

데이터베이스 Term Project 최종 결과 보고서

분 반	3분반		
작 품 명	Health Machine Share		
개발기간	2023년 5 월 11 일 ~ 20 년 6 월 18 일		
지도교수	컴퓨터공학과	오병우	
구 분	학년	학 번	성 명
제출자	3	20190633	송제용
<p>본인은 데이터베이스 Term Project 최종 결과 보고서를 첨부와 같이 제출한다. 제출한 보고서는 본인이 직접 개발 및 작성하였으며, 거짓이나 부정이 있다면 F학점을 받고 학칙에 의거하여 처벌받겠습니다 (학적부에 등재).</p> <p style="text-align: right;">2023년 6월 23일</p> <p style="text-align: right;">제 출 자 송제용</p> <p style="text-align: center;">컴퓨터공학심화프로그램</p>			

최종 결과 보고서

Health Machine Share

0. 결과 요약

- (1) 3개 테이블 이상 Join 수행한 결과를 React (+MUI) 및 Express로 출력 구현 완료
 - ○
- (2) 이미지 저장 방법 및 출력
 - ○ 이미지 저장 방법 : 이미지 파일 이름 저장
- (3) 게시판 또는 블로그 형태 HTML 템플릿 사용
 - × 게시판 형태로 만들긴 했으나 템플릿을 사용하지 않았다.

1. 서론

- 개발 배경

현대 사회에서 헬스와 운동은 매우 중요한 가치를 갖게 됐다. 아래 사진과 같이 많은 사람들이 건강과 웰빙을 추구하며 헬스장을 이용하고 있다.



이에 따라 헬스 머신은 운동을 위해 필수적인 도구가 되었다. 그러나 아래 사진과 같이 헬스 머신의 다양성과 성능 차이로 인해 사용자들은 어떤 머신을 선택해야 하는지 혼란을 겪을 수 있다.



또한, 사용자들은 자신이 사용한 헬스 머신에 대한 경험과 평가를 공유하고 다른 사람들의 의견을 참고하고 싶다. 하지만 해당 기능들을 지원하는 웹사이트는 존재하지 않는다. 이러한 이유로 "Health Machine Share" 프로그램을 개발하게 되었다.

작품에 대한 설명

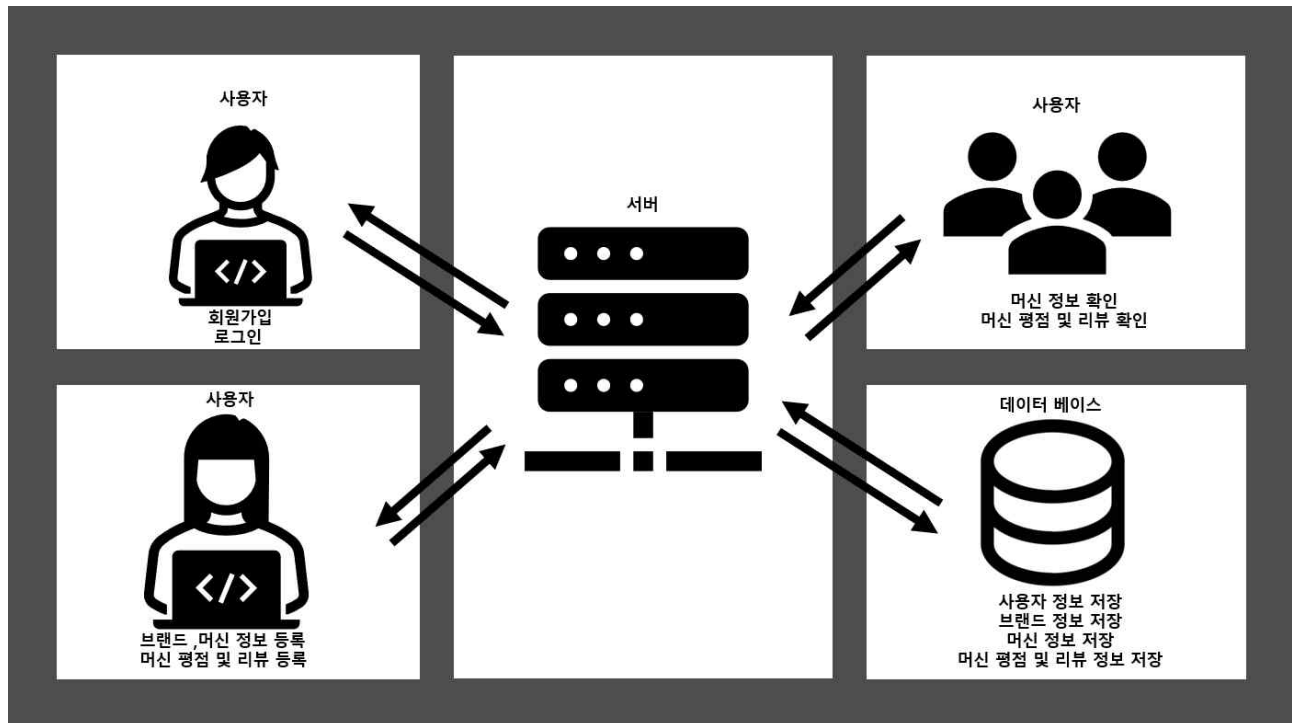
"Health Machine Share"는 사용자들이 헬스 머신에 대한 평가와 리뷰를 관리하고 공유할 수 있는 웹 기반 프로그램이다. 사용자들은 다양한 종류의 헬스 머신에 대한 평가와 리뷰를 작성하고, 다른 사용자들의 평가와 리뷰를 확인할 수 있다. 이를 통해 사용자들은 헬스 머신의 성능, 사용 편의성, 효과 등에 대한 정보를 얻을 수 있으며, 자신의 경험을 공유하고 다른 사람들과 소통할 수 있다.

작품명 결정 이유

"Health Machine Share"라는 작품명은 프로그램의 핵심 기능과 목적을 명확하게 표현한 것이다. "Health"는 건강과 관련된 의미를 가지고 있고, "Machine"은 헬스 머신을 의미한다. "Share"는 사용자들이 자신의 평가와 리뷰를 공유하는 기능을 강조한다. 간결하고 명확한 작품명을 선택함으로써 사용자들에게 프로그램의 내용과 목적을 쉽게 전달하고자 하였다.

2. 작품 개요

2.1 전체 구성도



- 사용자 관리 기능: 사용자는 회원 가입을 통해 개인 계정을 생성하고, 로그인하여 평점을 남길 수 있다.
- 브랜드 관리 기능: 헬스 머신 브랜드에 대한 정보를 관리한다. 브랜드의 이름, 국가, 설립 연도 및 브랜드 이미지를 저장하고 사용자에게 제공한다.
- 헬스 머신 관리 기능: 다양한 종류의 헬스 머신에 대한 정보를 관리한다. 헬스 머신의 브랜드, 이름, 설명, 머신 이미지 등을 저장하고 사용자에게 제공한다.
- 평점 및 리뷰 관리 기능: 사용자는 특정 헬스 머신에 대한 평점과 리뷰를 작성할 수 있다. 이를 통해 사용자들은 자신의 경험을 공유하고, 다른 사용자들은 평가와 리뷰를 확인하여 헬스 머신에 대한 정보를 얻을 수 있다. 또한 리뷰 삭제 기능을 제공한다.

2.2 설계구성요소 및 설계제한 요소

<설계구성요소 및 설계제한 요소>

설계 구성요소						설계 제한요소						
목표 설정	합성	분석	구현/ 제작	시험/ 평가	결과 도출	성능	규격/ 표준	경제 성	미 학	신뢰성	안정성/ 내구성	환경
○		○	○				○		○	○		○

설계 구성 요소

(1) 목표 설정

- 웹 기반의 헬스 머신에 대한 경험과 평가를 공유하는 게시판을 개발한다.
- 로그인 기능을 제공하여 사용자 식별과 보안을 유지한다.
- 사용자는 헬스 머신 정보에 대한 리뷰와 평점을 등록할 수 있다.

(2) 분석

선정한 취미는 헬스 머신이다. 사용자는 헬스 머신을 등록할 수 있고 해당 머신에 대해서 리뷰 및 평점을 등록할 수 있다. 해당 취미를 분석하여 필요한 테이블이 무엇인지 알게 되었다.

- users: 사용자 정보(아이디, 비밀번호, 생성일자)
- brands: 브랜드 정보(이름, 국가, 창립년도, 이미지)
- machines: 머신 정보(이름, 설명, 이미지, 브랜드 번호)
- ratings_reviews: 평점 및 리뷰 정보(평점, 리뷰, 사용자 번호, 머신 번호)

(3) 구현/제작

- 프론트엔드: React를 사용하여 사용자 인터페이스를 구현한다.
- 백엔드: Express.js를 사용하여 서버를 구축하고 데이터베이스와의 상호작용을 처리한다.
- 데이터베이스: MySQL을 사용하여 테이블을 생성하고 정규화 규칙 BCNF에 기반하여 설계한다.
- Join 연산을 활용하여 테이블 간의 관계를 구성하고 데이터를 출력한다.

설계 제한 요소

(1) 규격/표준

- HTML과 JSON을 사용한다. 테이블은 총 4개 이상 사용한다. Join 연산을 활용하여 테이블을 조인한다.

- 사용하는 테이블

users : 사용자 이름(아이디)(아이디), 비밀번호 , 생성일자를 저장한다. 생성일자는 자동으로 저장한다.

brands : 브랜드 이름, 국가, 창립년도, 이미지를 저장한다.

machines : 머신 이름, 설명, 이미지, 브랜드 번호를 저장한다.

ratings_reviews: 평점, 리뷰, 유저 번호, 머신 번호를 저장한다.

join 연산은 machines의 브랜드 번호를 이용하여 brands와 join하여 머신 정보를 출력할 것이고

ratings_reviews의 유저 번호와 머신번호를 이용하여 users와 machines를 join하여 평점 및 리뷰 정보를 출력할 것이다.

(2) 미학

- React의 Material-UI를 활용하여 미려하고 직관적인 사용자 인터페이스를 설계한다.
- 이미지 출력은 이미지 경로를 사용하여 사용자에게 이미지를 제공한다.

(3) 신뢰성

- 데이터베이스를 반영하여 웹페이지로부터 데이터를 삽입하는 사용자 정보, 브랜드 정보, 머신 정보, 평점 및 리뷰 정보를 등록하는 기능을 구현할 것이다.

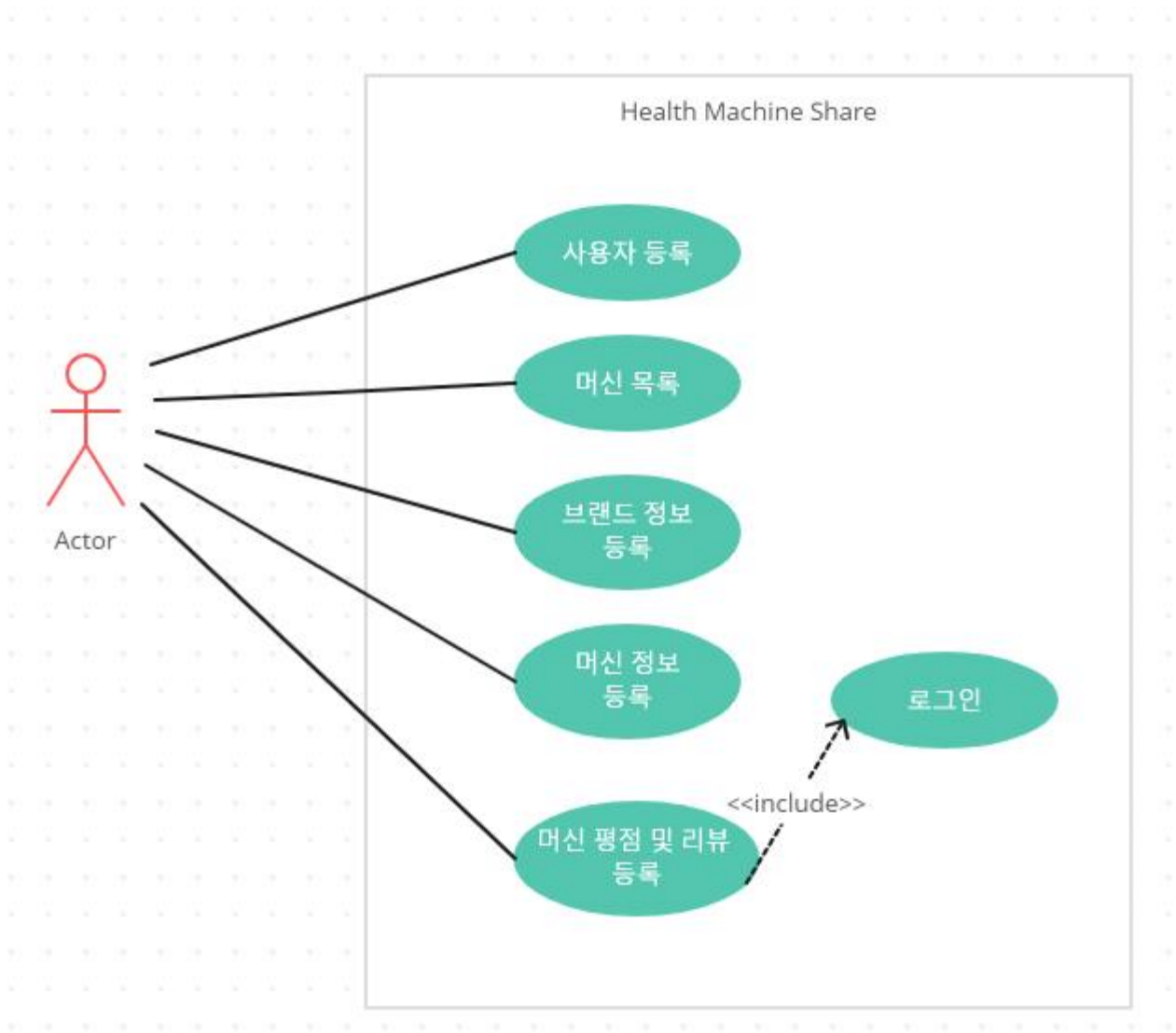
(4) 환경

- React, Express, MySQL, JSON을 사용하여 웹 환경에서 개발 및 구현한다.

3. 설계

3.1 Use Case

- Use Case 다이어그램



3.1.1 메인 화면

성공 시나리오

- 사용자가 메인 화면 페이지에 접속한다.
- 사용자는 회원 등록 화면 버튼을 눌러 사용자 등록 페이지로 이동할 수 있다.
- 사용자는 머신 목록 화면 버튼을 눌러 머신 목록 페이지로 이동할 수 있다.
- 사용자는 브랜드 등록 화면 버튼을 눌러 브랜드 등록 페이지로 이동할 수 있다.
- 사용자는 머신 등록 화면 버튼을 눌러 머신 등록 페이지로 이동할 수 있다.

3.1.2 사용자 등록

성공 시나리오

- 사용자가 프로그램의 사용자 등록 페이지에 접속한다.
- 사용자가 사용자 이름(아이디), 비밀번호를 입력한다.
- 사용자가 등록 양식을 제출한다.
- 프로그램은 입력된 정보를 확인하고, 중복된 사용자 이름(아이디)이 있는지 확인한다.
- 입력된 정보가 유효하고 고유한 경우, 프로그램은 새로운 사용자 계정을 생성한다.
- 프로그램은 등록된 사용자에게 고유한 사용자 ID를 생성한다.
- 프로그램은 사용자 이름(아이디), 비밀번호를 데이터베이스의 "users" 테이블에 저장한다.
- 프로그램은 사용자에게 등록이 성공적으로 완료되었다고 알려주기 위해 입력 칸을 비운다.
- 이제 사용자는 등록한 사용자 이름(아이디)과 비밀번호를 사용하여 머신 평점 및 리뷰를 남길 수 있다.

실패 시나리오

- 필수 정보가 누락된 경우, 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.
- 입력된 사용자 이름(아이디)이 이미 데이터베이스에 존재하는 경우, 프로그램은 사용자에게 오류 메시지를 표시하고 다른 사용자 이름(아이디)을 입력하도록 안내한다.

3.1.3 머신 목록 보기

성공 시나리오

- 사용자가 머신 목록 페이지에 접속한다.
- 여러 머신의 사진과 간단한 정보가 보여진다.
- 각 머신은 사진과 함께 제목, 요약 정보 등을 포함한다.

3.1.4 브랜드 정보 등록

성공 시나리오

- 사용자는 브랜드 정보 등록을 위한 메인화면에서 브랜드 정보 등록 화면으로 이동한다.
- 사용자는 새로운 브랜드의 로고를 업로드한다. (이미지 파일 형식: JPG, PNG 등)
- 사용자는 브랜드의 이름, 설명 등의 정보를 입력한다.
- 사용자는 등록 양식을 제출한다.
- 프로그램은 입력된 정보를 확인하고, 필수 정보가 누락되지 않았는지 검사한다.
- 프로그램은 머신의 정보를 데이터베이스의 "brands" 테이블에 저장하고, 브랜드에 대한 고유한 브

랜드 ID를 생성한다.

실패 시나리오

- 필수 정보가 누락된 경우, 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.
- 이미 동일한 브랜드 이름이 데이터베이스에 존재하는 경우 프로그램은 오류 메시지를 표시하도록 한다.

