2015-2 데이터베이스(컴퓨터공학과)

카메라 촬영 정보 기록 시스템

일시 : 2015. 12. 6

조장 : 이현규

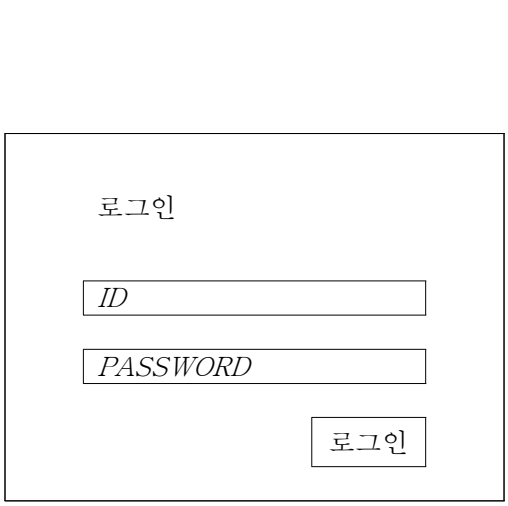
조원 : 박정재

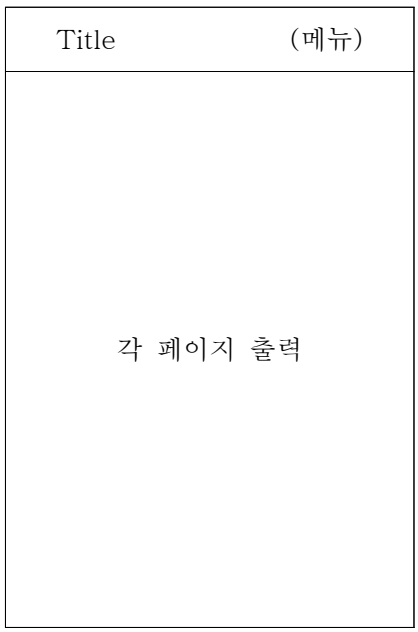
담당교수 : 김진덕

1. 프로젝트의 최종 목표

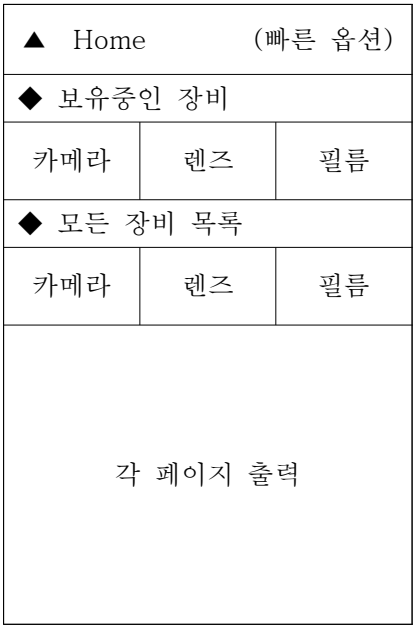
기존의 필름을 사용하던 사용자들은 자신이 촬영한 사진의 정보를 따로 기록하지 않으면 남지 않았기 때문에 감각적으로 사진을 찍어나 손으로 촬영했던 설정을 기록해야 했다. 하지만 감각적으로 사진을 찍으면 실력을 늘리기 힘들고, 매번 손으로 기록하기에는 번거롭다. 이것을 보완하기 위해 이 프로젝트에서는 사용자가 가진 전자기기를 이용해 35mm 필름과 렌즈 교환 방식의 카메라를 사용하여 촬영한 사진의 정보를 어떤 환경이라도 신속하게 기록할 수 있도록 하는 시스템을 구현하는 것을 목표로 한다.

