## INSCRIBIRSE A UNA SALA

### RoomJava:

package entities;

import entities.User;

import java.util.ArrayList;

import java.text.SimpleDateFormat;

import java.sql.Date;

public class Room {

private int id;

private String name;

private String code;

private int amountParticipants;

private int maxAmountParticipants;

private Date initDate;

private Date endDate;

private boolean deleted = false;

private User admin;

private ArrayList<User> users;

private ArrayList<Quiz> quizzes;

SimpleDateFormat outputFormat = new SimpleDateFormat("dd/MM/yyyy");

public User getAdmin() {

return admin;

}

public void setAdmin(User admin) {

this.admin = admin;

}

public ArrayList<Quiz> getQuizzes() {

return quizzes;

}

public void setQuizzes(ArrayList<Quiz> quizzes) {

this.quizzes = quizzes;

}

public ArrayList<User> getUsers() {

return users;

}

public void setUsers(ArrayList<User> users) {

this.users = users;

}

public boolean isDeleted() {

return deleted;

}

public void setDeleted(boolean deleted) {

this.deleted = deleted;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public int getAmountParticipants() {

return amountParticipants;

}

public void setAmountParticipants(int amountParticipants) {

this.amountParticipants = amountParticipants;

}

public int getMaxAmountParticipants() {

return maxAmountParticipants;

}

public void setMaxAmountParticipants(int maxAmountParticipants) {

this.maxAmountParticipants = maxAmountParticipants;

}

public Date getInitDate() {

return initDate;

}

public String getInitDateAsString() {

if (initDate != null) {

return outputFormat.format(initDate);

}

return "";

}

public void setInitDate(Date initDate) {

this.initDate = initDate;

}

public Date getEndDate() {

return endDate;

}

public String getEndDateAsString() {

if (endDate != null) {

return outputFormat.format(endDate);

}

return "";

}

public void setEndDate(Date endDate) {

this.endDate = endDate;

}

public void addUser(User u) {

this.users.add(u);

this.amountParticipants=+1;

}

public Room() {}

public Room(int id, String name, String code, int amountParticipants, int maxAmountParticipants, Date initDate, Date endDate, boolean deleted) {

super();

this.name = name;

this.id = id;

this.code = code;

this.amountParticipants = amountParticipants;

this.maxAmountParticipants = maxAmountParticipants;

this.initDate = initDate;

this.endDate = endDate;

this.deleted = deleted;

}

public boolean isValid() {

return endDate.after( new Date(System.currentTimeMillis() ));

}

}

### IndexJSP(fragmento):

<div>

<article>

<strong>Únete a una sala</strong>

<footer>

<form action="room" method="post">

<fieldset role="group">

<input type="text" maxlength="36" name="code" placeholder="Código" aria-label="Text"/>

<button type="submit">Ingresar</button>

</fieldset>

</form>

</footer>

</article>

</div>

### RoomServlet (Post method):

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

RoomLogic roomLogic = new RoomLogic();

User usr = (User) request.getSession().getAttribute("user");

String roomcode = (String) request.getParameter("code");

Room room = new Room();

room.setCode(roomcode);

if (usr == null) {

request.setAttribute("headTitle", "Acceso denegado");

request.setAttribute("bodyTitle", "Debe iniciar sesión para entrar a una sala");

request.setAttribute("buttonAction", "login.html");

request.setAttribute("buttonMessage", "Iniciar sesión");

request.getRequestDispatcher("WEB-INF/message.jsp").forward(request, response);

return;

}

if (roomcode == "") {

response.sendRedirect("index.jsp");

return;

}

Room foundRoom = roomLogic.getRoomByCode(room);

if (foundRoom == null) {

request.setAttribute("headTitle", "Sala no encontrada");

request.setAttribute("bodyTitle", "La sala no existe");

request.setAttribute("buttonAction", "index.jsp");

request.setAttribute("buttonMessage", "Aceptar");

request.getRequestDispatcher("WEB-INF/message.jsp").forward(request, response);

return;

