CODE ZERO

박의용(팀장) & 강민영

GUI

3인이상이 참여하는
멀티 통신

DB

DB1. 채팅방 대화내용
DB2. 유저 정보리스트

ARCHIVE

지난 채팅방
최근 내용 불러오기

MEMBERS

박의용

3인이상 참여 서버 구현
GUI 연결구현(채팅방)

강민영

클라이언트(채팅) 구현
클라이언트 DB 구현

SERVER 1. 소켓 생성 & 연결대기

- 소켓 생성 [socket.socket()]
- 연결 대기 [listen()]
- Thread 생성 [threading.Thread()]
  Thread1. 메세지 송.수신
  Thread2. 서버 닫기

```
# 유저로부터 메시지를 받을 쓰레드 생성
thread = threading.Thread(target=self.process, args=())
thread.start()
```

```
# 서버를 종료할 수 있어야 한다. ( 엔터 입력시 종료 )
exit = threading.Thread(target=self.closeServer, args=())
exit.start()
```

SERVER 2. 인코딩

- 메세지 UTF-8 형식으로 인코딩
  [str().encode(format)]
- 보낼 메세지의 길이를 인코딩
- return message, send_length

```python
def encodeMsg(msg):
    # 메세지 텍스트를 UTF-8 형식으로 인코딩
    message = str(msg).encode(FORMAT)
    print(message)
    # print(message)
    # 인코딩 된 메세지의 길이 저장
    msg_length = len(message)
    # print(msg_length)
    # 메세지 길이를 UTF-8 형식으로 인코딩
    send_length = str(msg_length).encode(FORMAT)
    # print(send_length)
    # 정의된 HEADER 와 같을 때까지 공백으로 길이 메시지를 완성한다.
    # 반환 메시지 = send_length
    send_length += b' ' * (HEADER - len(send_length))
    # print(send_length)

    return message, send_length
```

SERVER 3. 인코딩

- 메세지 UTF-8 형식으로 인코딩
  [str().encode(format)]
- 보낼 메세지의 길이를 인코딩
- return message, send_length

```python
def encodeMsg(msg):
    # 메세지 텍스트를 UTF-8 형식으로 인코딩
    message = str(msg).encode(FORMAT)
    print(message)
    # print(message)
    # 인코딩 된 메세지의 길이 저장
    msg_length = len(message)
    # print(msg_length)
    # 메세지 길이를 UTF-8 형식으로 인코딩
    send_length = str(msg_length).encode(FORMAT)
    # print(send_length)
    # 정의된 HEADER 와 같을 때까지 공백으로 길이 메시지를 완성한다.
    # 반환 메시지 = send_length
    send_length += b' ' * (HEADER - len(send_length))
    # print(send_length)

    return message, send_length
```

SERVER 4-1. 수신 메세지 처리

메세지 변수 저장 및 번호 분리

```
NEW_MESSAGE = '0'
NAME_LIST = '1'
CLEAR_LIST = '2'
DISCONNECT_MESSAGE = '3'
SAVE_LIST = '4'
MEMBER_INVITE = '5'
```

- if re == NEW_MESSAGE
  새로운 메세지를 표시하는 번호 ->
  연결된 사용자에게 메세지를 보냄

- elif (re == DISCONNECT_MESSAGE):
  오프라인 유저를 표시하는 번호 ->
  서버에 연결 해제 요청

```
def handleMsg(self, msg):

    # 메세지 변수 저장 (번호 분리)
    re = msg[0]  # 첫 번째 문자 저장
    msg_list = list(msg)  # 리스트로 변환
    msg_list.pop(0)  # 번호 삭제
    msg = "".join(msg_list)  # 문자열에 저장

    # 수행할 작업
    if (re == NEW_MESSAGE):  # 새로운 메세지 번호인 경우
        self.server.userMsg(msg, self)  # 연결된 사용자에게 보내기

    elif (re == DISCONNECT_MESSAGE):

        self.userOnline = False  # 유저 오프라인으로 설정
        self.server.cancleConnection()  # 서버에 연결 해제 요청

    elif (re == MEMBER_INVITE):
        print(f"{msg}님을 채팅방에 초대하고있습니다...")
        self.invite(msg)
        print(self.invite(msg))
```

SERVER 4-2. 대화내용 DB저장

```
if msg_length:
    msg_length = int(msg_length)  # 값을 인트형 저장
    # 메세지 수신 및 디코딩
    msg = self.conn.recv(msg_length).decode(FORMAT)
```

- self.conn.recv(msg).decode(FORMAT)
- 수신 메세지 디코딩

- 수신메세지를 decoding 한 후, DB에 저장:
  f"insert into newschema.chatting1(user_id, message, ip_address, port_number, time) values('{self.username}', '{self.dbmsg}', '{self.addr[0]}', '{self.addr[1]}', '{date_}')

SERVER 1. 연결 & THREAD

- 소켓 생성 및 서버와 연결
  - socket.socket()
  - client.connect(ADDR)