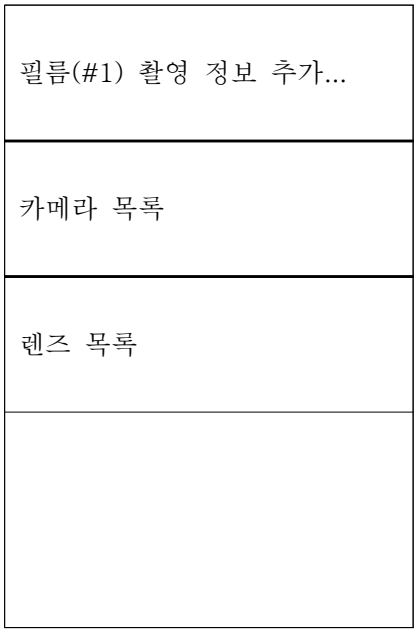
2. 프로그램의 UI 및 기능



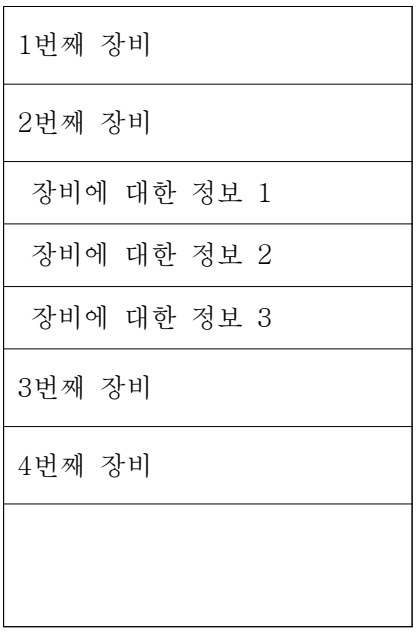


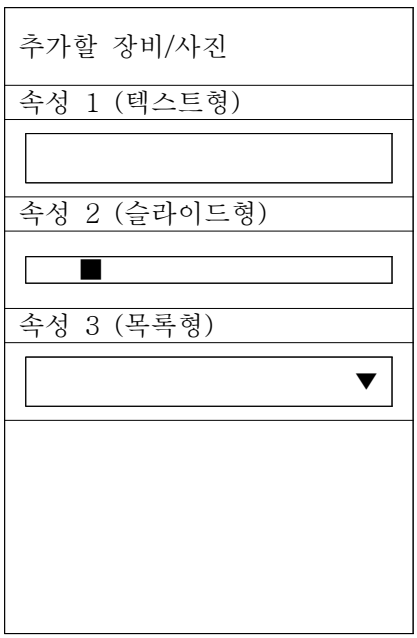
로그인 화면	기본 틀 페이지
• 접속 시 기본 틀에 메인 페이지를 출력하고, 로그인 정보를 확인한다. • 로그인 페이지는 화면의 가운데에 나타나며, 나머지 화면은 보이지 않도록 한다. 정보가 없다면 자동으로 추가한다. • 기본 틀에서 메뉴 확장 버튼을 누르면, 메뉴가 페이지를 가리는 형식으로 확장된다. 빠른 옵션은 기본으로 촬영 버튼이 있고, 필요에 따라 각 장비 추가 버튼이 추가된다. • Home 문자를 누르면 메인 페이지로 이동한다.	





확장된 메뉴	메인 페이지
• 메뉴를 확장하면 보유중인 장비 목록과 모든 장비 목록을 확인할 수 있다. • 메인 페이지는 현재 삽입되어 있는 필름에 빠르게 출력하고, 각 장비 목록으로 이동할 수 있도록 하고, 해당 장비를 등록하기 쉽게 만들도록 한다.	



