



TRAVEL MILES

COMMUNITY VIAGGI

Progetto Web and Real Time Communication Systems

Applicazione Web per community e prenotazione viaggi

a.a. 2024/2025

Candidati:

Mario Migliuolo M63001617

Francesco Riccio M63001646



TRAVEL MILES

COMMUNITY VIAGGI

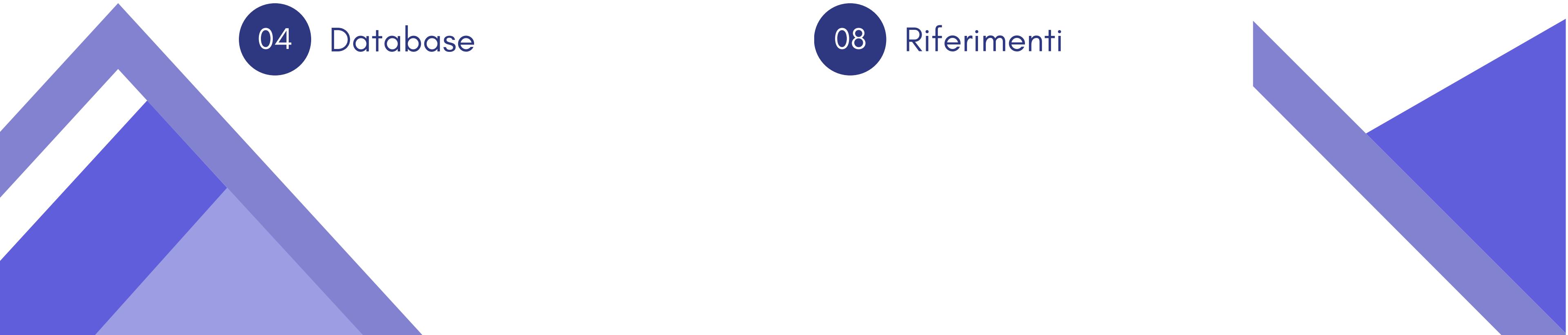
Hello!

Non siamo una semplice agenzia di viaggi: siamo una community appassionata di avventure!

Qui puoi scoprire, condividere e ispirarti a itinerari e luoghi indimenticabili.

Trasforma i tuoi viaggi in ricordi indelebili, un'esperienza alla volta!

Overview

- 
- 01 Presentazione
 - 02 Obiettivi
 - 03 Framework
 - 04 Database
 - 05 Javascript
 - 06 Real Time Feature
 - 07 Signaling Server
 - 08 Riferimenti



TRAVEL MILES

COMMUNITY VIAGGI

Stanco di perdere ore a cercare tra migliaia di offerte per organizzare il tuo viaggio?

Ora puoi rilassarti: il nostro team di esperti di viaggi ed itinerari si occuperà di tutto!

Scegli la tua destinazione, invia i documenti con un semplice click e lascia a noi il resto. Con noi, viaggiare diventa facile e senza pensieri!



Obiettivi



Viaggia con un click

Parti senza pensieri: scegli la tua destinazione e lascia che noi ci occupiamo del resto!



Arriva dritto al punto

Scegli la tua destinazione e lascia a noi ogni dettaglio: hotel, attività, esperienze. Tu rilassati, noi pensiamo a tutto.



Team di Esperti

Affida il tuo viaggio al nostro team di esperti! Grazie alla chat in tempo reale chiedici tutto ciò che desideri ed invia documenti in un lampo per finalizzare la tua prenotazione!

Framework



Alcune Funzionalità:

- RestController
- Controllo Complessità Password
- Interazione con MySQL
- Operazioni CRUD
- Creazione/Modifica Commenti

```
@RestController
public class Controller {

    @Autowired
    private UtentiRepository utentiRepository;

    @Autowired
    private ConsulentiRepository consulentiRepository;

    @Autowired
    private AccountRepository accountRepository;

    @Autowired
    private CityRepository cityRepository;

    @Autowired
    private CityCommentsRepository cityCommentsRepository;

    private boolean isValidPassword(String password) {

        String specialChars = "!£$%&/()=?^'[]@#*";
        boolean hasSpecialChar = false;

        for (char c : specialChars.toCharArray()) {
            if (password.indexOf(c) >= 0) {
                hasSpecialChar = true;
                break;
            }
        }

        return password.length() > 15 && hasSpecialChar;
    }
}
```

Framework



Alcune Funzionalità:

- RestController
- Controllo Complessità Password
- Interazione con MySQL
- Operazioni CRUD
- Creazione/Modifica Commenti

```
@PostMapping("/login")
public ResponseEntity<String> loginUser(
    @RequestParam("email") String email,
    @RequestParam("password") String password,
    HttpServletResponse response,
    HttpServletRequest request
) throws IOException {

    Utenti utenti = utentiRepository.findByEmail(email);

    if (utenti == null) {
        response.sendRedirect("/problem.html?error=Email non registrata");
        return ResponseEntity.badRequest().body("Email non registrata.");
    }

    String username = utenti.getUsername();

    Account account = accountRepository.findByUsername(username);

    if (account == null) {
        response.sendRedirect("/problem.html?error=Account non trovato.");
        return ResponseEntity.badRequest().body("Account non trovato.");
    }

    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    if (!passwordEncoder.matches(password, account.getPassword())) {
        response.sendRedirect("/problem.html?error=Password errata.");
        return ResponseEntity.badRequest().body("Password errata.");
    }

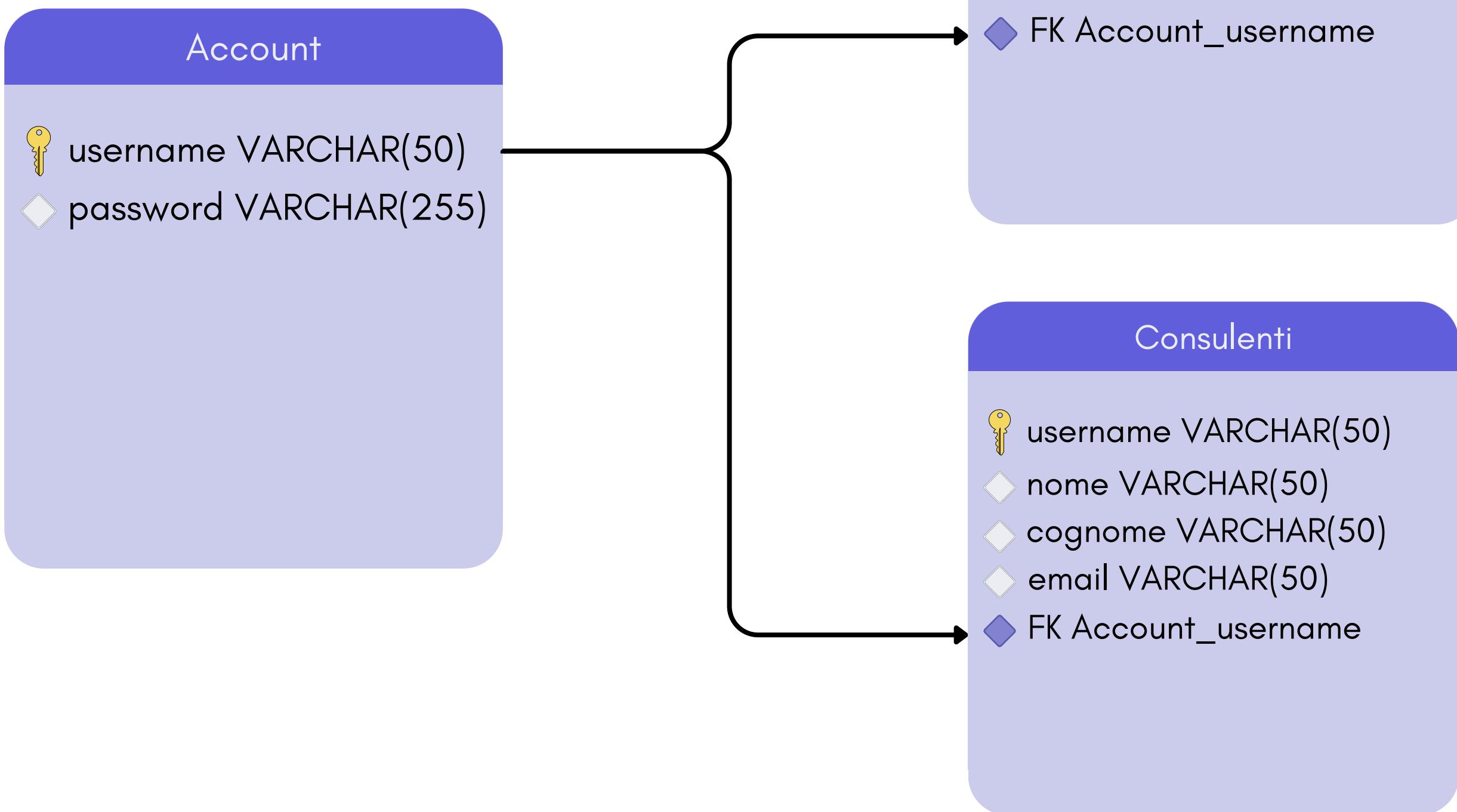
    try {
        request.getSession().setAttribute("username", username);
        response.sendRedirect("/home.html?username=" + username);

        return ResponseEntity.ok("Redirecting to home...");
    } catch (IOException e) {
        e.printStackTrace();
    }

    return ResponseEntity.status(500).body("Errore durante il reindirizzamento.");
}
```

