**클라이언트서버프로그래밍**

**중간 리포트**

|  |  |
| --- | --- |
| **학번** | **60201945** |
| **이름** | **성유진** |
| **제출일** | **2023-10-24** |

**목차**

|  |
| --- |
| 1. **분석 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**    1. **개요** 2. **설계 및 구현 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**    1. **프로그램 설계**       1. **전체 설계 구조**       2. **Client 설계 구조**       3. **Server 설계 구조**       4. **DBMS-DAO 설계 구조**    2. **주요 기능 메서드 및 로직 구현**       1. **<가입> 클라이언트 요청에 의한 토큰 발행**       2. **<로그인> 토큰 인증 로그인을 이용한 권한 관리**       3. **<정보보안> 비밀번호 암호화 작업**       4. **<학생/강의> 조회, 추가, 삭제**       5. **<수강신청> 유효성 확인, 조회, 등록, 취소**    3. **로깅**       1. **서버 로깅**       2. **토큰 로깅**    4. **예외 처리**       1. **NullDataException**       2. **DuplicateDataException**       3. **TokenDifferentException**       4. **TokenTimeOutException** 3. **Demo - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**    1. **프로그램 실행** 4. **결론 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -**    1. **느낀 점** |

**보고서 및 논문 윤리 서약**

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.

2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.

3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.

4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁 받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

예술, 그래픽, 폰트, 원이(가) 표시된 사진

자동 생성된 설명

**과목:** 클라이언트서버프로그래밍

**담당 교수:** 김정호 교수님

텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명**학번/성명**  60201945/성유진 (서명)

1. **분석**
   1. **개요**

사용자가 수강신청 서비스를 이용할 때 고려해야 하는 점은 무엇일까? 사용자는 서비스가 제공하는 인터페이스 또는 입력란에 자신이 적고 싶은 정보를 작성하면 될 뿐 고려해야 되는 것은 없다. 그렇다면, 수강신청 서비스에서 어떤 부분이 클라이언트고 또 어떤 부분이 서버일까, 사용자가 입력한 정보는 어디로 넘어가 저장되는 걸까?

물리적으로 분리되어 있는 클라이언트와 서버는 각각 어떤 역할을 담당하고, 어떤 방식으로 상호작용하는 지를 고민하는 것이 클라이언트서버 프로그래밍의 시작이다. 클라이언트와 서버의 관심사를 분리하기, 클라이언트서버 프로그래밍에 있어서 가장 중요한 개념 및 목적이다. 서버 각각의 관점으로 분석하고, 이를 기반으로 설계 및 구현하고자 한다. 리포트는 프로그램 설계 구조, 주요 기능 구현, 프로그램 실행 순서로 전개된다.

1. **설계 및 구현**
   1. **프로그램 설계**
      1. **전체 설계 구조**

전반적인 프로그램 구조는 [Client], [Server], [DBMS] 구조로 구분되며, 각 프로젝트에서 각각의 역할과 기능을 수행할 수 있도록 코드 및 관심사를 분리하여 설계하였다. 기존 데이터 관리에 사용하던 Data 파일은 더욱 안정적인 [Server]와 데이터의 보안을 위해 데이터베이스 연결로 변경하여 설계하였다. [Server]의 DAO클래스를 이용한 [DBMS] 호출 및 연결을 통해 데이터 관리에 용이 할 수 있도록 설계하였다. [Client]와 [Server]는 서버 인터페이스를 공유하고, RMI 통신을 통해 통신이 가능하다.

아래는 전반적인 프로그램 설계 구조이다.

텍스트, 폰트, 번호, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* + 1. **Client 설계 구조**

[Client]는 [서버]에 요청을 하는 역할로, [Client]에서는 [서버]에 요청을 한 뒤 반환 값을 받아 사용자에게 결과를 보여주는 기능을 수행하도록 설계하였다. [Client]를 설계하면서 가장 집중한 부분은 [Client]에서 ‘판단’을 하게해서는 안 된다는 점이었다. 그러므로, [Client]에서는 [Server]에게 직관적인 요청을 할 수 있고, [Server]로부터 오는 반환 값을 잘 받을 수 있도록 하였다. 더불어 사용자 친화적인 [Client]가 될 수 있도록 예외 메시지를 보여주고, 발행된 토큰이 사용자용 로그에 기록되는 과정을 통하여 사용자에게 온전한 정보 전달을 할 수 있도록 설계하였다.

아래는 [Client] 주요 클래스 및 인터페이스이다.

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

* + 1. **Server 설계 구조**

Client-Server프로그래밍에 있어서, [Client]가 사용자에게 결과를 보여줄 수 있도록 요청을 받아 온전한 반환 값을 보내는 비즈니스 로직이 필요한데, [Server]가 수강신청 시스템에서 학생, 수업, 수강과 관련된 비즈니스 로직을 수행할 수 있도록 설계하였다. [Server] 내 DAO 클래스 설계를 통해 데이터베이스 호출 및 연결 과정을 통해 다양한 데이터를 관리할 수 있고, 예외 처리 및 로깅을 통하여 [Client] 와의 통신 문제를 발견 및 해결할 수 있도록 하였다. [Client]의 요청에 따른 토큰을 발행하여 Client-Server 프로그래밍에서 보다 확실하고 안정성 있게 로그인 절차를 수행할 수 있도록 하였다. 더불어 사용자의 로그인 정보를 기반으로 암호화를 위한 Salt를 생성하고, 생성한 Salt를 Hashing 하여 저장하는 과정을 통해 [Server]의 보안성을 향상시킬 수 있도록 설계하였다.

