**2018학년도 1학기**

**분산 시스템 및 컴퓨팅**

**- 팀 프로젝트 최종 보고서 -**

****

|  |  |  |
| --- | --- | --- |
| 과목 | 네트워크 프로그래밍 | |
| 담당교수 | 김기천/진정하 교수님 | |
| 학과 | 컴퓨터공학과 | |
| 학번 | 201311292, 201311259 | |
| 이름 | 유효상, 권오승 | |
| 제출일 | 2017.12.16 | |
|  | **프로젝트 소개** | |

|  |
| --- |
| 1. 프로젝트 목표 및 결과 |

* 최종 목표
* 프로젝트 이름
* Straw : 빨대 같은 스트리밍 서비스
* 프로젝트 구현 목표
* 서버는 클라이언트(사용자)에게 끊김 없는 음악 스트리밍 서비스를 제공해주기 위해 실시간으로 클라이언트의 밴드위쓰를 검사하여 음악이 끊기지 않는 선에서 최적의 음질을 제공해준다. 또한 프로젝트에 분산시스템을 접목시켜 서버를 마스터와 슬레이브로 나누어 음원을 여러 음질로 인코딩 하는 일을 슬레이브들에게 맡긴다. 클라이언트는 모든 슬레이브과 연결할 필요없이 슬레이브들과 클라이언트 사이에 브로커 서버를 하나 두어 브로커와 연결만 하면 서비스를 제공 받을 수 있게 한다.
* 프로젝트 결과
* 구현 결과
* 클라이언트(사용자)는 GUI에 재생버튼 클릭을 통해 손 쉽게 음악을 재생할 수 있다.
* 서버(마스터)는 성공적으로 소켓 통신을 통해 서버(슬레이브)들에게 원본 음원을 전송하고 각 서버(슬레이브)들은 각자 다른 음질로 받은 음원을 성공적으로 인코딩 한다.
* 클라이언트는 자신이 필요한 음원의 이름, 음원의 시퀀스 넘버, 음질 정보를 브로커에게 전송하고 해당 음원을 받을 때까지 기다린다.
* 서버(슬레이브)들은 인코딩한 음원을 작은 데이터 그램으로 쪼개어 저장해 두고 브로커에게 요청이 오면 필요한 음원의 음질과 시퀀스넘버를 확인하여 전송해 준다.
* 브로커는 클라이언트에게 요청받은 해당 음원 파일(음질, 시퀀스넘버)을 서버(슬레이브)에게 받아서 그대로 클라이언트에게 전송해 준다.
* 위 모든 작업이 순식간에 진행되므로 클라이언트(사용자)는 끊김없이 음원을 재생할 수 있다.

|  |
| --- |
| 2. 프로젝트 수행 내역 |

* 프로젝트 진행 일정
* 프로젝트를 진행하는 중간에 기말고사도 있고 팀원 중 한명이 다른 팀 프로젝트가 2개 더 있어 지연이 있었지만 프로젝트를 완성하는데 무리없이 진행 되었다.

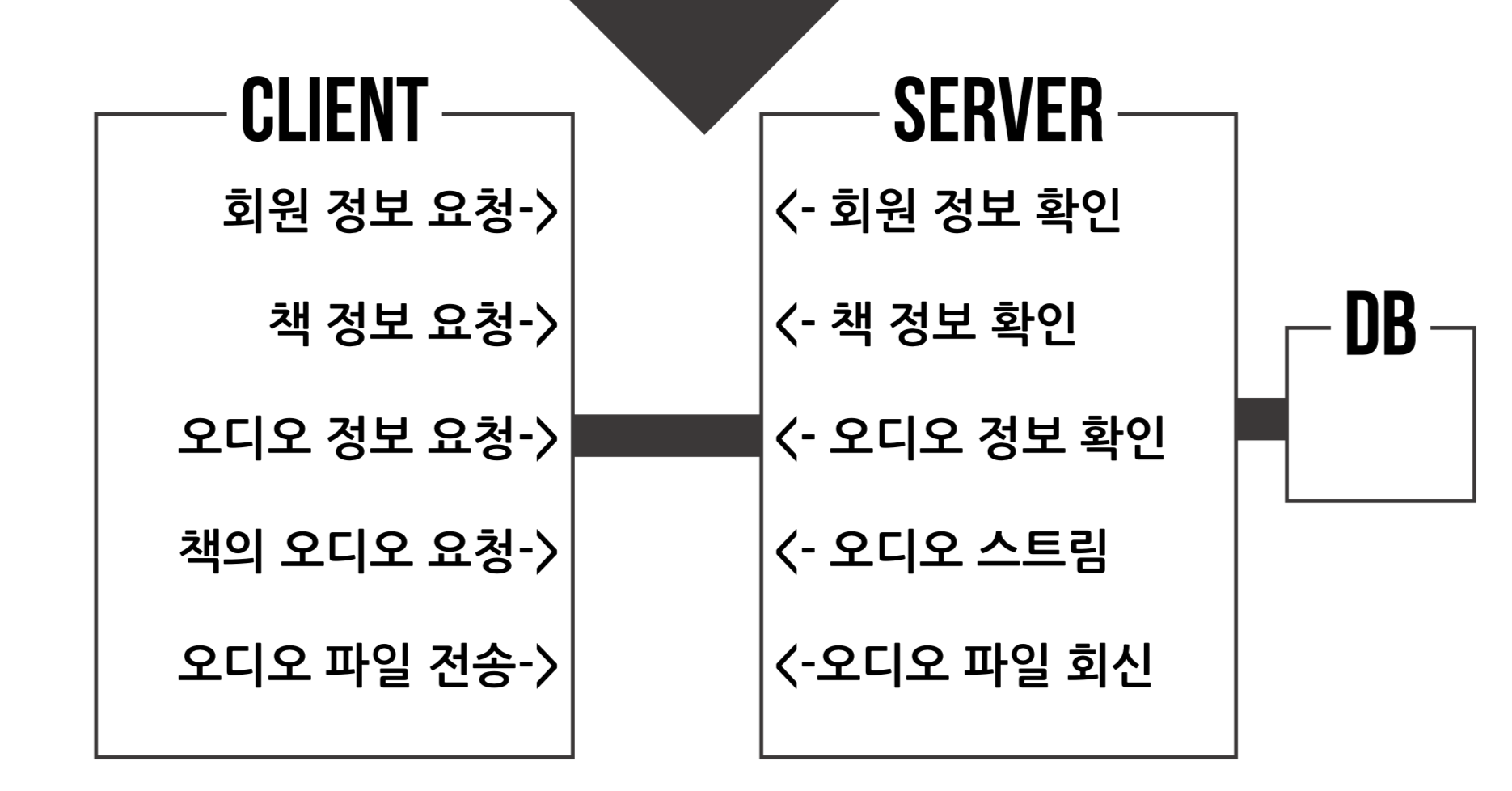
.