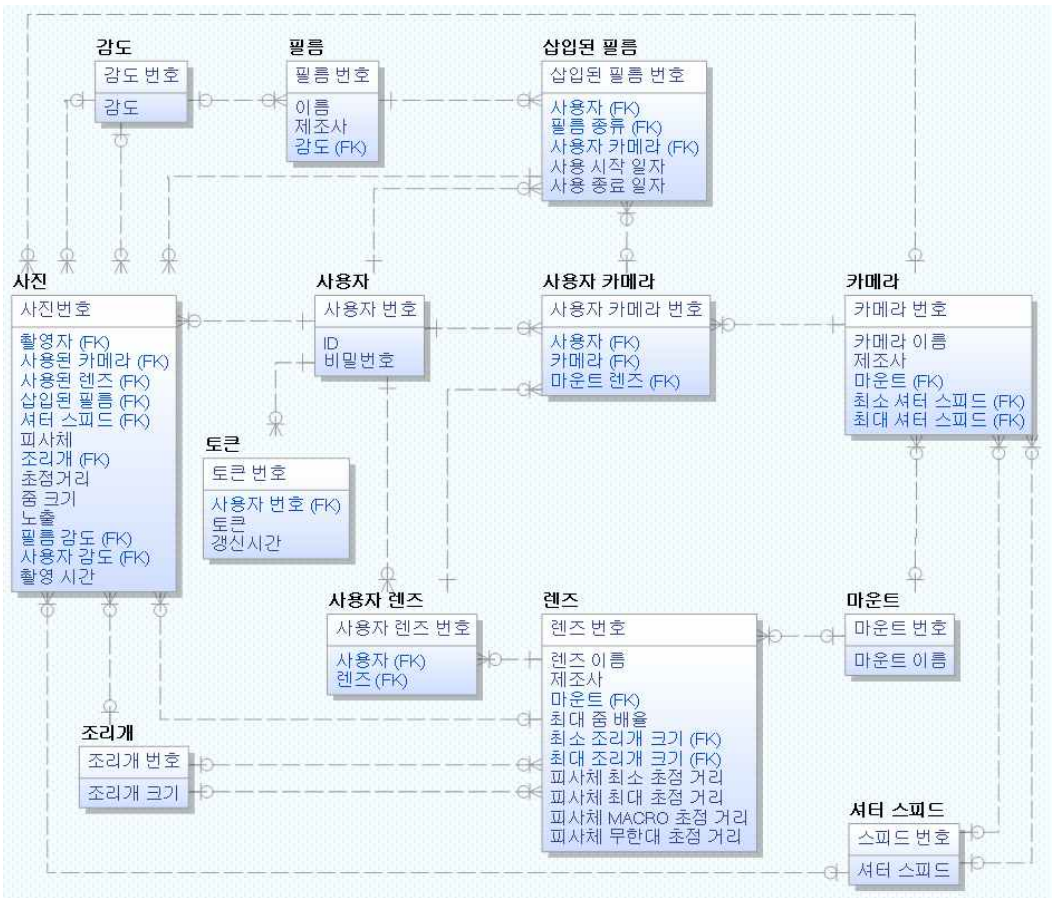


각 장비 정보 페이지	새로운 장비/사진 추가 페이지
• 장비에 대한 정보페이지에는 장비 리스트가 보여지고, 해당 장비를 누르면 상세 설명을 확인할 수 있다. 그리고 보유 장비에서 보면 연결되어 있는 장비가 나온다. 상세 설명은 하나만 출력 가능하다(이전에 출력했던 상세 설명은 보이지 않게 된다). • 보유 장비에서의 페이지라면 새로운 장비를 추가할 수 있는 버튼이 상단 메뉴에 나타난다. • 새로운 장비/사진 추가 페이지에는 각 항목의 입력을 빠르게 받을 수 있도록 슬라이드, 목록 형식을 최대한 많이 사용하여 구현한다. 그리고 각 항목은 캐시(Cache)하여 다음 추가 시 입력되어 있도록 한다.	

3. 문서화된 요구사항

- 렌즈 교환식 카메라 촬영 정보 기록시스템을 구축하고자 한다.
- 사용자는 고유 번호와 ID, 비밀번호를 가진다.
- 카메라는 품목 번호와 이름, 제조사, 최대/최소 셔터 스피드 정보를 가진다.
- 렌즈는 품목 번호와 이름, 제조사, 렌즈 최대/최소 초점 거리, 최대/최소 조리개 너비, 피사체 최대/최소 초점 거리, 피사체 MACRO, 무한대 초점 거리 가능 유무에 대한 정보를 가진다.
- 필름은 품목 번호와 이름, 제조사, 감도를 가진다.
- 사용자는 여러 개의 카메라를 가질 수 있다.
- 사용자는 여러 개의 렌즈를 가질 수 있다.
- 사용자는 자신의 카메라에 렌즈를 마운트하고 필름을 삽입하여 사용한다.
- 카메라와 렌즈는 같은 마운트를 사용해야 한다.
- 삽입된 필름은 사용자, 사용 카메라, 필름 종류, 사용 시작 시간, 사용 종료 시간에 대한 정보를 가진다.
- 사용자는 자신의 카메라를 사용해 사진을 촬영한다.
- 사진은 사진 번호와 촬영자, 사용된 카메라, 사용된 렌즈, 삽입된 필름, 셔터 스피드, 조리개, 초점거리, 노출, 감도, 촬영 시간에 대한 정보를 가진다.
- 사용자는 토큰을 사용해 인증한다.
- 토큰은 사용자 정보, 토큰 값, 갱신 시간에 대한 정보를 가진다.