3.1.5 머신 정보 등록

성공 시나리오

- 사용자는 머신 정보 등록을 위한 메인화면에서 정보 등록 화면으로 이동한다.
- 사용자는 새로운 머신의 사진을 업로드한다. (이미지 파일 형식: JPG, PNG 등)
- 사용자는 머신의 브랜드, 모델명, 설명 등의 정보를 입력한다.
- 사용자는 등록 양식을 제출한다.
- 프로그램은 입력된 정보를 확인하고, 필수 정보가 누락되지 않았는지 검사한다.
- 입력된 정보가 유효하고 완전한 경우, 프로그램은 새로운 머신을 등록한다.
- 프로그램은 머신의 정보를 데이터베이스의 “machines” 테이블에 저장하고, 머신에 대한 고유한 머신 ID를 생성한다.
- 등록된 머신의 사진과 정보가 프로그램에서 표시되고 사용자들이 해당 정보를 확인할 수 있다.

실패 시나리오

- 필수 정보가 누락된 경우, 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.

3.1.6 리뷰 목록 보기 및 머신 평점 리뷰 등록 및 삭제

성공 시나리오

- 사용자는 머신 목록 화면에서 해당 머신의 사진을 클릭하여, 해당 머신의 평점 및 리뷰 페이지로 이동하여서 "평가 및 리뷰 작성" 버튼을 클릭하여 평가 및 리뷰 작성 페이지로 이동한다.
- 사용자는 자신의 이름(아이디)을 선택하고 비밀번호를 입력한다.
- 사용자는 리뷰를 작성할 수 있는 텍스트 입력란에 머신에 대한 자세한 의견과 경험을 기록한다.
- 사용자는 머신에 대한 평점 또한 입력한다.
- 사용자는 작성한 평가와 리뷰를 제출한다.
- 프로그램은 사용자가 유저의 이름과 비밀번호를 이용하여 등록된 사용자인지 검사하고 만약 등록된

사용자라면 작성한 평가와 리뷰를 확인하고, 데이터베이스에 저장한다.

- 등록된 평가와 리뷰는 해당 머신의 상세 페이지에서 사용자들에게 표시되고, 다른 사용자들은 이를 확인할 수 있다.
- 등록된 평가와 리뷰는 해당 머신의 리뷰 페이지에서 삭제 버튼을 눌러 삭제할 수 있다.

실패 시나리오

- 필수 정보가 누락된 경우 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.
- 사용자가 잘못된 비밀번호를 입력할 경우, 프로그램은 사용자에게 오류 메시지를 표시하고 제대로 된 비밀번호를 입력하도록 안내한다.

3.2 UI Design

3.2.1 메인 화면 (APP.js)

메인 화면

회원 등록 화면

머신 목록 화면

브랜드 등록화면

머신 등록 화면

- 메인 화면에 접근하면 회원등록, 머신 목록, 브랜드 등록, 머신 등록에 버튼을 눌러 접근할 수 있다.

3.1.2 사용자 등록 (HealthMachineUserRegisterTable.js)

회원 등록 화면

머신 목록 화면

브랜드 등록화면

머신 등록 화면

사용자 등록

사용자 이름 입력

비밀번호 입력

등록

- 사용자 등록 화면에서는 사용자 이름과 비밀번호를 입력하여 사용자를 등록할 수 있다.

3.2.3 머신 목록 보기 (HealthMachineTable.js)

회원 등록 화면
머신 목록 화면
브랜드 등록화면
머신 등록 화면

머신 이름	머신 설명	머신 이미지	브랜드 이름	브랜드 창립 년도	브랜드 이미지
바벨로우	등 운동		라이프 피트니스	1978	
랫플다운	등 운동		해머스트렝스	1960	
체스트 프레스	가슴 운동		사이백스	1989	

- 머신 목록 보기에서는 사용자가 입력한 머신들의 정보를 볼 수 있다. 각각의 머신들에 대해서 머신 이름, 머신 설명, 머신 이미지, 브랜드 이름, 브랜드 창립년도, 브랜드 이미지를 제공해준다.

3.2.4 브랜드 등록 (HealthMachineBrandTable.js)

회원 등록 화면
머신 목록 화면
브랜드 등록화면
머신 등록 화면

헬스 머신 브랜드 등록

브랜드 이름 입력

브랜드 국가 입력

창립 년도 입력

파일 선택

선택된 파일 없음

등록

- 헬스 머신 브랜드 등록 화면에서는 브랜드를 등록할 수 있다. 사용자가 브랜드 이름, 국가, 창립년도, 로고 이미지를 제공해주면 브랜드가 저장된다.

3.2.5 머신 정보 등록 (HealthMachineRegisterTable.js)

회원 등록 화면

머신 목록 화면

브랜드 등록화면

머신 등록 화면

헬스 머신 정보 등록

브랜드 선택

머신 이름 입력

머신 설명 입력

파일 선택 선택된 파일 없음

등록

- 머신 정보 등록 화면에서는 사용자가 머신 브랜드를 결정하고 머신 이름, 머신 설명, 머신 이미지를 입력하면 머신 정보를 등록할 수 있다.

3.2.6 머신 리뷰 및 평점 목록 보기 / 리뷰 삭제 (HealthMachineReviewRegisterTable.js)

회원 등록 화면

머신 목록 화면

브랜드 등록화면

머신 등록 화면

번호	머신 이름	사용자 이름	리뷰	평점	
1	랫풀다운	송제용	등이 잘 먹어서 좋아요	★★★★☆	삭제
2	랫풀다운	송송송	다른 브랜드가 더 좋아요	★☆☆☆☆	삭제
3	랫풀다운	용용용	그냥 저장.. 할만한듯	★★★★☆	삭제

- 사용자는 머신 목록 보기에서 머신을 선택하여 머신 리뷰 및 평점 목록을 볼 수 있다. 머신 리뷰 및 평점 목록에서는 평점 번호, 머신 이름, 사용자 이름, 리뷰, 평점을 제공해준다 또한 삭제 버튼을 눌러 해당 리뷰를 삭제 가능하다

3.2.7 머신 리뷰 및 평점 등록 (HealthMachineReviewTable.js)

회원 등록 화면

머신 목록 화면

브랜드 등록화면

머신 등록 화면

평점 및 리뷰 등록

유저 선택

비밀번호

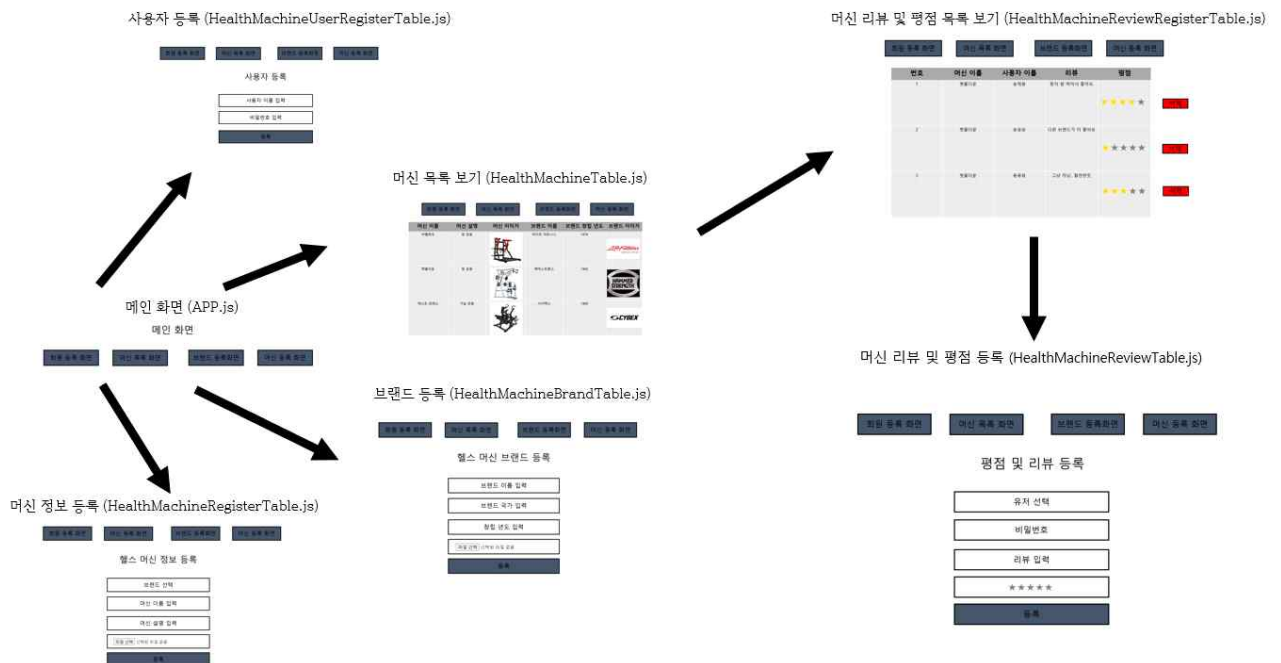
리뷰 입력

★★★★★

등록

- 평점 및 리뷰 등록 화면에서는 유저를 선택하여 비밀번호를 입력하고 리뷰를 남기고 평점을 선택하여 머신에 대해서 평점 및 리뷰를 등록할 수 있다. 만약 비밀번호가 틀릴 경우 오류 메시지를 표시해 사용자에게 올바른 비밀번호를 입력할 수 있도록 해준다.

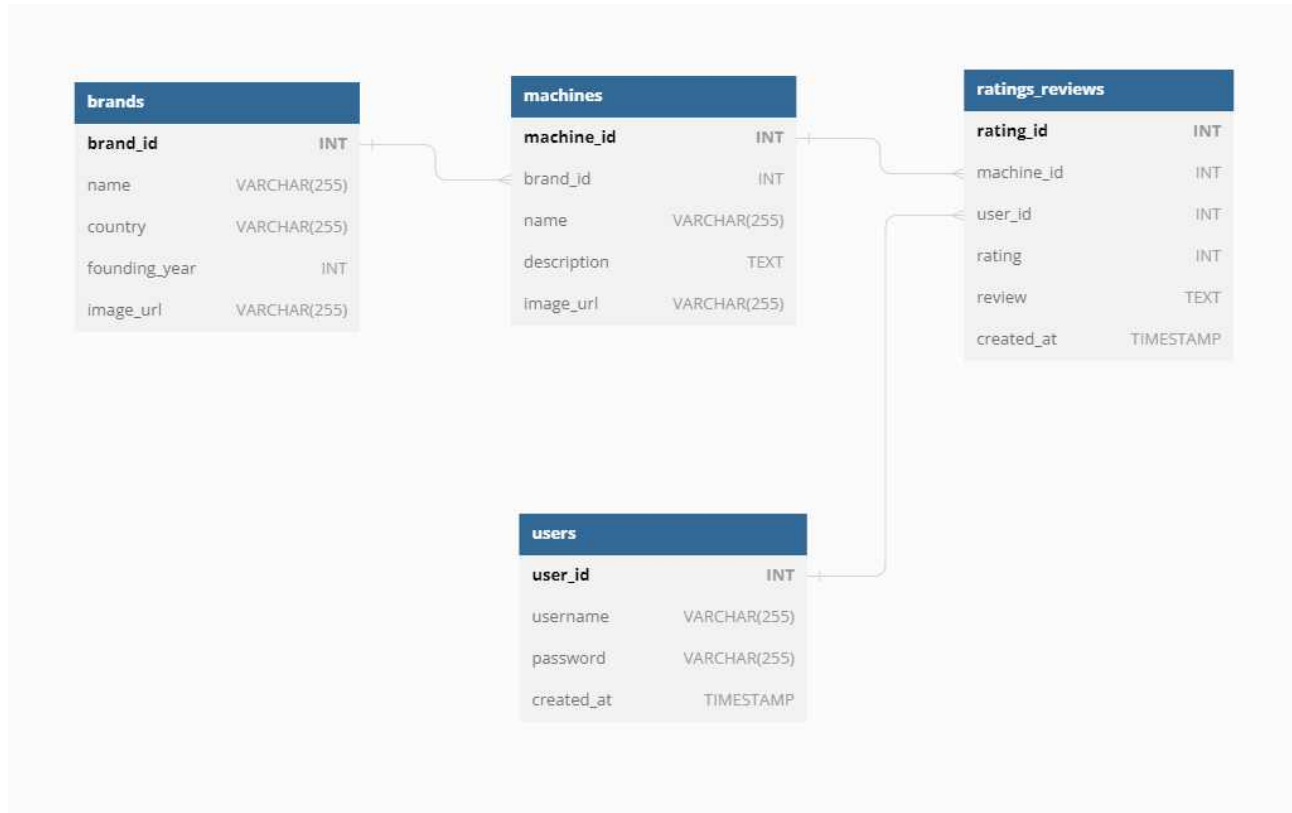
3.2.8 전체 흐름도



-
- 메인화면에서 사용자 등록화면, 머신 정보등록화면, 머신 목록 보기 화면, 브랜드 등록 화면으로 버튼을 눌러 이동할 수 있으며 머신 목록 보기에서 각각의 머신들에 대한 정보를 클릭하여 머신 리뷰 및 평점 목록 보기 화면으로 넘어갈 수 있다. 또한 머신 리뷰 및 평점 보기 화면에서 리뷰 및 평점 등록 화면을 통해서 평점 및 리뷰 등록 화면으로 넘어갈 수 있다.
 - 해당 방법 말고도 사용자의 편의성을 위해서 이동할 일이 많은 사용자 등록화면, 머신 정보등록화면, 머신 목록 보기 화면, 브랜드 등록 화면으로 이동하는 버튼은 모든 화면에서 만들어 두어 어디에서나 이동할 수 있도록 하였다.

3.3 Conceptual Design

- Crow's foot 다이어그램



Health Machine Share 프로그램에 대한 Crow's foot 다이어그램이다. 사용자는 리뷰를 여러개 쓸 수 있기 때문에 사용자는 리뷰와 1: N 관계이다. 브랜드는 여러개의 머신에 등록 될 수있기 때문에 브랜드와 머신은 1:N 관계이다. 머신은 여러개의 리뷰가 작성될 수 있기 때문에 머신과 리뷰는 1:N 관계이다.

모든 테이블에서 기본 키를 제외한 나머지 속성들이 후보 키에 종속되지 않는다. 또한 각 테이블의 모든 결정자가 후보 키에 속하는 정규화된 형태로 정의되어 있다. 따라서, 주어진 코드는 BCNF 규칙을 만족한다.

3.4 Logical Design

사용자 테이블 생성

```
mysql> DESC users;
```

Field	Type	Null	Key	Default	Extra
user_id	int(11)	NO	PRI	NULL	auto_increment
username	varchar(255)	NO		NULL	
password	varchar(255)	NO		NULL	
created_at	timestamp	NO		CURRENT_TIMESTAMP	

```
CREATE TABLE users (  
  user_id INT AUTO_INCREMENT PRIMARY KEY,  
  username VARCHAR(255) NOT NULL,  
  password VARCHAR(255) NOT NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Brands 테이블 생성

```
mysql> DESC brands;
```

Field	Type	Null	Key	Default	Extra
brand_id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
country	varchar(255)	NO		NULL	
founding_year	int(11)	NO		NULL	
image_name	varchar(255)	YES		NULL	

```
CREATE TABLE brands (  
  brand_id INT AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(255) NOT NULL,  
  country VARCHAR(255) NOT NULL,  
  founding_year INT NOT NULL,  
  image_name VARCHAR(255)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Machines 테이블 생성

```
mysql> DESC machines;
```

Field	Type	Null	Key	Default	Extra
machine_id	int(11)	NO	PRI	NULL	auto_increment
brand_id	int(11)	NO	MUL	NULL	
name	varchar(255)	NO		NULL	
description	text	YES		NULL	
image_name	varchar(255)	YES		NULL	

```
CREATE TABLE machines (
  machine_id INT AUTO_INCREMENT PRIMARY KEY,
  brand_id INT NOT NULL,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  image_name VARCHAR(255),
  FOREIGN KEY (brand_id) REFERENCES brands(brand_id) ON DELETE CASCADE ON UP
DATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

평점 및 리뷰 테이블 생성

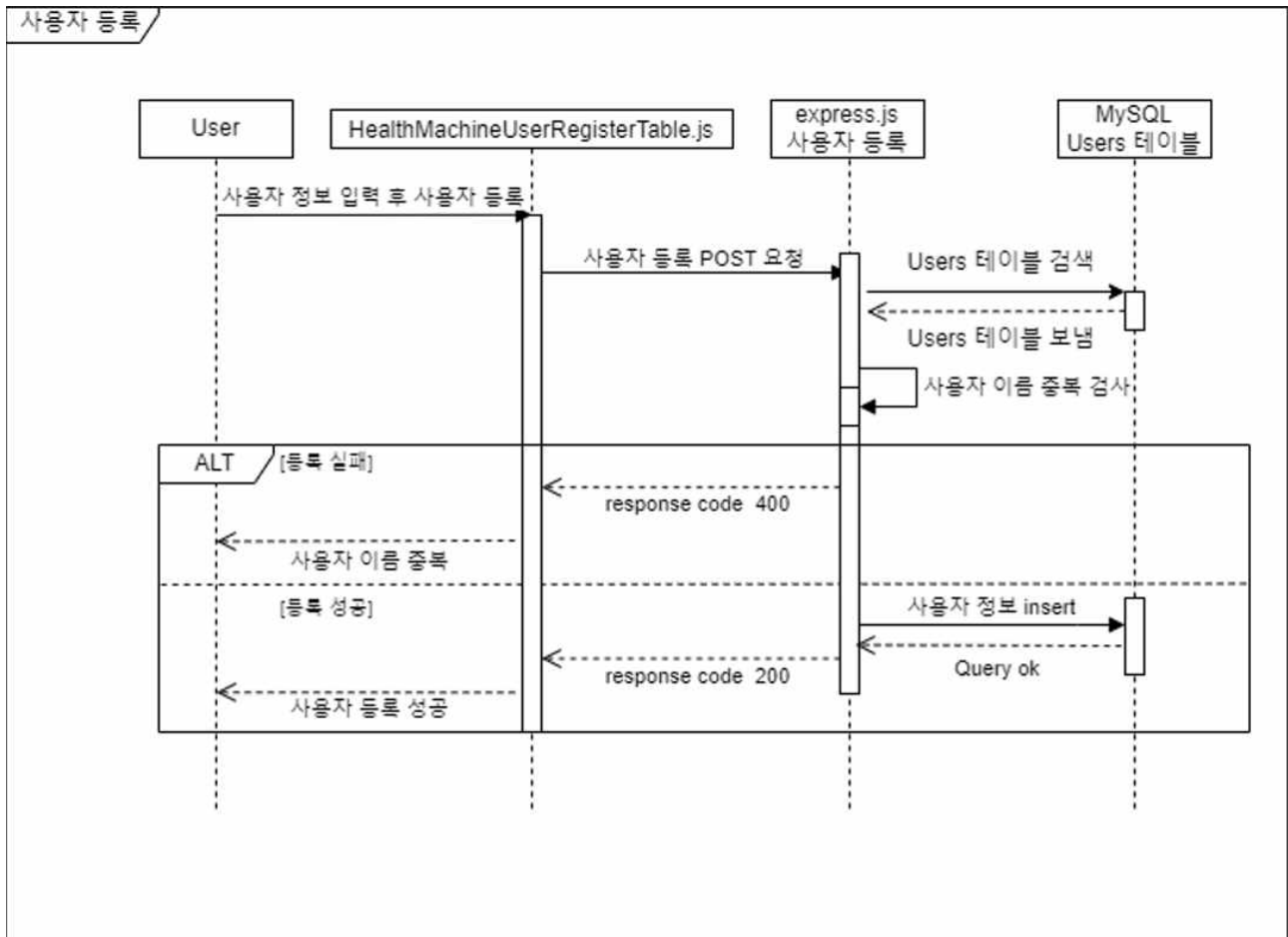
```
mysql> DESC ratings_reviews;
```

Field	Type	Null	Key	Default	Extra
rating_id	int(11)	NO	PRI	NULL	auto_increment
machine_id	int(11)	NO	MUL	NULL	
user_id	int(11)	NO	MUL	NULL	
rating	int(11)	NO		NULL	
review	text	NO		NULL	
created_at	timestamp	NO		CURRENT_TIMESTAMP	