아래는 [Server] 주요 클래스 및 인터페이스이다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

* + 1. **DBMS-DAO 설계 구조**

[Server]에서 [DBMS]의 호출 및 연결을 담당하여 데이터의 CRUD가 구현되도록 하는 건 [DAO]클래스이다. [DAO]에서는 [Server]가 비즈니스 로직 수행을 위하여 필요한 데이터를 생성, 조회, 수정, 삭제와 관련된 기능만 가능하도록 설계하였고, 그 이외의 메서드는 [Server]에서 구현될 수 있도록 분리하였다.

아래는 [Server]-[DAO]-[DBMS] 호출 및 연결 과정이다.

[Server]-[DAO]-[DBMS] 호출 및 연결 과정:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[DAO]-[DBMS] 연결 확인 (예시: courses table): 텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 블랙, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명

* 1. **주요 기능 메서드 및 로직 구현**

**2.2.1 <가입> 클라이언트 요청에 의한 토큰 발행**

Token은 [Client]와 [Server] 간 사용자 정보 확인의 통신을 보다 확실하고 안정적으로 하기 위하여 필요한 개념이다. 기존 Client-Server 프로그래밍 내 로그인 단계에서 [Client]에서 입력한 사용자ID 값을 [Server]에서 받아 학생 데이터와 비교하여 일치하면 로그인 성공으로 [Client]에 반환하였으나, 보안성 보장 불가 및 양방향 통신에서는 옳지 않은 방법이라는 생각이 들었다. 따라서 사용자 정보 확인에 있어 [Client]가 토큰 발행을 요청하고, [Server]는 사용자 정보 확인 로직 확인을 통해 토큰을 발행 및 반환하여 인증하는 방식으로 인증 토큰 로그인 기능으로 설계 및 구현하였다.

아래는 [Client]-[Server]-[DAO] 간의 토큰 요청 및 발행, 로그인 설계 로직이다.

텍스트, 스크린샷, 도표, 직사각형이(가) 표시된 사진

자동 생성된 설명

아래는 [Client] 요청에 의한 [Server]의 토큰 발행 흐름이다.

[Client] signForUser 메서드 => 로그인 전 사용자의 토큰을 발행하기 위하여 사용자 정보를 입력 받는 메서드이다. 사용자에게 정보를 입력 받은 뒤 [Server]에게 사용자 정보의 유효성 검사 및 토큰 발행을 요청한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[Server] signForUser 메서드 => [Client]에게 받은 사용자 정보 및 토큰 발행 요청을 받은 것을 기반으로 사용자 정보 검사 및 토큰 발행을 하는 메서드이다. 기존 학생 정보에 있는 사용자 정보 일치 시 [Client]에 발행된 토큰 값을 반환하고, 사용자 정보를 데이터베이스에 저장하기 위해 [DAO].addUser에도 입력 및 토큰 값을 전송한다.

텍스트, 스크린샷이(가) 표시된 사진

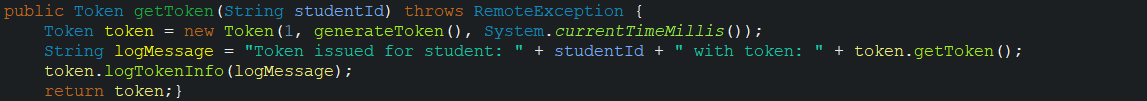
자동 생성된 설명

[Server] generateToken메서드 => [Server]로 토큰 발행 요청이 들어왔을 때, 요청 받은 토큰을 발행하기 위한 메서드이다. 알파벳 A 부터 숫자 9까지의 문자를 무작위로 섞어서 보안성 있는 토큰을 발행한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[Server] getToken 메서드 => generateToken 메서드에서 발행한 토큰을 받아 형식을 지정해 signForUser 메서드로 반환하는 메서드이다.



[Token] Token 클래스 => 토큰 값과 토큰 발행 시간을 저장하는 클래스이다. 인증 토큰의 경우 유효성 검사 시 발행 시간과 현재 시간을 비교하여 검사를 하기에 필요하다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[DAO] addUser메서드 => [Server]의 signForUser 메서드의 반환 값인 사용자 정보를 user 테이블에 추가하는 메서드이다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[결과] 사용자 토큰 발행 성공 시 저장되는 users 테이블의 token 데이터:

**텍스트, 스크린샷, 블랙, 폰트이(가) 표시된 사진

자동 생성된 설명**

**2.2.2 <로그인> 토큰 인증 로그인을 이용한 권한 관리**

아래는 토큰 인증 로그인을 통한 사용자 권한 관리 흐름이다.

[Client] loginUser메서드 => 사용자에게 로그인을 위한 정보 (id, password, token) 값을 입력 받아 [Server]로 로그인 요청을 보내는 메서드이다. [Server]로부터 로그인 성공 값을 반환 받으면 수강신청 단계로 사용자를 안내한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

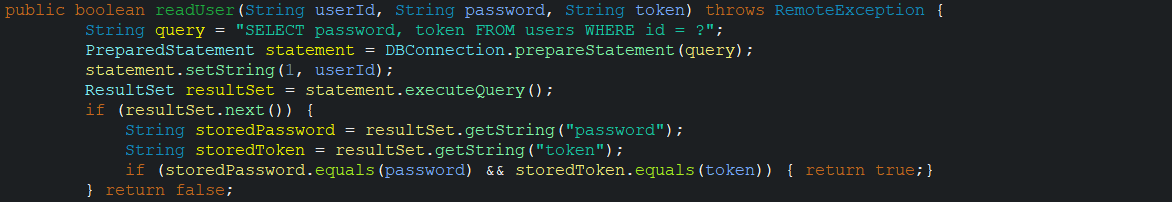
자동 생성된 설명**

[Server] loginUser메서드 => [Client]로부터 받은 사용자 정보를 통해 유효성 검사를 한 뒤, 정보가 일치하면 로그인 성공을 [Client]로 반환하는 메서드이다. 사용자 정보를 DB에 저장하기 위해 [DAO].readUser에도 입력 값을 전송한다.