4. E-R Diagram (최종)



5. 데이터베이스 스키마 (최종)

릴레이션 **스키마**

사용자
usr

사용자 번호		ID		비밀번호	
u_no	NUMBER(10)	ID	CHAR(20)	pwd	CHAR(64)

카메라
camera

카메라 번호		카메라 이름		제조사		*마운트*	
c_no	NUMBER(5)	name	CHAR(20)	maker	CHAR(10)	*mount*	*NUMBER(2)*
최소 셔터 스피드		*최대 셔터 스피드*					
minSS	*NUMBER(2)*	*maxSS*	*NUMBER(2)*				

렌즈
lens

렌즈 번호		렌즈 이름		제조사		*마운트*	
l_no	NUMBER(4)	name	CHAR(20)	maker	CHAR(10)	*mount*	*NUMBER(2)*
최대 줌 배율		*최소 조리개 너비*		*최대 조리개 너비*		피사체 최소 초점거리	
zoom	NUMBER(2,2)	*minF*	*NUMBER(2)*	*maxF*	*NUMBER(2)*	minfcs	NUMBER(4)
피사체 최대 초점거리		피사체 MACRO 초점거리		피사체 무한대 초점거리			
maxfcs	NUMBER(4)	macfcs	NUMBER(1)	inffcs	NUMBER(1)		

필름
film

필름 번호		필름 이름		제조사		*감도*	
f_no	NUMBER(3)	name	CHAR(10)	maker	CHAR(10)	*ISO*	*NUMBER(2)*

마운트
mount

마운트 번호		마운트 이름	
m_no	NUMBER(2)	name	CHAR(10)

사용자 카메라
uesr_camera

사용자 카메라 번호		*사용자*		*카메라*		*마운트 렌즈*	
uc_id	NUMBER(6)	*usr*	*NUMBER(10)*	*camera*	*NUMBER(5)*	*lens*	*NUMBER(4)*

사용자 렌즈
uesr_lens

사용자 렌즈 번호		*사용자*		*렌즈*	
ul_no	NUMBER(5)	*usr*	*NUMBER(10)*	*lens*	NUMBER(4)

삽입된 필름
inserted_film

삽입된 필름 번호		*사용자*		*필름 종류*		*사용자 카메라*	
uf_no	NUMBER(10)	*usr*	*NUMBER(10)*	*film*	*NUMBER(3)*	*camera*	NUMBER(6)
사용 시작 일자		사용 종료 일자					
start	DATE	end	DATE				

사진
picture_meta

사진번호		*촬영자*		*사용된 카메라*		*사용된 렌즈*	
p_no	NUMBER(15)	*usr*	*NUMBER(10)*	*camera*	*NUMBER(6)*	*lens*	*NUMBER(5)*
사용된 필름		*피사체*		*셔터 스피드*		*조리개*	
film	*NUMBER(10)*	target	CHAR(20)	*shut*	*NUMBER(2)*	*F_size*	*NUMBER(2)*
초점거리		줌 크기		노출		*필름 감도*	
range	NUMBER(4)	zoom	NUMBER(2,2)	ev	NUMBER(1)	*ISO*	*NUMBER(2)*
촬영 시간							
sttime	DATETIME						

셔터 스피드
shutter_speed

스피드 번호		셔터 스피드	
ss_no	NUMBER(2)	speed	CHAR(7)