```
CREATE TABLE ratings_reviews (
  rating_id INT AUTO_INCREMENT PRIMARY KEY,
  machine_id INT NOT NULL,
  user_id INT NOT NULL,
  rating INT NOT NULL,
  review TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (machine_id) REFERENCES machines(machine_id) ON DELETE CASCADE
ON UPDATE CASCADE,
  FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE ON UPDAT
E CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

3.5 Sequence Diagram

- 사용자 등록 Sequence Diagram

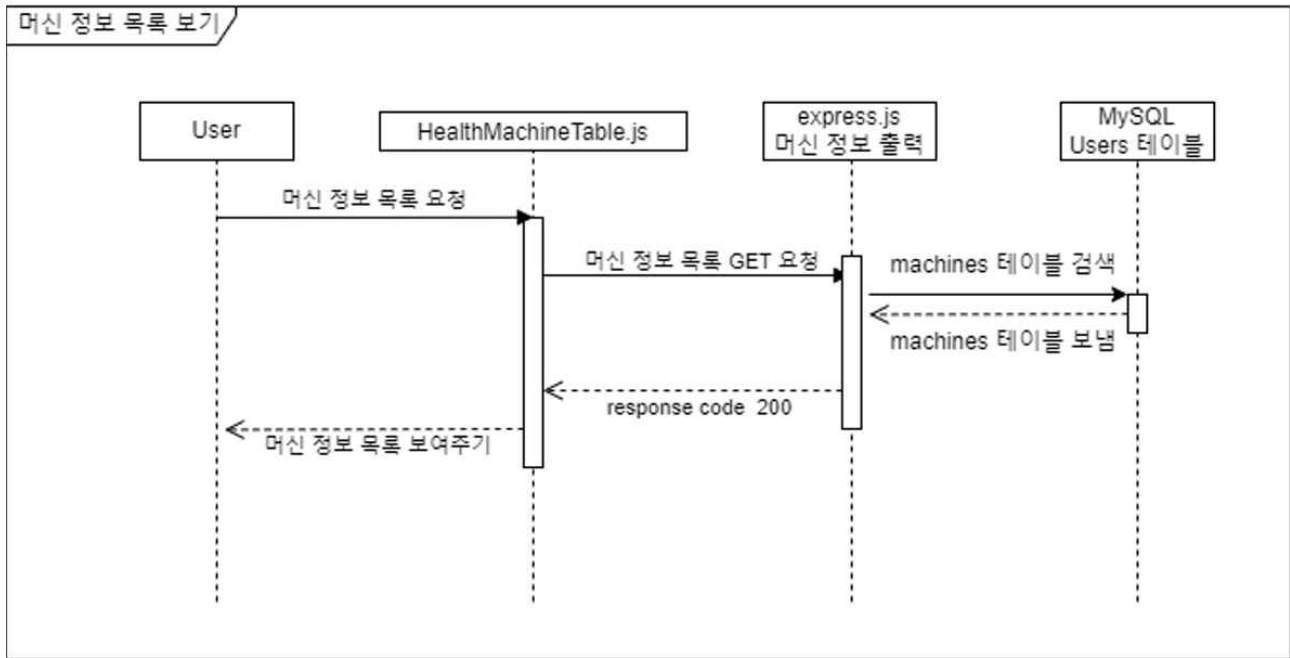


사용자가 사용자 정보 입력 후 사용자 등록을 하면 HealthMachineUserRegisterTable.js(프론트)에서 사용자 등록을 위해 express.js(백엔드)로 post 요청을 전송해준다. express.js(백엔드)에서 사용자 등록을 수행하는 API는 MySQL(데이터베이스)로 User 테이블을 검색하여 사용자 정보들을 들고온다. 그리고 express.js(백엔드)의 사용자 등록 API는 사용자 이름(아이디) 중복검사를 실행한다.

사용자 이름(아이디)이 중복될 경우 400 Bad Request를 보내주고 HealthMachineUserRegisterTable.js(프론트)는 해당 정보를 받아 사용자에게 사용자 이름(아이디)이 중복되어 사용자 등록이 실패했다고 알려준다.

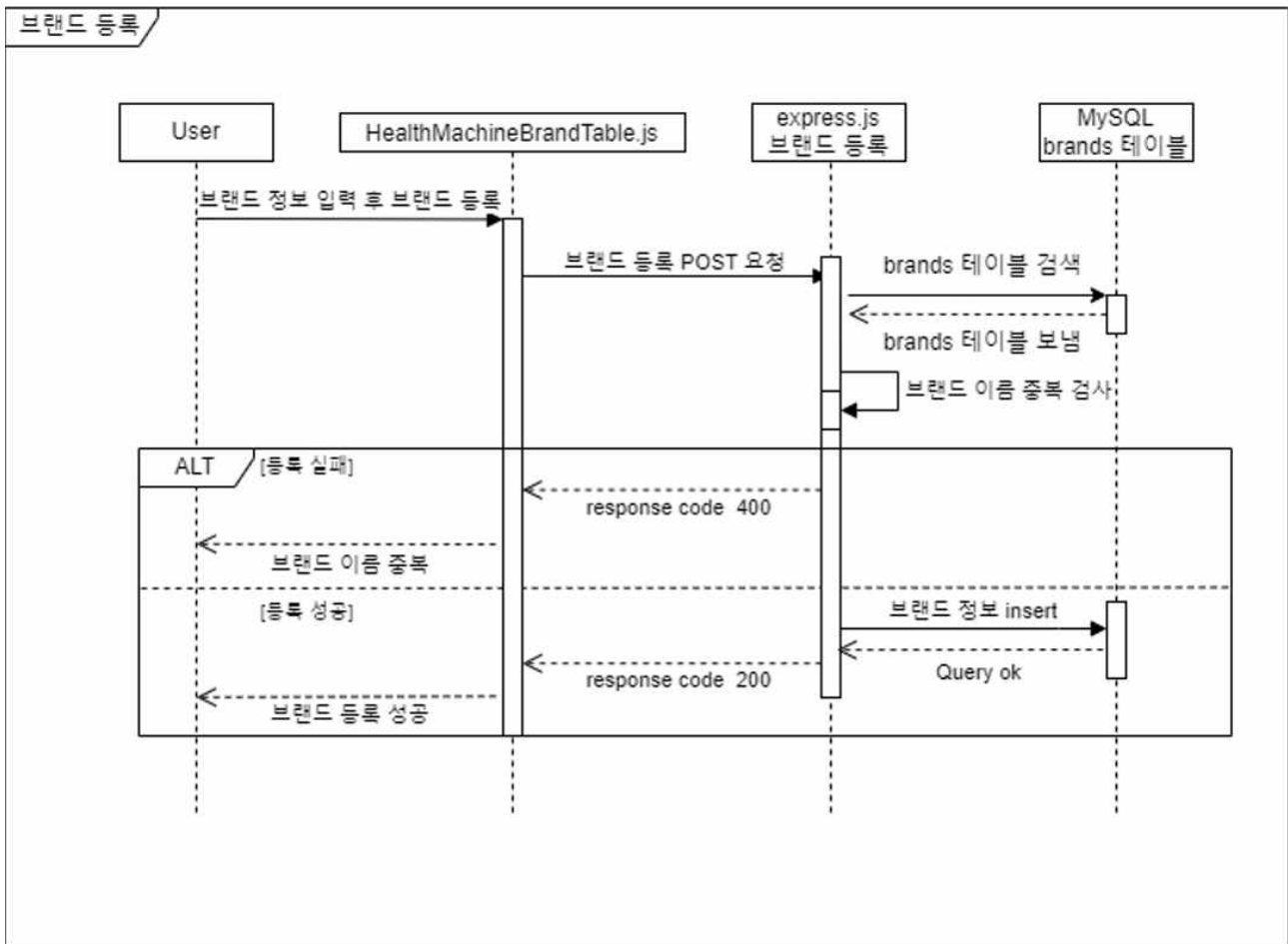
사용자 이름(아이디)이 중복되지 않았을 경우 express.js(백엔드)의 사용자 등록 API에서 200 OK를 보내주고 HealthMachineUserRegisterTable.js(프론트)는 해당 정보를 받고 사용자 등록을 성공했다고 사용자에게 알려준다.

- 머신 정보 목록 보기 Sequence Diagram



사용자가 머신 정보 목록을 요청하면 HealthMachineTable.js(프론트)에서 express.is(백엔드)로 머신 정보 목록 GET 요청을 한다. 그럼 express.is(백엔드)의 머신 정보 목록을 출력하는 API에서 MySQL(데이터베이스)로 machines 테이블에 검색하여 정보들을 받아온다. 그리고 HealthMachineTable.js(프론트)로 200 ok와 머신 목록을 보내준다. HealthMachineTable.js(프론트)는 해당 정보를 받은 뒤 사용자에게 머신 정보 목록을 보여준다.

- 브랜드 등록 Sequence Diagram

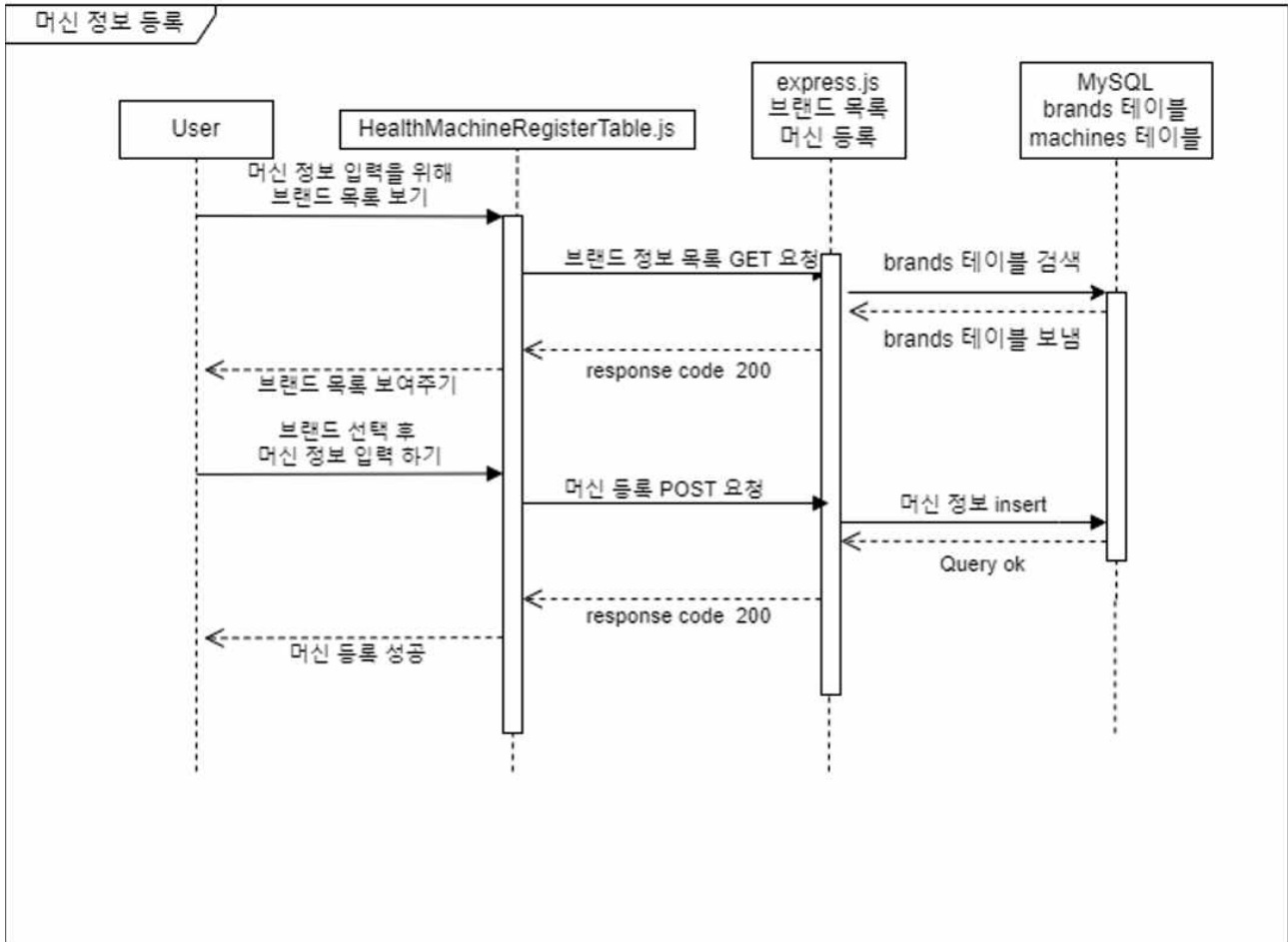


사용자가 브랜드 정보 입력 후 브랜드 등록을 하면 HealthMachineBrandTable.js(프론트)에서 브랜드 등록을 위해 express.js(백엔드)로 post 요청을 전송해준다. express.js(백엔드)에서 브랜드 등록을 수행하는 API는 MySQL(데이터베이스)로 brands 테이블을 검색하여 브랜드 정보들을 들고온다. 그리고 express.js(백엔드)의 브랜드 등록 API는 브랜드 이름 중복검사를 실행한다.

브랜드 이름이 중복될 경우 400 Bad Request를 보내주고 HealthMachineBrandTable.js(프론트)는 해당 정보를 받아 사용자에게 브랜드 이름이 중복되어 브랜드 등록이 실패했다고 알려준다.

브랜드 이름이 중복되지 않았을 경우 express.js(백엔드)의 브랜드 등록 API에서 200 OK를 보내주고 HealthMachineBrandTable.js(프론트)는 해당 정보를 받고 브랜드 등록을 성공했다고 사용자에게 알려준다.

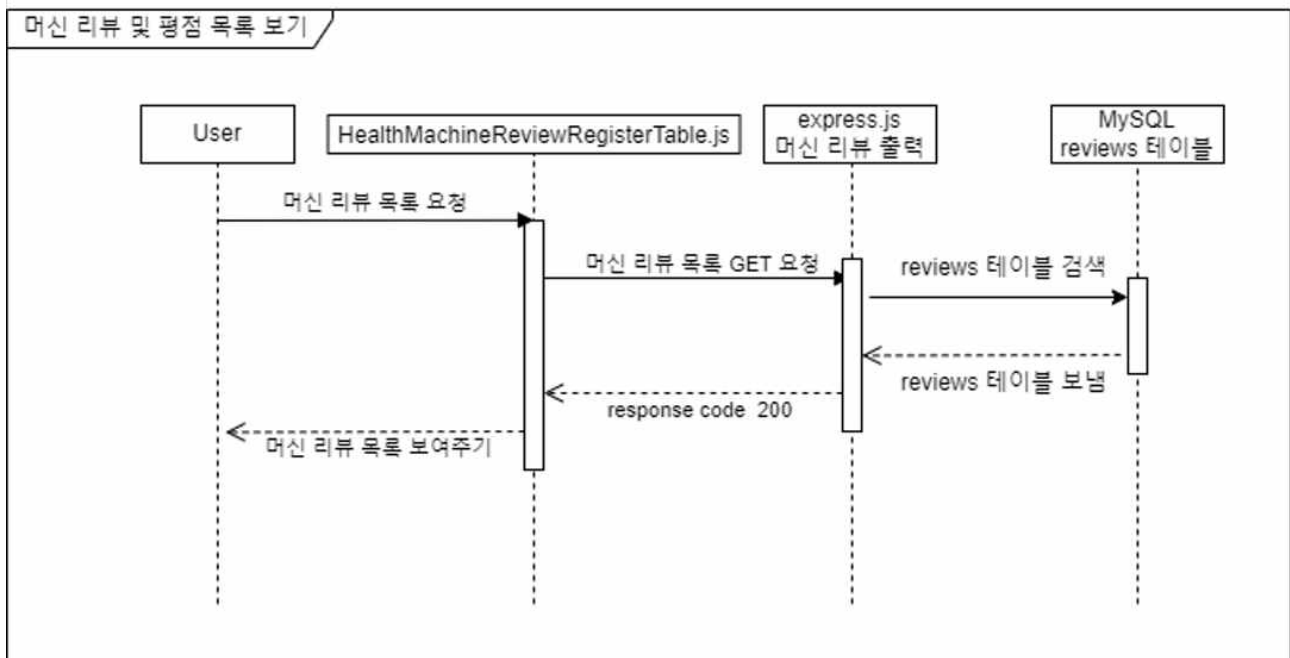
- 머신 정보 등록 Sequence Diagram



사용자가 머신 정보 입력을 위해서 브랜드 목록보기를 하면 HealthMachineRegisterTable.js(프론트)는 express.js(백엔드)로 브랜드 정보 목록 GET 요청을 보낸다. express.js(백엔드)의 브랜드 목록 보여주는 기능을 수행하는 API는 해당 요청을 받은 뒤 MySQL(데이터베이스)에 brands 테이블 검색을 하여 저장된 brands 정보를 받아온다. 그리고 HealthMachineRegisterTable.js(프론트)에 200 ok 와 brands 정보를 보내준다. HealthMachineRegisterTable.js(프론트)는 해당 정보를 받은뒤 사용자에게 브랜드 목록을 보여준다.