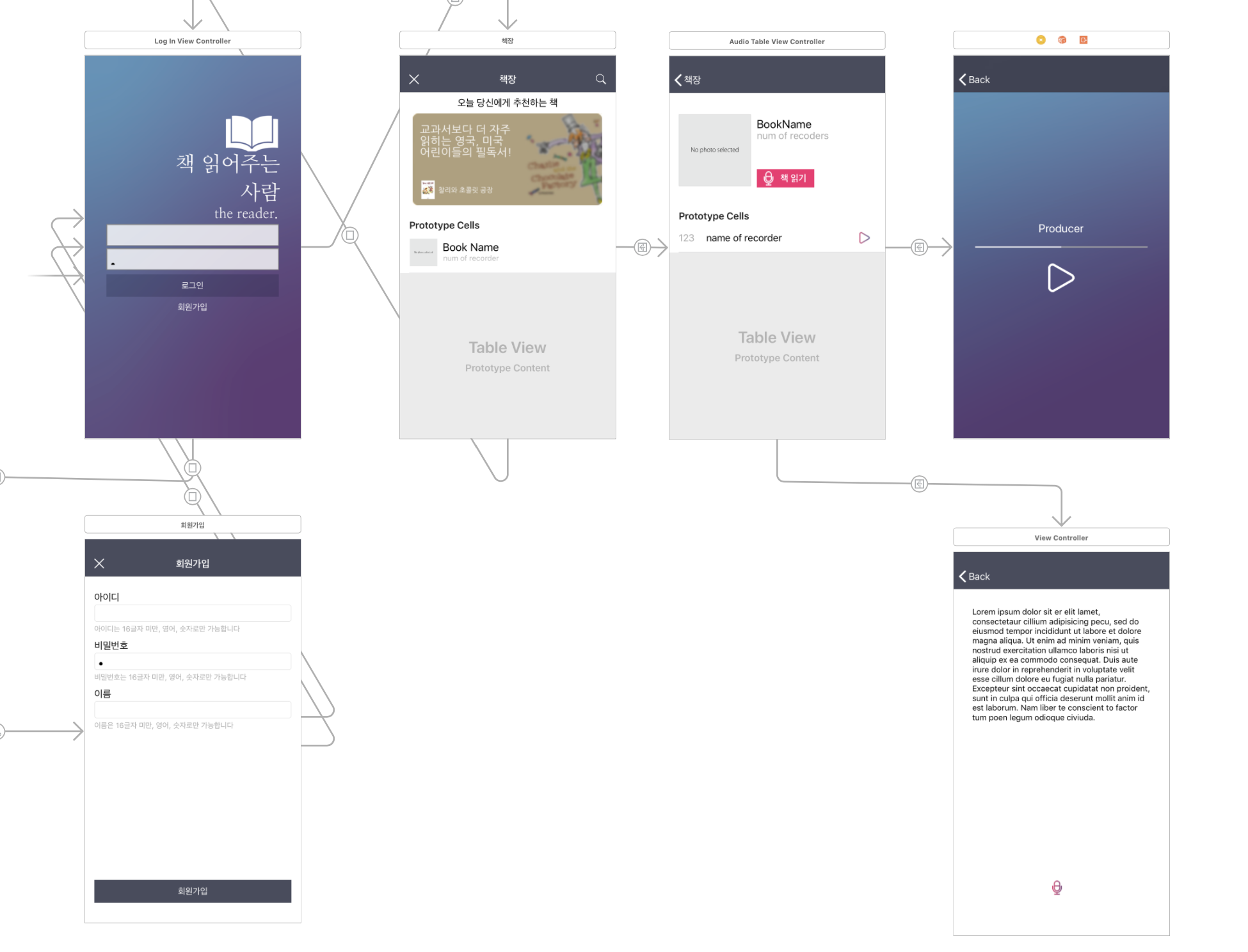
* 상세 진행 일정
* 1 주차 : 음원 인코딩 구현
* 2 주차 : 기본 설계, 음원 세그먼트화 및 매끄럽게 재생하기
* 3 주차 : 기본 설계, 네트워크 프로그래밍(서버, 클라이언트 생성)
* 4 주차 : 분할 실시간 음원 재생, GUI
* 5 주차 : GUI, 분산 시스템 적용(마스터, 슬레이브 구조 생성)
* 6 주차 : 분산 시스템 적용(브로커 서버 추가)
* 프로젝트 개발 환경
* 클라이언트 개발 환경
* 클라이언트 개발 언어 : JAVA
* 개발 언어 용 사용 IDE : intelliJ
* 클라이언트 사용 OS : Mac OSX
* 서버(마스터) 개발 환경
* 서버 개발 언어 : Java
* 개발 언어 용 사용IDE : intelliJ
* 서버(마스터) 사용 OS : Mac OSX
* 서버(슬레이브) 개발 환경
* 서버 개발 언어 : Java
* 개발 언어 용 사용IDE : 메모장
* 서버(슬레이브) 사용 OS : Debian 9
* 서버(브로커) 개발 환경
* 서버 개발 언어 : Java
* 개발 언어 용 사용IDE : intelliJ
* 서버(브로커) 사용 OS : Mac OSX
* 공통 사항
* 개발 환경 및 테스트 실행 환경 : Mac OSX, Debian 9
* 프로젝트 수행 방법
* 클라이언트 수행방법
* GUI를 만들기 전에 서버에서 전송하는 세그먼트화 된 음원을 실시간으로 연속 재생하여 자연스럽게 스트리밍 되는지 확인했다.
* WAV파일과 MP3파일 모두 재생이 가능하나 WAV파일 같은 경우 세그먼트화가 힘들어 MP3파일 용 플레이어를 사용했다.
* 브로커에게 현재 필요한 음원의 음질과 시퀀스 넘버를 순차적으로 요청하고 요청한 파일을 순차적으로 받아 재생할 수 있다.
* GUI를 만들고 음원을 요청하는 동시에 재생하는 쓰레드를 GUI에 넣었다.
* 서버(마스터) 수행방법
* 처음 구현 당시 서버와 클라이언트가 단순히 음원 파일을 세그먼트화하여 전송하고 클라이언트가 재생하는 형식으로 구현했다.
* 서버 DB의 경우 DB서버를 따로 두려고 했으나, 프로젝트의 DB규모가 크지 않아 서버가 로컬에 폴더를 두어 음원파일을 가져오도록 하였다.
* 서버를 마스터와 슬레이브 구조로 나누어 서버에서 음질 별로 인코딩하는 부분을 분산시스템화 하여 수행하였다.
* 마스터에서는 클라이언트에게 요청받은 파일을 슬레이브들에게 전송만하면 된다.
* 서버(슬레이브) 수행방법
* 마스터에게 받은 음원을 각자의 슬레이브가 서로 다른 음질로 인코딩을 하도록 하였다.
* 인코딩이 완료 된 후에는 인코딩된 음원을 세그먼트화하여 시퀀스넘버 별로 파일이름을 수정하여 폴더에 저장하도록 하였다.
* 브로커에게 요청이 오면 해당 음질의 시퀀스 넘버 음원파일을 브로커에게 전송하도록 하였다.
* 서버(브로커)수행방법
* 브로커는 클라이언트에게 먼저 파일이름을 받아 서버에 현재 파일이 있는지부터 검사하고 있으면 파일전송 메소드를 실행한다.
* 클라이언트에게 받은 음질 정보과 시퀀스번호를 가지고 슬레이브에게 요청하여 해당 파일을 받아오도록 한다.
* 슬레이브에게 받아온 파일을 저장하지 않고 그대로 클라이언트(사용자)에게 전송하므로써 브로커의 무게를 줄였다.
* 중간발표 후 교수님의 제안으로 슬레이브와 클라이언트 사이에 브로커를 둠으로써 서버(마스터, 슬레이브)를 숨길 수 있고 클라이언트들의 관리가 훨씬 수월해 졌다.