}

if (foundRoom.isDeleted()) {

request.setAttribute("headTitle", "Sala eliminada");

request.setAttribute("bodyTitle", "La sala a la que intenta ingresar fue eliminada");

request.setAttribute("buttonAction", "index.jsp");

request.setAttribute("buttonMessage", "Aceptar");

request.getRequestDispatcher("WEB-INF/message.jsp").forward(request, response);

return;

}

boolean possible = roomLogic.joinPossible(usr, foundRoom);

if (!possible) {

request.setAttribute("headTitle", "Sala encontrada");

request.setAttribute("bodyTitle", "No es posible unirlo a la sala");

request.setAttribute("buttonAction", "index.jsp");

request.setAttribute("buttonMessage", "Aceptar");

request.getRequestDispatcher("WEB-INF/message.jsp").forward(request, response);

return;

}

roomLogic.addUserToRoom(usr, foundRoom);

foundRoom.setAdmin(roomLogic.getRoomAdmin(foundRoom));

foundRoom.setQuizzes(roomLogic.getRoomQuizzes(foundRoom));

List<RankedUser> rankedUsers = roomLogic.getRanking(foundRoom);

request.setAttribute("rankedUsers", rankedUsers);

request.setAttribute("room", foundRoom);

request.getRequestDispatcher("WEB-INF/room.jsp").forward(request, response);

}

### RoomLogic (métodos utilizados):

public Room getRoomByCode(Room r) {

Room room = roomDB.getByCode(r);

if (room != null) {

return room;

} else {

return null;

}

}

public boolean joinPossible(User u, Room r) {

ArrayList<Room> roomsFromDb = userRoomDB.getRoomsByUserId(u);

for (Room aRoom: roomsFromDb) {

if (aRoom.getId() == r.getId()) {

return false;

}

}

if (r.getAmountParticipants() >= r.getMaxAmountParticipants()) {

return false;

}

if (!r.isValid()) {

return false;

}

return true;

}

public void addUserToRoom(User u, Room r) {

userRoomDB.addUserToRoom(u, r);;

}

public User getRoomAdmin(Room r) {

User admin = userRoomDB.getRoomAdmin(r);

return admin;

}

public ArrayList<Quiz> getRoomQuizzes(Room r) {

ArrayList<Quiz> quizzes = roomQuizDB.getQuizzesByRoomId(r);

return quizzes;

}

public List<RankedUser> getRanking(Room room) {

return RoomDB.getUsersbyRanking(room);

}

### RoomDB(métodos utilizados):

public Room getByCode(Room r) {

String sqlSelect = "select r.\*, (SELECT COUNT(\*) FROM users\_rooms ur WHERE ur.room\_id = r.id) AS amount\_participants from rooms r where r.code = ?";

Connect connect = new Connect();

Connection con = connect.getConnection();

Room room = null;

if (con != null) {

try {

PreparedStatement stm = con.prepareStatement(sqlSelect);

stm.setString(1, r.getCode());

ResultSet rs = stm.executeQuery();

if (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String code = rs.getString("code");

int amountParticipants = rs.getInt("amount\_participants");

int maxAmountParticipants = rs.getInt("max\_amount\_participants");

Date initDate = rs.getDate("init\_date");

Date endDate = rs.getDate("end\_date");

boolean deleted = rs.getBoolean("deleted");

room = new Room(id, name, code, amountParticipants, maxAmountParticipants, initDate, endDate, deleted);

}

con.close();

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

return room;

}

public static ArrayList<RankedUser> getUsersbyRanking(Room r) {

ArrayList<RankedUser> rankedUsers = new ArrayList<>();

String sqlSelect = "SELECT u.id, u.name, u.surname, SUM(a.amountRight) AS totalPuntos "

+ "FROM users u "

+ "INNER JOIN answers a ON a.user\_id = u.id "

+ "INNER JOIN quizzes q ON q.id = a.quizz\_id "

+ "WHERE q.room\_id = ? "

+ "GROUP BY u.id "

+ "ORDER BY totalPuntos DESC";

Connect connect = new Connect();

Connection con = connect.getConnection();

if (con != null) {

try {

PreparedStatement stm = con.prepareStatement(sqlSelect);

stm.setInt(1, r.getId());