조리개
diaphragm

조리개 번호		조리개 크기	
f_no	NUMBER(2)	F_size	NUMBER(2,1)

감도
iso

감도 번호		감도	
iso_no	NUMBER(2)	iso	NUMBER(5)

6. 조원의 역할

역할 구분	이현규	박정재
주 역할	• 클라이언트(Web-UI) 개발 • 서버-클라이언트 연동	• 서버-데이터베이스 접속부 구현 • 데이터베이스 쿼리 작성
보조 역할	◦ 데이터베이스 쿼리 작성 보조 ◦ 서버 개발 환경 구축	◦ 클라이언트 프로그래밍 보조

7. 일정

	이현규	박정재
1주차 11. 01 ~ 11. 07	• 서버 환경 구축 • 서버 인터페이스 파트 구현 • 서버 인터페이스-처리부 설계	• 데이터베이스 테이블 구성 • 사용될 쿼리 작성
2주차 11. 08 ~ 11. 14	• 클라이언트 작성 및 서버와 연계	• 데이터 처리부 개발
3주차 11. 15 ~ 11. 21	• 클라이언트-서버 테스트	• 서버 처리부 테스트
	• 프로젝트 통합	
4주차 11. 22 ~ 11. 28	• 테스트 및 버그 수정 • 보완점 확인	

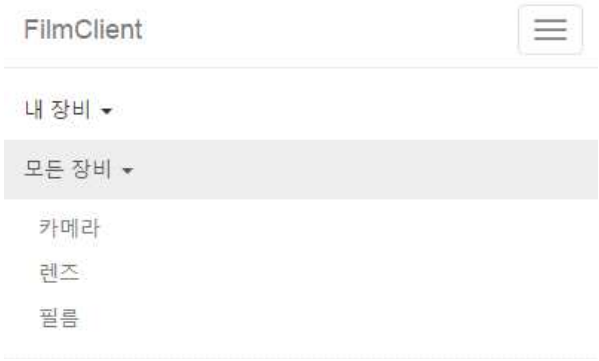
8. 프로그램 개요

※ 프로그램 개발 환경

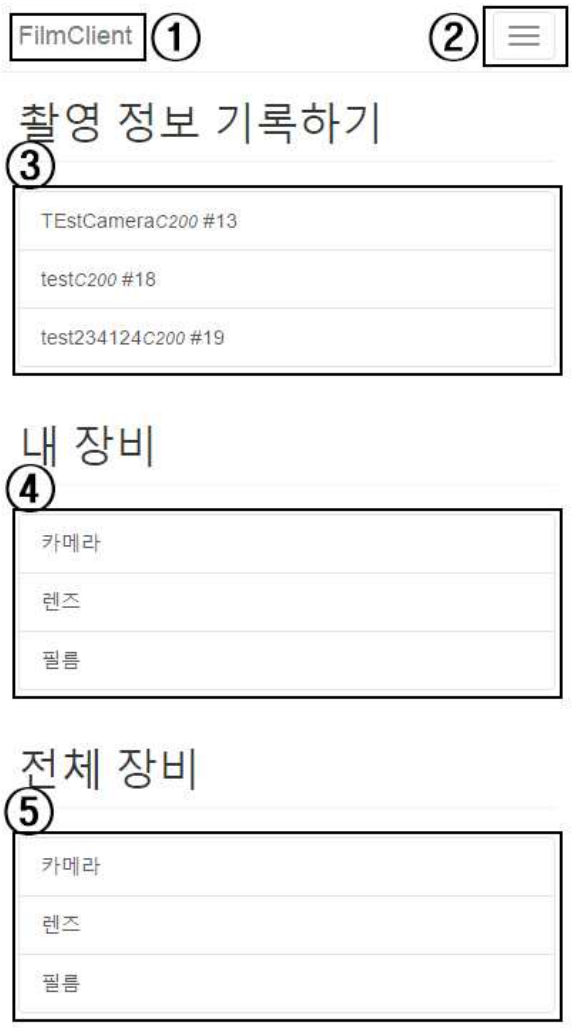
	이현규	박정재
서버 환경	• Apache Tomcat 8 • Nginx (Reverse Proxy) • Git	
개발 도구	• Eclipse Mars • VIM • Oracle SQL Developer • yoeman, grunt, angularjs	• Eclipse Mars • Oracle SQL Developer
언어	• Java 1.8 (Servlet) • HTML • javascript	• Java 1.8 (Servlet)