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

[DAO] readUser메서드 => [Server]의 loginUser메서드에서 넘긴 사용자 정보 (id, password, token)를 user 테이블에서 조회하는 메서드이다.

****

[결과] 사용자 토큰을 이용하여 사용자 권한 관리 (로그인 성공/실패 예시):

텍스트, 스크린샷, 블랙, 폰트이(가) 표시된 사진

자동 생성된 설명

**2.2.3 <정보보안> 비밀번호 암호화 작업**

[Client]와 [Server]의 Concern분리에 초점을 맞추어 설계 및 구현을 진행하다 보니, [Server]에서 보안성을 향상하여 사용자 정보를 보관하는 방향으로 비밀번호 암호화를 구상하게 되었다. Client-Server 프로그래밍에서 로그인 과정을 설계할 때, 높은 보안성을 위하여 도입하는 게 토큰, 솔트 알고리즘 및 해시화와 같은 방법이라고 알게 되었다. 발행 받은 토큰을 관리하는 영역은 [Client]라고 생각하여, [Server] 내에서 사용자의 정보를 더욱 안전하게 보관 및 관리하기 위하여 솔트 및 해시화를 선택하였다.

아래는 솔트, 해시 알고리즘을 이용한 비밀번호 암호화 과정이다.

[DAO] addUser 메서드 => 앞서 [Server]에서 토큰을 발행 성공 시 사용자 정보 추가에서 사용하였던 사용자 정보 추가 메서드이다. 사용자가 [Client]에 정보를 입력할 때 id, password를 입력하는데 이때 입력 받은 password를 토대로 솔트, 해시화가 진행되어 암호화된 비밀번호가 생성 및 정보에 추가가 된다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[SaltGenerator] generateSalt메서드 => [DAO]에 비밀번호를 암호화된 솔트를 제공하기 위해서 Salt를 만드는 메서드이다. SHA1PRNG 알고리즘을 통하여 난수를 만들어서 암호화된 비밀번호 형태를 갖도록 한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[SaltGenerator] hashPassword메서드 => generateSalt 메서드와 마찬가지로, 암호화된 비밀번호를 제공하는 메서드이다. 난수화 된 솔트를 다시 한번 해시화를 하는 과정인데, SHA-256 알고리즘을 이용하여 더욱 강력한 암호를 갖도록 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[결과] 사용자 토큰 발행 성공 시 저장되는 users 테이블의 salt 및 hash (pwSalt) 데이터:

**텍스트, 스크린샷, 키보드, 컴퓨터이(가) 표시된 사진

자동 생성된 설명**

**2.2.4 <학생/강의> 조회, 추가, 삭제**

Data Server를 이용하던 방식에서 데이터 CRUD 및 관리에 용이하도록 [DBMS]를 호출하여 데이터 관련 기능을 수행하는 방법으로 전환하였다. 해당 비즈니스 로직을 수행에 필요한 데이터의 CRUD는 모두 [DAO]에서 구현할 수 있도록 하였다. [Client]로부터 사용자의 입력 값을 받아 [Server]가 [DAO]로 전달하는 로직을 통해 데이터 CRUD가 원활하게 가능할 수 있도록 구현하였다.

아래는 DBMS를 연결하여 <학생> 정보를 조회, 추가, 삭제하는 과정이다.

[DAO] getAllStudentData메서드 => DB의 students 테이블로부터 모든 정보를 조회하는 메서드이다. DB 연결 후 테이블 내 정보를 리스트 형식으로 읽어 온다.

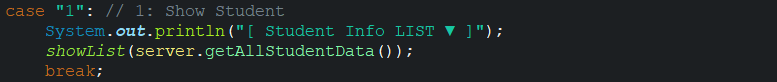
텍스트, 스크린샷이(가) 표시된 사진

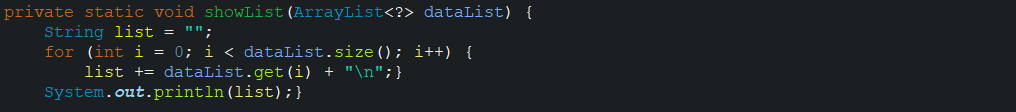
자동 생성된 설명

[Server] getAllStudentData메서드 => [Client] 요청에 의한 학생 정보를 리스트로 보여줄 수 있도록 [DAO]로부터 데이터 값을 받아온다.



[Client] showList메서드 => 선택한 ArrayList<>에 맞는 데이터 리스트를 가져와 보여주는 메서드이다. 학생 정보를 보여주기 위하여 [Server].getAllStudentData 메서드를 연결한다.





