

Real-time video call system - with WebRTC

R10525073 工科碩一 徐聖淮

錄影連結：https://drive.google.com/file/d/1hL38HTIQz7NEU4LHwQ-Zb8RioxuU3M_Y/view?usp=sharing

Outline

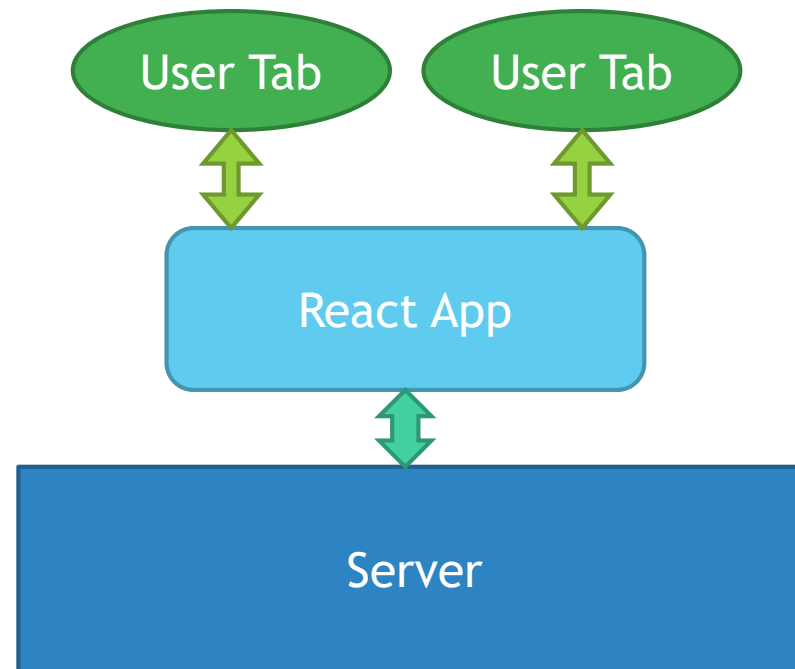
1. Introduction
 2. Architecture
 3. Code Review
 4. Live Demo
 5. Conclusion
- Reference

1. Introduction

- ▶ WebRTC is a API that enables Web applications to:
 - ▶ audio and video conferencing
 - ▶ file exchange
 - ▶ screen sharingEtc...
- ▶ This project uses WebRTC to make a real-time video call system.
 - ▶ User does not need to install the system and can also execute it on the web page.

2. Architecture

- ▶ System Type
 - ▶ Web interactive system
- ▶ System platform
 - ▶ Python, React, Node.js
 - ▶ Mouse and keyboard



3. Code Review-1

► Server.py

```
# event handler for the join event
@socketio.on('join')
def join(message):
    username = message['username']
    room = message['room']
    join_room(room)
    print('RoomEvent: {} has joined the room {}\n'.format(username, room))
    emit('ready', {username: username}, to=room, skip_sid=request.sid)

# event handler for the data event
@socketio.on('data')
def transfer_data(message):
    username = message['username']
    room = message['room']
    data = message['data']
    print('DataEvent: {} has sent the data:\n {} \n'.format(username, data))
    emit('data', data, to=room, skip_sid=request.sid)

# error handler
@socketio.on_error_default
def default_error_handler(e):
    print("Error: {}".format(e))
    socketio.stop()
```

