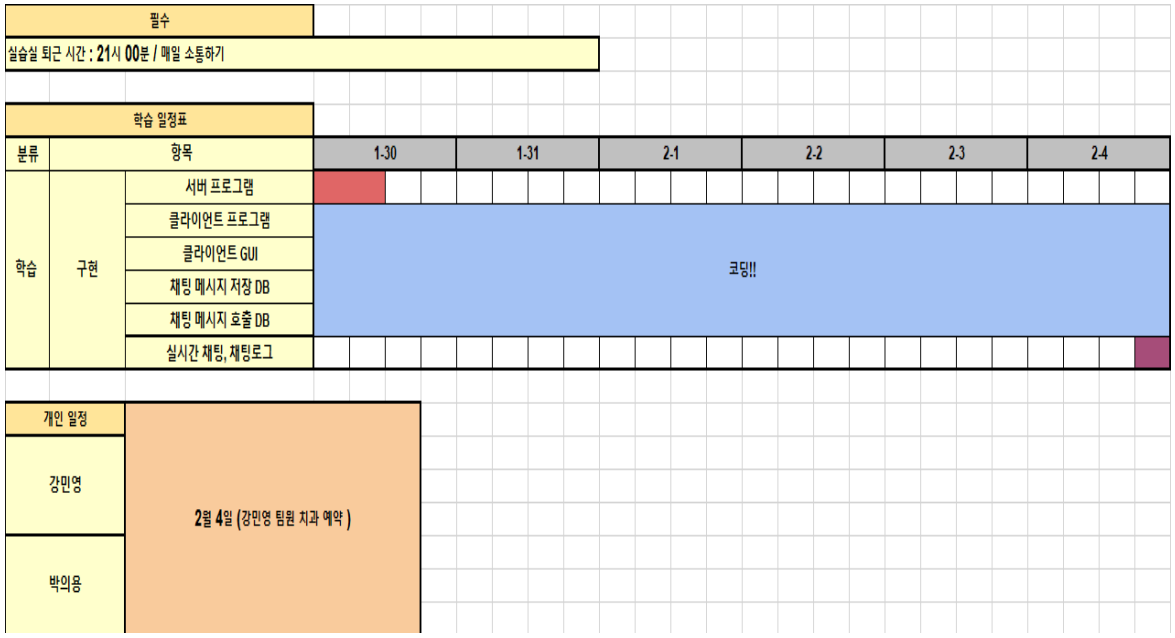
개발 완료 보고서

제출일 : 2023년 02월 04일

<table>
<tr><td>팀명</td><td colspan="3">코드제로</td></tr>
<tr><td>참여인원</td><td colspan="3">강민영, 박의용</td></tr>
<tr><td colspan="4">프로젝트 소개 (8번째 팀 프로젝트)</td></tr>
<tr><td>프로젝트 명</td><td colspan="3">네트워크 프로젝트 (다중 접속 서버 프로그램 구현)</td></tr>
<tr><td>활동일시</td><td>01 / 30(월) ~ 02 / 04(토) (21:00)</td><td>장소</td><td>광주인력개발원 공학 1관
드론융합실</td></tr>
<tr><td>주요 주제</td><td colspan="3">다중 접속 서버 프로그램 구현하기</td></tr>
<tr><td>개발 목적</td><td colspan="3">● TCP/IP 를 활용하여 다중 접속 서버 프로그램을 구현한다.
● Pyqt5 (GUI) 와 Python, MySQL 을 연결하여 프로그램을 구현한다.
● Python Socket으로 실시간 채팅을 구현한다.
● 데이터베이스(DB)를 사용하는데 익숙 해 진다.</td></tr>
<tr><td rowspan="4">개인
임무 분담</td><td rowspan="2">강민영</td><td colspan="2">클라이언트(채팅) 구현</td></tr>
<tr><td colspan="2">클라이언트 DB 구현
(채팅방 대화내용 저장, 지난 채팅방 내용 불러오기)</td></tr>
<tr><td rowspan="2">박의용</td><td colspan="2">3인이상 참여 서버 구현</td></tr>
<tr><td colspan="2">GUI 연결 구현 (채팅방)</td></tr>
<tr><td>개발환경</td><td colspan="3">vscode / python / mySQL / Github
/ window OS / PYQT5 /드론융합실 / TCP.IP</td></tr>
</table>