[Client] addStudet 메서드 => 추가할 학생 정보를 사용자에게 입력 받아 [Server]에 전달 및 Add 요청을 하는 메서드이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[Server] addStudet 메서드 => [Client]로부터 받은 추가할 학생 정보를 [DAO]로 전달하는 메서드이다. Add 성공 시 [Client]에게 결과 값을 반환한다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[DAO] addStudet 메서드 => [Server]로부터 받은 추가할 학생 정보를 DB의 students 테이블에 INSERT문을 사용하여 추가한다. getAllStudentData 메서드를 활용하여 중복 데이터 추가가 발생하지 않도록 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

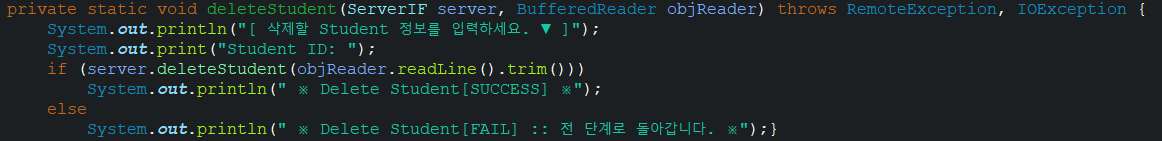
자동 생성된 설명

[Student] Student 클래스 => [DAO]에서 새로운 학생 정보를 Add 할 때 기존 학생 정보 및 문자열 확인을 위하여 사용한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[Client] deleteStudet 메서드 => 삭제할 학생 정보를 사용자에게 입력 받아 [Server]에 전달 및 Delete 요청을 하는 메서드이다.



[Server] deleteStudet 메서드 => [Client]로부터 받은 삭제할 학생 정보를 [DAO]로 전달하는 메서드이다. Delete 성공 시 [Client]에게 결과 값을 반환한다.



[DAO] deleteStudet 메서드 => [Server]로부터 받은 삭제할 학생 정보를 DB의 students 테이블에 DELETE문을 사용하여 추가한다. 열이 1개 이상으로 존재해야 삭제가 가능하도록 구현하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

아래는 DBMS를 연결하여 <강의> 정보를 조회, 추가, 삭제하는 과정이다.

[DAO] getAllCourseData메서드 => DB의 courses테이블로부터 모든 정보를 조회하는 메서드이다. DB 연결 후 테이블 내 정보를 리스트 형식으로 읽어 온다.

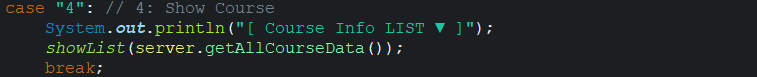
텍스트, 스크린샷, 폰트이(가) 표시된 사진

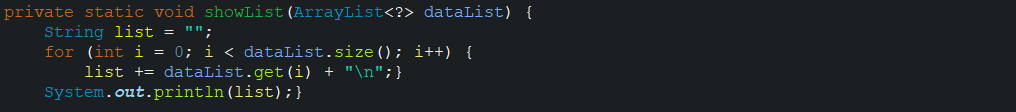
자동 생성된 설명

[Server] getAllCourseData메서드 => [Client] 요청에 의한 학생 정보를 리스트로 보여줄 수 있도록 [DAO]로부터 데이터 값을 받아온다.



[Client] showList메서드 => 선택한 ArrayList<>에 맞는 데이터 리스트를 가져와 보여주는 메서드이다. 학생 정보를 보여주기 위하여 [Server]. getAllCourseData메서드를 연결한다.





[Client] addCourse 메서드 => 추가할 강의 정보를 사용자에게 입력 받아 [Server]에 전달 및 Add 요청을 하는 메서드이다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[Server] addCourse 메서드 => [Client]로부터 받은 추가할 강의 정보를 [DAO]로 전달하는 메서드이다. Add 성공 시 [Client]에게 결과 값을 반환한다.



[DAO] addCourse 메서드 => [Server]로부터 받은 추가할 강의 정보를 DB의 courses테이블에 INSERT문을 사용하여 추가한다. getAllCourseData 메서드를 활용하여 중복 데이터 추가가 발생하지 않도록 한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

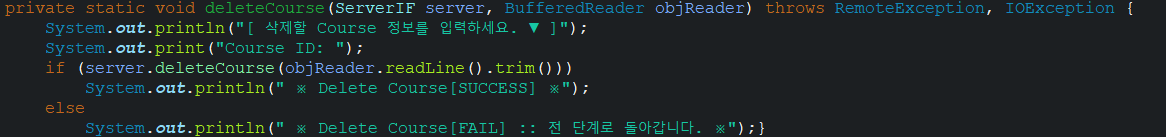
자동 생성된 설명

[Course] Course 클래스 => [DAO]에서 새로운 학생 정보를 Add 할 때 기존 강의 정보 및 문자열 확인을 위하여 사용한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[Client] deleteCourse 메서드 => 삭제할 강의 정보를 사용자에게 입력 받아 [Server]에 전달 및 Delete 요청을 하는 메서드이다.



[Server] deleteCourse 메서드 => [Client]로부터 받은 삭제할 강의 정보를 [DAO]로 전달하는 메서드이다. Delete 성공 시 [Client]에게 결과 값을 반환한다.



[DAO] deleteCourse 메서드 => [Server]로부터 받은 삭제할 강의 정보를 DB의 courses테이블에 DELETE문을 사용하여 추가한다. 열이 1개 이상으로 존재해야 삭제가 가능하도록 구현하였다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**2.2.5 <수강신청> 유효성 확인, 조회, 등록, 취소**

수강신청 테이블을 새로 만들어, 학생/강의 테이블 및 데이터를 기반으로 수강신청이 가능한 비즈니스 로직을 구현하였다. 수강신청이 이전 단계인 <학생/강의> CRUD와 다른 점이 있는데, 바로 두 가지의 테이블의 데이터를 확인하는 과정이 필요하다는 것이다. 수강신청 CRUD 전 확인해야 하는 두 가지는 수강할 강의의 선제 강의 수강 여부와 수강신청 내역 중복이다. 수강할 강의의 선제 강의 수강 여부를 확인하기 위해서는 Student테이블의 수강완료 데이터와 Course 테이블의 선제 강의 종류를 비교하여, 두 데이터가 일치하면 수강신청이 완료하도록 구현하였다.

