
        <td align="center" colspan="3" >
  <!--        <video width="450px" height="450" id="remoteVideo" autoplay> </video> -->
          <p align="center"><video width="27%" id="remoteVideo" autoplay> </video><p>
        </td>
      </tr>
```

```

      <tr>
        <td align="center">
          <textarea rows="7" cols="30" id="dataChannelSend" disabled placeholder="This will be enabled once the data channel is up..."></textarea>
        </td>
        <td>

          <p align="center"> <video width="25%" id="localVideo"  autoplay></video> </p>
        </td>
        <td align="center">
          <textarea rows="7" cols="30" id="dataChannelReceive" disabled placeholder="This will be enabled once the data channel is up..."></textarea>
        </td>
      </tr>

      <tr>
        <td align="center">
              <button id="sendButton" class="btn btn-success" disabled>Invia</button>
        </td>
      <td align="center" style="color: #483d8b">
          <strong>Video Locale</strong>
      </td>
      </tr>
    </table>

  </div>
  <div id="roomID" hidden></div>


    <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

  <script src='socket.io/socket.io.js'></script>
  <script src='js/lib/adapter.js'></script>
  <script src='js/NodeClient.js'></script>
  <br/><br/>
  <center><a href="index.html" class="btn  btn-danger">Termina</a></center>

</body>
</html>
```

Codice A.5: "WebRTCPage.html"

A.2 Servlet

A.2.1 Professore

```
package it.musicLearning;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

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
       ServletException, IOException {
64     // TODO Auto-generated method stub
65     doPost(request, response);
66     // DEBUG
```

```
    System.out.println(request.getQueryString());
    System.out.println("DEBUG: [Professor_Servlet: doGet] -> doPost");


  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {
    // TODO Auto-generated method stub

    System.out.println("DEBUG: [ProfList_Servlet: doPost]");


    String ProfName = request.getParameter("ProfName");
    String ProfEmail = request.getParameter("ProfEmail");
    String ProfInstrument = request.getParameter("Instrument");
    String ProfLevel = request.getParameter("Level");




    if((ProfName == null || ProfName.equals("")) || (ProfEmail == null || ProfEmail.equals("
        "))
      || (ProfInstrument == null || ProfInstrument.equals("")) || (ProfLevel == null ||
          ProfLevel.equals("")) ) {
        System.out.println("DEBUG: [Professor_Servlet: doPost] -> No Content for Professor "
            );
        response.sendError(HttpServletResponse.SC_NO_CONTENT);
        return;
    }
    else{
      incrementID();
      RoomDB room= new RoomDB(lastID, ProfName, ProfEmail, ProfInstrument, ProfLevel,1);
      room.SaveRoom();
      System.out.println("DEBUG: [Professor_Servlet: doPost] -> Room saved into DB!");



      File input = new File("/var/lib/tomcat8/webapps/MusicLearning/WebRtcPage.html");

      Document html = (Document) Jsoup.parse(input, "UTF-8");

      Element tmp= html.getElementById("roomID");

      tmp.text(Integer.toString(lastID));

      tmp= html.getElementById("pageTitle");

      tmp.text("VideoLezione RealTime di "+ ProfInstrument + ". Difficolta': "+ ProfLevel);

      response.setContentType("text/html");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().append(html.toString());

    }

  }
```

```

}
```

Codice A.6: "Servlet Professore"

A.2.2 Studente

```
package it.musicLearning;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;


/**
 * Servlet implementation class Student
 */
@WebServlet("/Student")
public class Student extends HttpServlet {
  private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Student() {
        super();
        // TODO Auto-generated constructor stub
    }

  /**
   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
   */
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {
    // TODO Auto-generated method stub
    doPost(request, response);
    // DEBUG
    //System.out.println(request.getQueryString());
    System.out.println("DEBUG: [Student_Servlet: doGet] -> doPost");
  }

  /**
   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
   */
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {
    // TODO Auto-generated method stub
    System.out.println("DEBUG: [Student_Servlet: doPost]");
```

```
    String comand= request.getParameter("comand"); //true or false

    String StudentName = request.getParameter("StudentName");
    String StudentEmail = request.getParameter("StudentEmail");


    if(comand == null || comand.equals("")){

      if((StudentName == null || StudentName.equals("")) || (StudentEmail == null ||
          StudentEmail.equals"")))  {
        System.out.println("DEBUG: [Student_Servlet: doPost] -> No Content for Student ");
        response.sendError(HttpServletResponse.SC_NO_CONTENT);
        return;
      }

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

      Element r= html.getElementById("pageTitle");

      r.text("Ciao "+ StudentName +"!  Scegli un Professore");

      response.setContentType("text/html");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().append(html.toString());
    }
    else{
      switch (comand) {
      case "getProfList":
        ArrayList<RoomDB> rooms= RoomDB.RetrieveAvailableRoom();