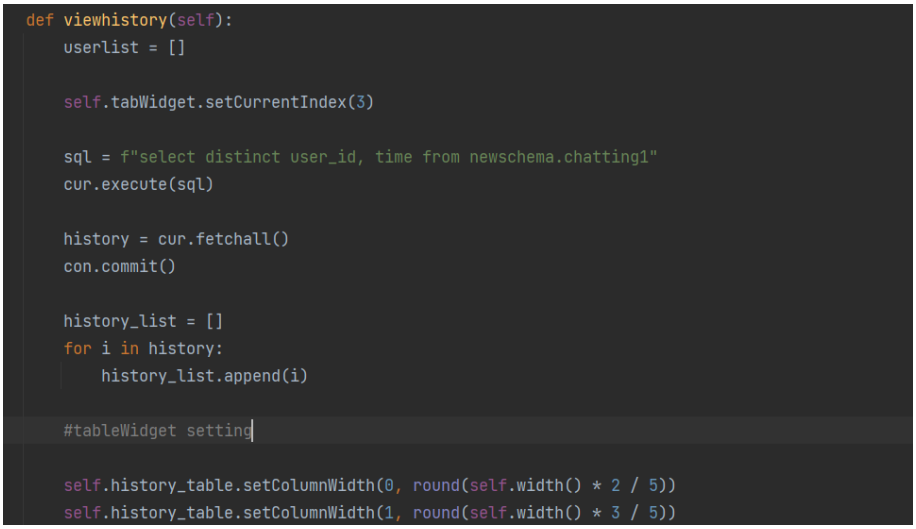
일정표



요구분석서

3팀 코드제로(강민영,박의용) 요구사항 분석서

번호	유형	요약	요구사항	요구분석 내용
1	서버 프로그램	3인 이상이 참여하는 멀티 통신 GUI(Pyqt5) 구현 <TCP/IP Python socket 사용 >	1. 멀티통신 서버 GUI 2. 채팅방 만들기	1. 서버는 연결 클라이언트를 저장 2. 서버는 연결을 수신하고 메시지 요청을 처리 3. 메시지는 서버로 먼저 전달 후 다른 사용자에게 전달 4. 클라이언트에게 응답을 요청하고 반환 5. TCP/IP , PYTHON SOCKET 으로 구현
2	클라이언트 프로그램	서버에 연결 및 메세지 전송, 수신 , MySQL을 이용하여 채팅방 내용 저장 및 불러오기 <TCP/IP Python socket 사용 >	1. 채팅방 대화내용 DB 저장 2. 지난 채팅방 최근 내용 불러오기	1. 서버에 연결 (이름, 주소, 포트) 2. 서버에 메세지 전송, 새로운 메시지 수신 3. 채팅방 대화 내용을 DB에 저장 4. 채팅방 지난 대화 내용을 DB에서 호출 5. TCP/IP, PYTHON SOCKET으로 구현
3	추가 기능	멀티 채팅방을 활용한 게임 만들기	1. 초대기능 구현 (수락 시 게임 진행)	1. 멀티 채팅방을 활용한 게임 만들기 2. 초대 기능 구현(수락 시 게임 진행) 3. 스코어(점수) 와 승부 결과를 DB에 저장 4. 특정 단어 맞추기 (영화, 만화, 고양이 이름등)

코드설명	<강민영> <server.py>  클라이언트 소켓에서 받은 메시지를 서버에서 받아서, DB에 저장하는 역할을 한다. <client.py>  사용자가 자신이 대화한 내역을 날짜별로 표시해준다. 서버에서 생성한 DB를 Distinct 로 날짜 중복을 제거하여 진행한다.

```
if msg_length:
    msg_length = int(msg_length)  # 값을 인트형 저장
    # 메세지 수신 및 디코딩
    msg = self.conn.recv(msg_length).decode(FORMAT)
    print(msg[1:], 'coconut')
    self.dbmsg = msg[1:]
    dbinput = f"insert into newschema.chatting1(user_id, message, ip_address, port_number, time) values"\
              f"('{self.username}', '{self.dbmsg}', '{self.addr[0]}', '{self.addr[1]}', '{date_}')"
    # idon = f"select * from newschema.chatting1"
    cur.execute(dbinput)

    # avg = cur.fetchall()
    con.commit()

    # 메세지 처리를 위한 함수 호출
    self.handleMsg(msg)
```

```
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()

    history_list = []
    for i in history:
        history_list.append(i)

    #tableWidget setting

    self.history_table.setColumnWidth(0, round(self.width() * 2 / 5))
    self.history_table.setColumnWidth(1, round(self.width() * 3 / 5))
```

```
def history_show(self, row, col):
    data = self.history_table.item(row, col)
    print("셀 클릭 셀 값 : ", data.text())
    print(data.text())
    # sql1 = f"update restaurant.inquiries set reply = '{reply1}' where customer_id = '{self.clicked_inquiry}' or in
    sql = f"select user_id, message, time from newschema.chatting1 where time = '{data.text()}' or user_id = '{data.
    # sql = f"select * from newschema.chatting1 where time like 'str(2023)'"

    cur.execute(sql)

    message = cur.fetchall()
    print(message)
    con.commit()

    message_list = []
    for i in message:
        message_list.append(i)

    self.message_table.setRowCount(len(message))  # 테이블의 행 갯수를 rows의 길이로 정함

    self.message_table.setColumnCount(len(message[0]))

    for i in range(len(message_list)):
        for j in range(len(message_list[i])):
            self.message_table.setItem(i, j, QTableWidgetItem(str(message_list[i][j])))
```