|  |  |
| --- | --- |
|  | **프로젝트 산출물** |

|  |
| --- |
| 3. 알고리즘 및 순서도 |

* 핵심 알고리즘



* 기본적으로 CLIENT는 서비스 요청을 보내며, 서비스 요청은 서로 규약 된 message type으로 정의 된다. Message type에 따라 서버는 해당되는 서비스 를 제공하며, 자세한 정보는 서버에 저장되어 있는 데이터베이스에 저장된 정보를 제공 및 수정 한다.
* TCP 통신이 기반이기 때문에 서로에게 전달된 정보에 대한 ack message 가 오지 않는다면 각 서비스는 실행 되지 않는다.
* 순서도

**4**

**3**

**1**

**2**

프로그램의 큰 순서도는 4개로 나눌 수 있다.

* 1. 회원 가입을 통해 사용자의 정보를 서버에 전달, 서버는 ID중복 및 회원 가입 양식의 타당성을 검사한 후에 회원 가입의 진행의 성공 여부를 클라이언트에 전달한다.
* 2. 회원 가입이 성공적으로 이루어진 후에는 클라이언트는 로그인을 진행 하며, 로그인 정보를 서버에 전달, 서버는 사용자 정보를 확인 타당할 경우 확인 메시지를 보내 클라이언트를 다음 VIEW로 넘어 갈 수 있게 한다.
* 3. 로그인이 이루어진 후에는 서버는 클라이언트에게 DB에 저장되어있는 책의 정보를 불러와서 클라이언트에게 전달한다. 클라이언트는 책의 정보를 형식에 맞추어 책의 이름, 책커버 사진, 그리고 책에 담겨져 있는 오디오 파일 리스트를 정리하여 화면에 띄운다.
* 4. 이제 사용자는 사용자가 원하는 책의 녹음 혹은 텍스트 파일을 볼 수 있으며 또는 책의 원하는 오디오 파일을 선택하여 재생 시킬 수 있다.

