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

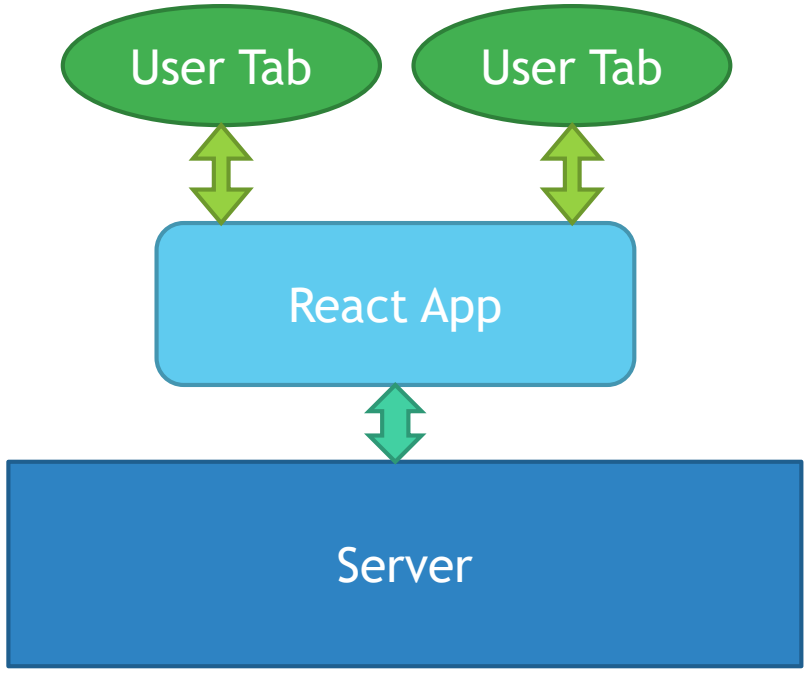
1. Introduction

- WebRTC is a API that enables Web applications to:
  - audio and video conferencing
  - file exchange
  - screen sharing

  Etc...
- 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

```python
# 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()
```