사용자가 클릭한 날짜의 대화내역을 출력한다.
출력 내용은 사용자 아이디, 메세지, 날짜 순으로 출력된다.

```
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)

    self.msg_send_btn.clicked.connect(self.newmsg)
    self.chat_tableWidget.cellDoubleClicked.connect(self.friendinvite)
```

UI 에서 서버 연결 클래스를 실행시키는 중요한 부분의 코드이다.
사용자 정의 함수 실행을 위해 Mysignal() 을 실행시킨다.

```python
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

        message, send_length = encodeMsg(self.username)
        self.client.send(send_length)  # 메세지 크기
        self.client.send(message)  # 메세지

        # 메세지를 받을 쓰레드 생성
        self.thread_recv = threading.Thread(target=self.recvMsg, args=())
        self.thread_recv.start()
```

클라이언트의 소켓을 생성하고, 서버와 연결을 한 후, 메세지 수신을 위한 쓰레드를 생성하여 메세지 수신 함수를 실행시킨다.

```
def sendMsg(self, msg, re):

    if (self.online):  # 사용자가 온라인 상태인 경우
        try:
            msg = re + msg  # 메세지에 태그 추가
            message, send_length = encodeMsg(msg)
            self.client.send(send_length)
            self.client.send(message)


        except:  # 통신이 실패할 경우
            self.disconnect()  # 연결 해제
```

사용자가 메세지 보내기 버튼을 클릭할 때마다 실행시켜서 보낼 메세지를 인코딩 하고 보낸다.

<박의용>

```
HEADER = 64 # 기본 메세지 크기 (바이트)
FORMAT = "utf-8" # 인코딩 형식

SERVER=socket.gethostbyname(socket.gethostname())
# IP 주소 (로컬)
PORT = 9058 # 통신용 포트
ADDR = (SERVER, PORT)
```

- 메세지 크기 , 인코딩 형식, ip, port 정의

```
class Server():

    # 서버 초기화 및 종료 기능
```

```
    def __init__(self):
        # 연결 유형 정리
   self.server   =   socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

        # 정의된 서버 및 포트에 대한 바인딩 연결
        self.server.bind(ADDR)
```

- 서버 클래스 시작, 소켓 객체 생성 및 주소 연결

```
   def receiveUser(self):
```

- 유저 받아들이는 함수

```
def cancleConnection(self, user):
```

- 유저 연결 취소 함수

```
def serverMsg(self, msg):
```

- 서버 메시지 처리 함수

```
def userMsg(self, msg, user):
```

- 연결된 사용자에게 메시지를 전달하는 함수

```
 def userListUpdate(self):
```

- 연결된 유저리스트를 보내는 함수

```
def closeServer(self):
```

- 서버 종료 함수

```
def date():
```

- 날짜 반환 함수

```
def encodeMsg(msg):
```

- 메시지 인코드 후 반환 함수

```
class User():

    # 서버에서 유저 인스턴스 초기화
    def __init__(self, server, username, conn,
addr):
```