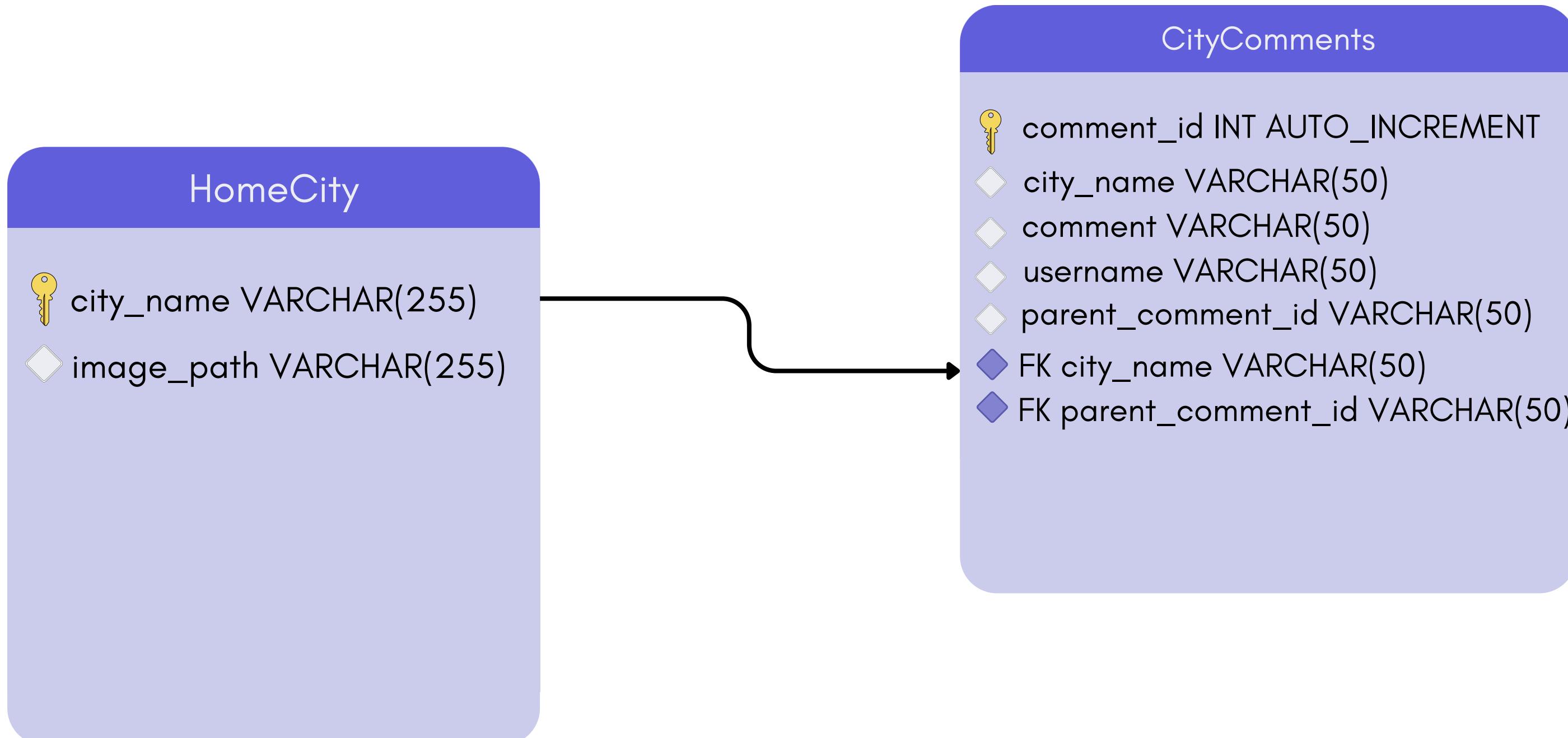
MySQL Database

Account, Utenti, Consulenti



MySQL Database

HomeCity, CityComments



JavaScript



Utilizzo di AJAX

```
// Invio dei dati modificati
document.getElementById('form-modifica').onsubmit = function(event) {
    event.preventDefault();

    const campo = document.getElementById('campo-nome').value;
    const nuovoValore = document.getElementById('nuovo-valore').value;
    const username = document.getElementById('username').value;

    // Richiesta HTTP per aggiornare i dati (AJAX)
    var req = new XMLHttpRequest();
    req.open("POST", "/modifica", true);
    req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    req.onload = function() {
        if (req.status === 200) {
            alert("Aggiornamento riuscito!");
            if (campo !== 'password') {
                document.getElementById(campo).textContent = nuovoValore;
            }
            nascondiForm();
        } else {
            alert("Errore nell'aggiornamento.");
        }
    };
    req.onerror = function() {
        alert("Errore nella richiesta.");
    };
    req.send(`campo=${campo}&valore=${nuovoValore}&username=${username}`);
};
```

JavaScript



Homepage dinamica

```
async function displayCities() {
    const cities = await getCities();
    const cityGrid = document.getElementById('cityGrid');
    cityGrid.innerHTML = '';

    if (cities) {
        cities.forEach(city => {
            const cityItem = createCityElement(city);
            cityGrid.appendChild(cityItem);
        });
    } else {
        const message = document.createElement('div');
        message.textContent = "Nessuna città disponibile.";
        cityGrid.appendChild(message);
    }
}
```

Homepage Dinamica



Benvenuto, Bob!

[Parla con un consulente](#) [Visualizza informazioni personali](#)

Esplora le nostre destinazioni
Scopri le meraviglie del mondo con Travel Miles

Selezione una città



Barcellona



Oslo



Parigi



Roma



Tokyo



Vienna

© 2024 Travel Miles. Tutti i diritti riservati.

Meteo in tempo reale

Con l'utilizzo di dell' API di openweathermap.org

The screenshot shows the OpenWeather API keys management interface. At the top, there's a navigation bar with links like Guide, API, Dashboard, Marketplace, Pricing, Maps, Our Initiatives, Partners, Blog, and For Business. Below the navigation, there are links for New Products, Services, API keys (which is underlined), Billing plans, Payments, Block logs, My orders, My profile, and Ask a question.