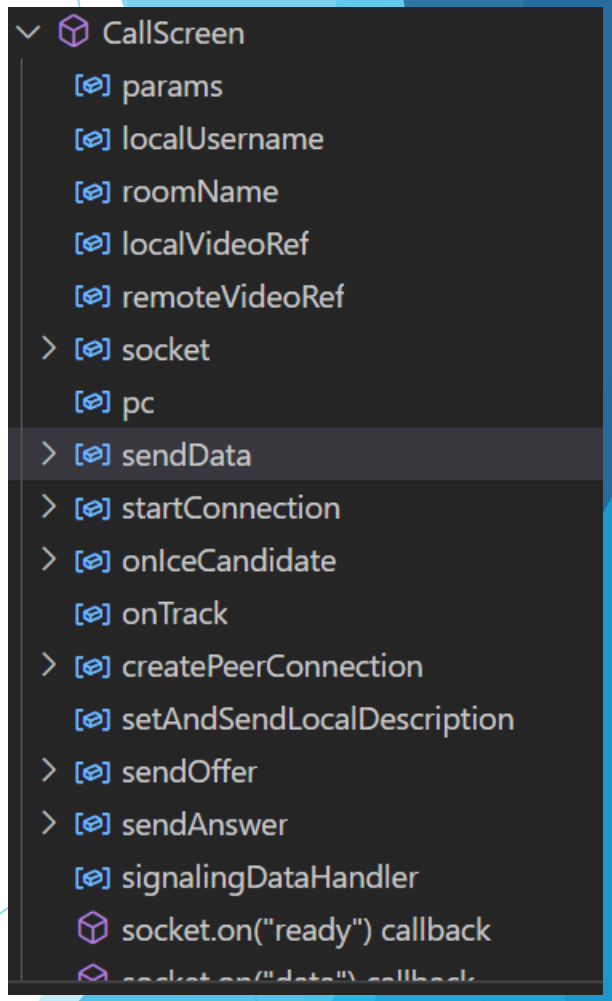
3. Code Review-2

- 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>
  );
}
```

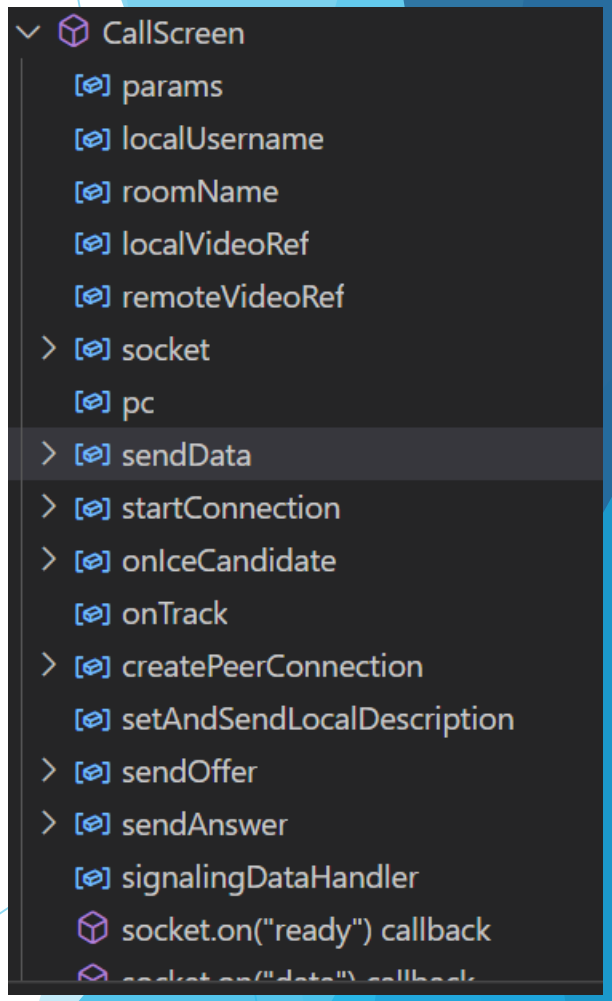
3. Code Review-3

- 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.



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.

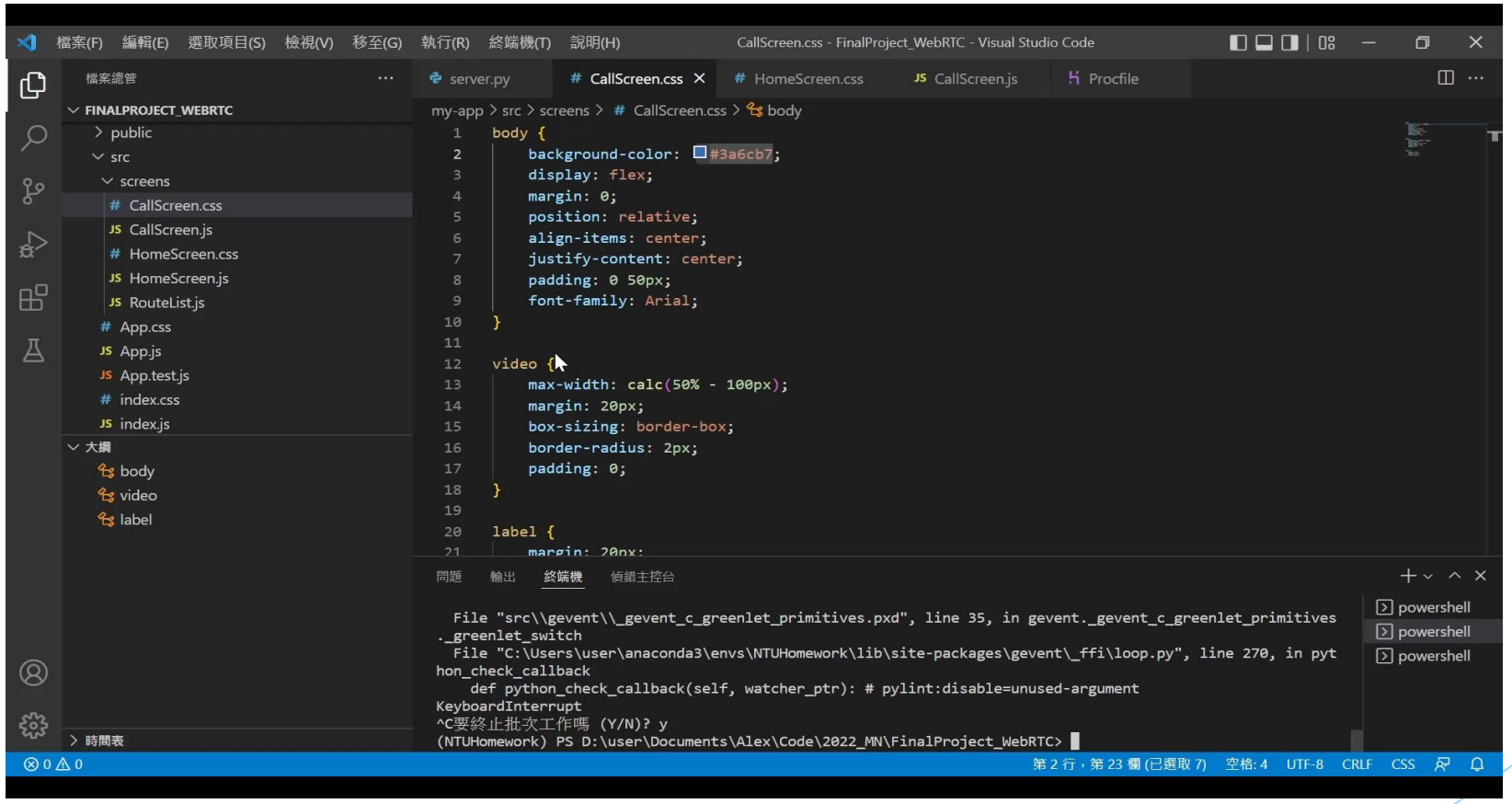


3. Code Review-5

- RouteList.js

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

4. Live Demo


檔案(F) 編輯(E) 選取項目(S) 檢視(V) 移至(G) 執行(R) 終端機(T) 說明(H)
CallScreen.css - FinalProject_WebRTC - Visual Studio Code
檔案總管
FINALPROJECT_WEBRTC
public
src
screens
CallScreen.css
CallScreen.js
HomeScreen.css
HomeScreen.js
RouteList.js
App.css
App.js
App.test.js
index.css
index.js
大綱
body
video
label
時間表
server.py
CallScreen.css
HomeScreen.css
CallScreen.js
Procfile
my-app > src > screens > CallScreen.css > body
body {
background-color: #3a6cb7;
display: flex;
margin: 0;
position: relative;
align-items: center;
justify-content: center;
padding: 0 50px;
font-family: Arial;
}
video {
max-width: calc(50% - 100px);
margin: 20px;
box-sizing: border-box;
border-radius: 2px;
padding: 0;
}
label {
問題 輸出 終端機 偵錯主控台
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 python_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>
powershell
powershell
powershell
第 2 行，第 23 欄 (已選取 7) 空格: 4 UTF-8 CRLF CSS

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 End

Thank you!