ResultSet rs = stm.executeQuery();

while (rs.next()) {

int userId = rs.getInt("id");

String name = rs.getString("name");

String surname = rs.getString("surname");

int totalScore = rs.getInt("totalPuntos");

User user = new User(userId, name, surname);

RankedUser rankedUser = new RankedUser(user, totalScore);

rankedUsers.add(rankedUser);

}

rs.close();

stm.close();

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

con.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

return rankedUsers;

}

### UserRoomDB (métodos utilizados):

public User getRoomAdmin(Room r) {

String sqlSelect = "SELECT users.\* FROM rooms INNER JOIN users ON users.id = rooms.admin WHERE rooms.id = ?";

Connect connect = new Connect();

Connection con = connect.getConnection();

User user = null;

if (con != null) {

try {

PreparedStatement stm = con.prepareStatement(sqlSelect);

stm.setInt(1, r.getId());

ResultSet rs = stm.executeQuery();

if (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String surname = rs.getString("surname");

String email = rs.getString("email");

String password = rs.getString("password");

user = new User(id, name, surname, email, password);

}

con.close();

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

return user;

}

public ArrayList<Room> getRoomsByUserId(User u) {

String sqlSelect = "SELECT r.\*, (SELECT COUNT(\*) FROM users\_rooms ur WHERE ur.room\_id = r.id) AS amount\_participants FROM rooms r INNER JOIN users\_rooms ur ON ur.room\_id = r.id WHERE ur.user\_id = ?";

Connect connect = new Connect();

Connection con = connect.getConnection();

ArrayList<Room> roomsFromUser = new ArrayList<>();

if (con != null) {

try {

PreparedStatement stm = con.prepareStatement(sqlSelect);

stm.setInt(1, u.getId());

ResultSet rs = stm.executeQuery();

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String code = rs.getString("code");

int amountParticipants = rs.getInt("amount\_participants");

int maxAmountParticipants = rs.getInt("max\_amount\_participants");

Date initDate = rs.getDate("init\_date");

Date endDate = rs.getDate("end\_date");

boolean deleted = rs.getBoolean("deleted");

roomsFromUser.add(new Room(id, name, code, amountParticipants, maxAmountParticipants, initDate, endDate, deleted));

}

con.close();

} catch(SQLException e) {

e.printStackTrace();

}

}

return roomsFromUser;

}

public void addUserToRoom(User u, Room r){

String sqlSelect = "INSERT INTO users\_rooms (user\_id, room\_id) VALUES (?, ?)";

Connect connect = new Connect();

Connection con = connect.getConnection();

if (con != null) {

try {

PreparedStatement stm = con.prepareStatement(sqlSelect);

stm.setInt(1, u.getId());

stm.setInt(2, r.getId());

stm.executeUpdate();

con.close();

} catch(SQLException e){

e.printStackTrace();

}

}

}

### RoomQuizDB(métodos utilizados):

public ArrayList<Quiz> getQuizzesByRoomId(Room r){

String sqlSelect = "select quizzes.id, quizzes.name, quizzes.max\_duration, quizzes.topic\_id, topics.name topic\_name, quizzes.deleted from rooms inner join quizzes on quizzes.room\_id = rooms.id inner join topics on topics.id = quizzes.topic\_id where rooms.id = ?";

Connect connect = new Connect();

Connection con = connect.getConnection();

ArrayList<Quiz> quizzesFromRoom = new ArrayList<Quiz>();

if (con != null) {

try {

PreparedStatement stm = con.prepareStatement(sqlSelect);

stm.setInt(1, r.getId());

ResultSet rs = stm.executeQuery();

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

Time maxDuration = rs.getTime("max\_duration");

int topicId = rs.getInt("topic\_id");

String topicName = rs.getString("topic\_name");

boolean deleted = rs.getBoolean("deleted");

quizzesFromRoom.add(new Quiz(id, name, maxDuration, new Topic(topicId, topicName), deleted));

}

con.close();

} catch(SQLException e) {

e.printStackTrace();

}

}

return quizzesFromRoom;

}