아래는 DBMS를 연결하여 <수강신청> 유효성 확인, 조회, 등록, 취소하는 과정이다.

[DAO] getAllRegisterData메서드 => DB의 registrations테이블로부터 모든 정보를 조회하는 메서드이다. DB 연결 후 테이블 내 정보를 리스트 형식으로 읽어 온다.

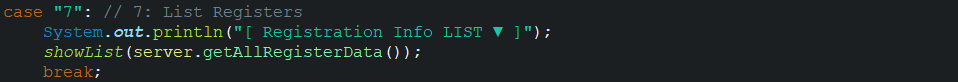
텍스트, 스크린샷, 폰트이(가) 표시된 사진

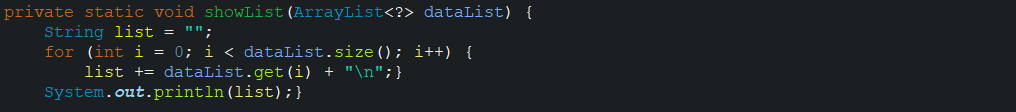
자동 생성된 설명

[Server] getAllRegisterData메서드 => [Client] 요청에 의한 수강 정보를 리스트로 보여줄 수 있도록 [DAO]로부터 데이터 값을 받아온다.



[Client] showList메서드 => 선택한 ArrayList<>에 맞는 데이터 리스트를 가져와 보여주는 메서드이다. 학생 정보를 보여주기 위하여 [Server]. getAllCourseData메서드를 연결한다



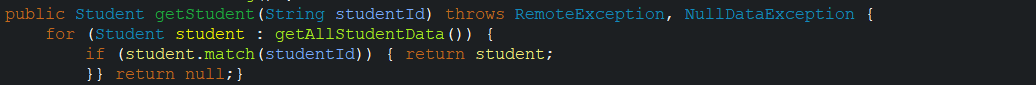


[Client] registerCourse메서드 => 등록할 수강 정보를 사용자에게 입력 받아 [Server]에 전달 및 등록 요청을 하는 메서드이다.

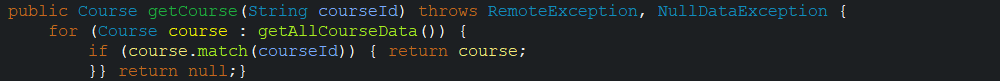
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[Server] getStudent메서드 => getAllStudentData 리스트에서 Student의 Id를 가져오는 메서드이다. 이후 수강신청을 위한 유효성 검사를 위하여 사용된다.



[Server] getCourse메서드 => getAllCourseData 리스트에서 Course의 Id를 가져오는 메서드이다. 이후 수강신청을 위한 유효성 검사를 위하여 사용된다.

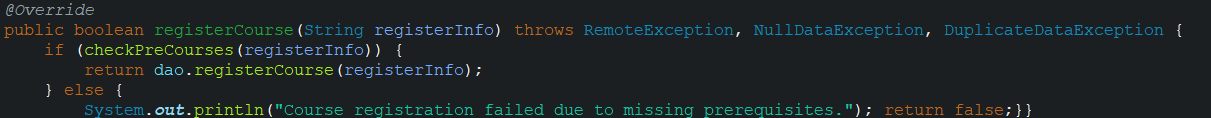


[Server] checkPreCourses메서드 => 학생이 수강하고자 하는 강의 정보 내 선제 강의와 학생의 수강 완료 강의가 일치해야 수강신청이 가능하도록 하는 메서드이다. 각각 다른 테이블에 있는 데이터를 가져와서 비교해야 하기 때문에 getStudent, getCourse 메서드를 통하여 각각의 Id를 확인해 수강 정보의 유효성을 검사한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[Server] registerCourse메서드 => [Client]로부터 받은 등록할 수강정보를 [DAO]로 전달하는 메서드이다. checkPreCourses 메서드를 통하여 수강할 강의의 선제 강의 수강 여부를 확인하고, 수강했을 시 수강신청이 가능해지는 로직으로 구현하였다. 수강 등록 성공 시 [Client]에게 결과 값을 반환한다



[Register] Register 클래스 => [DAO]에서 새로운 수강신청 정보를 등록할 때 기존 수강 정보 및 문자열 확인을 위하여 사용한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[DAO] registerCourse메서드 => [Server]로부터 받은 등록할 수강정보를 DB의 registrations테이블에 INSERT문을 사용하여 추가한다. getAllRegisterData 메서드를 활용하여 중복 데이터 추가가 발생하지 않도록 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

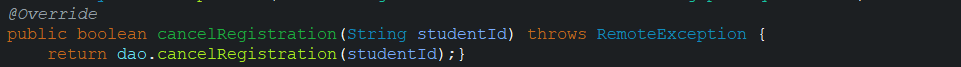
자동 생성된 설명

[Client] cancelRegistration 메서드 => 삭제할 수강 정보를 사용자에게 입력 받아 [Server]에 전달 및 Cancel요청을 하는 메서드이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[Server] cancelRegistration 메서드 => [Client]로부터 받은 삭제할 수강 정보를 [DAO]로 전달하는 메서드이다. Delete 성공 시 [Client]에게 결과 값을 반환한다.