|  |
| --- |
| 4. 소스코드 |

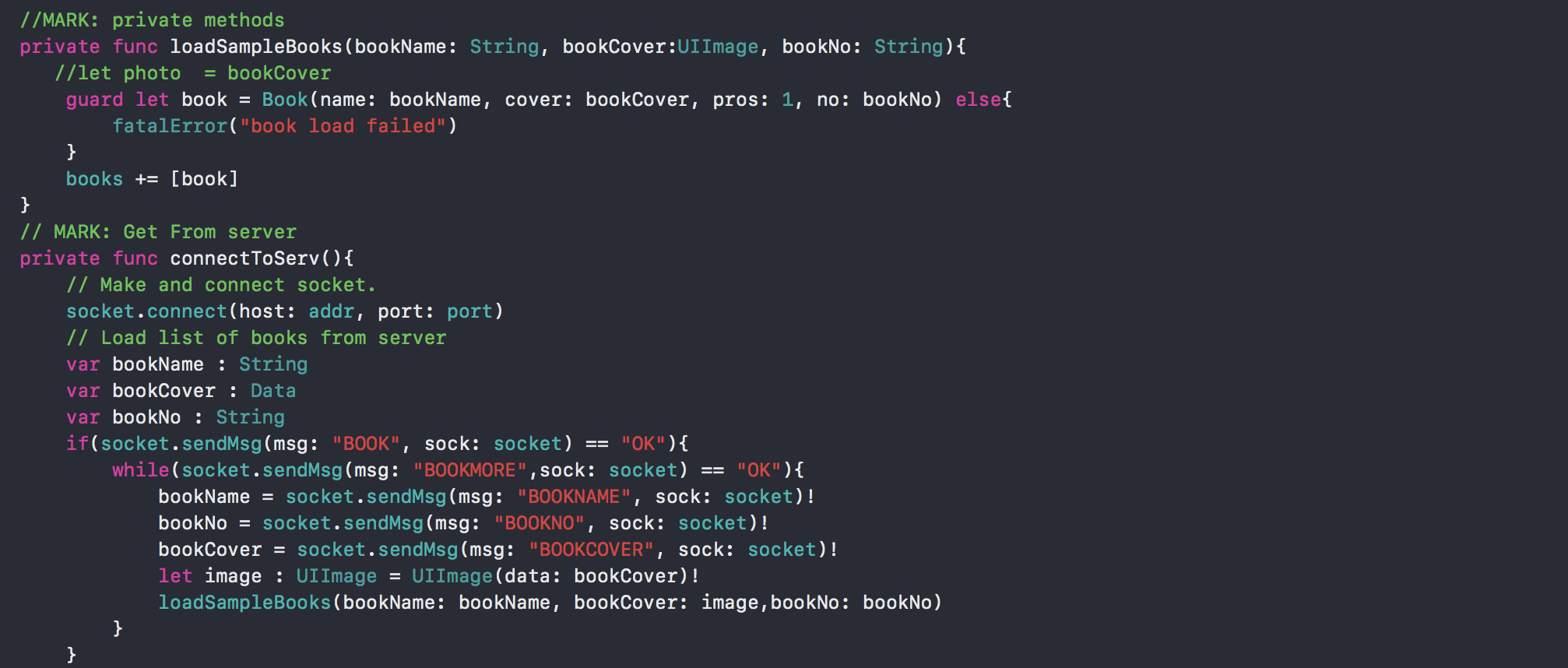
* 클라이언트 소스 코드
* 구현한 스트림 소켓



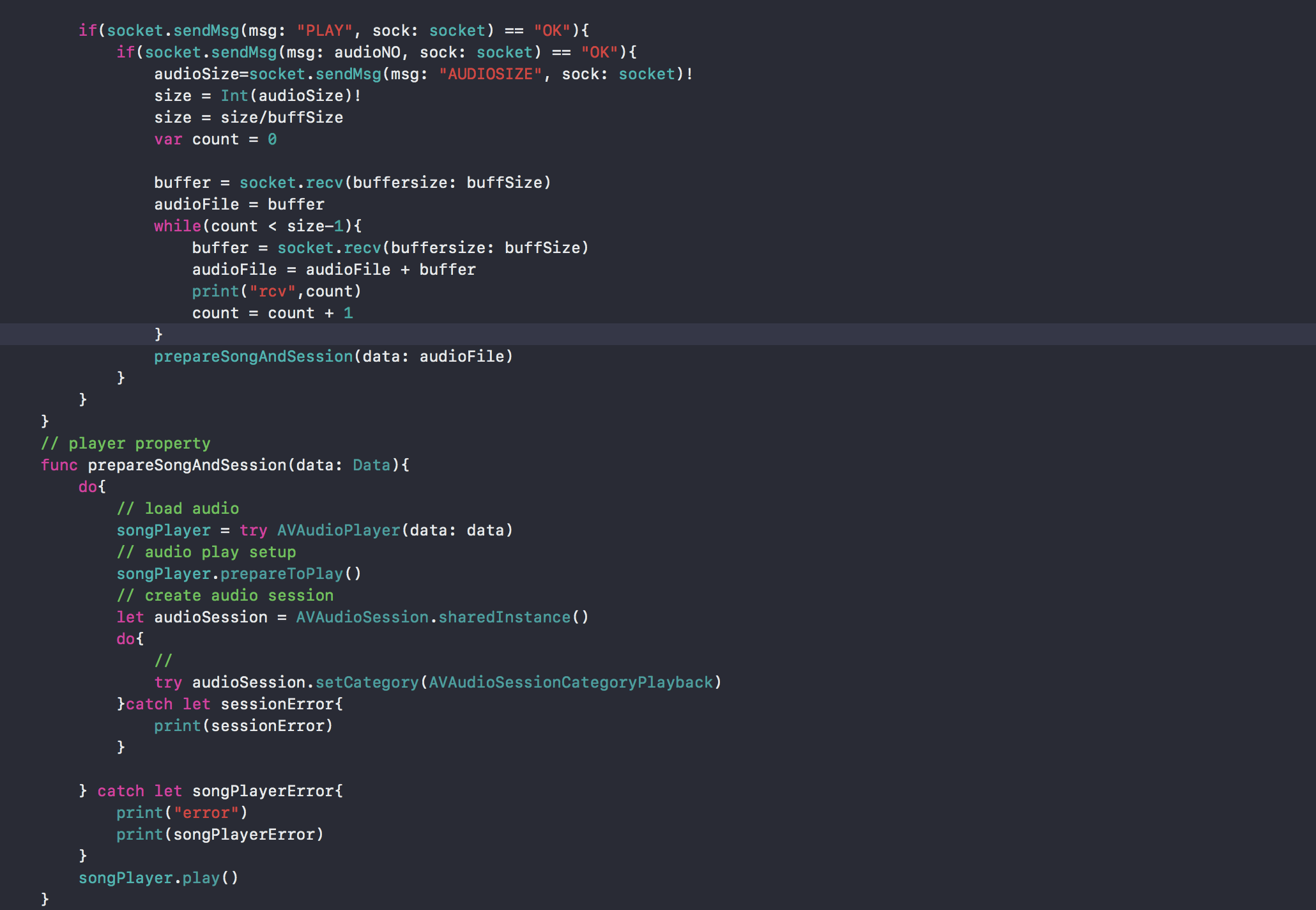
* 클라이언트내 소켓 연결 활용



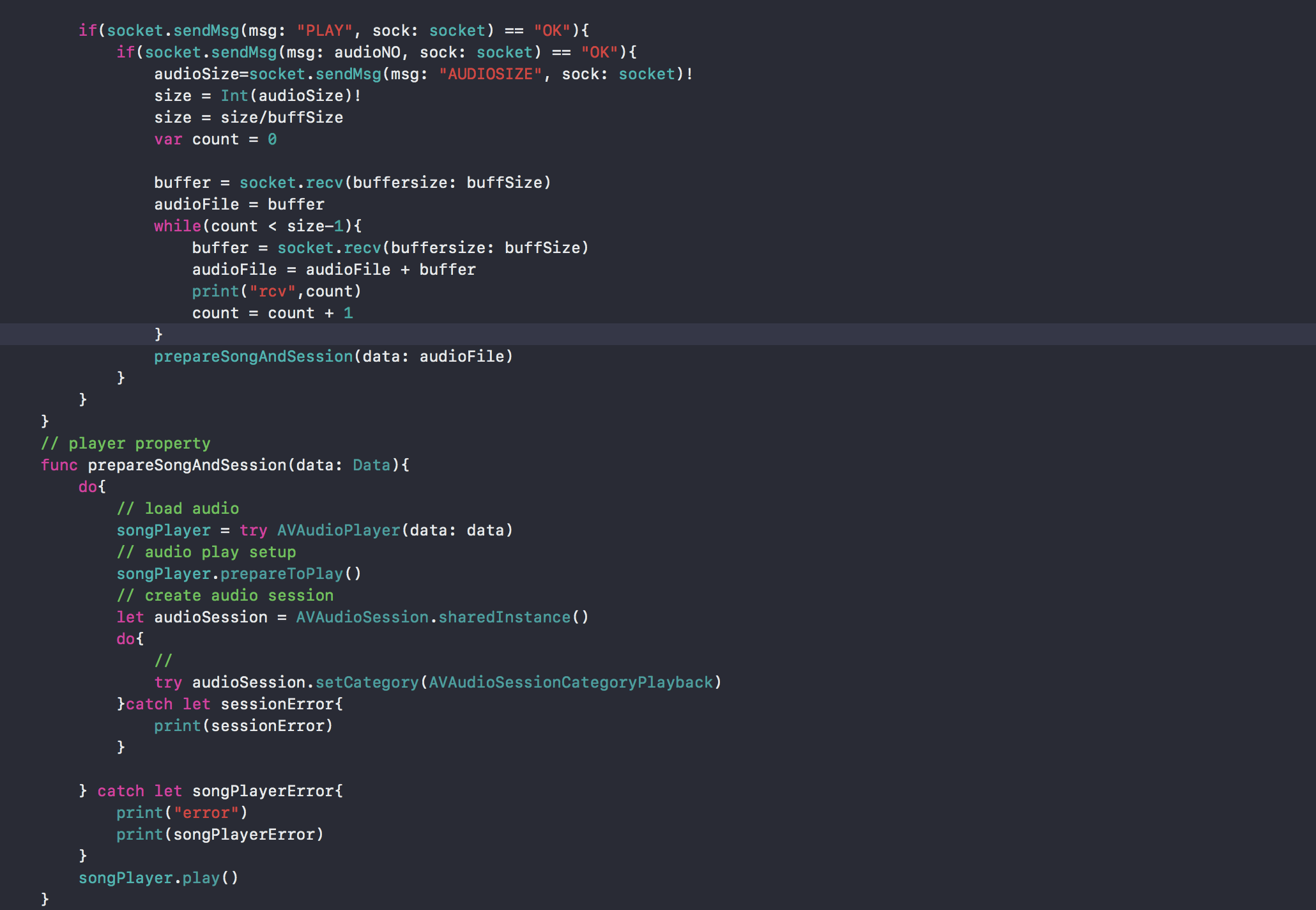
* 소켓 연결을 확인 하고 “LOGIN” message type을 서버에 보내고 “OK”이라는 ACK message를 확인하고 login process를 진행 한다.
* 책 목록 불러오기