사용자가 브랜드 선택 후 머신 정보 입력 후 머신 정보 등록을 하면HealthMachineRegisterTable.js (프론트)에서 머신 등록을 위해 express.js(백엔드)로 post 요청을 전송해준다. express.js(백엔드)에서 머신 등록을 수행하는 API는 MySQL(데이터베이스)로 machines 테이블에 사용자가 입력해준 정보를 입력한다. express.js(백엔드)의 머신 등록 API에서 200 OK를 보내주고 HealthMachineRegisterTable.js(프론트)는 해당 정보를 받고 머신 등록을 성공했다고 사용자에게 알려준다. 참고로 머신 등록은 머신 이름에 대해 중복 검사를 시행하지 않는다. 왜냐하면 헬스 머신의 특성상 같은 이름으로 나오는 제품이 많기 때문이다.

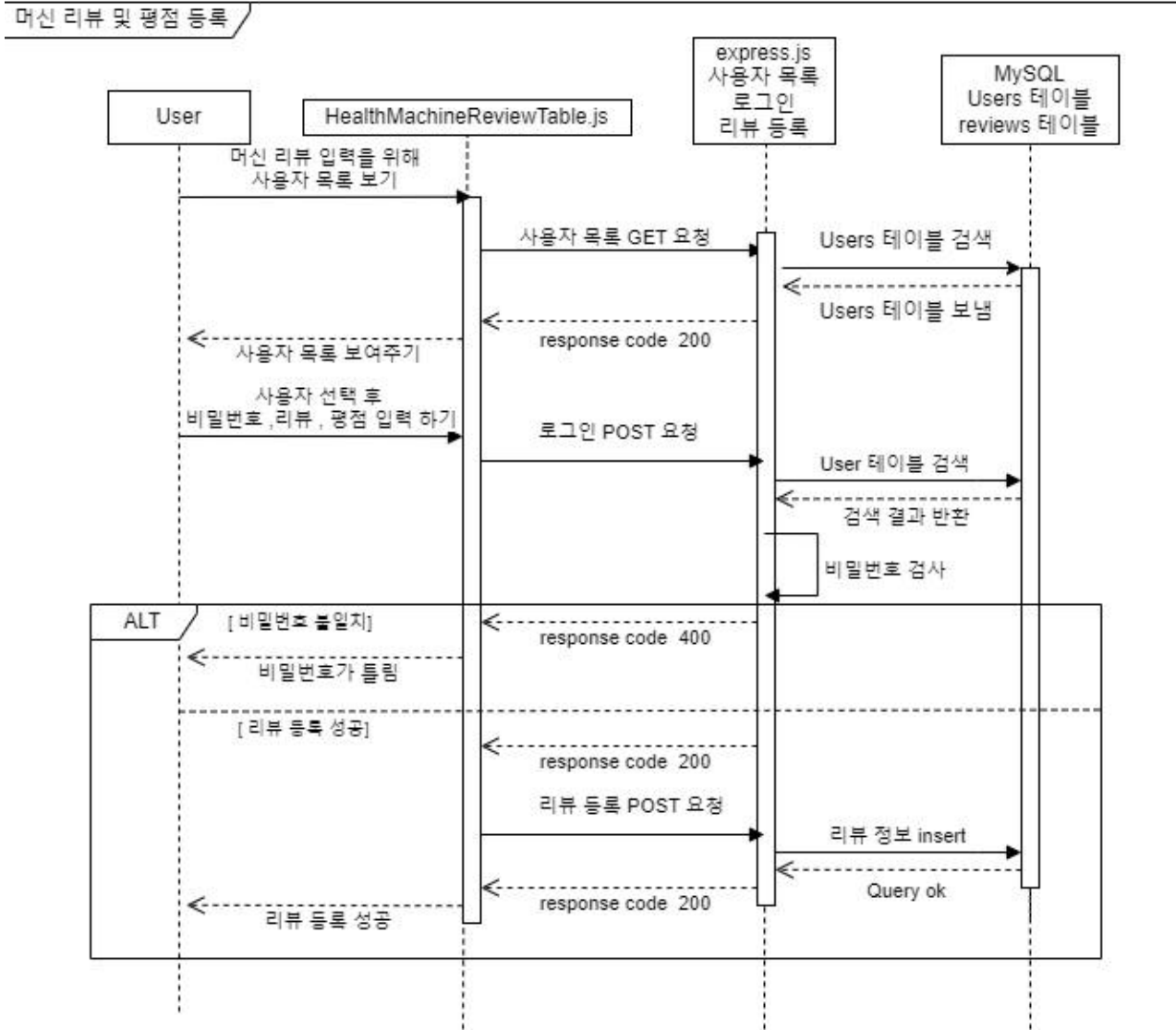
- 머신 리뷰 및 평점 목록 보기 Sequence Diagram



사용자가 머신 리뷰 및 평점 정보 목록을 요청하면 HealthMachineReviewRegisterTable.js(프론트)에서 express.is(백엔드)로 머신 리뷰 목록 GET 요청을 한다.

express.is(백엔드)는 해당 요청을 받은 뒤 머신 리뷰 목록을 출력하는 API에서 MySQL(데이터베이스)로 reviews 테이블에 검색하여 정보들을 받아온다. 그리고 HealthMachineReviewRegisterTable.js(프론트)로 200 ok와 리뷰 목록을 보내준다. HealthMachineReviewRegisterTable.js(프론트)는 해당 정보를 받은 뒤 사용자에게 머신 리뷰 및 평점 목록을 보여준다.

- 머신 리뷰 및 평점 등록 Sequence Diagram



사용자가 머신 리뷰 입력을 위해서 사용자 목록보기를 하면 HealthMachineReviewTable.js(프론트)는 express.js(백엔드)로 사용자 정보 목록 GET 요청을 보낸다. express.js(백엔드)의 사용자 목록 보여주는 기능을 수행하는 API는 해당 요청을 받은 뒤 MySQL(데이터베이스)에 Users 테이블 검색을 하여 저장된 사용자 정보를 받아온다. 그리고 HealthMachineReviewTable.js(프론트)에 200 ok 와 사용자 정보를 보내준다. HealthMachineReviewTable.js(프론트)는 해당 정보를 받은뒤 사용자에게 사용자 목록을 보여준다.

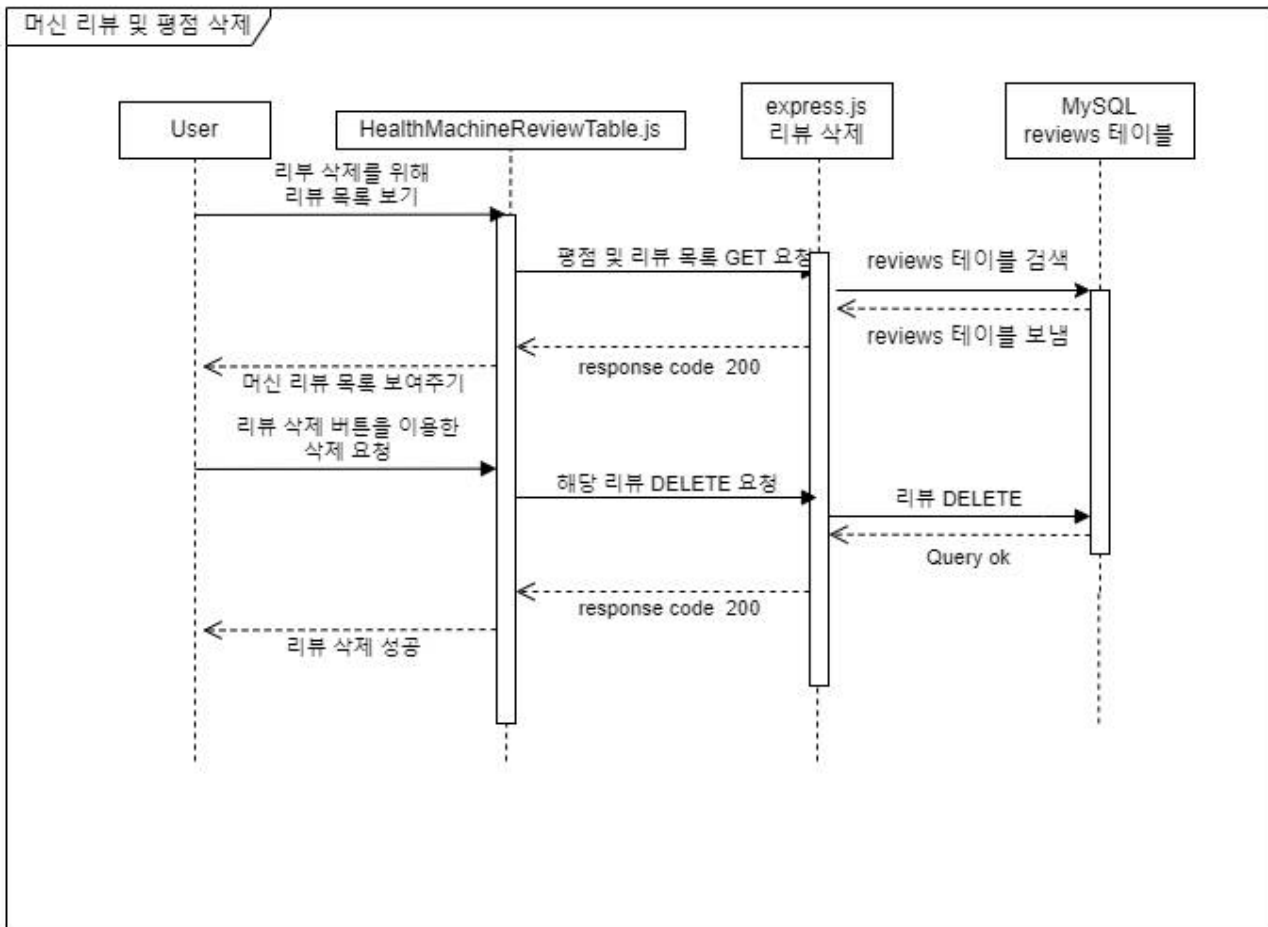
사용자가 사용자 선택 후 비밀번호, 리뷰, 평점 입력 후 머신 리뷰 및 평점 등록을 수행 하면 Health MachineReviewTable.js(프론트)에서 머신 리뷰 및 평점 등록을 위해 express.js(백엔드)로 로그인 post 요청을 전송해준다. express.js(백엔드)에서 로그인 기능을 수행하는 API는 MySQL(데이터베이스)로 Users 테이블에 사용자가 입력해준 정보로 검색한다. MySQL(데이터베이스)에서 받아온 정보를 바탕으로 비밀번호 검사를 수행한다.

사용자가 입력한 비밀번호가 저장된 비밀번호와 일치하지 않을 경우 express.js(백엔드)의 로그인 API 에서 400 Bad Request를 보내준다. HealthMachineReviewTable.js(프론트)에서 해당 정보를 받은

뒤 사용자에게 비밀번호가 틀렸다고 알려준다.

사용자가 입력한 비밀번호가 저장된 비밀번호와 일치할 경우 express.js(백엔드)의 로그인 API에서 200 OK를 보내주고 HealthMachineReviewTable.js(프론트)는 해당 정보를 받고 다시 MySQL(데이터베이스)로 사용자가 입력한 리뷰, 평점 정보를 express.js(백엔드)에 POST해준다. express.js(백엔드)에서 리뷰 등록 기능을 수행하는 API는 리뷰, 평점 정보를 받아 MySQL(데이터베이스)의 reviews 테이블에 리뷰 정보를 입력해준다. 그리고 200 Ok를 보내준다. HealthMachineReviewTable.js(프론트)는 해당 정보를 받은 뒤 머신 리뷰 및 평점 등록을 성공했다고 사용자에게 알려준다.

- 머신 리뷰 및 평점 삭제 Sequence Diagram



사용자가 리뷰 및 평점 정보 목록을 요청하면 HealthMachineReviewRegisterTable.js(프론트)에서 express.js(백엔드)로 머신 리뷰 목록 GET 요청을 한다.

express.js(백엔드)는 해당 요청을 받은 뒤 머신 리뷰 목록을 출력하는 API에서 MySQL(데이터베이스)로 reviews 테이블에 검색하여 정보들을 받아온다. 그리고 HealthMachineReviewRegisterTable.js(프론트)로 200 ok와 리뷰 목록을 보내준다. HealthMachineReviewRegisterTable.js(프론트)는 해당 정보를 받은 뒤 사용자에게 머신 리뷰 및 평점 목록을 보여준다.

사용자는 자신이 원하는 리뷰를 삭제버튼을 이용해서 삭제 요청하면 HealthMachineReviewRegisterTable.js(프론트)에서 express.js(백엔드)로 해당 리뷰를 DELETE 요청을 한다. express.js(백엔드)는 해당 요청을 받은 뒤 머신 리뷰를 삭제해주는 API에서 MySQL(데이터베이스)에서 해당 리뷰를 reviews 테이블에서 삭제해준다. 그리고 HealthMachineReviewRegisterTable.js(프론트)로 200 ok와 리뷰 목록을 보내준다. HealthMachineReviewRegisterTable.js(프론트)는 해당 정보를 받은 뒤 사용자에게 머신 리뷰 및 평점이 삭제 된 목록을 보여준다.

3.6 자료 구조 설계

- 사용자

```
{
  "users": [
    {
      "user_id": 1,
      "username": "송제용",
      "password": "*****",
      "created_at": "2023-06-12T10:00:00Z"
    },
    {
      "user_id": 2,
      "username": "용제송",
      "password": "*****",
      "created_at": "2023-06-11T14:30:00Z"
    },
    ...
  ]
}
```

- 헬스 머신

```
{
  "machines": [
    {
      "machine_id": 1,
      "machine_name": "Machine A",
      "description": "Machine A description",
      "machine_image": "machine_a.jpg",
      "brand_name": "Brand A",
      "founding_year": 2005,
      "brand_image": "brand_a.jpg"
    },
    {
      "machine_id": 2,
      "machine_name": "Machine B",
      "description": "Machine B description",
      "machine_image": "machine_b.jpg",
      "brand_name": "Brand B",
      "founding_year": 2010,
      "brand_image": "brand_b.jpg"
    },
    ...
  ]
}
```

- 브랜드

```
{
  "brands": [
    {
      "brand_id": 1,
      "name": "Brand A",
      "country": "Korea",
      "founding_year": 2005,
      "image_name": "brand_a.jpg"
    },
    {
      "brand_id": 2,
      "name": "Brand B",
      "country": "USA",
      "founding_year": 2010,
      "image_name": "brand_b.jpg"
    },
    ...
  ]
}
```

- 평점 및 리뷰

```
{
  "reviews": [
    {
      "rating_id": 1,
      "machine_name": "Machine A",
      "user_name": "송제용",
      "rating": 4,
      "review": "Machine A review by 송제용"
    },
    {
      "rating_id": 2,
      "machine_name": "Machine A",
      "user_name": "용제송",
      "rating": 5,
      "review": "Machine A review by 용제송"
    },
    ...
  ]
}
```

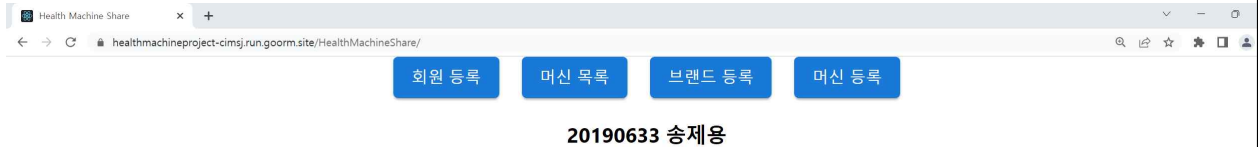
4. 구현

4.1.1 메인 화면

- 실행 화면

성공 시나리오

- 사용자가 메인 화면 페이지에 접속한다.