A callout box states: "You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them."

Key	Name	Status	Actions	Create key
a175cd432f8d4f5ccb7b954479b4ef9f	Default	Active		<input type="text" value="API key name"/>
c7711539ee71271ffa4e0a3c3fb1e19	TravelMiles	Active		

```
9 @Service
10 public class WeatherService {
11
12     private final String API_KEY = "c7711539ee71271ffa4e0a3c3fb1e19";
13     private final String BASE_URL = "http://api.openweathermap.org/data/2.5/weather?q={city}&appid={apiKey}";
14
15     public String getWeather(String city) {
16         RestTemplate restTemplate = new RestTemplate();
```

Città



Tokyo

[Torna indietro](#)

[Chatta con un nostro consulente!](#)



Meteo

Temperatura: 9.3°C

Previsione: clear sky

Alba e Tramonto

Alba: 22:43:43

Tramonto: 08:29:07

Precipitazioni

Nessuna precipitazione

Community

Marco

Per gli appassionati di videogiochi consiglio di andare ad Akhiabara!

[Modifica](#)[Rispondi](#)

Bob

Grazie, ci sono stato e ho trovato tutto ciò che cercavo!

[Modifica](#)[Rispondi](#)

Alice

Sono da qualche giorno a Tokyo, riesci a consigliarmi qualche negozio in particolare?

[Modifica](#)[Rispondi](#)

Scrivi il tuo commento

[Invia commento](#)

WebRTC Data Channel



WebRTC

 TRAVEL MILES
COMMUNITY VIAGGI

WebRTC Chat & File Sender

Inizia a chattare

Chat:

Scrivi un messaggio e premi invio

Connesso a un consulente.
Bob: Salve, vorrei alcuni consigli per un viaggio
Consulente1: Salve, mi dica tutto!

File Transfer:

Sfoglia... document.pdf

Invia file

 TRAVEL MILES
COMMUNITY VIAGGI

WebRTC Chat & File Sender

Inizia a chattare

Chat:

Scrivi un messaggio e premi invio

Un utente si è connesso.
Bob: Salve, vorrei alcuni consigli per un viaggio
Consulente1: Salve, mi dica tutto!

File Transfer:

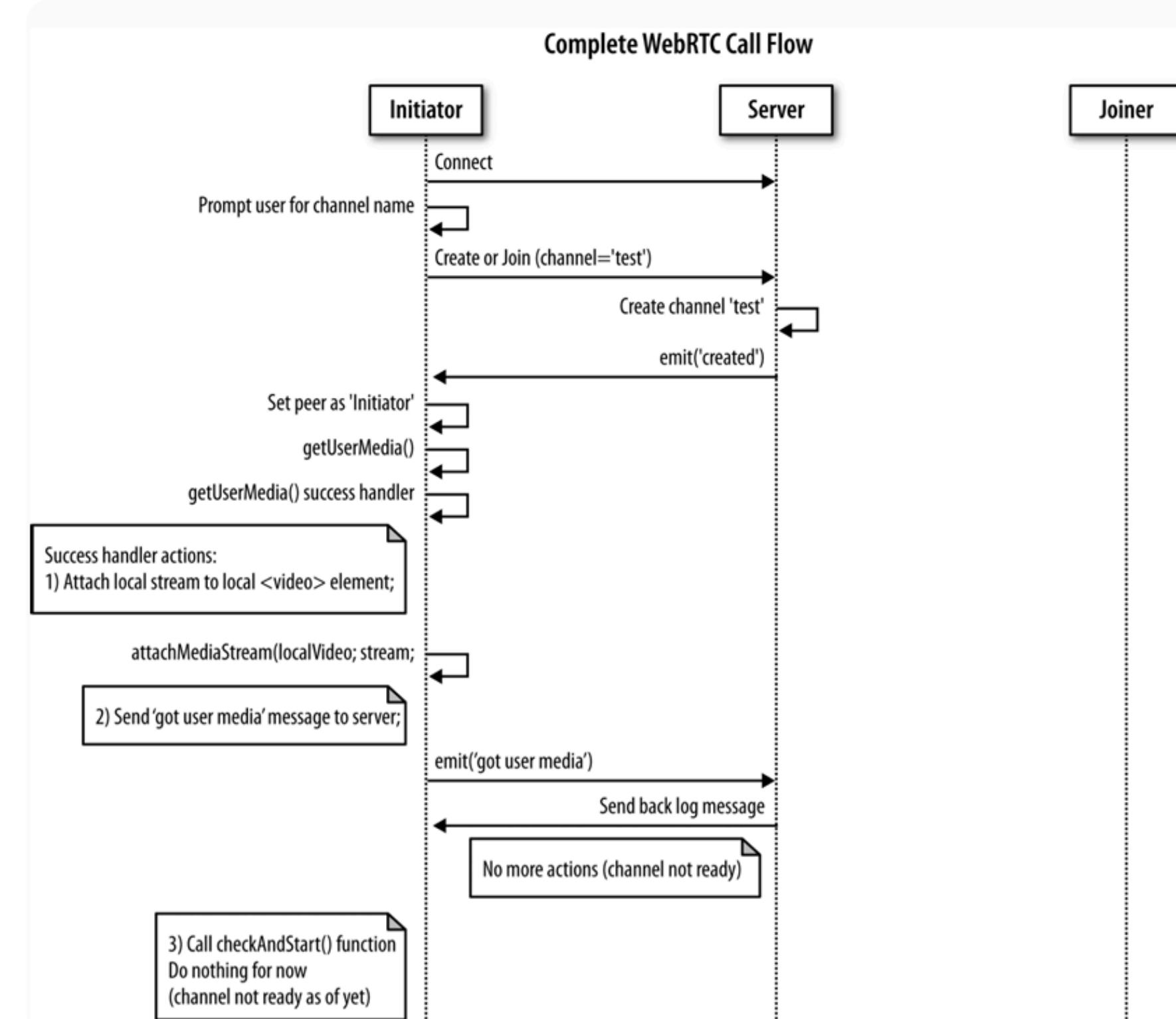
Scegli file Nessun file selezionato

Invia file

Download document.pdf

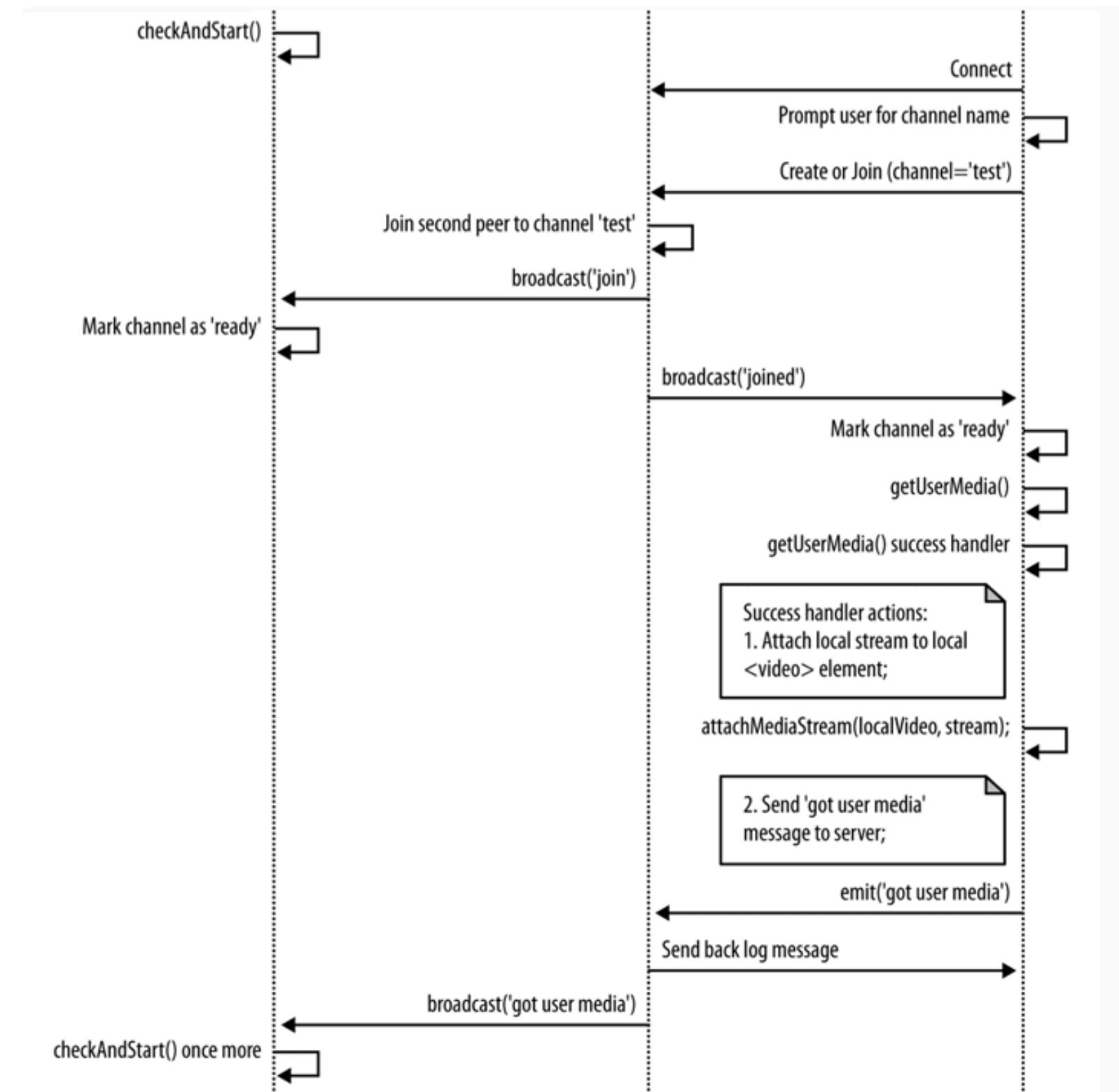
WebRTC General Sequence

Diagram 1/6



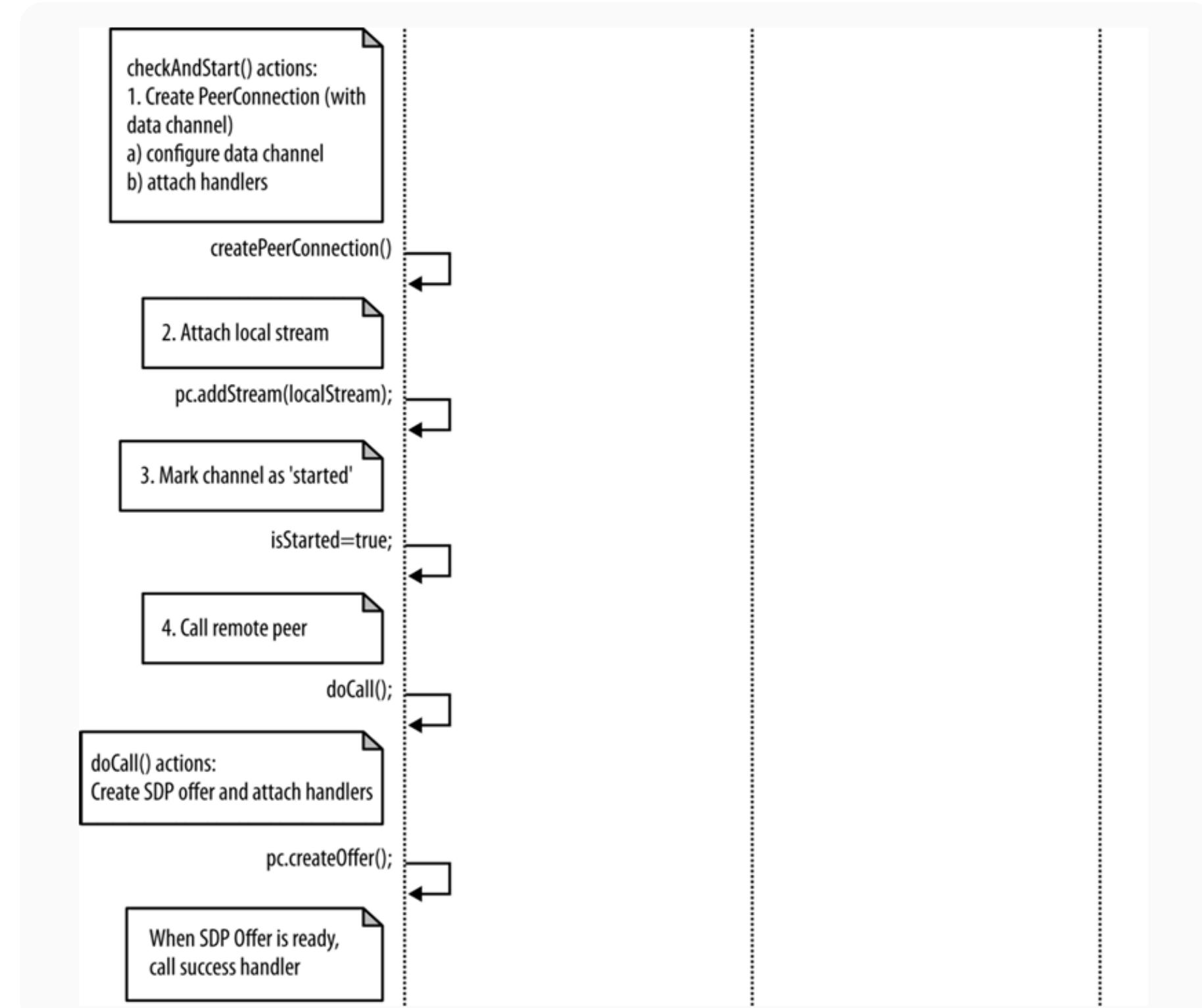
WebRTC General Sequence

Diagram 2/6



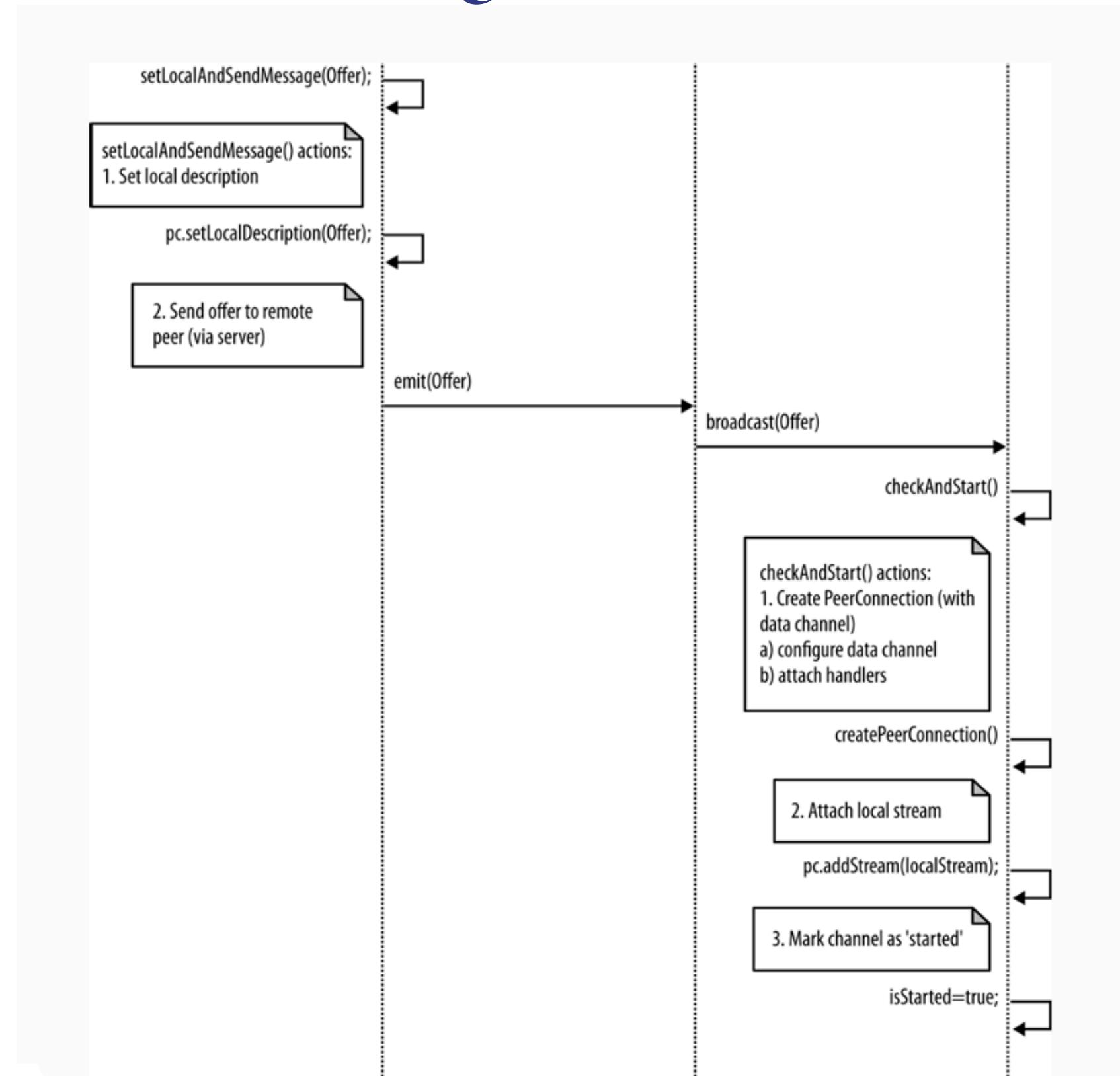
WebRTC General Sequence

Diagram 3/6

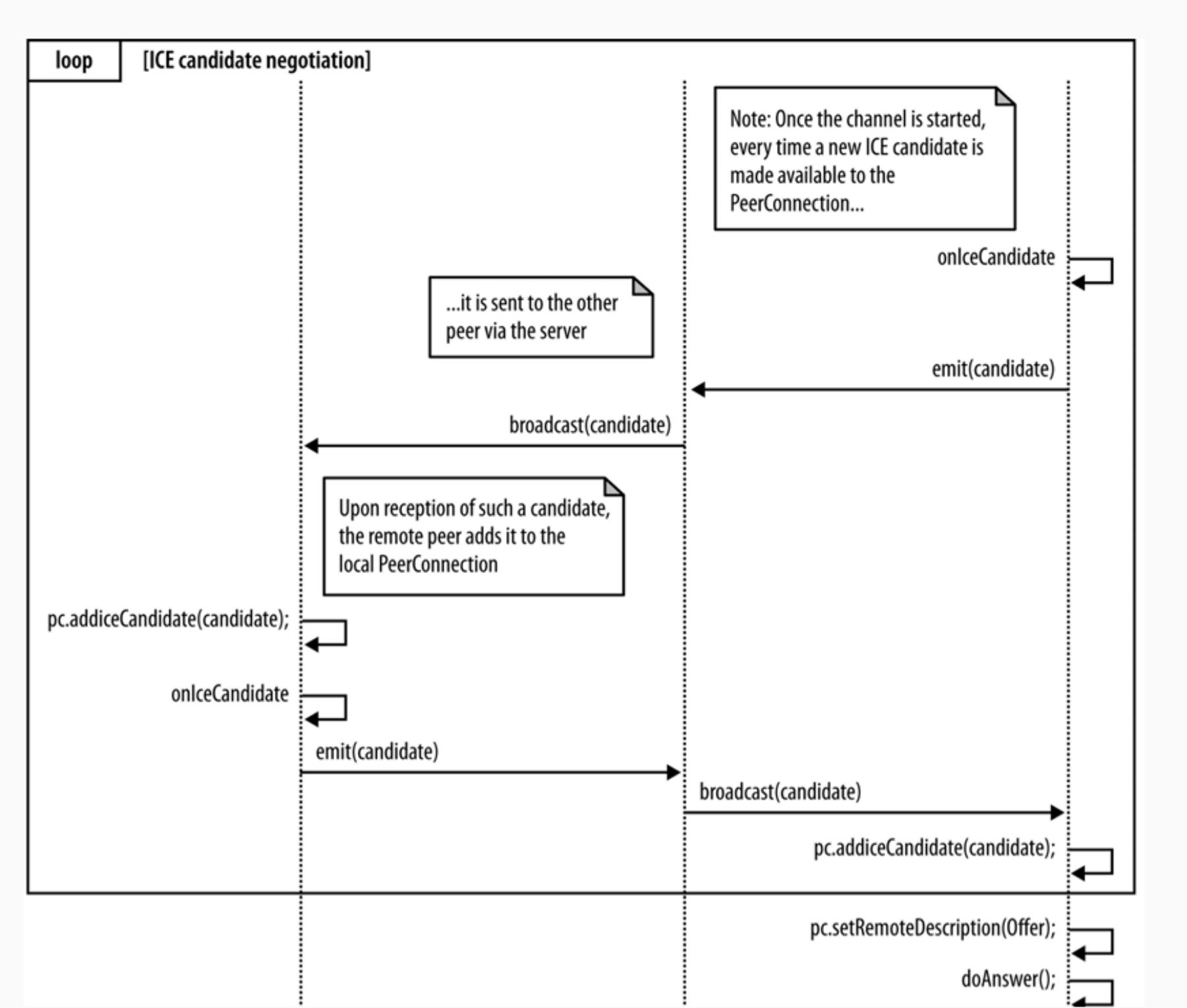


WebRTC General Sequence