※ 프로그램 동작

<로그인 페이지>

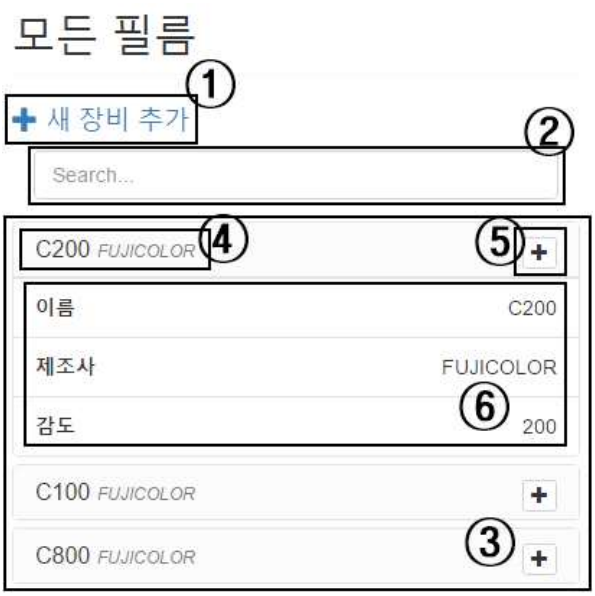


<네비게이션>



<메인 페이지>

① 메인 페이지로 이동할 수 있는 타이틀
② 메뉴 열기/닫기 <네비게이션>
③ 빠른 사진 정보 입력 페이지
④ 사용자 본인의 장비 목록 페이지 (미구현)
⑤ 모든 장비 목록 페이지 <장비 목록 페이지>



<장비 목록 페이지>

① 새 장비를 추가하는 페이지 <장비 추가 페이지>
② ③ 목록 검색
③ 장비 목록
④ 상세 정보 보기 (⑥ 출력)
⑤ 장비 추가 액션 버튼 <선택 팝업>
⑥ 해당 장비의 상세 정보

<장비 추가 페이지>
① 메인 페이지로 이동할 수 있는 타이틀
② 메뉴 열기/닫기 <네비게이션>

<선택 팝업>
① 메인 페이지로 이동할 수 있는 타이틀
② 메뉴 열기/닫기 <네비게이션>
③ 빠른 사진 정보 입력 페이지

※ 프로그램 소스 설명

1. 페이지 오픈 (Clinet)
2. 서버에 기본 정보 요청 (Clinet)
3. 기본 정보 반환 (Server)
4. 정보 해석 및 데이터 바인딩 (Client)
5. 사용자가 특정 동작 (Client)
6. 동작에 대한 정보 전송 (Client)
7. 정보를 읽어서 처리 (Server)
8. 처리 완료 후 동작 (Client)

이 프로그램의 모든 페이지는 다음과 같은 구성을 따른다. 정보를 처리하는 페이지(Server)에서는 요청받은 정보에 필요한 질의문을 구성하고, 데이터베이스에 질의하여 반환받은 값을 특정 서식으로 바꿔서(Formatting) 출력해준다. 정보를 요청할 때는 HTTP 함수인 GET을 사용했고, 처리를 요청할 때는 POST를 사용했다.

```
$scope.init = function() {
    var token =  {
        token:TokenService.getToken('/')
    };
    if(token.token) {
        $http({
            method: 'GET',
            url: 'http://film.codefict.com/server/InsertedFilmList',
            params: token,
            headers: {'Content-Type': 'application/x-www-form-urlencoded; charset=utf-8'}
        }).success(function(data, status, headers, config) {
            if (data) {
                $scope.filmList = data;
                $scope.noneInsertedFilm = (data.length>0);
            }
        }).error(function(data, status, headers, config){
            console.log(status);
        });
    }
}
```