- 사용자는 회원 등록 화면 버튼을 눌러 사용자 등록 페이지로 이동할 수 있다.

healthmachineproject-cimsj.run.goorm.site/HealthMachineShare/users/register

- 사용자는 머신 목록 화면 버튼을 눌러 머신 목록 페이지로 이동할 수 있다

healthmachineproject-cimsj.run.goorm.site/HealthMachineShare/machines

- 사용자는 브랜드 등록 화면 버튼을 눌러 브랜드 등록 페이지로 이동할 수 있다.

healthmachineproject-cimsj.run.goorm.site/HealthMachineShare/brands

- 사용자는 머신 등록 화면 버튼을 눌러 머신 등록 페이지로 이동할 수 있다.

healthmachineproject-cimsj.run.goorm.site/HealthMachineShare/machines/register

- 소스코드

App.jsx

//20190633 송제용

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';
import { styled } from '@mui/system';
import { Button } from '@mui/material';
import HealthMachineTable from './HealthMachineTable';
import HealthMachineReviewTable from './HealthMachineReviewTable';
import HealthMachineBrandTable from './HealthMachineBrandTable';
import HealthMachineRegisterTable from './HealthMachineRegisterTable';
import HealthMachineReviewRegisterTable from './HealthMachineReviewRegisterTable';
import HealthMachineUserRegisterTable from './HealthMachineUserRegisterTable';
```

// 스타일링된 컴포넌트들의 스타일을 정의한 CSS-in-JS 코드

```
const ButtonContainer = styled('div')({
  display: 'flex',
  justifyContent: 'center',
  marginBottom: '20px',
});
```

```
const StyledLink = styled(Link)({
  textDecoration: 'none',
```

```

margin: '0 10px',
});

const Name = styled('div')({
  textAlign: 'center',
  marginTop: '20px',
  fontWeight: 'bold',
  fontSize: '18px',
});

function App() {
  return (
    <div className="App">
      <Router>
        <ButtonContainer>
          {/* 각 링크를 통해 다른 경로로 이동할 수 있는 버튼들 */}
          <StyledLink to="/HealthMachineShare/users/register">
            <Button variant="contained" className="button">
              회원 등록
            </Button>
          </StyledLink>
          <StyledLink to="/HealthMachineShare/machines">
            <Button variant="contained" className="button">
              머신 목록
            </Button>
          </StyledLink>
          <StyledLink to="/HealthMachineShare/brands">
            <Button variant="contained" className="button">
              브랜드 등록
            </Button>
          </StyledLink>
          <StyledLink to="/HealthMachineShare/machines/register">
            <Button variant="contained" className="button">
              머신 등록
            </Button>
          </StyledLink>
        </ButtonContainer>

        <Routes>
          {/* 각 경로에 맞는 컴포넌트를 렌더링 */}
          <Route path="/HealthMachineShare/users/register" element={<HealthMachineUserRegisterTable />} />
          <Route path="/HealthMachineShare/machines" element={<HealthMachineTable />} />
          <Route path="/HealthMachineShare/machines/reviews/:machineId" element={<HealthMachineReviewTable />} />
          <Route path="/HealthMachineShare/brands" element={<HealthMachineBrandTable />} />
          <Route path="/HealthMachineShare/machines/register" element={<HealthMachineRegisterTable />} />
          <Route
            path="/HealthMachineShare/machines/reviews/register/:machineId"
            element={<HealthMachineReviewRegisterTable />}
          />
        </Routes>
      </div>
    );
  }
}

```

```

</Router>

<Name>20190633 송제용</Name>
</div>
);
}

export default App;

```

- 컴포넌트 렌더링 : HealthMachineUserRegisterTable, HealthMachineTable, HealthMachineReviewTable, HealthMachineBrandTable, HealthMachineRegisterTable, HealthMachineReviewRegisterTable 등의 컴포넌트들이 사용자의 요청에 따라 렌더링된다.
- 사용자 인터페이스: StyledLink 컴포넌트를 사용하여 버튼을 생성하고, Link 컴포넌트를 통해 클릭 시 해당 경로로 이동할 수 있도록 한다.
- ▶ 주어진 경로에 따라 적절한 컴포넌트를 렌더링하고, 사용자가 버튼을 클릭하거나 경로를 탐색함에 따라 웹 애플리케이션의 화면을 업데이트하는 역할을 수행한다.

4.1.2 사용자 등록

- 실행 화면

성공 시나리오

- 사용자가 프로그램의 사용자 등록 페이지에 접속한다.

20190633 송제용

- 사용자가 사용자 이름(아이디)(아이디), 비밀번호를 입력한다.

- 사용자가 등록 양식을 제출한다.

- 프로그램은 입력된 정보를 확인하고, 중복된 사용자 이름(아이디)이 있는지 확인한다.

```
const checkUserQuery = `SELECT * FROM users WHERE username = '${username}'`;
db.query(checkUserQuery, (err, result) => {
  if (err) {
    res.status(500).json({ message: '서버 오류' });
    return console.log(err);
  }

  if (result.length > 0) {
    res.status(400).json({ message: '이미 등록된 사용자 이름입니다.' });
  }
});
```

- 입력된 정보가 유효하고 고유한 경우, 프로그램은 새로운 사용자 계정을 생성한다.

```
const insertUserQuery = `INSERT INTO users (username, password) VALUES ('${username}', '${password}')`;
db.query(insertUserQuery, (err, result) => {
  if (err) {
    res.status(500).json({ message: '서버 오류' });
    return console.log(err);
  }
  res.json({ userId: result.insertId, message: '사용자 등록이 완료되었습니다.' });
});
```

- 프로그램은 사용자 이름(아이디), 비밀번호를 데이터베이스의 "users" 테이블에 저장한다.

```
mysql> select * from users;
+-----+-----+-----+-----+
| user_id | username | password | created_at |
+-----+-----+-----+-----+
| 1 | 용용용 | 1234 | 2023-06-17 07:00:03 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

- 프로그램은 사용자에게 등록이 성공적으로 완료되었다고 알려주기 위해 입력 칸을 비운다.

사용자 이름 *

비밀번호 *

실패 시나리오

- 필수 정보가 누락된 경우, 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.

사용자 이름 *

송제용

비밀번호 *

!

이 입력란을 작성하세요.

- 입력된 사용자 이름(아이디)이 이미 데이터베이스에 존재하는 경우, 프로그램은 사용자에게 오류 메시지를 표시하고 다른 사용자 이름(아이디)을 입력하도록 안내한다.

이미 등록된 사용자 이름입니다.

사용자 이름 *

응응응

비밀번호 *

....

등록

- 소스코드

HealthMachineUserRegisterTable.js

```
//20190633 송제용
import React, { useState } from 'react';
import axios from 'axios';
import {
  Typography,
  TextField,
  Button
} from '@mui/material';
import { styled } from '@mui/system';

const EXPRESS_URL = 'https://healthmachineproject.run.goorm.site';

// 컴포넌트 스타일링을 위한 CSS-in-JS 코드
const Container = styled('div')({
  maxWidth: '400px',
  margin: '0 auto',
  padding: '20px',
});

const Heading = styled(Typography)({
  textAlign: 'center',
  marginBottom: '20px',
});

const Form = styled('form')({
  display: 'flex',
  flexDirection: 'column',
  alignItems: 'center',
});

const InputField = styled(TextField)({
  width: '100%',
  marginBottom: '10px',
});

const ErrorMessage = styled(Typography)({
```

```

    color: 'red',
    marginBottom: '10px',
  });

const SubmitButton = styled(Button)({
  width: '100%',
});

function HealthMachineUserRegisterTable() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [errorMessage, setErrorMessage] = useState('');

  // 사용자 이름 입력 변경 핸들러
  const handleUsernameChange = (e) => {
    setUsername(e.target.value);
  };

  // 비밀번호 입력 변경 핸들러
  const handlePasswordChange = (e) => {
    setPassword(e.target.value);
  };

  // 폼 제출 핸들러
  const handleFormSubmit = async (e) => {
    e.preventDefault();

    try {
      const response = await axios.post(`${EXPRESS_URL}/HealthMachineShare/users/register`, {
        username,
        password,
      });
      // 등록 후 필드 초기화
      setUsername('');
      setPassword('');
      setErrorMessage('');
      console.log(response.data.message);
    } catch (error) {
      console.log(error.response);
    }

    // 오류 응답이 있을 경우 에러 메시지 표시
    if (error.response && error.response.data) {
      setErrorMessage(error.response.data.message);
    } else {
      setErrorMessage('서버 오류');
    }
  };

  return (

```

```

    <Container>
      <Heading variant="h4">사용자 등록</Heading>
      {errorMessage && <ErrorMessage>{errorMessage}</ErrorMessage>}
      <Form onSubmit={handleFormSubmit}>
        <InputField
          label="사용자 이름"
          type="text"
          value={username}
          onChange={handleUsernameChange}
          required
          variant="outlined"
          className="input-field"
        />
        <InputField
          label="비밀번호"
          type="password"
          value={password}
          onChange={handlePasswordChange}
          required
          variant="outlined"
          className="input-field"
        />
        <SubmitButton type="submit" variant="contained">
          등록
        </SubmitButton>
      </Form>
    </Container>
  );
}

```

export default HealthMachineUserRegisterTable;

- 상태 관리: useState를 사용하여 username, password, errorMessage 상태를 관리한다. username은 사용자 이름을, password는 비밀번호를 저장하며, errorMessage는 서버로부터의 오류 메시지를 저장한다.
- 입력값 변경 핸들러: handleUsernameChange와 handlePasswordChange 함수를 사용하여 사용자 이름과 비밀번호 입력값의 변경을 감지하고, 해당 값을 상태에 업데이트한다.
- 폼 제출 핸들러: handleFormSubmit 함수는 폼 제출 이벤트를 처리한다. axios를 사용하여 서버에 사용자 이름과 비밀번호를 전송한다. 성공적인 응답을 받으면 필드를 초기화하고 오류 메시지를 초기화한다. 오류 응답이 있는 경우, 해당 오류 메시지를 errorMessage 상태에 저장한다.
- JSX 렌더링: 사용자 등록 폼을 구성하는 JSX 요소들을 렌더링한다. Container 컴포넌트로 전체 폼을 감싸고, Heading 컴포넌트로 제목을 표시한다. 오류 메시지가 존재할 경우 ErrorMessage 컴포넌트로 표시한다. 사용자 이름과 비밀번호를 입력하는 InputField 컴포넌트들과 등록 버튼을 나타내는 SubmitButton 컴포넌트를 포함한 폼을 제공한다.
- ▶ 사용자가 이름과 비밀번호를 입력하고 등록 버튼을 클릭하여 서버로 사용자 정보를 전송하는 역할을 담당한다.

express.js

// 사용자 등록 API에 대한 POST 요청 처리

```
app.post('/HealthMachineShare/users/register', (req, res) => {
  const { username, password } = req.body;

  // 등록된 사용자인지 확인하기 위해 DB에서 사용자 검색
  const checkUserQuery = `SELECT * FROM users WHERE username = '${username}'`;
  db.query(checkUserQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }

    if (result.length > 0) {
      res.status(400).json({ message: '이미 등록된 사용자 이름이다.' });
    } else {
      // 새로운 사용자 등록
      const insertUserQuery = `INSERT INTO users (username, password) VALUES ('${username}', '${password}')`;
      db.query(insertUserQuery, (err, result) => {
        if (err) {
          res.status(500).json({ message: '서버 오류' });
          return console.log(err);
        }
        res.json({ userId: result.insertId, message: '사용자 등록이 완료되었다.' });
      });
    }
  });
});
```

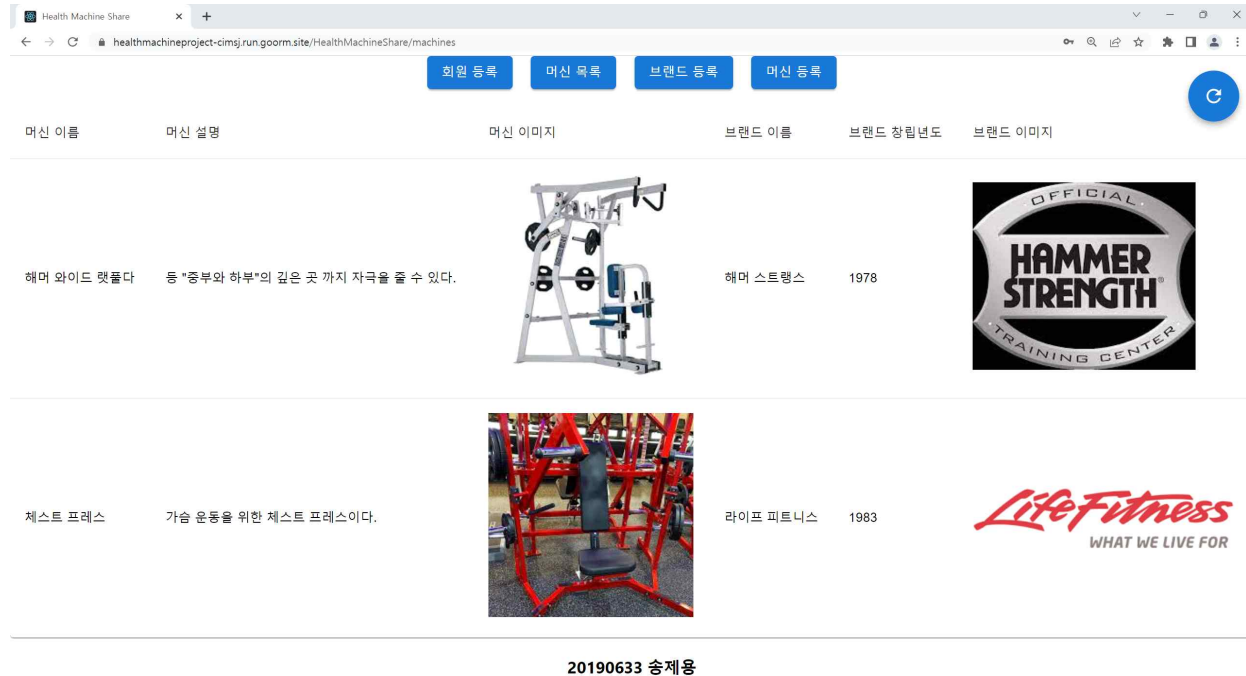
- 요청 파라미터 추출: POST 요청에서 username과 password를 추출한다.
 - 사용자 확인: DB에서 등록된 사용자인지 확인하기 위해 username을 사용하여 사용자를 검색한다.
 - 등록된 사용자 확인: 검색 결과를 통해 사용자가 이미 등록되어 있는지 확인한다. 결과가 존재하는 경우, 이미 등록된 사용자 이름임을 나타내는 오류 응답을 전송한다.
 - 새로운 사용자 등록: 검색 결과가 없는 경우, 새로운 사용자를 등록한다. 등록 쿼리를 실행하여 DB에 사용자 정보를 삽입한다.
- ▶ 사용자 등록 API의 POST 요청을 처리하는 부분이다.

4.1.3 머신 목록 보기

- 실행 화면

성공 시나리오

- 사용자가 머신 목록 페이지에 접속한다. 각 머신은 사진과 함께 제목, 요약 정보 등을 포함한다.



- 소스코드

HealthMachineTable.js

```
//20190633 송제용
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';
import {
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  Paper,
  Fab,
} from '@mui/material';
import { styled } from '@mui/system';
import RefreshIcon from '@mui/icons-material/Refresh';

const EXPRESS_URL = 'https://healthmachineproject.run.goorm.site';

// 컴포넌트 스타일링을 위한 CSS-in-JS 코드
const Container = styled('div')({
  width: '100%',
```

```

height: '100%',
display: 'flex',
flexDirection: 'column',
alignItems: 'center',
justifyContent: 'center',
});

const StyledTable = styled(Table)({
  minWidth: 650,
});

const StyledTableCell = styled(TableCell)({
  '& a': {
    display: 'block',
    width: '100%',
    textDecoration: 'none',
    color: '#000',
  },
});

function HealthMachineTable() {
  const [items, setItems] = useState([]);

  // 머신 목록을 서버에서 가져오는 비동기 함수
  const fetchData = async () => {
    try {
      const res = await axios.get(`${EXPRESS_URL}/HealthMachineShare/machines`);
      setItems(res.data.machines);
    } catch (error) {
      console.log(error);
    }
  };

  useEffect(() => {
    fetchData();
  }, []);

  // 데이터를 새로고침하는 함수
  const refresh = () => {
    fetchData();
  };

  return (
    <Container>
      <Paper sx={{ width: '100%', flexGrow: 1, overflow: 'hidden' }}>
        <Fab
          color="primary"
          sx={{
            position: 'fixed',
            top: (theme) => theme.spacing(2),

```

```

        right: (theme) => theme.spacing(2),
      }}
      onClick={refresh}
    >
      <RefreshIcon />
    </Fab>
    <TableContainer sx={{ maxHeight: '100%', overflow: 'auto' }}>
      <StyledTable stickyHeader aria-label="sticky table">
        <TableHead>
          <TableRow>
            <TableCell>머신 이름</TableCell>
            <TableCell>머신 설명</TableCell>
            <TableCell>머신 이미지</TableCell>
            <TableCell>브랜드 이름</TableCell>
            <TableCell>브랜드 창립년도</TableCell>
            <TableCell>브랜드 이미지</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {/* 머신 목록을 동적으로 렌더링 */}
          {items.map((machine, index) => (
            <TableRow key={index}>
              <StyledTableCell>
                <Link
                  to={` /HealthMachineShare/machines/reviews/${machine.machine_id}`}
                >
                  {machine.machine_name}
                </Link>
              </StyledTableCell>
              <StyledTableCell>
                <Link
                  to={` /HealthMachineShare/machines/reviews/${machine.machine_id}`}
                >
                  {machine.description}
                </Link>
              </StyledTableCell>
              <StyledTableCell>
                <Link
                  to={` /HealthMachineShare/machines/reviews/${machine.machine_id}`}
                >
                  <img
                    src={` ${process.env.PUBLIC_URL}/machine_images/${machine.machine_image}`}
                    alt="Machine"
                  />
                </Link>
              </StyledTableCell>
              <StyledTableCell>
                <Link
                  to={` /HealthMachineShare/machines/reviews/${machine.machine_id}`}