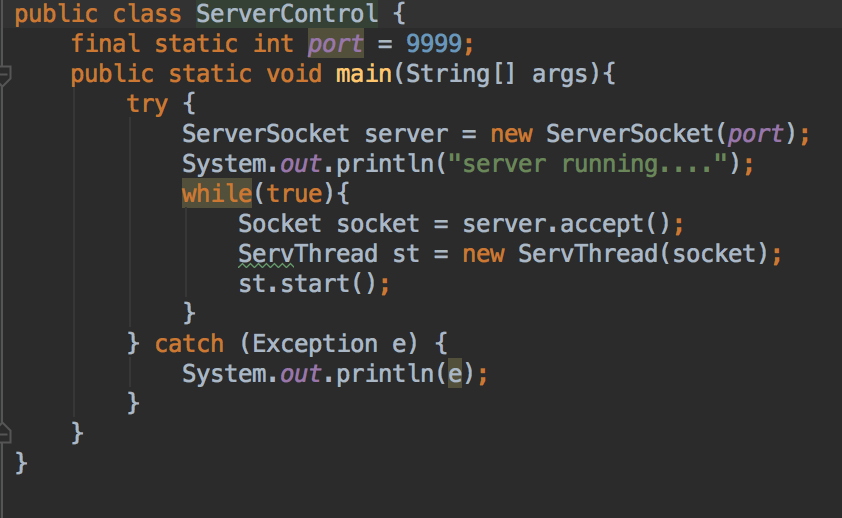
* 소켓을 통해 책 정보를 불러 오는데, 책 정보가 남아 있을 때까지 반복하여 정보를 불러오면, 받아온 정보를 loadSampleBoook()함수를 통해 Book의 객체를 만들어 준다.
* 오디오 파일 또한 같은 흐름으로 진행되기 때문에 보고서에는 따로 첨부 하지 않고 같이 첨부된 소스코드를 참고.
* 오디오 스트림



* 우선 서버에게 받을 총 데이터의 양을 전달 받은 후 미리 지정된 버퍼 크기 만큼의 잘라 반복하여 받을 수 있게 반복문을 통해 분할 시켜준다.
* 오디오 파일을 byte array의 형식으로 서버에게서 제공받는다.