토큰 정보를 가지고 서버에 페이지 구성에 필요한 데이터를 요청한다. ‘server/InsertedFilmList’로 데이터를 요청함을 알 수 있다.

```
response.setContentType("application/json;charset=utf-8");
response.setCharacterEncoding("utf-8");
Database db = (Database) getServletContext().getAttribute("film_database");
PrintWriter out = response.getWriter();

Token token = Token.ServerInstance( db );
String sendToken = MessageTool.ConvertToSQLSafeString(request.getParameter("token"));

String UserId;
try {
    UserId = token.getUserID(sendToken);
}
catch (Exception e) {
    out.print("Invalid Token");
    return;
}
String getInsertedFilmList = String.format("SELECT cif.uf_no, c.c_name, film.f_name, cif.uf_count "
                                           + "FROM count_inserted_films cif "
                                           + "JOIN film ON film.f_no=cif.film "
                                           + "JOIN user_camera uc ON cif.camera=uc.uc_no "
                                           + "JOIN camera c ON c.c_no=uc.camera "
                                           + "WHERE cif.usr=(SELECT u_no FROM usr WHERE id='%s') AND cif.f_end IS NULL ORDER BY cif.uf_count"
                                           , UserId);

try {
    ResultSet rs = db.runSql(getInsertedFilmList);
    int count = 0;
    String InsertedFilmList = "";
    while(rs.next()) {
        int uf_no = rs.getInt(1);
        String c_name = rs.getString(2).trim();
        String f_name = rs.getString(3).trim();
        int f_count = rs.getInt(4);

        InsertedFilmList += String.format((count++>0?
                ",{\"id\":%d,\"c_name\":\"%s\",\"f_name\":\"%s\",\"f_count\":%d}"
               :"{\"id\":%d,\"c_name\":\"%s\",\"f_name\":\"%s\",\"f_count\":%d}")
               , uf_no, c_name, f_name, f_count);
    }
    out.print( String.format("[%s]", InsertedFilmList) );
} catch (Exception e) {
    out.print("failed");
}
```

서버에서는 클라이언트와 인코딩을 일치시키고, 서버에 Singleton 패턴으로 구현되어있는 데이터베이스 객체를 가져오고 파라미터로 받은 토큰 정보를 점검하고, 토큰의 사용자 ID를 받는다.
다음으로 필요한 정보를 얻기 위한 질의문을 함수를 통해 구성한다.

구성된 질의문을 선언되어 있는 질의 함수(Sql)을 사용해서 서버에 데이터를 요청한다. 그리고 반환받은 데이터 객체를 반복(while)하여 각 필드의 데이터를 받고, 이를 다시 구성해서 마지막에 반환해준다.

```
Database db = (Database) getServletContext().getAttribute("film_database");
PrintWriter out = response.getWriter();

Token token = Token.ServerInstance( db );

String sendToken = MessageTool.ConvertToSQLSafeString(request.getParameter("token"));
String insertedFilmId = MessageTool.ConvertToSQLSafeString(request.getParameter("id"));
String target = MessageTool.ConvertToSQLSafeString(request.getParameter("target"));
String shutterSpeed = MessageTool.ConvertToSQLSafeString(request.getParameter("shutter_speed"));
String diaphragm = MessageTool.ConvertToSQLSafeString(request.getParameter("diaphragm"));
String range = MessageTool.ConvertToSQLSafeString(request.getParameter("range"));
String zoom = MessageTool.ConvertToSQLSafeString(request.getParameter("zoom"));
String EV = MessageTool.ConvertToSQLSafeString(request.getParameter("ev"));
String ISO = MessageTool.ConvertToSQLSafeString(request.getParameter("iso"));

String UserId;
try {
    UserId = token.getUserID(sendToken);
}
catch (Exception e) {
    out.print("Invalid Token");
    return;
}

String getCameraIdSql = String.format("SELECT camera FROM user_camera WHERE uc_no=(SELECT camera FROM inserted_film WHERE uf_no=%s)", insertedFilmId);
String getLensIdSql = String.format("SELECT lens FROM user_lens WHERE ul_no=(SELECT lens FROM user_camera WHERE uc_no=(SELECT camera FROM inserted_film WHERE uf_no=%s
String getFilmIdSql = String.format("SELECT film FROM inserted_film WHERE uf_no=%s", insertedFilmId);


String AddPictureSql = String.format("INSERT INTO picture_meta (p_no, usr, camera, lens, film, target, shut, f_size, range, zoom, ev, iso) VALUES ("
                                    + "picture_meta_ai.NEXTVAL"
                                    +", (SELECT u_no FROM usr WHERE id='%s')"
                                    +", (%s)"
                                    +", (%s)"
                                    +", (%s)"
                                    +", '%s'"
                                    +", (SELECT ss_no FROM shutter_speed WHERE speed='%s')"
                                    +", (SELECT f_no FROM diaphragm WHERE f_size=%s)"
                                    +", '%s'"
                                    +", %s"
                                    +", %s"
                                    +", (SELECT iso_no FROM iso WHERE iso=%s))"
                                    , UserId
                                    , getCameraIdSql
                                    , getLensIdSql
                                    , getFilmIdSql
                                    , target
                                    , shutterSpeed
```