        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().append("<response>");


        if( rooms != null){

          for(RoomDB iter: rooms){

            response.getWriter().append("<prof>");

              response.getWriter().append("<name>");
                response.getWriter().append(iter.getProfName());
              response.getWriter().append("</name>");

              response.getWriter().append("<level>");
                response.getWriter().append(iter.getLevel());
              response.getWriter().append("</level>");

              response.getWriter().append("<email>");
                response.getWriter().append(iter.getProfEmail());
              response.getWriter().append("</email>");

              response.getWriter().append("<linkImage>");
                response.getWriter().append("img/"+ iter.getMusicalInstrument() +".png");
              response.getWriter().append("</linkImage>");

```

```
            response.getWriter().append("<room>");
              response.getWriter().append(Integer.toString(iter.getID()));
            response.getWriter().append("</room>");

          response.getWriter().append("</prof>");
        }

      }

      response.getWriter().append("</response>");

      break;

    case "call":

      RoomDB temp= RoomDB.RetrieveRoomByID(Integer.parseInt(request.getParameter("room")))
          ;
      if(temp.getNclient() == 1){
        temp.setNclient(2);
        temp.UpdateRoom();


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

        Element tmp= html.getElementById("roomID");

        tmp.text(request.getParameter("room"));

        tmp= html.getElementById("pageTitle");

        tmp.text("VideoLezione RealTime di "+ temp.getMusicalInstrument() + ". Difficolta
            ': "+ temp.getLevel());

        response.setContentType("text/html");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().append(html.toString());


      }
      else{

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

        Element r= html.getElementById("pageTitle");

        r.text("Sessione avviata da un altro studente! Scegli un Professore");

        response.setContentType("text/html");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().append(html.toString());


      }



      break;
```

```
      default:
        System.out.println("ERROR: [Student_Servlet: doPost] -> Get Professor List");
        response.sendError(HttpServletResponse.SC_NO_CONTENT);
        break;
      }



    }
  }

}
```

Codice A.7: "Servlet Studente"

A.3 Node Client

```
'use strict';

// When I close the window, hangup() function starts.
window.onbeforeunload = function(e){
hangup();
}

// Send channel and receive channel
var sendChannel, receiveChannel;
// Variables associated with HTML5 elements
var sendButton = document.getElementById("sendButton");
var sendTextarea = document.getElementById("dataChannelSend");
var receiveTextarea = document.getElementById("dataChannelReceive");
var localVideo = document.querySelector('#localVideo');
var remoteVideo = document.querySelector('#remoteVideo');
// Handler associated with 'Send' button
sendButton.onclick = sendData;
// Flags
var isChannelReady;
var isInitiator;
var isStarted;
// WebRTC local and remote stream
var localStream;
var remoteStream;
// Peer Connection
var pc;
// Peer Connection ICE protocol configuration and constraints
var pc_config = {'iceServers': [{'urls': 'stun:stun.l.google.com:19302'}]};
var pc_constraints = null;
var sdpConstraints = {'mandatory': {'OfferToReceiveAudio':true, 'OfferToReceiveVideo':true
    }};

var room = document.getElementById("roomID").textContent;
// Connect to signaling server
var socket = io.connect(window.location.hostname+':8181');
// Create or join' message to signaling server
if (room !== '') {
console.log('Create or join room', room);
socket.emit('create or join', room);
}
//GetUserMedia()
var constraints = {video: true, audio: true};
```

```
navigator.mediaDevices.getUserMedia(constraints)
.then(handleSuccess)
.catch( function(error) {
console.log("navigator.mediaDevices.getUserMedia error: ", error);
});
console.log('Getting user media with constraints', constraints);
// GetUserMedia() handler succes
function handleSuccess(stream) {
  localStream = stream;
  localVideo.srcObject=stream;
  console.log('Adding local stream.');
  sendMessage('got user media');
  if (isInitiator) {
    checkAndStart();
    console.log('Initiator');
}
}
//-----------------------------------------------------------------------------------------
// 1. SERVER ---> CLIENT
// Handle 'created' message coming back from server to initiator peer
socket.on('created', function (room){
  console.log('Created room ' + room);
  isInitiator = true;
});

// Handle 'full' message coming back from server
socket.on('full', function (room){
  console.log('Room ' + room + ' is full');
});
// Handle 'join' message coming back from server to initiator peer
socket.on('join', function (room){
  console.log('Another peer made a request to join room ' + room);
  console.log('This peer is the initiator of room ' + room + '!');
  isChannelReady = true;
});

// Handle 'joined' message coming back from server to joiner peer
socket.on('joined', function (room){
  console.log('This peer has joined room ' + room);
  isChannelReady = true;
});