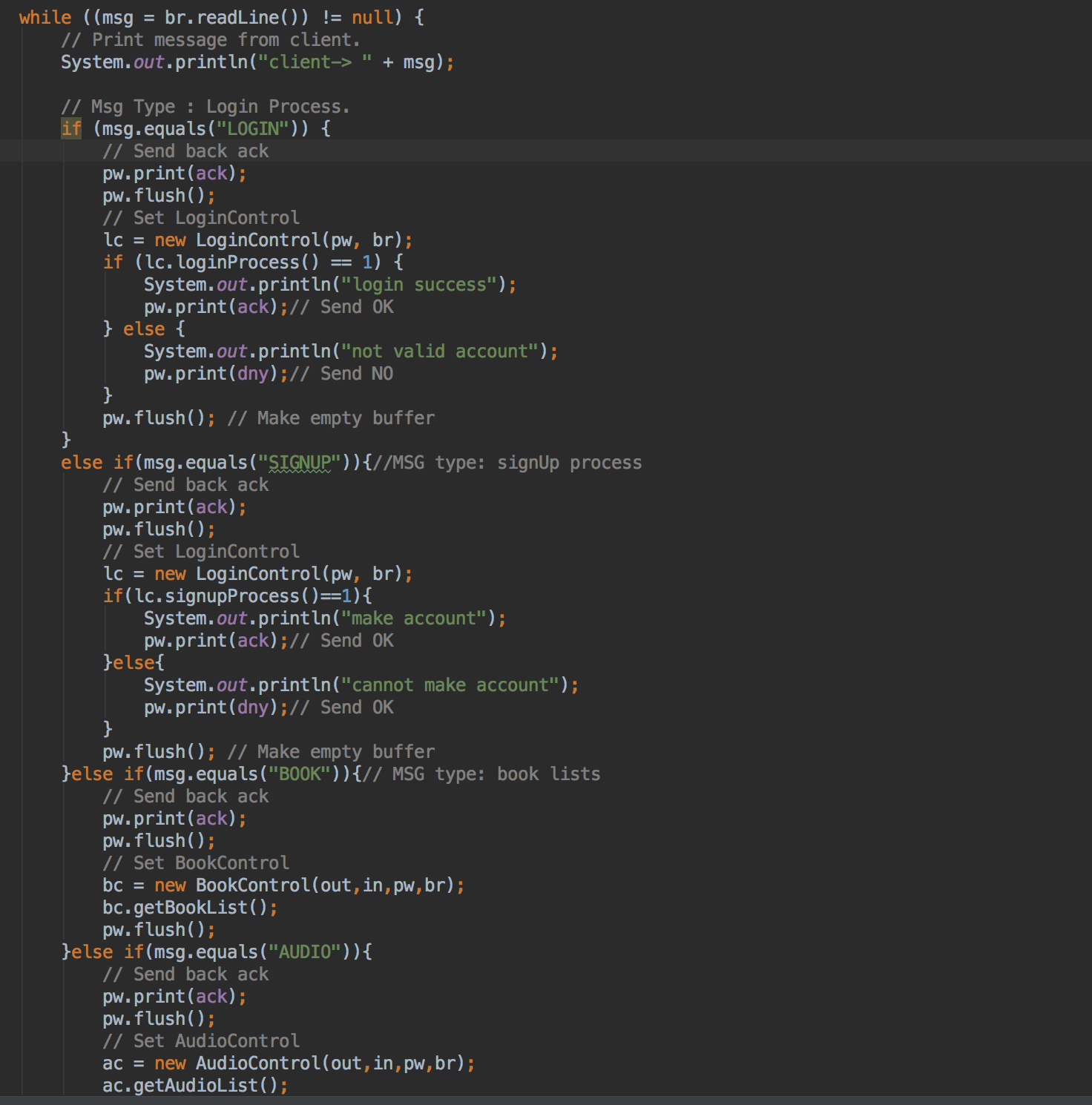
* 받은 오디오 파일을 AVAudioPlayet에 넣어주어 재생될 수 있게 만들어 준다.
* 서버 소스 코드
* 서버 메인 컨트롤



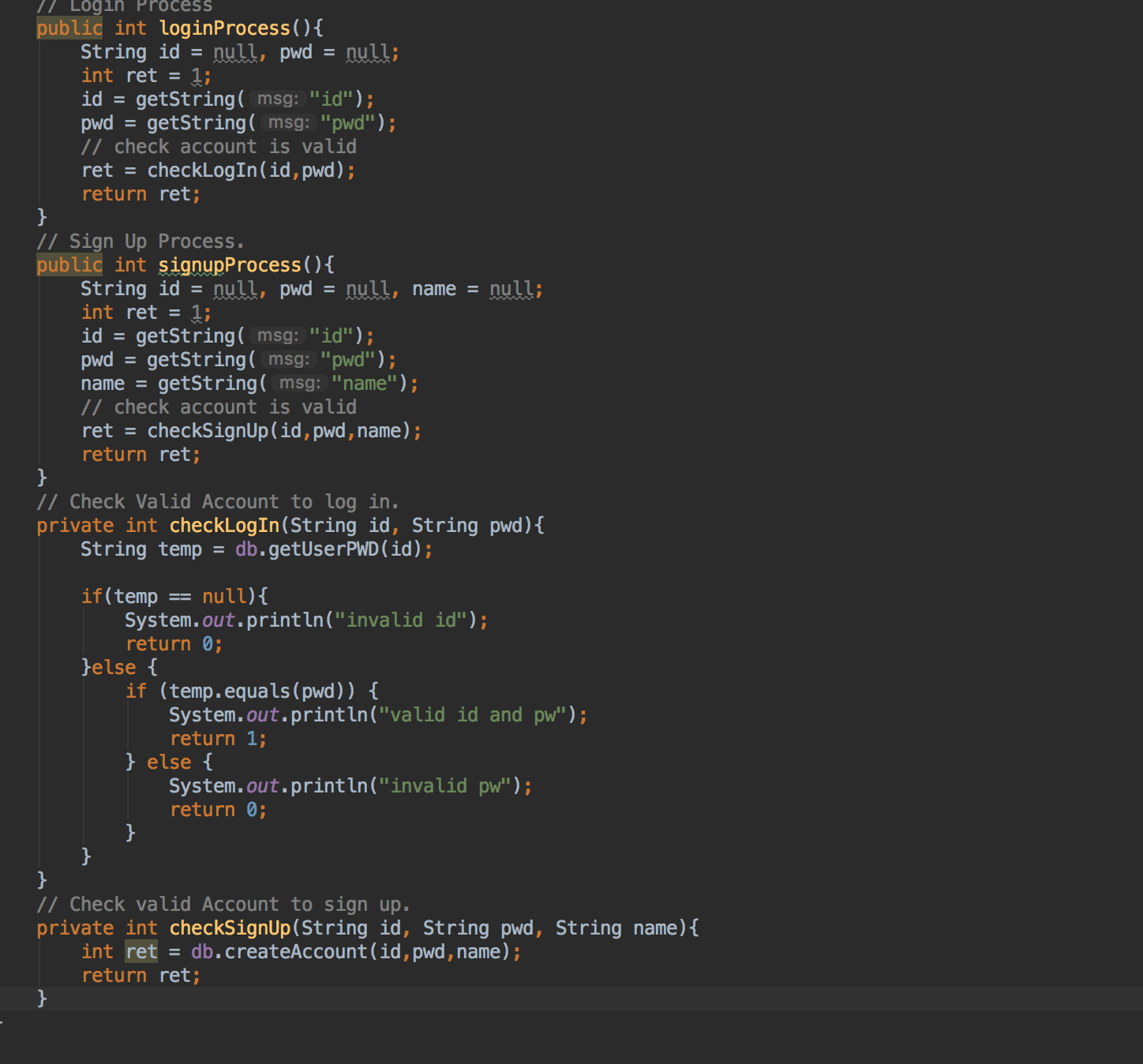
* 포트 번호를 지정해 주고 서버를 쓰레드화 시켜서 멀티 클라이언트를 지원 할 수 있게 만들었다.
* 서버 메인 쓰레드