- 유저 이름, 소켓 받아서 쓰레드 시작하는 클래스

```
    def process(self):
```

- 유저 메시지 받는 함수

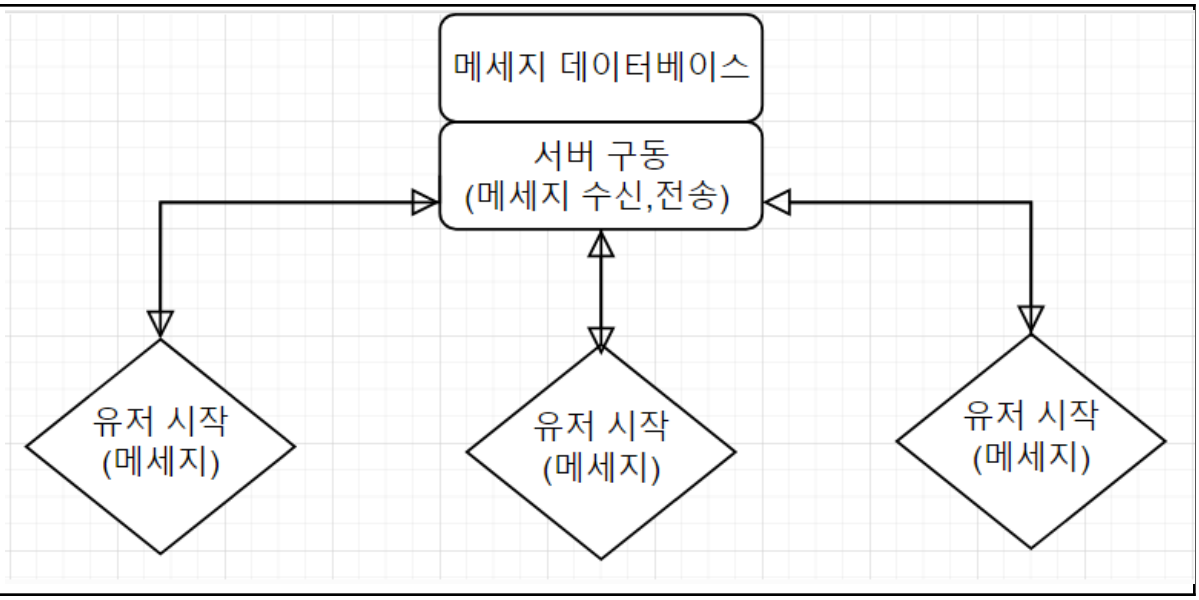
```
    def handleMsg(self, msg):
```

- 유저 메시지 처리 함수

```
if __name__ == "__main__":
    s = Server()
```

- 위 전체 클래스를 실행하는 구문

개인후기	<강민영> 이번 프로젝트를 진행하면서, 클래스에 대한 이해가 좀 더 잘 되는 듯한 느낌이 들었다. 또한 서버와 클라이언트 간의 상호작용과, TCP/IP 를 구성하는 필수요소에 대해서 알아가는 과정을 거쳤다. 초반에 select 를 사용하여 ui가 없이 작동되는 채팅 프로그램을 기반으로 대화내용을 db에 저장하여 불러오는 기능을 구현해 보았고, 그 후, 팀장인 의용님이 Thread를 사용하여 구현한 틀로 정했기 때문에, 새로운 코드에 같은 기능을 적용하여 구현하였다. 그러한 과정에서 낯설기만 했던 네트워크에 대해 친숙해 지는 시간이 되었다고 생각한다. <박의용> TCP/IP 에 대한 전반적인 이해를 하려고 노력하였다. 하지만 실제 코드를 작성하고, 프로그래밍화 시킨다는 것은 매우 어려웠다. 요구분석 또한 TCP/IP를 정확히 이해하지 못한 상태에서 진행 하다 보니 놓친 부분이 많았다. 일례로, 프로젝트 마감 날, 데이터에 관한 부분 (UI를 사용하는 사용자 입장에서 데이터를 다루었다.) 주 기능인 메시지는 서버측에서 관리하는 구조로 되었지만, 사용자 기능에 많은 것을 추가해버렸고, 결국 서버에서 데이터를 다루어야 하는데, 이 부분을 놓쳤다. 프로젝트 마감 기한까지 꼭 하고 싶었지만, 현재의 나로써는 많이 부족하여 문제점을 해결 할 수 없었다. 다음 프로젝트를 진행 할 시, 서버측에서 데이터를 다루고, 그 데이터를 다시 유저로 보내주어서 UI창에 나타나는 결과가 나와야겠다고 생각했다.

구조도	 메세지 데이터베이스 서버 구동 (메세지 수신,전송) 유저 시작 (메세지) 유저 시작 (메세지) 유저 시작 (메세지)