[DAO] cancelRegistration 메서드 => [Server]로부터 받은 삭제할 수강 정보를 DB의 registrations테이블에 DELETE문을 사용하여 추가한다. 열이 1개 이상으로 존재해야 삭제가 가능하도록 구현하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 1. **로깅**
     1. **서버 로깅**

Logging은 [Server]에 있어 디버깅 역할을 하여 [Server] 동작의 로그를 남겨주는 중요한 역할이다. [Server]의 로깅 기능은 ServerLogger 클래스를 통해 [Server] 로깅에 포함되어야 하는 정보를 기재하고, [Server]에서 일어나는 예외 발생, 각 메서드 구현 시 [Server]의 작업으로 기록되어 이후 로그를 통해 작업을 확인할 수 있도록 설계하였다. [Server] 로그에는 [Server] 내 메서드 및 기능이 처리되는 과정, 반환 값, 예외 등이 기록되도록 구현하였다.

아래는 ServerLogger를 이용하여 [Server]에 [Server] 로깅을 하는 과정이다.

[ServerLogger] ServerLogger 클래스 => [Server]에서 호출만으로 로그를 남길 수 있도록 로그에 관련된 메서드들을 모아둔 클래스이다. 로그를 기록하는 시간, 로그 메시지 등의 정보를 포함하고 있다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

[Server] ServerLogger 호출 => [Server]에서 로그를 남기고자 하는 곳에 로거를 기록한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[결과] ServerLog.log 파일에 기록된 Log:

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + 1. **토큰 로깅**

실제 서비스에서 사용자가 인증 토큰을 발행 받을 때, 서비스에서는 토큰을 잃어버리지 않도록 주의하라고 안내한다. 일반적으로 로깅이라는 게 개발의 편의성을 높이기 위하여 [Server]의 기록을 남기는 이유지만, 현재 프로젝트에서 실제 사용자가 있는 게 아니기에 발행되는 토큰을 임시적으로 [Client]에서 기록되었으면 하였다. 토큰 로깅은 [Client]에서 [Server]에게 토큰을 요청한 후 받은 토큰 값을 저장 및 처리하기 용이하도록 임시적으로 토큰 값이 기록될 수 있도록 하였다.

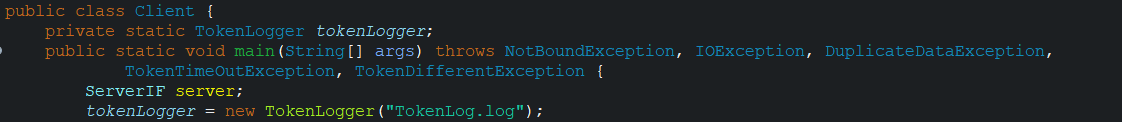
아래는 TokenLogger를 이용하여 [Client]에 [Token] 로깅을 하는 과정이다.

[TokenLogger] TokenLogger 클래스 => [Client]에서 호출만으로 토큰 로그를 남길 수 있도록 로그에 관련된 메서드들을 모아둔 클래스이다. 로그를 기록하는 시간, 로그 메시지 등의 정보를 포함하고 있다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

[Client] TokenLogger 호출 => [Server]로부터 토큰이 발행되면 로그가 기록된다.



[결과] TokenLog.log 파일에 기록된 Log:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* 1. **예외 처리**

Client-Server프로그래밍 중 완전한 예외 처리가 된다는 것은 [Sever]에서 throw로 던진 exception을 [Client]에서 try/catch문을 이용해서 catch를 할 수 있어야 함을 의미한다. 각 예외 클래스에서 doaction 메서드를 만들어, 상황별로 알맞은 메시지가 발생하고 예외 처리 확인이 원활하게 되도록 하였다. 설계 및 구현한 로직에서 발생할 수 있는 가능성이 있는 NullData, DuplicateData, TokenDifferent, TokenTimeOut 예외 처리를 구현하였다.

* + 1. **NullDataException**

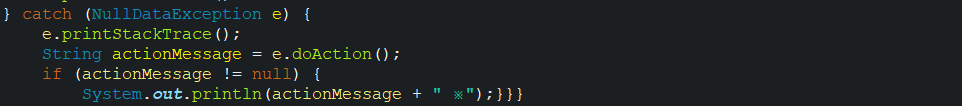
아래는 NullDataException 발생 및 처리 과정이다.

[NullDataException] NullDataException 클래스 => 테이블에 데이터가 존재하지 않을 때 발생하는 Exception이다.

**스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명**

[Server]에서 throw 한 [NullDataException]을 [Client]에서 catch 한다.

****

[결과] Registration 테이블에 데이터가 없는 경우:

****

* + 1. **DuplicateDataException**

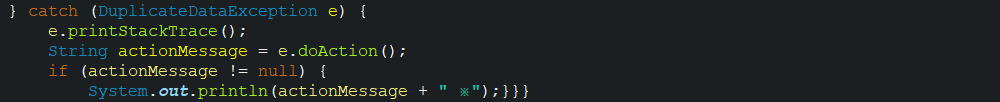
아래는 DuplicateDataException발생 및 처리 과정이다.

[DuplicateDataException] DuplicateDataException 클래스 => 테이블에 데이터 추가 시 중복 데이터가 존재할 때 발생하는 Exception이다.

**스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명**

[Server]에서 throw 한 [DuplicateDataException]을 [Client]에서 catch 한다.



[결과] Registration 테이블에 데이터가 없는 경우:

****

* + 1. **TokenDifferentException**

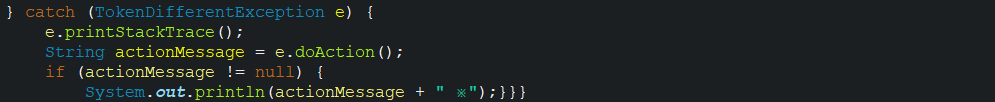
아래는 TokenDifferentException발생 및 처리 과정이다.

[TokenDifferentException] TokenDifferentException 클래스 => 사용자에게 발급된 토큰과 입력한 토큰이 다른 경우 발생하는 Exception이다.