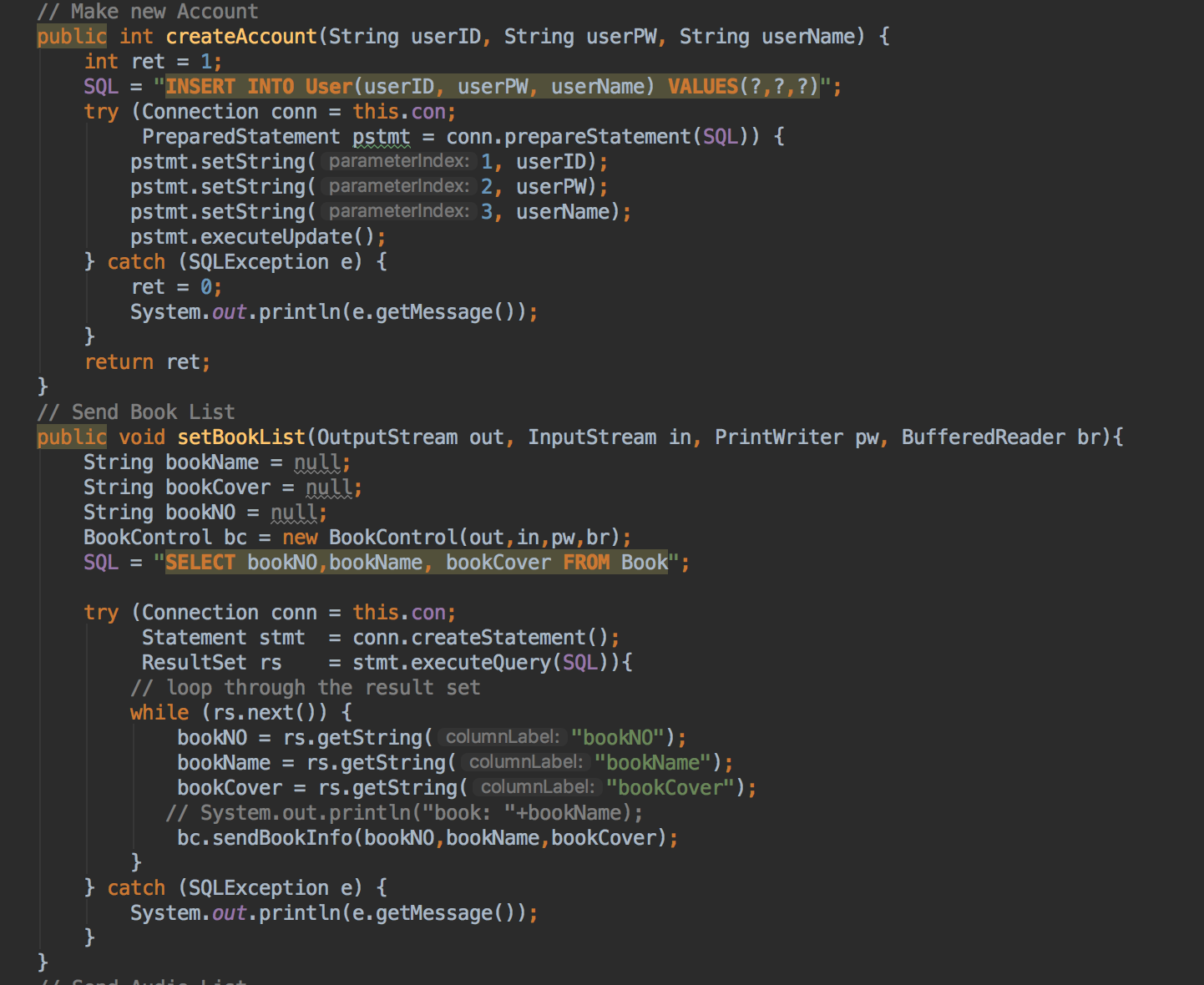
* 서버 또한 스트림 방식의 소켓 통신으로 운영된다.