3. Code Review-2

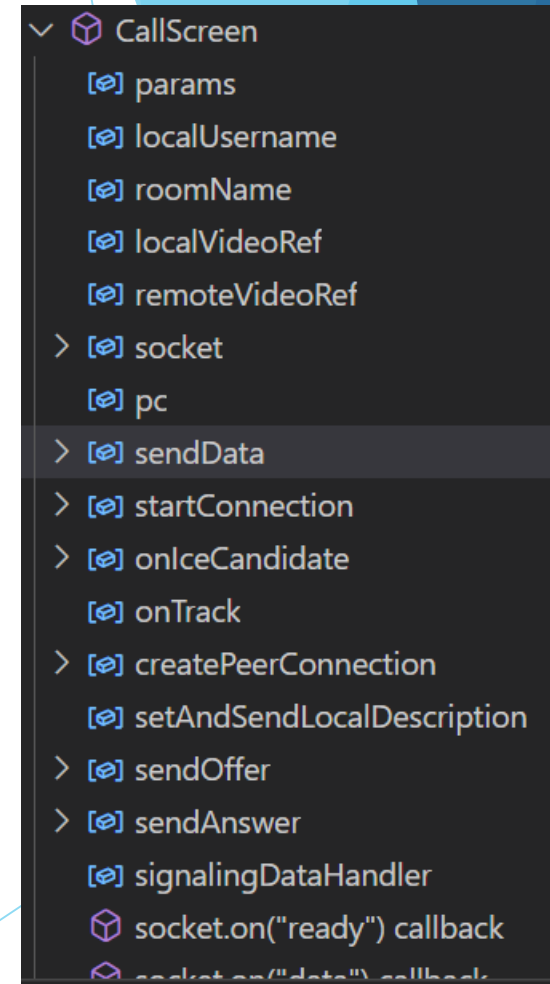
► HomeScreen.js

```
function HomeScreen() {  
  const [room, setRoom] = useState("");  
  const [username, setUsername] = useState("");  
  
  return (  
    <form method="post" action="">  
      <label for="username">Username</label>  
  
      <input  
        value={username}  
        title="username"  
        onInput={(e) => setUsername(e.target.value)}  
      />  
  
      <label for="room">Room</label>  
  
      <input  
        value={room}  
        title="room"  
        onInput={(e) => setRoom(e.target.value)}  
      />  
  
      <Link to={` /call/${username}/${room}`}>  
        <input type="submit" name="submit" value="Join Room" />  
      </Link>  
    </form>  
  );  
}
```

3. Code Review-3

► CallScreen.js

1. Peer A creates a `RTCPeerConnection` object for the connection.
2. Peer A creates an offer SDP message with `createOffer()` and calls `setLocalDescription()` to set it as the local SDP description.
3. Peer A now sends this offer in a stringified form to Peer B via a signaling server.
4. Peer B creates a `RTCPeerConnection` object and calls `setRemoteDescription()` with Peer A's offer to know about its setup.

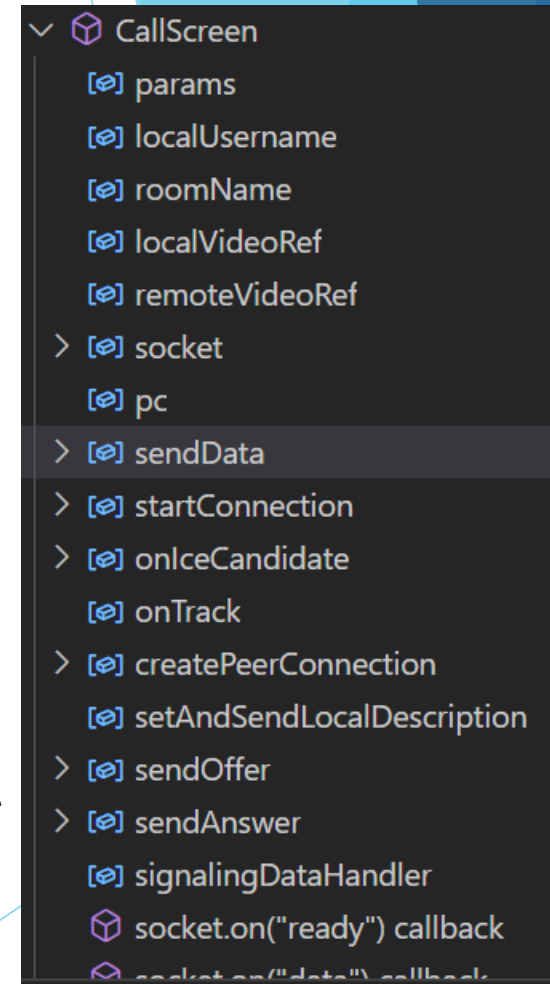


```
CallScreen
├── [x] params
├── [x] localUsername
├── [x] roomName
├── [x] localVideoRef
├── [x] remoteVideoRef
├── [x] socket
├── [x] pc
├── [x] sendData
├── [x] startConnection
├── [x] onIceCandidate
├── [x] onTrack
├── [x] createPeerConnection
├── [x] setAndSendLocalDescription
├── [x] sendOffer
├── [x] sendAnswer
├── [x] signalingDataHandler
├── [x] socket.on("ready") callback
└── [x] socket.on("data") callback
```