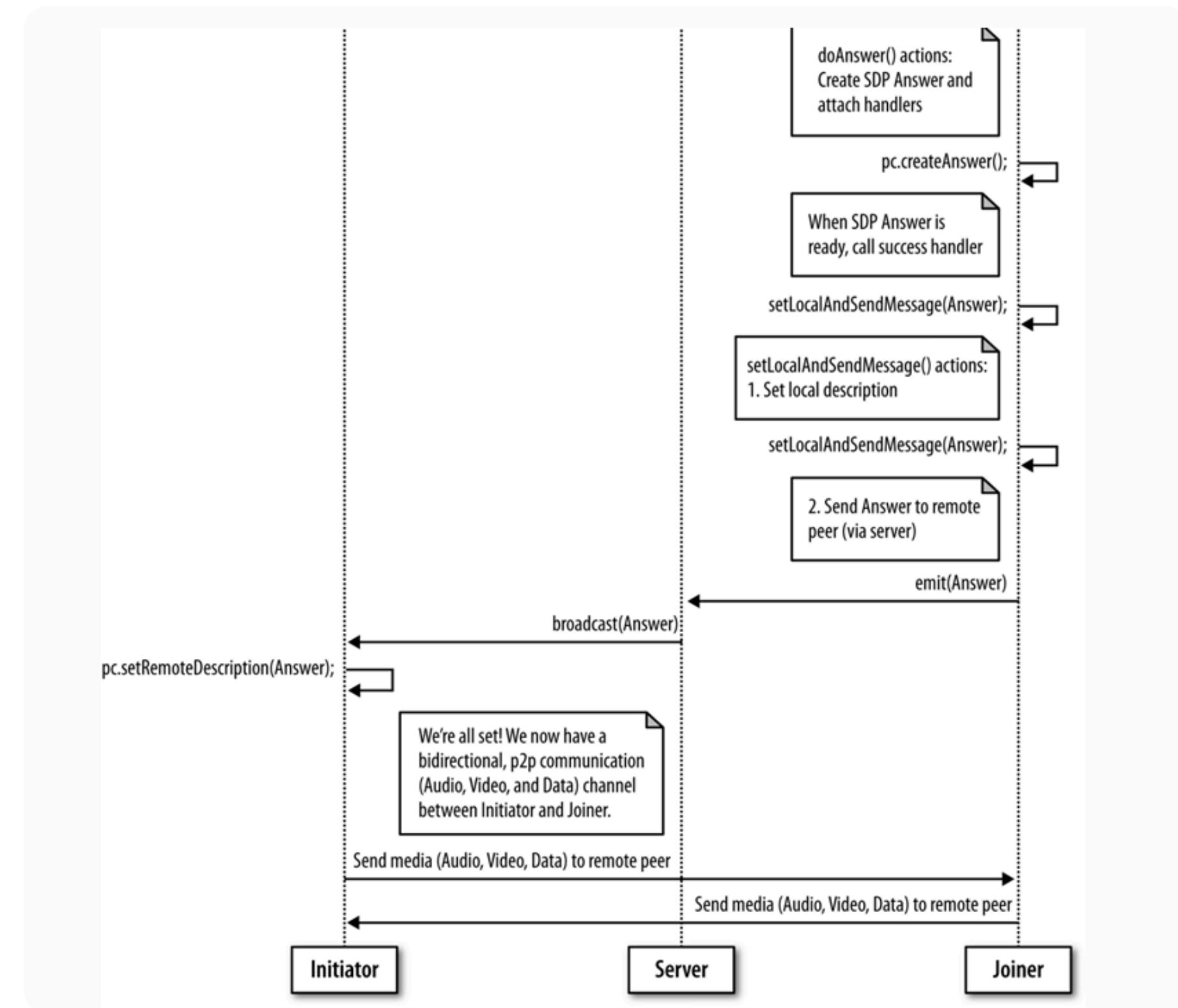
Diagram 4/6



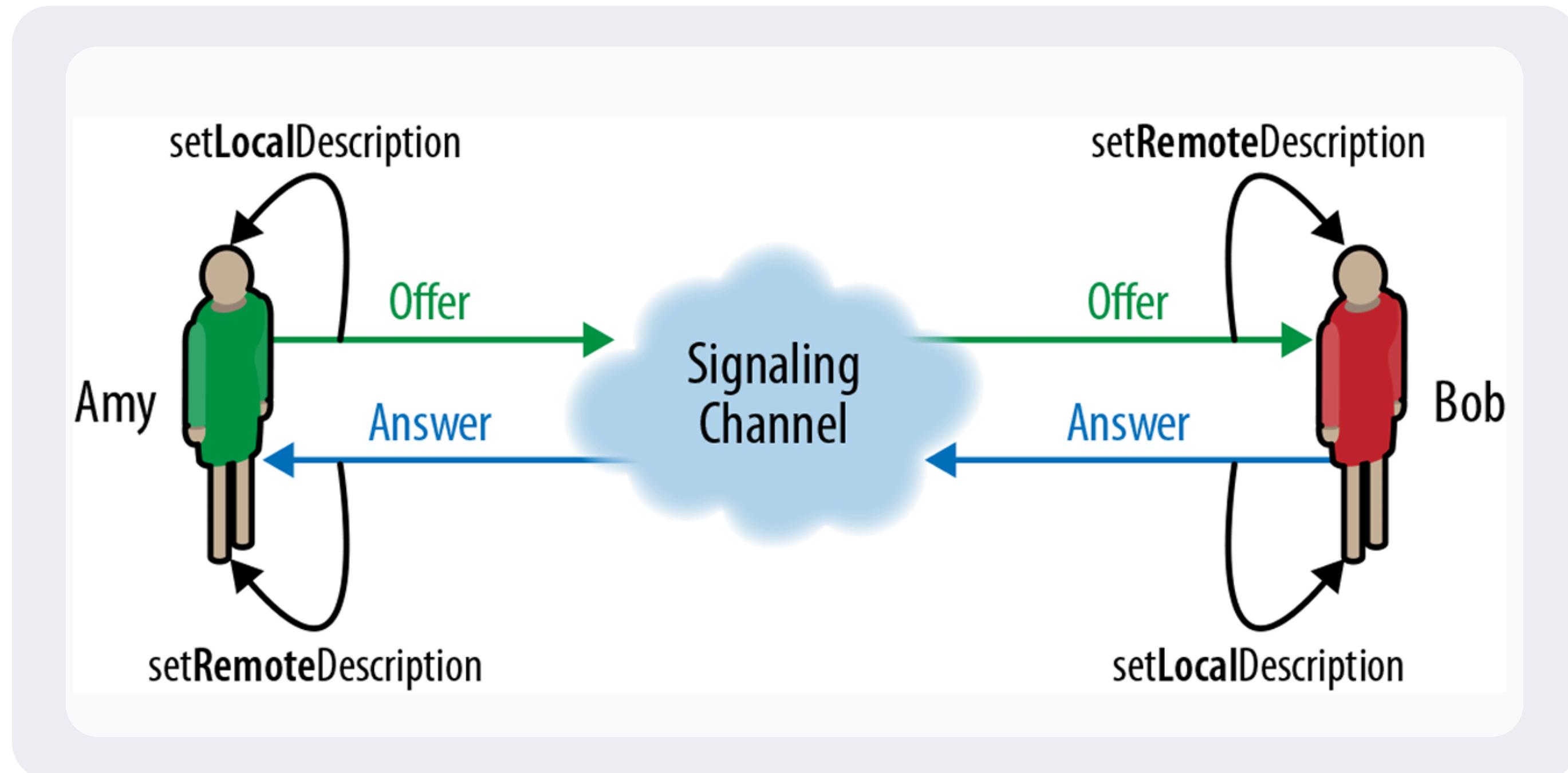
WebRTC General Sequence Diagram 5/6



WebRTC General Sequence Diagram 6/6



Signaling Server



Signaling Server



```
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
        registry.addHandler(new SignalingHandler(), "/signaling")
            .setAllowedOrigins("*")
            .addInterceptors(new CustomHandshakeInterceptor());
    }
}
```

1. Abilita WebSocket con l'annotazione **@EnableWebSocket**
2. Registra un *handler* che gestisce i messaggi al percorso **/signaling**
3. Definisce le origini dalle quali accettare le connessioni
4. **CustomHandshakeInterceptor** è un metodo customizzato da noi per permettere di personalizzare la fase di connessione, prestando attenzione ai ruoli dei peer, permettendo la comunicazione solo tra utenti e consulenti.