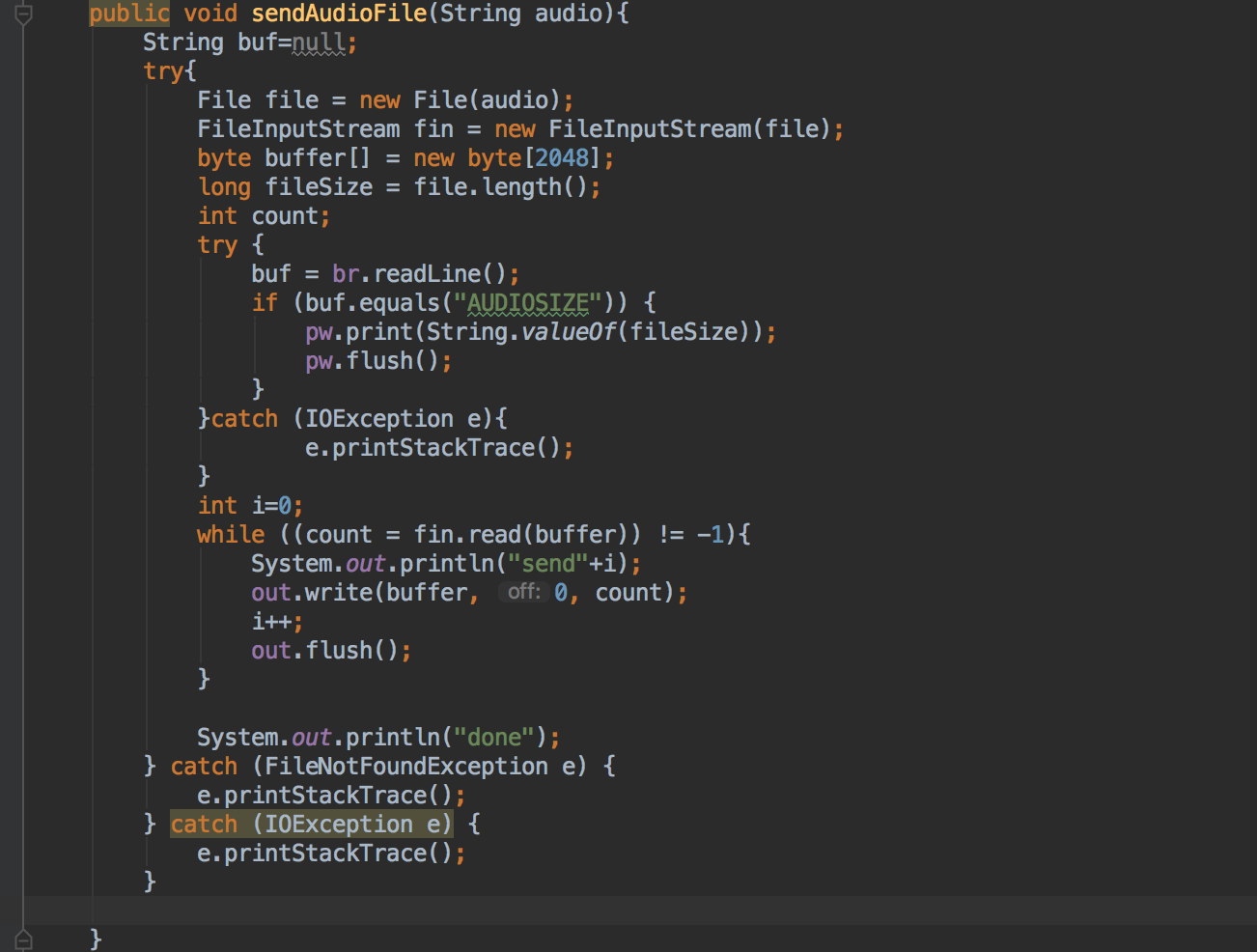
* 각 Message type에 맞추어 controller들을 생성하고, ack message 를 클라이언트에 보내준다.
* Login & sign up process를 통해 보는 데이터베이스 활용



* 서버에서는 기본적인 정보를 다루며 중복 error나 데이터의 타당성의 경우 데이터베이스의 SQL문을 통해 검사 한다.
* 데이터베이스 Controller



* 오디오 정보 전송



* 데이터베이스에서 오디오 파일을 불러온 후 클라이언트에 오디오 파일사이즈의 크기를 먼저 전송, 그 다음 지정된 버퍼 크기에 맞추어 클라이언트에 전송한다.

|  |  |
| --- | --- |
|  | **프로젝트 진행결과** |

|  |
| --- |
| 5. 프로젝트 진행에 따른 문제점 및 개선 방안 |

* 문제점
* 불안정한 네트워크
  + 현재 로컬에서 마스터, 슬레이브, 브로커를 모두 실행하면 문제없이 돌아가지만 마스터와 슬레이브, 브로커 모두 각자 다른 컴퓨터로 실행할 경우 파일을 전송하는 과정에서 패킷 손실이 일어났다.
  + 처음 프로젝트를 구현할 당시에는 단순히 서버가 클라이언트의 밴드위스가 낮으면 저용량(낮은 음질)의 파일을 전송하면 된다고 생각했는데 프로젝트를 진행하면서 마스터와 슬레이브 구조에서도 슬레이브의 밴드위스가 낮은 경우를 생각해야 한다는 것을 깨달았다.
* 밴드위스 측정
* 클라이언트의 밴드위스를 측정하는 법을 아직 몰라서 현재는 실시간으로 커맨드를 입력받아 “h”, “m”, “l” 중 하나를 입력하면 고음질, 중음질, 저음질 중 하나로 요청할 파일의 음질 정보를 세팅해 주도록 하였다.
* 개선 방안
* 시뮬레이션
* 먼저 프로젝트를 구상하고 나서 여러 번 시뮬레이션을 해보았더라면 마스터와 슬레이브 구조 안에서의 밴드위스 문제를 좀 더 빨리 알아내고 설계 당시 고려하여 설계했을 것이다.
* 네트워크 패킷 제어
* 현재 각자 다른 컴퓨터에서 하면 패킷 데이터를 출력해본 결과 전송하는 패킷이 없어질 때가 있다. 패킷이 하나라도 유실되면 인코딩 과정에서부터 오류가 나기 때문에 패킷을 전송할 때 패킷에 헤더를 붙여 검사하고 유실된 패킷은 다시 전송하도록 해야 한다.