3. Code Review-4

► CallScreen.js

5. Peer B creates an answer SDP message with `createAnswer()` and calls `setLocalDescription()` to set it as the local SDP description.
6. Peer B now sends this answer in a stringified form to Peer A using a signaling server.
7. Peer A calls `setRemoteDescription()` with the answer received in order to know about Peer B's setup.
8. Either of these peers can send ICE Candidates to the other on generation, with the help of the `onIceCandidate` callback, and set the candidates received from the other using `addIceCandidate()`.
9. Connection is established by the end of this flow.



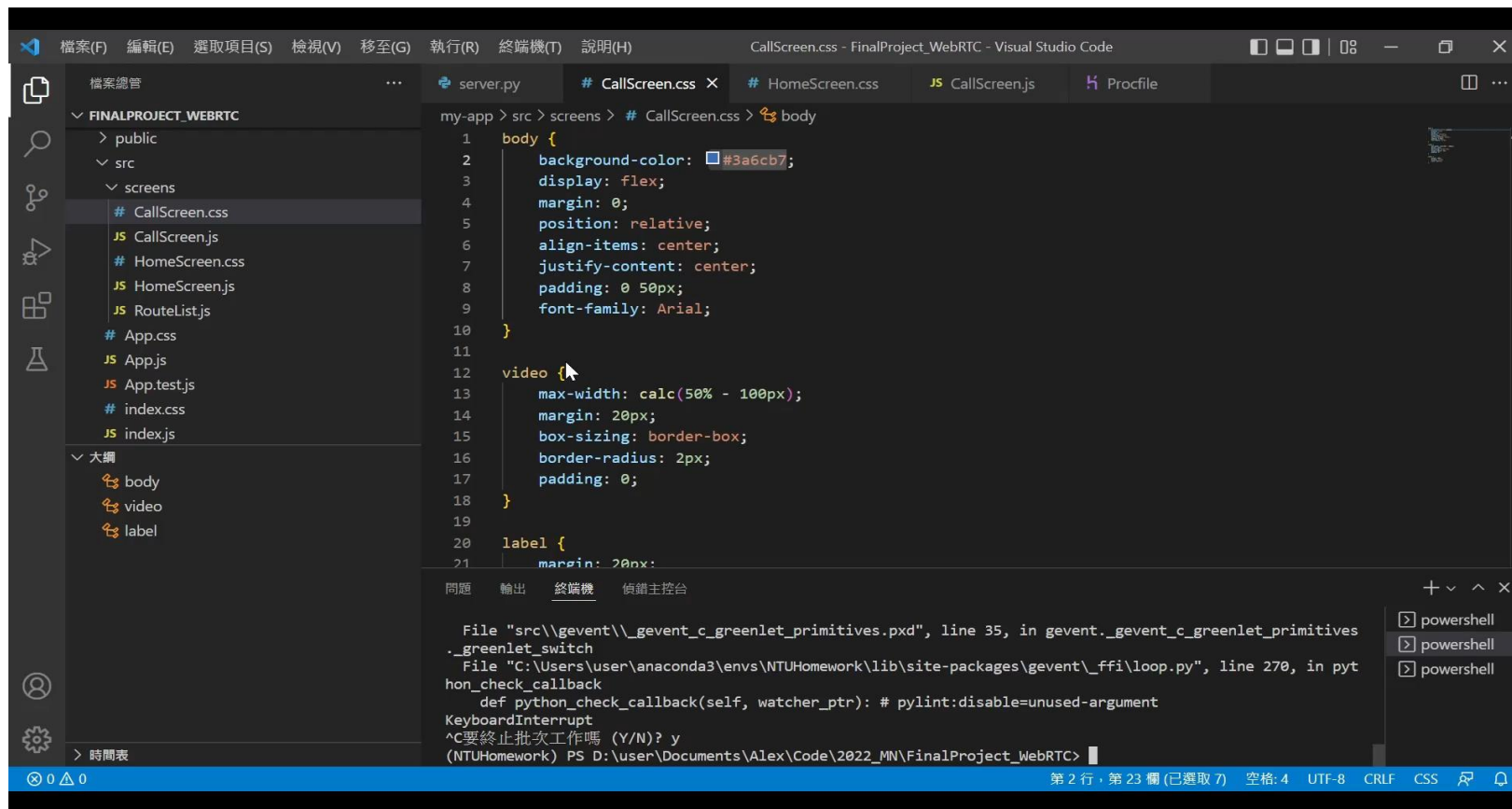
```
CallScreen
├── [🔗] params
├── [🔗] localUsername
├── [🔗] roomName
├── [🔗] localVideoRef
├── [🔗] remoteVideoRef
├── > [🔗] socket
├── [🔗] pc
├── > [🔗] sendData
├── > [🔗] startConnection
├── > [🔗] onIceCandidate
├── [🔗] onTrack
├── > [🔗] createPeerConnection
├── [🔗] setAndSendLocalDescription
├── > [🔗] sendOffer
├── > [🔗] sendAnswer
├── [🔗] signalingDataHandler
├── [🔗] socket.on("ready") callback
└── [🔗] socket.on("data") callback
```