```

```

        <Link>
          {machine.brand_name}
        </Link>
      </StyledTableCell>
    </StyledTableCell>
    <Link
      to={` /HealthMachineShare/machines/reviews/${machine.machine_id}`}
    >
      {machine.founding_year}
    </Link>
  </StyledTableCell>
</StyledTableCell>
  <Link
    to={` /HealthMachineShare/machines/reviews/${machine.machine_id}`}
  >
    <img
      src={` ${process.env.PUBLIC_URL}/brand_images/${machine.brand_image}`}
      alt="Brand"
    />
  </Link>
</StyledTableCell>
</TableRow>
  )}
</TableBody>
</StyledTable>
</TableContainer>
</Paper>
</Container>
);
}

export default HealthMachineTable;

```

- 데이터 가져오기: fetchData 함수를 통해 서버로부터 머신 목록 데이터를 가져온다. axios 라이브러리를 사용하여 GET 요청을 보내고, 응답으로 받은 데이터를 items에 저장한다.
- 데이터 새로고침: refresh 함수는 fetchData 함수를 이용해 화면을 새로고침한다. 사용자가 새로고침 버튼을 클릭하면 이 함수가 실행되며, 서버에서 최신 데이터를 다시 가져와 items 상태를 업데이트한다.
- 테이블 렌더링: 컴포넌트는 Table, TableHead, TableBody, TableRow, TableCell 등의 MUI 컴포넌트를 사용하여 테이블을 구성한다. items 값을 이용하여 각 머신에 대한 정보를 사용자에게 보여준다.
- 링크 생성: 각 머신 이름, 머신 설명, 머신 이미지, 브랜드 이름, 브랜드 창립년도, 브랜드 이미지의 셀은 Link 컴포넌트를 통해 해당 머신의 리뷰 페이지로 이동할 수 있는 링크를 생성한다. 각 셀은 클릭 가능한 링크로 구성되어 있으며, 사용자가 클릭하면 해당 머신의 리뷰 페이지로 이동한다.
- ▶ 머신 목록을 가져와서 사용자에게 보여주는 역할 및 해당 머신의 리뷰 페이지로 이동을 담당한다.

express.js

// 머신 목록 API에 대한 GET 요청 처리

```
app.get('/HealthMachineShare/machines', (req, res) => {
```

```
  // 머신 정보와 해당 머신의 브랜드 정보를 조인하여 가져오는 SQL 쿼리
```

```
  const sql = `SELECT machines.machine_id ,machines.name AS machine_name, machines.description, machines.image_name AS machine_image,
```

```
                brands.name AS brand_name, brands.founding_year, brands.image_name AS brand_image
```

```
                FROM machines
```

```
                INNER JOIN brands ON machines.brand_id = brands.brand_id;
```

```
`;
```

```
db.query(sql, (err, rows) => {
```

```
  if (err) {
```

```
    res.json({ result: 'error' });
```

```
    return console.log(err);
```

```
  }
```

```
  res.json({ machines: rows });
```

```
});
```

```
});
```

- SQL 쿼리 실행: machines 테이블과 brands 테이블을 INNER JOIN하여 머신 정보와 브랜드 정보를 연결하여 머신 정보와 해당 머신의 브랜드 정보를 조인하여 가져오는 SQL 쿼리를 실행한다.
- 결과 반환: SQL 쿼리 실행 결과인 rows를 JSON 형식으로 반환한다.
- ▶ 해당 코드는 머신 목록을 가져오는 API의 GET 요청 처리를 담당한다.

4.1.4 브랜드 정보 등록

- 실행 화면

성공 시나리오

- 사용자는 브랜드 정보 등록을 위한 메인화면에서 브랜드 정보 등록 화면으로 이동한다.

Health Machine Share

healthmachineproject-cimsj.run.goorm.site/HealthMachineShare/brands

회원 등록 머신 목록 브랜드 등록 머신 등록

헬스 머신 브랜드 등록

브랜드 이름 *

국가 *

창립 년도 *

파일 선택 선택된 파일 없음

등록

20190633 송재용

- 사용자는 새로운 브랜드의 로고를 업로드한다. (이미지 파일 형식: JPG, PNG 등)
- 사용자는 브랜드의 이름, 설명 등의 정보를 입력한다.

브랜드 이름*
싸이백스

국가*
미국

창립 년도*
1974

파일 선택 brand5.jpg

- 사용자는 등록 양식을 제출한다.

등록

- 프로그램은 입력된 정보를 확인하고, 중복된 브랜드 이름이 있는지 확인한다.

```
// 등록된 브랜드인지 확인하기 위해 DB에서 브랜드 검색
const checkBrandQuery = `SELECT * FROM brands WHERE name = ?`;
db.query(checkBrandQuery, [name], (err, result) => {
  if (err) {
    res.status(500).json({ message: '서버 오류' });
    return console.log(err);
  }

  if (result.length > 0) {
    res.status(400).json({ message: '이미 등록된 브랜드 이름입니다.' });
  }
});
```

- 프로그램은 머신의 정보를 데이터베이스의 “brands” 테이블에 저장하고, 브랜드에 대한 고유한 브랜드 ID를 생성한다.

```
mysql> select * from brands;
+-----+-----+-----+-----+-----+
| brand_id | name           | country | founding_year | image_name           |
+-----+-----+-----+-----+-----+
| 1 | 해머 스트렝스 | 미국   | 1978 | 1686985988762-brand3.jpg |
| 2 | 라이프 피트니스 | 유럽  | 1983 | 1686986155621-brand1.png |
| 3 | 싸이백스      | 미국   | 1974 | 1686987021467-brand5.jpg |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```

v public
v brand_images
  .jpg 1686985988762-brand3.jpg
  .png 1686986155621-brand1.png
  .jpg 1686987021467-brand5.jpg
```

실패 시나리오

- 필수 정보가 누락된 경우, 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.

헬스 머신 브랜드 등록

브랜드 이름*
제용 브랜드

국가*
한국

창립 년도*
2000

파일 선택 선택된 파일 없음

파일을 선택하세요.

20190633 송제용

- 이미 동일한 브랜드 이름이 데이터베이스에 존재하는 경우 프로그램은 오류 메시지를 표시하도록 한다.

헬스 머신 브랜드 등록

이미 등록된 브랜드 이름입니다.

브랜드 이름*
싸이백스

국가*
미국

창립 년도*
1974

파일 선택 brand5.jpg

등록

20190633 송제용

- 소스코드

HealthMachineBrandTable.js

```
//20190633 송제용
import React, { useState } from 'react';
import axios from 'axios';
import { Button, TextField, Typography } from '@mui/material';
import { styled } from '@mui/system';

const EXPRESS_URL = 'https://healthmachineproject.run.goorm.site';

// 스타일링된 컴포넌트들의 스타일을 정의한 CSS-in-JS 코드
const FormContainer = styled('div')({
  maxWidth: '400px',
```

```

margin: '0 auto',
padding: '20px',
});

const Heading = styled(Typography)({
  textAlign: 'center',
  marginBottom: '20px',
});

const InputField = styled(TextField)({
  width: '100%',
  marginBottom: '20px',
});

const ErrorMessage = styled(Typography)({
  color: 'red',
  marginBottom: '10px',
});

const SubmitButton = styled(Button)({
  width: '100%',
});

function HealthMachineBrandTable() {
  const [name, setName] = useState('');
  const [country, setCountry] = useState('');
  const [foundingYear, setFoundingYear] = useState('');
  const [errorMessage, setErrorMessage] = useState('');

  // 브랜드 이름 변경 시 상태 업데이트
  const handleNameChange = (e) => {
    setName(e.target.value);
  };

  // 국가 변경 시 상태 업데이트
  const handleCountryChange = (e) => {
    setCountry(e.target.value);
  };

  // 창립 년도 변경 시 상태 업데이트
  const handleFoundingYearChange = (e) => {
    setFoundingYear(e.target.value);
  };

  // 폼 제출 시 실행되는 함수
  const handleFormSubmit = async (e) => {
    e.preventDefault();

    try {
      // FormData 객체를 생성하여 필드와 값 추가

```

```

const formData = new FormData();
formData.append('name', name);
formData.append('country', country);
formData.append('founding_year', foundingYear);
formData.append('image', e.target.image.files[0]);

// axios를 사용하여 POST 요청을 서버로 보냄
const response = await axios.post(
  `${EXPRESS_URL}/HealthMachineShare/brands`,
  formData,
  {
    headers: { 'Content-Type': 'multipart/form-data' },
  }
);

// 등록 후 입력 필드 초기화
setName('');
setCountry('');
setFoundingYear('');

// 서버에서 전달된 메시지 출력
console.log(response.data.message);
} catch (error) {
  console.log(error.response);

  // 오류 응답이 있을 경우 예러 메시지 표시
  if (error.response && error.response.data) {
    setErrorMessage(error.response.data.message);
  } else {
    setErrorMessage('서버 오류');
  }
}
};

return (
  <div className="brand-table-container">
    <FormContainer>
      <Heading variant="h4">헬스 머신 브랜드 등록</Heading>
      {errorMessage && <ErrorMessage>{errorMessage}</ErrorMessage>}
      <form onSubmit={handleFormSubmit}>
        <InputField
          label="브랜드 이름"
          value={name}
          onChange={handleNameChange}
          required
        />
        <InputField
          label="국가"
          value={country}
          onChange={handleCountryChange}

```

```

        required
      />
      <InputField
        label="창립 년도"
        type="number"
        value={foundingYear}
        onChange={handleFoundingYearChange}
        required
      />
      <InputField
        type="file"
        name="image"
        inputProps={{ accept: 'image/*' }}
        required
      />
      <SubmitButton type="submit" variant="contained">
        등록
      </SubmitButton>
    </form>
  </FormContainer>
</div>
);
}

export default HealthMachineBrandTable;

```

- 상태 관리: useState를 사용하여 브랜드 이름(name), 국가(country), 창립 년도(foundingYear), 오류 메시지(errorMessage)의 상태를 관리한다.
- 입력값 변경 핸들러: handleNameChange와 handleCountryChange와 handleFoundingYearChange 함수를 사용하여 브랜드 이름과 국가, 창립 년도 입력값의 변경을 감지하고, 해당 값을 상태에 업데이트한다.
- 폼 제출 핸들러: handleFormSubmit 함수는 폼 제출 이벤트를 처리한다. axios를 사용하여 서버에 브랜드 정보를 전송한다. 성공적인 응답을 받으면 필드를 초기화하고 오류 메시지를 초기화한다. 오류 응답이 있는 경우, 해당 오류 메시지를 errorMessage 상태에 저장한다.
- JSX 렌더링: 브랜드 등록 폼을 구성하는 JSX 요소들을 렌더링한다. Container 컴포넌트로 전체 폼을 감싸고, Heading 컴포넌트로 제목을 표시한다. 오류 메시지가 존재할 경우 ErrorMessage 컴포넌트로 표시한다. 브랜드 이름, 국가, 창립 년도, 이미지를 입력하는 InputField 컴포넌트들과 등록 버튼을 나타내는 SubmitButton 컴포넌트를 포함한 폼을 제공한다.
- ▶ 브랜드 등록 기능을 제공하며, 사용자는 폼을 통해 브랜드 정보를 입력하고 이미지 파일을 업로드하여 서버로 전송할 수 있다.

```

express.js
// 브랜드이미지 저장 경로
const brandImageStorage = multer.diskStorage({
  destination: '/workspace/HealthMachineProject/client/public/brand_images',
  filename: (req, file, cb) => {

```

```

        cb(null, `${Date.now()}-${file.originalname}`);
    },
    });
// 브랜드 정보 등록 API에 대한 POST 요청 처리
app.post('/HealthMachineShare/brands', uploadBrandImage.single('image'), (req, res) => {
    const { name, country, founding_year } = req.body;
    const image = req.file.filename;

    // 등록된 브랜드인지 확인하기 위해 DB에서 브랜드 검색
    const checkBrandQuery = `SELECT * FROM brands WHERE name = ?`;
    db.query(checkBrandQuery, [name], (err, result) => {
        if (err) {
            res.status(500).json({ message: '서버 오류' });
            return console.log(err);
        }

        if (result.length > 0) {
            res.status(400).json({ message: '이미 등록된 브랜드 이름이다.' });
        } else {
            // 새로운 브랜드 등록
            const insertBrandQuery = `INSERT INTO brands (name, country, founding_year, image_name) VALUES (?, ?, ?, ?)`;

            db.query(insertBrandQuery, [name, country, founding_year, image], (err, result) => {
                if (err) {
                    res.status(500).json({ message: '서버 오류' });
                    return console.log(err);
                }
                res.json({
                    brandId: result.insertId,
                    message: '브랜드 정보 등록이 완료되었다.',
                });
            });
        }
    });
});

```

- 이미지 저장 경로 설정: multer.diskStorage를 사용하여 브랜드 이미지의 저장 경로와 파일명을 설정한다. 파일명은 중복방지를 위해 현재 시간과 원본 파일명을 조합하여 생성된다.
- 이미지 저장: uploadBrandImage.single('image') 미들웨어를 사용하여 이미지 파일을 업로드한다.
- 요청 파라미터 추출: POST 요청에서 브랜드 이름(name), 국가(country), 창립 년도(founding_year)를 추출하고, req.file.filename에서 업로드된 이미지 파일명을 추출한다.
- 등록된 브랜드 확인: 데이터베이스에서 이미 등록된 브랜드인지 확인하기 위해 checkBrandQuery 쿼리를 실행한다. 브랜드 이름을 기준으로 검색하여 결과를 확인한다. 만약 이미 등록된 브랜드인 경우 400 상태 코드와 함께 오류 메시지를 반환한다.
- 새로운 브랜드 등록: 이미 등록된 브랜드가 아닌 경우, insertBrandQuery 쿼리를 실행하여 새로운 브랜드 정보를 데이터베이스에 등록한다. 등록이 성공하면 200 상태 코드와 함께 브랜드 ID와

등록 완료 메시지를 반환한다.

- ▶ 클라이언트로부터 전달된 브랜드 정보를 데이터베이스에 저장하여 새로운 브랜드를 등록하는 기능을 제공한다.

4.1.5 머신 정보 등록

- 실행 화면

성공 시나리오

- 사용자는 머신 정보 등록을 위한 메인화면에서 정보 등록 화면으로 이동한다.

헬스 머신 정보 등록

머신 브랜드

머신 이름 *

머신 설명 *

파일 선택 선택된 파일 없음

등록

20190633 송재용

- 사용자는 새로운 머신의 사진을 업로드한다. (이미지 파일 형식: JPG, PNG 등)

- 사용자는 머신의 브랜드, 모델명, 설명 등의 정보를 입력한다.

(브랜드 정보 목록 API를 활용하여 브랜드를 선택할 수 있도록 구현)

```
// 브랜드 정보 목록 보기 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/brands', (req, res) => {
  // DB에서 모든 브랜드 가져오기
  const getBrandsQuery = 'SELECT * FROM brands;';
  db.query(getBrandsQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    res.json({ brands: result });
  });
});
```

머신 브랜드

브랜드 선택

해머 스트렝스

라이프 피트니스

싸이백스

->

헬스 머신 정보 등록

머신 브랜드
해머 스트렝스

머신 이름*
레그 프레스

머신 설명*
남자는 하체!

파일 선택 5.jpg

등록

20190633 송제용

- 입력된 정보가 유효하고 완전한 경우, 프로그램은 새로운 머신을 등록한다.

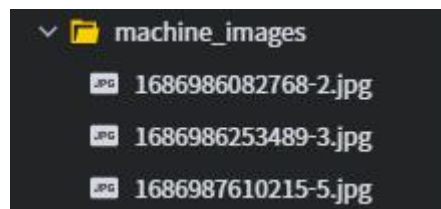
```
// 새로운 머신 등록
const insertMachineQuery =
  'INSERT INTO machines (brand_id, name, description, image_name) VALUES (?, ?, ?, ?)';
db.query(insertMachineQuery, [brandId, name, description, image], (err, result) => {
  if (err) {
    console.log(err);
    return res.status(500).json({ message: '서버 오류' });
  }
  res.json({
    machineId: result.insertId,
    message: '머신 정보 등록이 완료되었습니다.',
  });
});
```

- 프로그램은 머신의 정보를 데이터베이스의 “machines” 테이블에 저장하고, 머신에 대한 고유한 머신 ID를 생성한다.







```
mysql> select * from machines;
```

machine_id	brand_id	name	description	image_name
1	1	해머 와이드 랫플다	등 "중부와 하부"의 깊은 곳 까지 자극을 줄 수 있다.	1686986082768-2.jpg
2	2	체스트 프레스	가슴 운동을 위한 체스트 프레스이다.	1686986253489-3.jpg
3	1	레그 프레스	남자는 하체!	1686987610215-5.jpg

3 rows in set (0.00 sec)



- 등록된 머신의 사진과 정보가 프로그램에서 표시되고 사용자들이 해당 정보를 확인할 수 있다.

머신 이름	머신 설명	머신 이미지	브랜드 이름	브랜드 창립년도	브랜드 이미지
해머 와이드 벤치	통 "중부와 하부"의 깊은 곳 까지 자극을 줄 수 있다.		해머 스트렝스	1978	
제스트 프레스	가슴 운동을 위한 제스트 프레스이다.		라이프 피트니스	1983	
레그 프레스	남자는 학재		해머 스트렝스	1978	

20190633 송제웅

실패 시나리오

- 필수 정보가 누락된 경우, 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.

헬스 머신 정보 등록

머신 브랜드

머신 이름*
제용 머신

! 이 입력란을 작성하세요.