// Handle 'log' message coming back from server to "console peer"
socket.on('log', function (array){
  console.log.apply(console, array);
});

// Receive message from the other peer via the signaling server
socket.on('message', function (message){
  console.log('Received message:', message);
  if (message === 'got user media') {
    checkAndStart();
    }
  else if (message.type === 'offer') {
    if (!isInitiator && !isStarted) {
      checkAndStart();
    }
    pc.setRemoteDescription(new RTCSessionDescription(message));
    doAnswer();
  }
```

```
  else if (message.type === 'answer' && isStarted) {
    pc.setRemoteDescription(new RTCSessionDescription(message));
    }
  else if (message.type === 'candidate' && isStarted) {
    var candidate = new RTCIceCandidate({sdpMLineIndex:message.label,
    candidate:message.candidate});
    pc.addIceCandidate(candidate);
    }
  else if (message === 'bye' && isStarted) {
    handleRemoteHangup();
  }
});
//-----------------------------------------------------------------------------------------
// 2. CLIENT ---> SERVER
// Send message to the other peer via the signaling server
function sendMessage(message){
  console.log('Sending message: ', message);
  socket.emit('message', message);
}

// Check and Start
function checkAndStart() {
  if (!isStarted && typeof localStream != 'undefined' && isChannelReady) {
    createPeerConnection();
    pc.addStream(localStream);
    isStarted = true;
    if (isInitiator) {
      doCall();
    }
  }
}

//-----------------------------------------------------------------------------------------
// Peer connection and create data channel
function createPeerConnection() {
  console.log('call Function- createPeerConnection');
  try {
    pc = new RTCPeerConnection(pc_config, pc_constraints);
    pc.onicecandidate = handleIceCandidate;
    console.log('Created RTCPeerConnnection with:\n' +
    ' config: \'' + JSON.stringify(pc_config) + '\';\n' +
    ' constraints: \'' + JSON.stringify(pc_constraints) + '\'.');
    }
   catch (e) {
    console.log('Failed to create PeerConnection, exception: ' + e.message);
    alert('Cannot create RTCPeerConnection object.');
    return;
    }
  pc.onaddstream = handleRemoteStreamAdded;
  pc.onremovestream = handleRemoteStreamRemoved;
  if (isInitiator) {
  try {
    // Create a reliable data channel
    sendChannel = pc.createDataChannel("sendDataChannel",
    {reliable:true});
    console.log('Created send data channel');
    console.log('readystate:' +sendChannel.readyState);
    }
  catch (e) {
    alert('Failed to create data channel. ');
```

```
    }
  sendChannel.onopen = handleSendChannelStateChange;
  sendChannel.onmessage = handleMessage;
  sendChannel.onclose = handleSendChannelStateChange;
  }
   else { // Joiner
    pc.ondatachannel = gotReceiveChannel;
    }
}
// Send data from a peer to the other one
function sendData() {
  var data = sendTextarea.value;
  sendTextarea.value='';
  if(isInitiator) sendChannel.send(data);
  else receiveChannel.send(data);
  console.log('Sent data: ' + data);
}

//-----------------------------------------------------------------------------------------
// Handlers
function gotReceiveChannel(event) {
  console.log('Receive Channel Callback');
  receiveChannel = event.channel;
  receiveChannel.onmessage = handleMessage;
  receiveChannel.onopen = handleReceiveChannelStateChange;
  receiveChannel.onclose = handleReceiveChannelStateChange;
}

function handleMessage(event) {
  console.log('Received message: ' + event.data);
  receiveTextarea.value += event.data + '\n';
}

function handleSendChannelStateChange() {
  var readyState = sendChannel.readyState;
  console.log('Send channel state is: ' + readyState);
  // If channel ready, enable user's input
  if (readyState == "open") {
    dataChannelSend.disabled = false;
    dataChannelSend.focus();
    dataChannelSend.placeholder = "";
    sendButton.disabled = false;
  }
  else {
    dataChannelSend.disabled = true;
    sendButton.disabled = true;
  }
}

function handleReceiveChannelStateChange() {
  var readyState = receiveChannel.readyState;
console.log('Receive channel state is: ' + readyState);

// If channel ready, enable user's input
if (readyState == "open") {
  dataChannelSend.disabled = false;
  dataChannelSend.focus();
  dataChannelSend.placeholder = "";
  sendButton.disabled = false;
  } else {
```

```
  dataChannelSend.disabled = true;
  sendButton.disabled = true;
}
}
// ICE candidates management
function handleIceCandidate(event) {
  console.log('handleIceCandidate event: ', event);
  if (event.candidate) {
    sendMessage({
    type: 'candidate',
    label: event.candidate.sdpMLineIndex,
    id: event.candidate.sdpMid,
    candidate: event.candidate.candidate});
  } else {
    console.log('End of candidates.');
  }
}