요청을 처리하는 페이지에서도 같은 방식으로 질의문을 구성하고, 질의문의 길이가 길어지면 질의문을 나눠서 구성한 뒤 합쳐줬다.

```
                                    +", (SELECT ss_no FROM shutter_spe
                                    +", (SELECT f_no FROM diaphragm WH
                                    +", '%s'"
                                    +", %s"
                                    +", %s"
                                    +", (SELECT iso_no FROM iso WHERE
                                    , UserId
                                    , getCameraIdSql
                                    , getLensIdSql
                                    , getFilmIdSql
                                    , target
                                    , shutterSpeed
                                    , diaphragm
                                    , range
                                    , zoom
                                    , EV
                                    , ISO);

try {
    db.runSql(AddPictureSql);
    out.print("success");
}
catch(SQLException e) {
    out.print("failed" + AddPictureSql);
}
```

구성된 질의문은 결과를 반환받지 않고 질의에서 에러가 나면 try-catch를 통해 에러를 처리해줬다.

클라이언트에서는 처리 결과를 성공(success)과 실패(failed)로 반환 받으며, 이 결과로 이후 동작을 실행해준다.

9. 결론 및 애로 사항

지금까지의 프로젝트에서 데이터베이스를 사용할 때는 머릿속으로 스키마를 구성하고 테이블을 생성했었다. 하지만 이렇게 구성된 테이블들은 코드를 작성하다보면 계속 구조를 바꾸게 되고, 비효율적인 구조가 되기 쉬웠다. 이번 프로젝트에서 프로그램의 요구사항을 정리하고 정리된 요구사항을 바탕

으로 E-R 다이어그램을 그리고(ER Win) 테이블의 구조를 확실히 만든 뒤에 이를 바탕으로 테이블을 생성하고 사용해보니 테이블의 구조 변경이 비교적 적었다.

하나의 프로젝트를 구현해나가면서 처음에 간단하게 머릿속에서 구성한 데이터가 어떻게 정규형 데이터베이스에 들어가게 되는지 확실히 알 수 있었다. 그리고 명확한 관계를 정의하여 내가 필요한 데이터 조합을 더 쉽게 구현할 수 있었다.

다만 데이터 구성은 항상 변할 수 있기 때문에 구조 변경에 대해서 자유롭지는 못했다. 프로그래밍을 하게 되면 코드 내에서 질의문을 구성하여 데이터베이스로부터 데이터를 받게 되는데, 테이블의 구조 변경이 있을 때 코드 내에서 찾아서 수정해 줘야 했다. 이러한 코드의 의존성 때문에 구조 변경에 대해 소극적으로 만들기도 했고, 이후 코드 수정을 더 힘들게 했다. 이런 문제를 해결하기 위해 코드와 질의문의 의존성을 해소하는 방법이 필요하다고 생각된다.