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

[Server]에서 throw 한 [TokenDifferentException]을 [Client]에서 catch 한다.

****

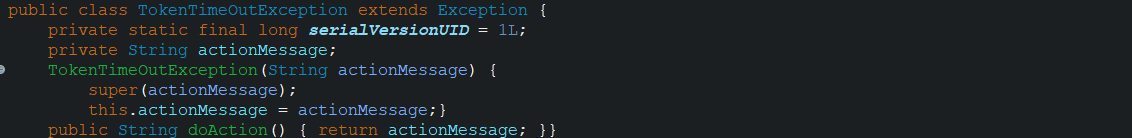
[결과] [Server]에서 발급한 토큰과 [Client] 입력 토큰이 다른 경우:

****

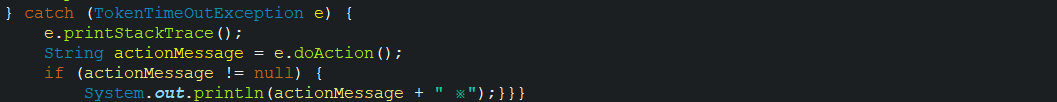
* + 1. **TokenTimeOutException**

아래는 TokenTimeOutException발생 및 처리 과정이다.

[TokenTimeOutException] TokenTimeOutException 클래스 => 사용자가 입력한 토큰이 보유 유효 시간을 초과했을 경우 발생하는 Exception이다.



[Server]에서 throw 한 [TokenTimeOutException]을 [Client]에서 catch 한다.



[결과] [Client]에 입력한 토큰이 보유 유효 시간을 초과한 경우:



1. **Demo**
   1. **프로그램 실행**

프로그램의 전체 실행 흐름을 나타내기 위한 Demo이다. 앞서 다뤘던 [Client], [Server], [DAO]의 모든 메서드가 유기적으로 통신이 되며 결과적으로 사용자에게 프로그램 실행이 가능하게 하였다. 프로젝트에서 다룬 모든 기능을 종합적으로 보여주는 실행 흐름이며, 데이터의 조회, 추가, 삭제 등의 작업과 예외 상황도 기재된다.

아래는 [Client]에서 사용자의 입력 값을 받는 로직 구현이다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

아래는 프로그램 전체 실행 흐름이다.

[메인] 가장 상단의 메뉴

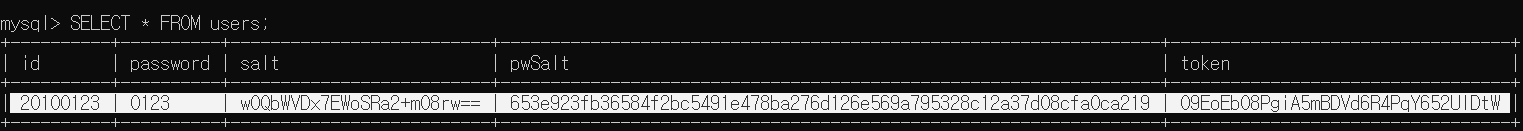
****

[토큰 발행 성공] DB의 학생 정보(id)와 입력한 정보가 일치하는 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[토큰 발행 성공] DB에 유저 데이터(id, pw, salt, pwsalt, token)가 정상적으로 추가된 경우



[토큰 발행 실패] DB의 학생 정보(id)와 입력한 정보가 일치하지 않는 경우

**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

[로그인 성공] DB의 User 정보(id, pw, token)와 입력한 정보가 일치하는 경우

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

[로그인 실패] 유저가 토큰 발행을 하지 않은 경우 (User 정보(id, pw, token) 불일치)

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

[수강신청 메인] 로그인 성공 시 진입 가능한 CRUD가 이뤄지는 메뉴

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

[학생 조회] 학생정보 조회 선택 시 보여지는 리스트

**텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명**

[학생 추가] 학생 추가가 정상적으로 완료된 경우

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

[학생 추가] DB에 학생 데이터가 정상적으로 추가된 경우

**텍스트, 스크린샷, 디스플레이이(가) 표시된 사진

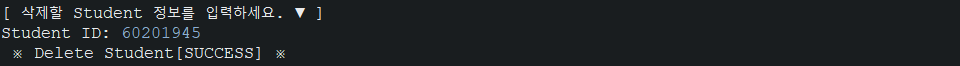
자동 생성된 설명**

[학생 추가] 추가하려는 학생 ID가 중복인 경우

**스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명**

[학생 삭제] 학생 삭제가 정상적으로 완료된 경우

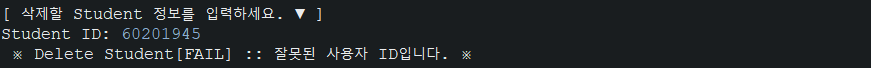
****

[학생 삭제] DB의 학생 데이터가 정상적으로 삭제된 경우

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

[학생 삭제] 삭제하려는 학생 ID를 찾을 수 없는 경우

****

[강의 조회] 강의정보 조회 선택 시 보여지는 리스트

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

[강의 추가] 강의 추가가 정상적으로 완료된 경우

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

[강의 추가] DB에 강의 데이터가 정상적으로 추가된 경우

**텍스트, 스크린샷, 블랙이(가) 표시된 사진

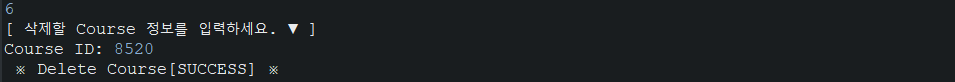
자동 생성된 설명**

[강의 추가] 추가하려는 강의 ID가 중복인 경우

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

[강의 삭제] 강의 삭제가 정상적으로 완료된 경우

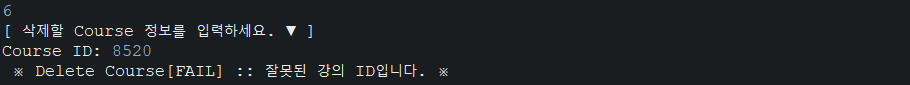
****

[강의 삭제] DB의 강의 데이터가 정상적으로 삭제된 경우

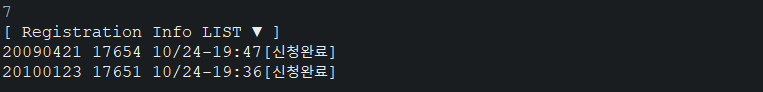
**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