Signaling Server



```
public class CustomHandshakeInterceptor extends HttpSessionHandshakeInterceptor {  
    @Override  
    public boolean beforeHandshake(ServerHttpRequest request, ServerHttpResponse response,  
                                   WebSocketHandler wsHandler, Map<String, Object> attributes) throws Exception {  
  
        String query = request.getURI().getQuery(); //Ottengo la query string dall'URI, ad esempio "?role=consulente"  
        if (query != null) {  
            attributes.put("role", query.split("=")[1]); //divido la stringa in due parti separata dal simbolo "=" e prendo la seconda parte [1] ad esempio -> "role=consulente"  
            //In questo modo associo l'attributo "ruolo" ad ogni sessione per poi usarlo in afterConnectionEstablished  
        }  
  
        return super.beforeHandshake(request, response, wsHandler, attributes);  
    }  
}
```

Recupera il ruolo ("utente" o "consulente") di ogni sessione WebSocket al momento della connessione, dalla query string:

1. Prendo la query string
2. Controllo se la query string è vuota
3. Se non è vuota allora prendo il valore a destra di "role="
4. Chiamo il metodo **beforeHandshake** della classe genitore con super, per associare una sessione HTTP alla sessione WebSocket

Signaling Server



```
public class SignalingHandler extends TextWebSocketHandler {  
    private final Set<WebSocketSession> consultants = new HashSet<>(); //Vogliamo un set perché non devono esserci duplicati  
    private final Queue<WebSocketSession> waitingUsers = new LinkedList<>(); //Una sorta di coda, preleviamo il primo utente libero per assegnarlo ad un consulente  
    private final Map<WebSocketSession, WebSocketSession> sessionPairs = new HashMap<>(); //HashMap chiave-valore  
  
    @Override  
    public void afterConnectionEstablished(WebSocketSession session) throws Exception {  
        String role = (String) session.getAttributes().get("role");  
        System.out.println("Connessione stabilita con ruolo: " + role);  
  
        if ("consulente".equals(role)) {  
            consultants.add(session); //Se il ruolo è consulente allora aggiungo la sessione al set  
            System.out.println("Consulente aggiunto: " + session.getId());  
  
            //Controllo se ci sono utenti in attesa (se è entrato prima un utente e poi il consulente)  
            if (!waitingUsers.isEmpty()) {  
                WebSocketSession userSession = waitingUsers.poll(); //Prelevo il primo utente in attesa  
                sessionPairs.put(userSession, session); //Associo l'utente al consulente  
                sessionPairs.put(session, userSession); //Poichè l'associazione è bidirezionale faccio anche il viceversa  
  
                userSession.sendMessage(new TextMessage("{\"type\":\"info\",\"message\":\"Connesso a un consulente.\""}));  
                session.sendMessage(new TextMessage("{\"type\":\"info\",\"message\":\"Un utente si è connesso.\""}));  
                System.out.println("Pairing utente " + userSession.getId() + " con consulente " + session.getId());  
            }  
        }  
    }  
}
```

Quando la connessione è stata stabilita verifico il ruolo dell'entità connessa; se essa è un consulente lo aggiungo all'*HashSet* **consultants** e verifico la presenza di eventuali utenti in coda, se sono presenti, prendo il primo utente in coda ed associo le due sessioni all'*hashmap* **sessionPairs** , dato che il flusso è bidirezionale inserisco nell'*hashmap* anche il viceversa.

Signaling Server



```
    } else if ("utente".equals(role)) {
        System.out.println("Utente connesso: " + session.getId());
        WebSocketSession consultantSession = consultants.stream().findFirst().orElse(null); //Se un utente si è connesso cerco *il primo* consulente disponibile

        if (consultantSession != null) {
            consultants.remove(consultantSession); //Rimuovo il consulente dal set perché è stato assegnato ad un utente
            sessionPairs.put(session, consultantSession); //Aggiorno l'hashmap sessionPairs -> sessioneUtente(session) + sessioneConsulente(assignedConsultant)
            sessionPairs.put(consultantSession, session); //Lo faccio due volte per avere un accoppiamento bidirezionale

            session.sendMessage(new TextMessage("{\"type\":\"info\",\"message\":\"Connesso a un consulente.\""}));
            consultantSession.sendMessage(new TextMessage("{\"type\":\"info\",\"message\":\"Un utente si è connesso.\""}));

            System.out.println("Pairing utente " + session.getId() + " con consulente " + consultantSession.getId());
        } else {
            waitingUsers.add(session); //Inserisco l'utente nella coda di attesa
            session.sendMessage(new TextMessage("{\"type\":\"info\",\"message\":\"In attesa di un consulente.\""}));
            System.out.println("Nessun consulente disponibile. Utente in attesa.");
        }
    }
}
```

Continuando: se invece l'entità connessa è un utente, allora associo a **consultantSession** la sessione del primo consulente disponibile, dopodiché rimuovo il consulente dal set perché è stato già assegnato, poi aggiorno nuovamente l'*hashmap* **sessionPairs**. Se invece non c'è nessun consulente disponibile, l'utente viene messo in attesa di un consulente che si connetta.

Signaling Server



```
@Override  
protected void handleTextMessage(WebSocketSession session, TextMessage message) throws Exception {  
    WebSocketSession pairedSession = sessionPairs.get(session); //Recupero il valore associato alla chiave, ad esempio ( key:userSession -> value:consultantSession)  
    if (pairedSession != null && pairedSession.isOpen()) {  
        System.out.println("Inoltro messaggio da " + session.getId() + " a " + pairedSession.getId());  
        pairedSession.sendMessage(message); //Se la sessione era di un Utente -> Consulente allora invio il msg al Consulente e viceversa (flusso bidirezionale)  
    }  
}
```

Implementa un vero e proprio flusso di comunicazione bidirezionale

1. Recupero il valore associato alla chiave di **sessionPairs**
2. Ci permette di riconoscere il tipo di flusso:

Utente -> Consulente

Consulente -> Utente

Signaling Server



```
@Override  
public void afterConnectionClosed(WebSocketSession session, CloseStatus status) throws Exception {  
    System.out.println("Connessione chiusa: " + session.getId());  
  
    WebSocketSession pairedSession = sessionPairs.remove(session); //Quando una connessione viene chiusa, cerco la sessione accoppiata (pairedSession) e la rimuovo dalla hashmap  
    //pairedSession avrà dunque la sessione duale a quella chiusa  
    if (pairedSession != null) {  
        sessionPairs.remove(pairedSession); //se la sessione chiusa aveva una sessione accoppiata allora la rimuovo dall'hashmap  
        pairedSession.sendMessage(new TextMessage("{\"type\":\"info\",\"message\":\"La connessione è stata chiusa.\""}));  
        pairedSession.close(); //chiudo la sessione accoppiata  
    }  
  
    consultants.remove(session);  
    waitingUsers.remove(session);  
}
```

Gestisce la chiusura di una connessione:

1. Rimuovo la sessione dell'utente che ha chiuso la sessione dall'*hashmap*
2. Se era presente una sessione accoppiata rimuove anch'essa dall'*hashmap*
3. Chiude definitivamente anche l'altra sessione

WebRTC DataChannel



WebRTC

```
1 const config = {  
2   iceServers: [{ urls: "stun:stun.l.google.com:19302" }]  
3 };  
4  
5 const pc = new RTCPeerConnection(config);  
6  
7 const role = "utente";  
8 const ws = new WebSocket(`ws://localhost:8080/signaling?role=${role}`);  
9  
10 const chatInput = document.getElementById("chatInput");  
11 const chatOutput = document.getElementById("chatOutput");  
12 const connectButton = document.getElementById("connectButton");  
13 const fileInput = document.getElementById("fileInput");  
14 const sendFileButton = document.getElementById("sendFileButton");  
15 const fileLinks = document.getElementById("fileLinks");  
16  
17  
18 let dataChannel;  
19 let fileBuffer = [];  
20 let receivedFileMetadata = null;  
21 let username = "";  
22  
23  
24 async function getUsernameFromSession() {  
25   try {  
26     const response = await fetch('http://localhost:8080/get-username');  
27     if (!response.ok) {  
28       throw new Error("Errore durante il recupero dell'username");  
29     }  
30     return await response.text();  
31   } catch (error) {  
32     console.error("Errore:", error);  
33     return null;  
34   }  
35 }
```

Per implementare il **DataChannel WebRTC** per prima cosa viene configuriamo il server STUN da utilizzare durante la fase di *ICE Gathering*, poi viene creato un oggetto **RTCPeerConnection** e passiamo come parametro la configurazione precedentemente discussa.

Definiamo il ruolo della chat e creiamo un oggetto **WebSocket** collegato all'endpoint del **Controller SpringBoot**.