- 메세지를 받을 쓰레드 생성
  threading.Thread()

```
class Client():

    # 소켓 클라이언트 초기화
    def __init__(self, username, address, port, win):
        # 연결 유형 정리
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # 소켓 서버와 연결
        ADDR = (address, int(port))
        self.client.connect(ADDR)

        # 매개변수 수신
        self.username = username  # 인스턴스 사용자 이름 설정
        self.win = win  # 통신창 참조 저장
        self.online = True  # 클라이언트를 온라인으로 설정

        # 사용자 이름을 서버로 보내기
```

SERVER 2. GUI [MAIN THREAD]

- Gui [main thread]
  메인 쓰레드에서 UI 실행

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = LogWindow()
    win.show()
    app.exec_()
```

- 로그인 버튼 클릭 -> 초대할 친구 클릭
  -> 클라이언트 서버 시작

```
def Chat(self, username, address, port):

    self.signal = MySignal()
    self.signal.listUser.connect(self.listUpdate)
    self.signal.chatLabel.connect(self.chatUpdate)

    self.client = Client(username, address, port, self)
```

```
class LogWindow(QMainWindow, form_class):

    def __init__(self):

        super(QMainWindow, self).__init__()
        self.setupUi(self)

        self.tabWidget.setCurrentIndex(0)
        self.profileList = []

        for i in range(1, 13):
            self.profileList.append(convertToBinaryData(f"C:\chatting\one{i}.png"))

        self.name = False
        self.addr = False
        self.prt = False
        self.act = ""

        self.user_tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.chat_tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)
        self.user_tableWidget.cellDoubleClicked.connect(self.friend)
        # self.chat_tableWidget.cellDoubleClicked.connect(self.friendinvite)

        self.entrance_btn.clicked.connect(self.login)

        self.chat_exit_btn.clicked.connect(self.waitroom)
```

SERVER 3. 사용자 정의 SIGNAL

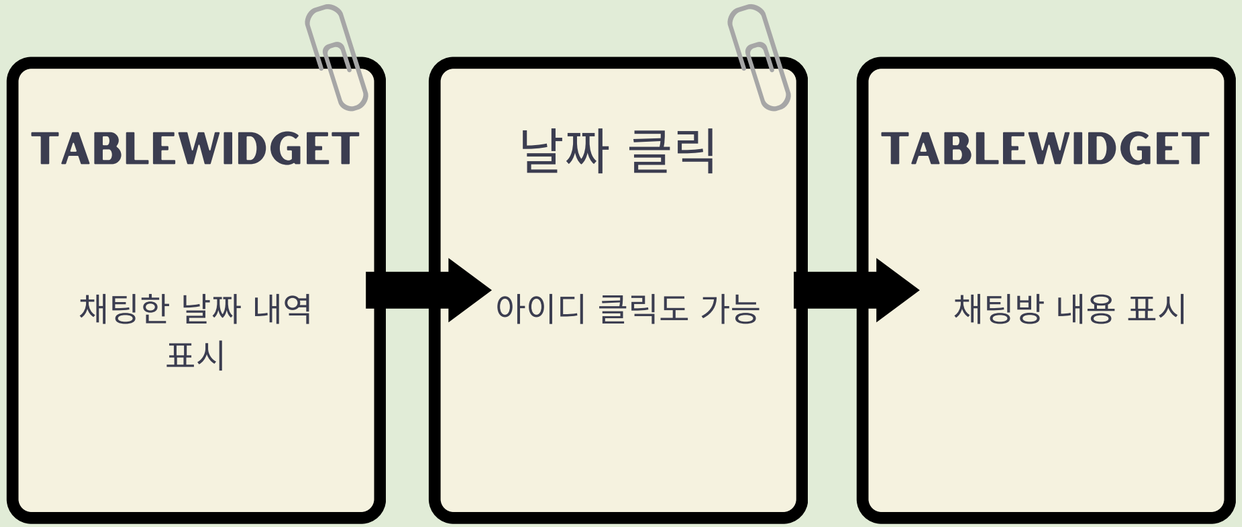
- 사용자 정의 Signal 사용
  Signal 1. listUser - user list 업데이트
  Signal 2. chatLabel - 메세지 나타내기

```
if (re == NEW_MESSAGE):  # 새로운 메세지 T
    self.win.signal.chatLabel.emit(msg)

elif (re == CLEAR_LIST):  # 목록을 지우는
    self.win.signal.listUser.emit('')  #

elif (re == NAME_LIST):  # 이름 목록인 경우
    self.win.signal.listUser.emit(msg)
```

```
class MySignal(QtCore.QObject):
    listUser = QtCore.pyqtSignal(str)
    chatLabel = QtCore.pyqtSignal(str)
```

```
def Chat(self, username, address, port):

    self.signal = MySignal()
    self.signal.listUser.connect(self.listUpdate)
    self.signal.chatLabel.connect(self.chatUpdate)
```

SERVER 4. 최근 대화내용 조회




```
def viewhistory(self):
    userlist = []

    self.tabWidget.setCurrentIndex(3)

    sql = f"select distinct user_id, time from newschema.chatting1"
    cur.execute(sql)

    history = cur.fetchall()
    con.commit()
```

구동영상