// Create Offer
function doCall() {
  console.log('Creating Offer...');
  pc.createOffer(sdpConstraints)
  .then(setLocalAndSendMessage)
  .catch(function(error) {
    console.log('Failed to create signaling message : ' + error.name);
  });
}

// Create Answer
function doAnswer() {
  console.log('Sending answer to peer.');
  pc.createAnswer(sdpConstraints)
  .then(setLocalAndSendMessage)
  .catch(function(error) {
    console.log('Failed to create signaling message : ' + error.name);
  });
}

// Handler success for Offer and Answer
function setLocalAndSendMessage(sessionDescription) {
  pc.setLocalDescription(sessionDescription);
  sendMessage(sessionDescription);
}


// ------------------------------------------------------------------------------------
// Remote stream handlers
function handleRemoteStreamAdded(event) {
  console.log('Remote stream added.');
  remoteVideo.srcObject=event.stream;
  remoteStream = event.stream;
}

function handleRemoteStreamRemoved(event) {
  console.log('Remote stream removed. Event: ', event);
}


//------------------------------------------------------------------------------------
// Clean-up functions
```

```
function hangup() {
  console.log('Hanging up.');
  stop();
  sendMessage('bye');
}

function handleRemoteHangup() {
  console.log('Session terminated.');
  stop();
  isInitiator = false;
}

function stop() {
  isStarted = false;
  if (sendChannel) sendChannel.close();
  if (receiveChannel) receiveChannel.close();
  if (pc) pc.close();
  pc = null;
  sendButton.disabled=true;
}
```

Codice A.8: "NodeClient.js"

A.4 Node Server

```
var static = require('node-static');
var http = require('http');
// Create a node-static server instance
var file = new(static.Server)();
// We use the http module createServer function and rely on our instance of node-static to
    serve the files
var app = http.createServer(function (req, res) {
  file.serve(req, res);
}).listen(8181);
console.log('Listening on ' + app.address().port);
// Use socket.io JavaScript library for real-time web applications
var io = require('socket.io').listen(app);
// Connection
io.sockets.on('connection', function (socket){

  // Handle 'message' messages
  socket.on('message', function (message) {
    log('S --> got message: ', message);
    // channel-only broadcast...
    socket.broadcast.emit('message', message);
  });

  // Handle 'create or join' messages
  socket.on('create or join', function (room) {
    var numClients = io.sockets.adapter.rooms[room]!=undefined ? Object.keys(io.sockets.
        adapter.rooms[room]).length:0;
    log('S --> Room ' + room + ' has ' + numClients + ' client(s)');
    log('S --> Request to create or join room', room);
    console.log('Stanza: = ' + room + " Clienti:" + numClients);
    // First client joining...
    if (numClients == 0){
      socket.join(room);
      socket.emit('created', room);
    }
```

```
    else if (numClients == 1) {
      // Second client joining...
      io.sockets.in(room).emit('join', room);
      socket.join(room);
      socket.emit('joined', room);
    }
    else { // max two clients
      socket.emit('full', room);
    }
  });

  function log(){
    var array = [">>> "];
    for (var i = 0; i < arguments.length; i++) {
      array.push(arguments[i]);
    }
    socket.emit('log', array);
  }
});
```

Codice A.9: "NodeServer.js"

A.5 Hibernate - MySQL

```
package it.musicLearning;

import java.util.ArrayList;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import org.hibernate.Query;
import org.hibernate.Session;

@Entity
public class RoomDB {

  @Id
//  @GeneratedValue
  @Column(name="id",unique=true,nullable=false)
  private int ID;


  @Column(name="prof_name",unique=false,nullable=false)
  private String ProfName;

  @Column(name="prof_email",unique=false,nullable=false)
  private String ProfEmail;

  @Column(name="musical_instrument",unique=false,nullable=false)
  private String MusicalInstrument;

  @Column(name="level",unique=false,nullable=false)
  private String Level;

  @Column(name="n_client",unique=false,nullable=false)
  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     }
197         catch (RuntimeException e) {
198           System.out.println("\n\nEccezione: Query1 Room!\n\n");
199         session.getTransaction().rollback();
200         throw e;
201     }
202     finally {
203         session.close();
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    }
267        catch (RuntimeException e) {
268          System.out.println("\n\nEccezione: Query3 Room!\n\n" + e);
269        session.getTransaction().rollback();
270        throw e;
271    }
272    finally {
273        session.close();
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     }
299         catch (RuntimeException e) {
300           System.out.println("\n\nEccezione: Query4 Room!\n\n");
301         session.getTransaction().rollback();
302         throw e;
303     }
304     finally {
305         session.close();
306     }
307   }
308 }
```

Codice A.10: "RoomDB"