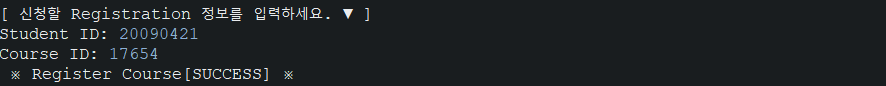
[강의 삭제] 삭제하려는 강의 ID를 찾을 수 없는 경우

****

[수강내역 조회] 수강내역 조회 선택 시 보여지는 리스트



[수강 신청] 수강 신청이 정상적으로 완료된 경우 (선제 강의 수강 완료)

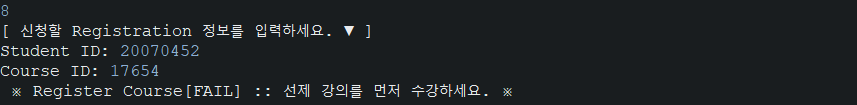
****

[수강 신청] DB에 수강 신청 데이터가 정상적으로 추가된 경우

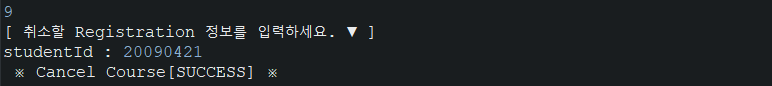
**텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명**

[수강 신청] 선제 강의 수강 미완료로 수강신청이 불가능한 경우

****

[수강 취소] 수강 취소가 정상적으로 완료된 경우

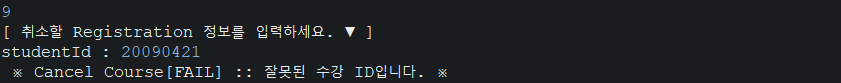
****

[수강 취소] DB의 수강 데이터가 정상적으로 삭제된 경우

**텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

자동 생성된 설명**

[수강 취소] 취소하려는 강의 ID를 찾을 수 없는 경우

****

1. **결론** 
   1. **느낀 점**

프로젝트를 진행하면서 가장 많은 노력과 고민을 해야 했던 부분은 [Client]와 [Server]의 관심사 분리였다. [Client]와 [Server]가 각각 어떤 기능을 수행해야 하고, 서로의 영역과 기능 수행의 역할을 절대 침범해서는 안 된다는 걸 여러 번 상기하면서 설계 및 구현을 진행하였다. [Client]에서는 절대 판단을 하게 해서는 안 되며 모든 판단은 [Server]가 진행한 뒤 [Client]에게 결과 값만 반환해야 한다는 것을 잊지 않기 위하여 노력하였다.

주요 기능인 토큰 인증과 로깅 단계를 수행하면서, 물리적으로 분리되어 있는 Client-Server 프로그래밍에 있어서 ‘확인’ 단계가 반드시 필요하다는 것을 느낄 수 있었다. 분리되어 있기 때문에 어느 한 쪽에 확인 단계를 느슨하게 하거나, 옳지 않은 요청 또는 응답이 된다면 프로그램 자체에 결함이 생기게 된다는 것을 깨닫게 되었다. 더불어, 서버의 로그를 기록하고 예외 처리를 함으로써 프로그램 처리 과정을 하나하나 밟고 확인하는 게 얼마나 중요한 과정인지 알 수 있었다. 기존에는 프로그램이 돌아가기만 하면 되겠지 라는 생각이었다면, 이제는 [Server]의 입장이 되어서 오류나 문제 상황이 발생했을 때 얼마나 더 효율적으로 문제를 해결하는 가에 집중할 수 있게 되었다.

중간 리포트 제출 전 마지막 멘토링에서 Data Server를 이용한 파일 기록도 다 하지 못한 채 멘토링을 받았다. 멘토링 이후 파일 기록을 수행하며 안정성 및 관리성이 더욱 보장되는 데이터베이스에 연결에 도전하고 싶은 마음이 생겼다. 기존 [Client]와 [Server]에 DB를 연결하기 위해서 코드를 수정하고, 결론적으로 [DAO]를 통하여 DB 호출 및 CRUD를 구현하게 되었다.

Client-Server 구조를 이해하고 프로젝트의 목적을 준수하기 위해서 프로그램 설계 및 구현 시 많은 고민을 하였다. 코드를 구현하며 단순히 if문으로 해결해도 되지 않을까 하다가 프로젝트의 목적을 상기하며 다른 방안을 생각해내었다. 마주치는 이슈를 어떤 방식으로 해결해야 할지 고민하는 과정을 통해 사고의 확장을 느낀 것 같다. 전공 수업에서 진행한 프로젝트 중 가장 많은 고민을 한 프로젝트였다고 생각한다. 아직 중간 리포트이지만, 더 많은 고민과 연구를 통해 더욱 완성도 있는 프로그램을 만들고 싶다는 생각이 들었다.