머신 설명*
제용이가 좋아함

파일 선택 3.jpg

등록

20190633 송제웅

- 소스코드

HealthMachineRegisterTable.js

//20190633 송제웅

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import {
  Button,
  FormControl,
  InputLabel,
  MenuItem,
  Select,
  TextField,
  Typography,

```

```

} from '@mui/material';
import { styled } from '@mui/system';

const EXPRESS_URL = 'https://healthmachineproject.run.goorm.site';

// 스타일링된 컴포넌트들의 스타일을 정의한 CSS-in-JS 코드
const Container = styled('div')({
  maxWidth: '400px',
  margin: '0 auto',
  padding: '20px',
});

const Heading = styled(Typography)({
  textAlign: 'center',
  marginBottom: '20px',
});

const Form = styled('form')({
  display: 'flex',
  flexDirection: 'column',
  alignItems: 'center',
});

const InputField = styled(TextField)({
  width: '100%',
  marginBottom: '20px',
});

const SubmitButton = styled(Button)({
  width: '100%',
});

const SelectWrapper = styled(FormControl)({
  width: '100%',
  marginBottom: '10px',
});

function HealthMachineRegisterTable() {
  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [brands, setBrands] = useState([]);
  const [selectedBrand, setSelectedBrand] = useState('');

  useEffect(() => {
    const fetchBrands = async () => {
      try {
        // 서버로부터 브랜드 목록을 가져옴
        const res = await axios.get(`${EXPRESS_URL}/HealthMachineShare/brands`);
        setBrands(res.data.brands);
      } catch (error) {

```

```

        console.log(error);
    }
};

fetchBrands();
}, []);

// 머신 이름 변경 시 상태 업데이트
const handleNameChange = (e) => {
    setName(e.target.value);
};

// 머신 설명 변경 시 상태 업데이트
const handleDescriptionChange = (e) => {
    setDescription(e.target.value);
};

// 브랜드 선택 시 상태 업데이트
const handleBrandChange = (e) => {
    setSelectedBrand(e.target.value);
};

// 폼 제출 시 실행되는 함수
const handleFormSubmit = async (e) => {
    e.preventDefault();

    try {
        // FormData 객체를 생성하여 필드와 값 추가
        const formData = new FormData();
        formData.append('name', name);
        formData.append('description', description);
        formData.append('image', e.target.image.files[0]);
        formData.append('brand', selectedBrand);

        // axios를 사용하여 POST 요청을 서버로 보냄
        await axios.post(`${EXPRESS_URL}/HealthMachineShare/machines/register`, formData, {
            headers: { 'Content-Type': 'multipart/form-data' },
        });

        // 등록 후 입력 필드 초기화
        setName('');
        setDescription('');
        setSelectedBrand('');

    } catch (error) {
        console.log(error);
    }
};

return (

```

```

<Container>
  <Heading variant="h4">헬스 머신 정보 등록</Heading>
  <Form onSubmit={handleFormSubmit}>
    <SelectWrapper>
      <InputLabel>머신 브랜드</InputLabel>
      <Select value={selectedBrand} onChange={handleBrandChange} required>
        <MenuItem value="">
          <em>브랜드 선택</em>
        </MenuItem>
        {brands.map((brand) => (
          <MenuItem key={brand.brandId} value={brand.name}>
            {brand.name}
          </MenuItem>
        ))}
      </Select>
    </SelectWrapper>
    <InputField label="머신 이름" value={name} onChange={handleNameChange} required />
    <InputField
      label="머신 설명"
      multiline
      rows={4}
      value={description}
      onChange={handleDescriptionChange}
      required
    />
    <InputField
      type="file"
      name="image"
      inputProps={{ accept: 'image/*' }}
      required
    />
    <SubmitButton type="submit" variant="contained">
      등록
    </SubmitButton>
  </Form>
</Container>
);
}

export default HealthMachineRegisterTable;

```

- 데이터 가져오기: fetchBrands 함수를 통해 서버로부터 브랜드 목록 데이터를 가져온다. axios 라이브러리를 사용하여 GET 요청을 보내고, 응답으로 받은 데이터를 brands에 저장한다.
- 상태 관리: useState를 사용하여 브랜드 이름(name), 머신 설명(description), 브랜드 목록(brands), 선택된 브랜드(selectedBrand)의 상태를 관리한다.
- 브랜드 선택 필드: 등록할 머신에 대한 브랜드를 선택할 수 있는 드롭다운 메뉴를 제공한다. selectedBrand 상태를 사용하여 선택한 브랜드를 관리하고, handleBrandChange 함수를 통해 선택된 브랜드를 업데이트한다.

- 폼 제출 핸들러: `handleFormSubmit` 함수는 폼 제출 이벤트를 처리한다. `axios`를 사용하여 서버에 머신 정보를 전송한다. 성공적인 응답을 받으면 필드를 초기화하고 오류 메시지를 초기화한다. 오류 응답이 있는 경우, 해당 오류 메시지를 `errorMessage` 상태에 저장한다.
- JSX 렌더링: 머신 등록 폼을 구성하는 JSX 요소들을 렌더링한다. `Container` 컴포넌트로 전체 폼을 감싸고, `Heading` 컴포넌트로 제목을 표시한다. 오류 메시지가 존재할 경우 `ErrorMessage` 컴포넌트로 표시한다. 브랜드를 선택하기 위해서 `SelectWrapper` 컴포넌트를 사용하고 `brands.map`를 사용하여 브랜드 목록을 순회하며 드롭다운 메뉴 아이템을 렌더링한다. 또한 머신 이름, 머신 설명, 머신 이미지를 입력하는 `InputField` 컴포넌트들과 등록 버튼을 나타내는 `SubmitButton` 컴포넌트를 포함한 폼을 제공한다.
- ▶ 사용자가 브랜드를 선택하고 머신 정보 입력 후 등록 버튼을 클릭하여 서버로 머신 정보를 전송하는 역할을 담당한다.

`express.js`

```
// 머신이미지 저장 경로
const machineImageStorage = multer.diskStorage({
  destination: '/workspace/HealthMachineProject/client/public/machine_images',
  filename: (req, file, cb) => {
    cb(null, `${Date.now()}-${file.originalname}`);
  },
});

// 머신 정보 등록 API에 대한 POST 요청 처리
app.post(
  '/HealthMachineShare/machines/register', uploadMachineImage.single('image'),
  (req, res) => {
    const { brand, name, description } = req.body;
    const image = req.file.filename;

    // 등록된 브랜드의 brand_id 가져오기
    const getBrandIdQuery = 'SELECT brand_id FROM brands WHERE name = ?';
    db.query(getBrandIdQuery, [brand], (err, brandResult) => {
      if (err) {
        console.log(err);
        return res.status(500).json({ message: '서버 오류' });
      }

      const brandId = brandResult[0].brand_id;

      // 새로운 머신 등록
      const insertMachineQuery =
        'INSERT INTO machines (brand_id, name, description, image_name) VALUES (?, ?, ?, ?)';

      db.query(insertMachineQuery, [brandId, name, description, image], (err, result) => {
        if (err) {
          console.log(err);
          return res.status(500).json({ message: '서버 오류' });
        }

        res.json({
```

```

        machineId: result.insertId,
        message: '머신 정보 등록이 완료되었다.',
    });
    });
    });
}
);
// 브랜드 정보 목록 보기 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/brands', (req, res) => {
    // DB에서 모든 브랜드 가져오기
    const getBrandsQuery = 'SELECT * FROM brands:.';
    db.query(getBrandsQuery, (err, result) => {
        if (err) {
            res.status(500).json({ message: '서버 오류' });
            return console.log(err);
        }

        res.json({ brands: result });
    });
});

```

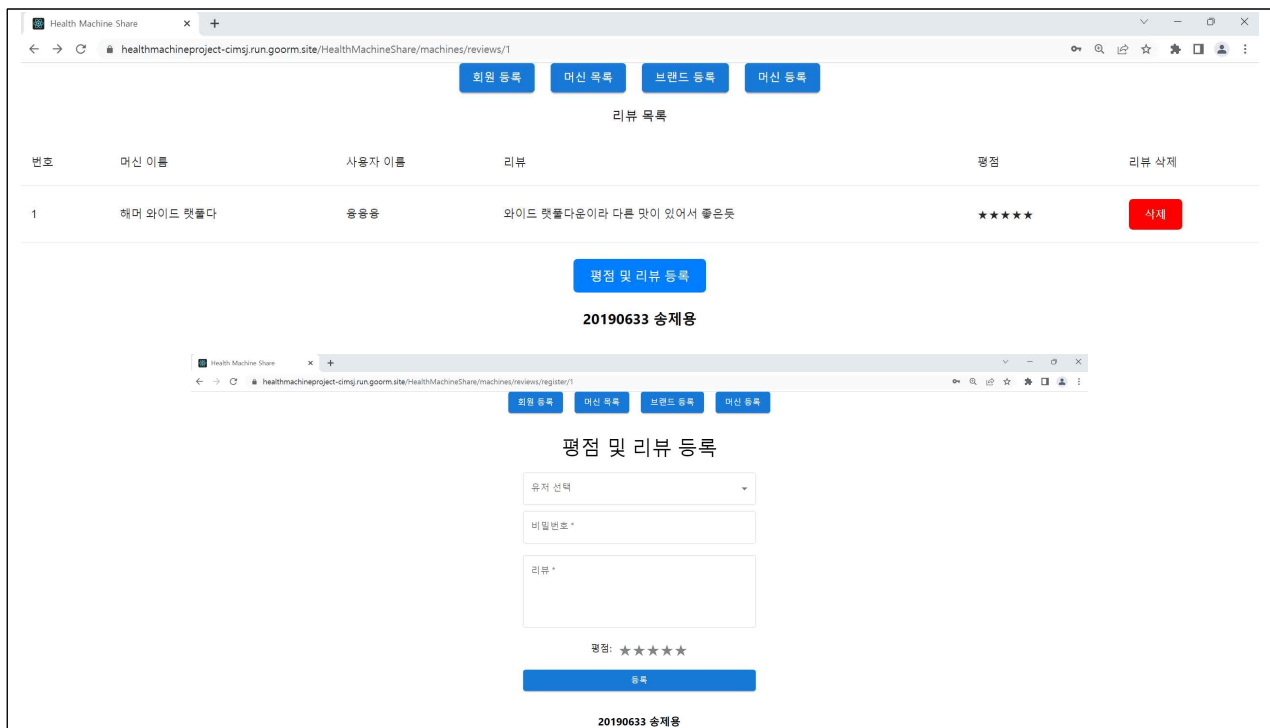
- 이미지 저장 경로 설정: multer.diskStorage를 사용하여 머신 이미지의 저장 경로와 파일명을 설정한다. 파일명은 중복방지를 위해 현재 시간과 원본 파일명을 조합하여 생성된다.
 - 이미지 저장: uploadBrandImage.single('image') 미들웨어를 사용하여 이미지 파일을 업로드한다.
 - 요청 파라미터 추출: POST 요청에서 브랜드(brand), 머신 이름(name), 머신 설명(description)을 추출하고, req.file.filename에서 업로드된 이미지 파일명을 추출한다.
 - 데이터 가져오기: brand 값으로 등록된 브랜드의 brand_id를 가져오기 위해 브랜드 이름을 사용하여 DB에서 조회한다.
 - 새로운 머신 등록: insertMachineQuery 쿼리를 실행하여 새로운 머신 정보를 데이터베이스에 등록한다. 등록이 성공하면 200 상태 코드와 함께 머신 ID와 등록 완료 메시지를 반환한다.
 - 브랜드 정보 목록 API: 사용자가 머신을 등록할 때 사용자 편의성을 위해서 브랜드를 드롭다운 방식으로 선택할 수 있게하기 위해서 사용한다.
- ▶ 헬스 머신 정보 등록과 브랜드 정보 목록 조회를 처리하는 API 부분이다.

4.1.6 리뷰 목록 보기 및 머신 평점 리뷰 등록 및 삭제

- 실행 화면

성공 시나리오

- 사용자는 머신 목록 화면에서 해당 머신의 사진을 클릭하여, 해당 머신의 평점 및 리뷰 페이지로 이동하여서 "평가 및 리뷰 작성" 버튼을 클릭하여 평가 및 리뷰 작성 페이지로 이동한다.



- 사용자는 자신의 이름(아이디)을 선택하고 비밀번호를 입력한다.

(사용자 목록 API를 이용하여 유저를 선택할 수 있도록 구현했음)

평점 및 리뷰 등록

```
// 사용자 목록 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/users', (req, res) => {
  // DB에서 모든 사용자 가져오기
  const getUsersQuery = 'SELECT * FROM users;';
  db.query(getUsersQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    res.json({ users: result });
  });
});
```

->

유저 선택

유저 선택

송제용

송제용

- 사용자는 리뷰를 작성할 수 있는 텍스트 입력란에 머신에 대한 자세한 의견과 경험을 기록한다.

평점 및 리뷰 등록

유저 선택

송제용

비밀번호 *

리뷰 *

와이드는 별로고 난 그냥 정석 랫폼다움이 좋다

평점: ★★★★★

등록

20190633 송제용

- 사용자는 머신에 대한 평점 또한 입력한다.

(마우스의 드래그를 이용해서 평점 조절 가능)

평점: ★★★★★

- 프로그램은 사용자가 유저의 이름과 비밀번호를 이용하여 등록된 사용자인지 검사하고 만약 등록된 사용자라면 작성한 평가와 리뷰를 확인하고, 데이터베이스에 저장한다.

```
mysql> select * from ratings_reviews;
+-----+-----+-----+-----+-----+-----+
| rating_id | machine_id | user_id | rating | review | created_at |
+-----+-----+-----+-----+-----+-----+
| 1 | 1 | 1 | 5 | 와이드 랫플다운이라 다른 맛이 있어서 좋은듯 | 2023-06-17 07:50:00 |
| 2 | 1 | 2 | 1 | 와이드는 별로고 난 그냥 정석 랫플다운이 좋더라 | 2023-06-17 07:53:59 |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

- 등록된 평가와 리뷰는 해당 머신의 상세 페이지에서 사용자들에게 표시되고, 다른 사용자들은 이를 확인할 수 있다.

번호	머신 이름	사용자 이름	리뷰	평점	리뷰 삭제
1	해머 와이드 랫플다	용용용	와이드 랫플다운이라 다른 맛이 있어서 좋은듯	★★★★★	삭제
2	해머 와이드 랫플다	송제용	와이드는 별로고 난 그냥 정석 랫플다운이 좋더라	★☆☆☆☆	삭제

- 등록된 평가와 리뷰는 해당 머신의 리뷰 페이지에서 삭제버튼을 눌러 삭제할 수 있다.

번호	머신 이름	사용자 이름	리뷰	평점	리뷰 삭제
1	해머 와이드 랫플다	용용용	와이드 랫플다운이라 다른 맛이 있어서 좋은듯	★★★★★	삭제

```
mysql> select * from ratings_reviews;
+-----+-----+-----+-----+-----+-----+
| rating_id | machine_id | user_id | rating | review | created_at |
+-----+-----+-----+-----+-----+-----+
| 1 | 1 | 1 | 5 | 와이드 랫플다운이라 다른 맛이 있어서 좋은듯 | 2023-06-17 07:50:00 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

실패 시나리오

- 필수 정보가 누락된 경우 프로그램은 사용자에게 오류 메시지를 표시하고, 필요한 정보를 입력하도록 안내한다.

평점 및 리뷰 등록

유저 선택

비밀번호*

.....

이 입력란을 작성하세요.

리뷰*

좋아용

평점: ★★★★★

등록

20190633 송제용

- 사용자가 잘못된 비밀번호를 입력할 경우, 프로그램은 사용자에게 오류 메시지를 표시하고 제대

로 된 비밀번호를 입력하도록 안내한다.

평점 및 리뷰 등록

유저 선택
송제용

비밀번호가 일치하지 않습니다.
비밀번호
.....

리뷰 *
비밀번호 테스트

평점: ★★☆☆☆

등록

20190633 송제용

- 소스코드

HealthMachineReviewTable.js

```
//20190633 송제용
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams, Link } from 'react-router-dom';
import {
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  Typography,
} from '@mui/material';
import { styled } from '@mui/system';

const EXPRESS_URL = 'https://healthmachineproject.run.goorm.site';

// 컴포넌트 스타일링을 위한 CSS-in-JS 코드
const Container = styled('div')({
  width: '100%',
  height: '100%',
  display: 'flex',
  flexDirection: 'column',
  alignItems: 'center',
  justifyContent: 'center',
});

const Heading = styled(Typography)({
```

```

    textAlign: 'center',
    marginBottom: '20px',
    width: '100%',
    fontSize: '100%',
  });

  const ButtonContainer = styled('div')({
    display: 'flex',
    justifyContent: 'center',
    marginTop: '20px',
  });

  const RegisterButton = styled(Link)({
    textDecoration: 'none',
    padding: '10px 20px',
    color: '#fff',
    backgroundColor: '#007bff',
    border: 'none',
    borderRadius: '5px',
    cursor: 'pointer',
  });

  const DeleteButton = styled('button')({
    padding: '10px 20px',
    color: '#fff',
    backgroundColor: 'red',
    border: 'none',
    borderRadius: '5px',
    cursor: 'pointer',
  });

  function HealthMachineReviewTable() {
    const [items, setItems] = useState([]); // 리뷰 목록을 저장하는 상태 변수
    const { machineId } = useParams();

    useEffect(() => {
      const fetchData = async () => {
        try {
          // 해당 머신의 리뷰 목록을 서버로부터 가져옴
          const res = await axios.get(
            `${EXPRESS_URL}/HealthMachineShare/machines/reviews/${machineId}`
          );
          setItems(res.data.reviews); // 가져온 리뷰 목록을 상태에 저장
        } catch (error) {
          console.log(error);
        }
      };

      fetchData();
    }, [machineId]);
  }

```

```

// 리뷰 삭제 함수
const deleteReview = async (reviewId) => {
  try {
    // 서버로 리뷰 삭제 요청을 보냄
    await axios.delete(`${EXPRESS_URL}/HealthMachineShare/machines/reviews/${reviewId}`);
    // 삭제 후 리뷰 목록을 다시 가져와서 상태를 업데이트
    const res = await axios.get(
      `${EXPRESS_URL}/HealthMachineShare/machines/reviews/${machineId}`
    );
    setItems(res.data.reviews);
  } catch (error) {
    console.log(error);
  }
};