3. Code Review-5

► RouteList.js

```
function RouteList() {  
  return (  
    <Routes>  
      <Route path="/" element={<HomeScreen />} />  
      <Route path="/call/:username/:room" element={<CallScreen />} />  
    </Routes>  
  );  
}
```

4. Live Demo



The screenshot displays the Visual Studio Code interface for a project named 'FinalProject_WebRTC'. The left sidebar shows the file explorer with a directory structure: 'FINALPROJECT_WEBRTC' containing 'public' and 'src'. Under 'src', there is a 'screens' directory with files like 'CallScreen.css', 'CallScreen.js', 'HomeScreen.css', 'HomeScreen.js', 'RouteList.js', 'App.css', 'App.js', 'App.test.js', 'index.css', and 'index.js'. Below this, a '大綱' (Outline) view shows 'body', 'video', and 'label' elements.

The main editor window shows the 'CallScreen.css' file with the following CSS rules:

```
1 body {  
2   background-color: #3a6cb7;  
3   display: flex;  
4   margin: 0;  
5   position: relative;  
6   align-items: center;  
7   justify-content: center;  
8   padding: 0 50px;  
9   font-family: Arial;  
10 }  
11  
12 video {  
13   max-width: calc(50% - 100px);  
14   margin: 20px;  
15   box-sizing: border-box;  
16   border-radius: 2px;  
17   padding: 0;  
18 }  
19  
20 label {  
21   margin: 20px;
```

The bottom panel shows the 'Terminal' with the following output:

```
File "src\gevent\_gevent_c_greenlet_primitives.pxd", line 35, in gevent._gevent_c_greenlet_primitives  
._greenlet_switch  
File "C:\Users\user\anaconda3\envs\NTUHomework\lib\site-packages\gevent\_ffi\loop.py", line 270, in pyt  
hon_check_callback  
def python_check_callback(self, watcher_ptr): # pylint:disable=unused-argument  
KeyboardInterrupt  
^C要終止批次工作嗎 (Y/N)? y  
(NTUHomework) PS D:\user\Documents\Alex\Code\2022_MN\FinalProject_WebRTC>
```

The status bar at the bottom indicates the current file is 'CallScreen.css', line 2, column 23, with a UTF-8 encoding and CRLF line endings.

5. Conclusion

- ▶ In this work, we implement a real-time interactive video call system.
- ▶ We also provide a user-friendly GUI to make the system more practical.
- ▶ In future work, We hope to integrate more functions, such as text chat, to allow users to have more diverse operation options.

Reference

- ▶ [Build your first WebRTC app with Python and React](#)
- ▶ [Python Flask 入門指南：輕量級網頁框架教學](#)
- ▶ [Welcome to Flask – Flask Documentation \(2.1.x\)](#)
- ▶ [React Documentation](#)
- ▶ [WebRTC API - Web APIs | MDN](#)

The background features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side of the frame, creating a modern, layered effect. The rest of the background is a solid, very light blue.

The End

Thank you!