Definiamo le variabili globali necessarie e la funzione per ottenere l'username dalla sessione, in modo tale da rendere disponibili gli username di utente e consulente all'interno della chat

WebRTC DataChannel



WebRTC

```
55 ws.onmessage = async (event) => {
56   const message = JSON.parse(event.data);
57   console.log("Messaggio dal WebSocket:", message);
58
59   if (message.type === "info") {
60     chatOutput.innerHTML += `<br><em>${message.message}</em>`;
61     loadUsername();
62
63   } else if (message.type === "offer") {
64     await pc.setRemoteDescription(new RTCSessionDescription(message))
65
66     const answer = await pc.createAnswer();
67     await pc.setLocalDescription(answer);
68     ws.send(JSON.stringify(pc.localDescription));
69
70   } else if (message.type === "answer") {
71     await pc.setRemoteDescription(new RTCSessionDescription(message));
72
73   } else if (message.candidate) {
74     await pc.addIceCandidate(new RTCIceCandidate(message));
75
76   } else if (message.type === "username") {
77     if (message.username) {
78       chatOutput.innerHTML += `<br><em>Consulente: ${message.username}</em>`;
79     } else {
80       console.log('Username del consulente non disponibile');
81     }
82   }
83 }
```

La seguente funzione freccia asincrona, "in ascolto" di un messaggio dalla WebSocket, permette di gestire il flusso di negoziazione per il *protocollo WebRTC*; In particolare se il messaggio è di tipo **info**, iniziamo a caricare l'username dell'utente. Se il messaggio è **l'offerta** dell'altro peer, utilizziamo la funzione **setRemoteDescription** di *Java Session Establishment Protocol (JSEP)* per impostare l'offerta remota, e in modo asincrono chiamiamo **createAnswer** e impostiamo la descrizione locale per poi poterla inviare al server di *signaling WebSocket*. Se invece il messaggio è di tipo **candidate** aggiungo il **candidato ICE** del peer remoto. Infine l'ultimo messaggio di tipo **username** è utilizzato per mostrare in chat l'username dell'utente.

WebRTC DataChannel



```
86 pc.onicecandidate = ({ candidate }) => {
87   if (candidate) {
88     ws.send(JSON.stringify(candidate));
89   }
90 };
91
92
93 connectButton.onclick = async () => {
94   dataChannel = pc.createDataChannel("chat");
95   setupDataChannel(dataChannel);
96
97   const offer = await pc.createOffer();
98   await pc.setLocalDescription(offer);
99   ws.send(JSON.stringify(pc.localDescription));
100 };
101
102 pc.ondatachannel = (event) => {
103   dataChannel = event.channel;
104   setupDataChannel(dataChannel);
105 };
```

Il metodo **onicecandidate** invia tramite WebSocket il candidato/i all'altro peer.

Il metodo **connectButton** permette di creare il *data channel* con **createDataChannel** e viene effettuato il setup con la funzione **setupDataChannel**, creata ad hoc al fine di inserire tutto il necessario al corretto funzionamento dello stesso. Viene poi creata l'offerta, per poi impostarla con **setLocalDescription** ed infine viene inviata al *signaling server*.

La funzione freccia **ondatachannel** permette al *peer* remoto di rilevare la creazione di un *data channel* e di effettuarne il setup.

WebRTC DataChannel



WebRTC

```
109 function setupDataChannel(channel) {  
110   channel.onopen = () => {  
111     console.log("DataChannel aperto, ora è possibile inviare messaggi e file");  
112   };  
113  
114   channel.onmessage = (e) => {  
115     const message = JSON.parse(e.data);  
116     if (message.type === "text") {  
117       chatOutput.innerHTML += `<br><strong>${message.username}</strong> ${message.data}`;  
118     } else if (message.type === "file-metadata") {  
119       receivedFileMetadata = message;  
120       fileBuffer = [];  
121     } else if (message.type === "file-chunk") {  
122       fileBuffer.push(new Uint8Array(message.data));  
123       if (fileBuffer.length === Math.ceil(receivedFileMetadata.size / 16000)) {  
124         const blob = new Blob(fileBuffer);  
125         const link = document.createElement("a");  
126         link.href = URL.createObjectURL(blob);  
127         link.download = receivedFileMetadata.name;  
128         link.textContent = `Download ${receivedFileMetadata.name}`;  
129         fileLinks.appendChild(link);  
130         fileLinks.appendChild(document.createElement("br"));  
131       }  
132     }  
133   };  
134  
135  
136   chatInput.onkeypress = (e) => {  
137     if (e.key === "Enter" && channel.readyState === "open") {  
138       const message = { type: "text", username: username, data: chatInput.value };  
139       channel.send(JSON.stringify(message));  
140       chatOutput.innerHTML += `<br><strong>${username}</strong> ${chatInput.value}`;  
141       chatInput.value = "";  
142     }  
143   };  
};
```

La funzione **setupDataChannel** prende in ingresso il channel e nel caso in cui arrivi un messaggio di tipo **text** non farà altro che stamparlo in chat. Se invece arriva il messaggio di tipo **file-metadata**, lo salvo perché servirà all'atto di ricezione del file per stabilire la grandezza in termini di byte. Se invece il messaggio è di tipo **file-chunk**, inserisco nel *fileBuffer* il chunk in formato *Uint8Array*, poi controllo se ho ricevuto tutti i chunk, in caso affermativo creo un *binary large object (BLOB)* e un link associato per permettere al ricevente di effettuare il download del file.

chatInput.onkeypress rileva l'input del tasto *enter* per inoltrare il contenuto della chat attraverso il *data channel*.

WebRTC DataChannel



WebRTC

```
145e  sendFileButton.onclick = () => {
146e    if (channel.readyState === "open") {
147      const file = inputFile.files[0];
148e    if (file) {
149      const reader = new FileReader();
150e    reader.onload = () => {
151      const arrayBuffer = reader.result;
152      channel.send(JSON.stringify({ type: "file-metadata", name: file.name, size: file.size }));
153     .sendFileChunks(arrayBuffer);
154    };
155      reader.readAsArrayBuffer(file);
156    }
157e  } else {
158    console.warn("DataChannel non disponibile o non aperto. Impossibile inviare file.");
159  }
160  };
161 }

164efunction sendFileChunks(arrayBuffer) {
165  const chunkSize = 16000;
166  const uint8Array = new Uint8Array(arrayBuffer);
167e  for (let i = 0; i < uint8Array.length; i += chunkSize) {
168    const chunk = uint8Array.slice(i, i + chunkSize);
169    dataChannel.send(JSON.stringify({ type: "file-chunk", data: Array.from(chunk) }));
170  }
171 }
```

sendFileButton.onclick controlla prima se il canale è aperto ed in caso affermativo prende il file allegato, crea un oggetto *FileReader* e lo converte in *ArrayBuffer*, quando è stato correttamente “elaborato”, invia sul *data channel* i metadati del file, per poi inviare i singoli *chunk* con la funzione **sendFileChunks**.

Quest’ultima funzione definisce la grandezza del singolo *chunk* e converte l’arraybuffer in un *Uint8Array* più comodo, calcola gli slice del file e poi li invia uno ad uno attraverso il *data channel* all’altro *peer*.



Riferimenti

- <https://docs.spring.io/spring-framework/reference/web/websoket/server.html>
- <https://docs.spring.io/spring-framework/docs/4.3.15.RELEASE/spring-framework-reference/html/websoket.html>
- <https://www.w3schools.com/js/>
- <https://webrtc.org/?hl=it>
- <https://webrtc.org/getting-started/peer-connections?hl=it>
- Romano, S. P. (2024). Lezione su WebRTC [Materiale didattico]. Università di Napoli Federico II.



TRAVEL MILES
COMMUNITY VIAGGI

Thank You

20 Dec, 2024