// 평점을 별로 표시하는 함수
const renderRatingStars = (rating) => {
  const fullStar = '★';
  const emptyStar = '☆';
  const maxRating = 5;

  const fullStarsCount = Math.round(rating);
  const emptyStarsCount = maxRating - fullStarsCount;

  const fullStars = fullStar.repeat(fullStarsCount);
  const emptyStars = emptyStar.repeat(emptyStarsCount);

  return fullStars + emptyStars;
};

return (
  <Container className="review-table-container">
    <Heading variant="h4">리뷰 목록</Heading>
    <TableContainer>
      <Table>
        <TableHead>
          <TableRow>
            <TableCell>번호</TableCell>
            <TableCell>머신 이름</TableCell>
            <TableCell>사용자 이름</TableCell>
            <TableCell>리뷰</TableCell>
            <TableCell>평점</TableCell>
            <TableCell>리뷰 삭제</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {/* 리뷰 목록을 동적으로 렌더링 */}
          {items.map((review, i) => (
            <TableRow key={i}>
              <TableCell>{review.rating_id}</TableCell>

```

```

        <TableCell>{review.machine_name}</TableCell>
        <TableCell>{review.user_name}</TableCell>
        <TableCell>{review.review}</TableCell>
        <TableCell>{renderRatingStars(review.rating)}</TableCell>
        <TableCell>
            {/* 리뷰 삭제 버튼 */}
            <DeleteButton onClick={() => deleteReview(review.rating_id)}>
                삭제
            </DeleteButton>
        </TableCell>
    </TableRow>
  </Table>
  </TableBody>
</Table>
</TableContainer>
{/* 리뷰 등록 페이지로 이동하는 버튼 */}
<ButtonContainer>
  <RegisterButton to={`/HealthMachineShare/machines/reviews/register/${machineId}`}>
    평점 및 리뷰 등록
  </RegisterButton>
</ButtonContainer>
</Container>
);
}

export default HealthMachineReviewTable;

```

- 데이터 가져오기: fetchData 함수를 통해 서버로부터 평점 및 목록 데이터를 가져온다. axios 라이브러리를 사용하여 GET 요청을 보내고, 응답으로 받은 데이터를 items에 설정한다.
- 데이터 삭제: deleteReview 함수는 리뷰를 삭제하는 역할을 한다. 해당 리뷰의 rating_id를 이용하고 axios.delete 메서드를 사용하여 DELETE 요청을 보낸다. 삭제 후에는 리뷰 목록을 다시 가져와서 업데이트한다.
- 별점 표시: renderRatingStars 함수는 평점을 별로 표시하는 역할을 한다. 평점에 따라 채워진 별과 비어있는 별을 조합하여 문자열 형태로 반환한다. rating 값에 따라 채워진 별과 비어있는 별을 조합하여 표시한다.
- 테이블 렌더링: 컴포넌트는 Table, TableHead, TableBody, TableRow, TableCell 등의 MUI 컴포넌트를 사용하여 테이블을 구성한다. items 값을 이용하여 각 평점에 대한 정보를 사용자에게 보여준다. 또한 삭제 버튼을 추가하고 삭제 버튼을 해당 리뷰를 삭제하는 deleteReview 함수가 실행된다.
- 리뷰 등록 페이지로 이동: 해당 머신의 ID를 이용하여서 리뷰 등록 페이지로 이동하는 버튼을 생성한다.
- ▶ 헬스 머신의 리뷰 목록을 표시하고 리뷰를 삭제할 수 있는 기능을 제공한다. 또한 해당 머신에 대한 리뷰 등록 페이지로 이동할 수 있는 기능을 제공한다.

HealthMachineReviewRegisterTable.js

```
//20190633 총제용
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams } from 'react-router-dom';
import {
  Button,
  FormControl,
  InputLabel,
  MenuItem,
  Select,
  TextField,
  Typography,
} from '@mui/material';
import Rating from 'react-rating-stars-component';
import { styled } from '@mui/system';

const EXPRESS_URL = 'https://healthmachineproject.run.goorm.site';

// 스타일링된 컴포넌트들의 스타일을 정의한 CSS-in-JS 코드
const FormContainer = styled('div')({
  maxWidth: '400px',
  margin: '0 auto',
  padding: '20px',
});

const FormHeading = styled(Typography)({
  textAlign: 'center',
  marginBottom: '20px',
});

const InputField = styled(TextField)({
  width: '100%',
  marginBottom: '20px',
});

const SelectField = styled(FormControl)({
  width: '100%',
  marginBottom: '10px',
  minWidth: '200px',
});

const SubmitButton = styled(Button)({
  width: '100%',
});

const ErrorMessage = styled(Typography)({
  color: 'red',
  margin: '10px 0 0 0',
});
```

```

const RatingContainer = styled('div')({
  display: 'flex',
  alignItems: 'center',
  justifyContent: 'center',
  marginBottom: '20px',
});

const RatingLabel = styled(Typography)({
  marginRight: '10px',
});

function HealthMachineReviewRegisterTable() {
  const [rating, setRating] = useState(0);
  const [review, setReview] = useState('');
  const [userId, setUserId] = useState('');
  const [password, setPassword] = useState('');
  const [users, setUsers] = useState([]);
  const [errorMessage, setErrorMessage] = useState('');
  const { machineId } = useParams();

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        // 서버로부터 사용자 목록을 가져옴
        const res = await axios.get(`${EXPRESS_URL}/HealthMachineShare/users`);
        setUsers(res.data.users);
      } catch (error) {
        console.log(error);
      }
    };

    fetchUsers();
  }, []);

  // 평점 변경 시 상태 업데이트
  const handleRatingChange = (value) => {
    setRating(value);
  };

  // 리뷰 내용 변경 시 상태 업데이트
  const handleReviewChange = (e) => {
    setReview(e.target.value);
  };

  // 사용자 선택 시 상태 업데이트
  const handleUserChange = (e) => {
    setUserId(e.target.value);
  };
}

```

```

// 비밀번호 입력 시 상태 업데이트
const handlePasswordChange = (e) => {
  setPassword(e.target.value);
};

// 폼 제출 시 실행되는 함수
const handleFormSubmit = async (e) => {
  e.preventDefault();

  try {
    // 로그인 요청을 보내 사용자 인증
    const loginResponse = await axios.post(
      `${EXPRESS_URL}/HealthMachineShare/users/login`,
      {
        username: userId,
        password,
      }
    );
    // 리뷰 등록을 진행
    await axios.post(
      `${EXPRESS_URL}/HealthMachineShare/machines/reviews/register/${machineId}`,
      {
        userId: loginResponse.data.userId,
        rating,
        review,
      }
    );

    // 등록 후 필드 초기화
    setRating(0);
    setReview('');
    setPassword('');
    setErrorMessage('');
  } catch (error) {
    // 오류 응답이 있을 경우 에러 메시지 표시
    if (error.response && error.response.data) {
      setErrorMessage(error.response.data.message);
    } else {
      setErrorMessage('서버 오류');
    }
  }
};

return (
  <div className="review-registration-container">
    <FormContainer>
      <FormHeading variant="h4">평점 및 리뷰 등록</FormHeading>
      <form onSubmit={handleFormSubmit}>
        <SelectField>
          <InputLabel>유저 선택</InputLabel>

```



```

        <Select value={userId} onChange={handleUserChange} required>
            <MenuItem value="">
                <em>유저 선택</em>
            </MenuItem>
            { /* 사용자 목록을 동적으로 렌더링 */ }
            {users.map((user) => (
                <MenuItem key={user.id} value={user.username}>
                    {user.username}
                </MenuItem>
            ))}
        </Select>
    </SelectField>

    {errorMessage && <ErrorMessage>{errorMessage}</ErrorMessage>}

    <InputField
        type="password"
        value={password}
        onChange={handlePasswordChange}
        required
        label="비밀번호"
    />
    <InputField
        value={review}
        onChange={handleReviewChange}
        required
        label="리뷰"
        multiline
        rows={4}
    />
    <RatingContainer>
        <RatingLabel>평점:</RatingLabel>
        { /* 평점을 입력받는 Rating 컴포넌트 */ }
        <Rating
            count={5}
            size={24}
            activeColor="#ffd700"
            value={rating}
            onChange={handleRatingChange}
            required
        />
    </RatingContainer>
    <SubmitButton type="submit" variant="contained">
        등록
    </SubmitButton>
</form>
</FormContainer>
</div>
);
}

export default HealthMachineReviewRegisterTable;

```

- 데이터 가져오기: fetchUsers 함수를 통해 서버로부터 사용자 목록 데이터를 가져온다. axios 라이브러리를 사용하여 GET 요청을 보내고, 응답으로 받은 데이터를 users에 저장한다.
- 상태 관리: useState를 사용하여 평점 및 리뷰등록에 필요한 정보들을 관리한다.
- 입력값 변경 핸들러: handleRatingChange 함수는 평점이 변경될 때 rating 상태 변수를 업데이트한다. handleReviewChange 함수는 리뷰 내용이 변경될 때 review 상태 변수를 업데이트한다. handleUserChange 함수는 사용자 선택이 변경될 때 userId 상태 변수를 업데이트한다. handlePasswordChange 함수는 비밀번호 입력이 변경될 때 password 상태 변수를 업데이트한다.
- 평점 입력: react-rating-stars-component 라이브러리를 사용하여 평점을 입력받는 기능을 제공한다.
- 폼 제출 핸들러: handleFormSubmit 함수는 폼 제출 이벤트를 처리한다. 먼저, 로그인 요청을 보내 사용자 인증을 진행한다. 성공적으로 인증된 경우, 서버에 평점 및 리뷰 정보를 전송한다. 성공적인 응답을 받으면 필드를 초기화하고 오류 메시지를 초기화한다. 오류 응답이 있는 경우, 해당 오류 메시지를 errorMessage 상태에 저장한다.
- ▶ 사용자 선택, 비밀번호 입력, 리뷰 내용, 평점을 입력받아 헬스 머신의 리뷰를 등록하는 기능을 제공한다.

express.js

```
// 머신 평점 및 리뷰 등록 API에 대한 POST 요청 처리
app.post('/HealthMachineShare/machines/reviews/register/:machineId', (req, res) => {
  const { machineId } = req.params;
  const { userId, rating, review } = req.body;

  // 머신 평점 및 리뷰 등록
  const insertReviewQuery = `INSERT INTO ratings_reviews (machine_id, user_id, rating, review) VALUES (${machineId}, ${userId}, ${rating}, '${review}')`;
  db.query(insertReviewQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    res.json({ message: '평점 및 리뷰가 제출되었다.' });
  });
});

// 리뷰 및 평점 목록 조회 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/machines/reviews/:machineId', (req, res) => {
  const { machineId } = req.params;
  const sql = `SELECT ratings_reviews.rating_id, machines.name AS machine_name, users.username AS user_name, ratings_reviews.rating, ratings_reviews.review
    FROM ratings_reviews
    LEFT JOIN machines ON ratings_reviews.machine_id = machines.machine_id
    LEFT JOIN users ON ratings_reviews.user_id = users.user_id WHERE machines.machine_id = ?`;
```

```

db.query(sql, [machineId], (err, rows) => {
  if (err) {
    console.log(err);
    return res.json({ result: 'error' });
  }
  res.json({ reviews: rows });
});

// 머신 리뷰 및 평점 삭제 API에 대한 DELETE 요청 처리
app.delete('/HealthMachineShare/machines/reviews/:reviewId', (req, res) => {
  const { reviewId } = req.params;

  // 리뷰 삭제
  const deleteReviewQuery = `DELETE FROM ratings_reviews WHERE rating_id = ?`;
  db.query(deleteReviewQuery, [reviewId], (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    if (result.affectedRows === 0) {
      res.status(404).json({ message: '존재하지 않는 리뷰이다.' });
    } else {
      res.json({ message: '리뷰가 삭제되었다.' });
    }
  });
});

// 사용자 목록 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/users', (req, res) => {
  // DB에서 모든 사용자 가져오기
  const getUsersQuery = 'SELECT * FROM users';
  db.query(getUsersQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    res.json({ users: result });
  });
});

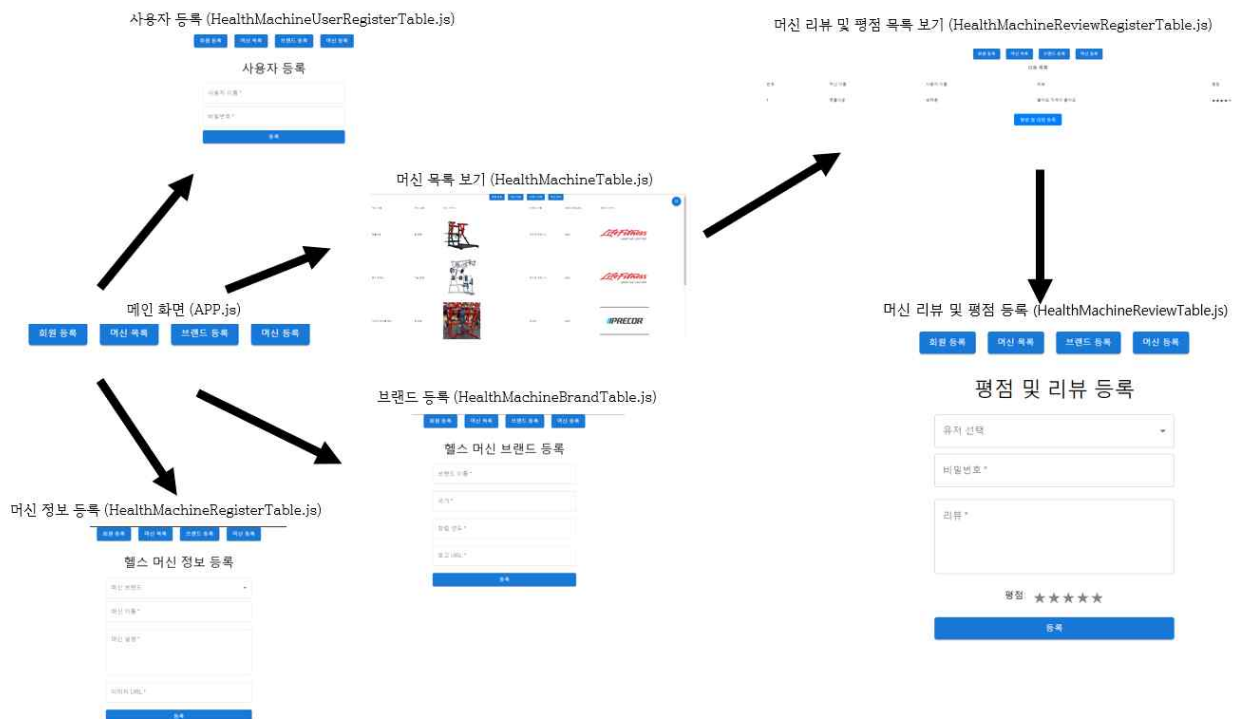
```

- 머신 평점 및 리뷰 등록 처리: 머신에 대한 평점과 리뷰를 등록한다. 파라미터에서 machineId를 추출하고, Body에서 userId, rating, review를 추출한다. 추출한 데이터를 사용하여 ratings_reviews 테이블에 INSERT 쿼리를 실행하여 평점과 리뷰를 등록한다.
- 리뷰 및 평점 목록 조회 처리: 특정 머신에 대한 리뷰와 평점 목록을 조회한다. 파라미터에서 machineId를 추출하고, 추출한 machineId를 사용하여 ratings_reviews, machines, users 테이블

을 LEFT JOIN 하는 SELECT 쿼리를 실행하여 반환한다.

- 머신 리뷰 및 평점 삭제 처리: 특정 리뷰와 평점을 삭제한다. 요청의 파라미터에서 reviewId를 추출하고, 추출한 reviewId를 사용하여 ratings_reviews 테이블에서 해당 리뷰를 DELETE 쿼리를 실행하여 삭제한다.
- 사용자 목록 조회 처리: 모든 사용자 목록을 조회한다. users 테이블에서 모든 사용자를 SELECT 쿼리를 실행하여 조회한 결과를 반환한다.
- ▶ 헬스 머신 평점과 리뷰를 등록하고 조회, 삭제하는 API 부분과 사용자 목록을 조회하는 API 부분이다.

- 전체 흐름도



- 전체 백엔드 코드

```

express.js
//20190633 송제용
import express from 'express';
import mysql from 'mysql';
import bodyParser from 'body-parser';
import cors from 'cors';
import multer from 'multer';

import dbconf from './conf/auth.js';

const app = express();
const port = 3010;

const db = mysql.createConnection(dbconf);

db.connect();
  
```

```

app.use(cors());
app.use(bodyParser.json());

// 브랜드이미지 저장 경로
const brandImageStorage = multer.diskStorage({
  destination: '/workspace/HealthMachineProject/client/public/brand_images',
  filename: (req, file, cb) => {
    cb(null, `${Date.now()}-${file.originalname}`);
  },
});

// 머신이미지 저장 경로
const machineImageStorage = multer.diskStorage({
  destination: '/workspace/HealthMachineProject/client/public/machine_images',
  filename: (req, file, cb) => {
    cb(null, `${Date.now()}-${file.originalname}`);
  },
});

const uploadBrandImage = multer({ storage: brandImageStorage });
const uploadMachineImage = multer({ storage: machineImageStorage });

// 사용자 등록 API에 대한 POST 요청 처리
app.post('/HealthMachineShare/users/register', (req, res) => {
  const { username, password } = req.body;

  // 등록된 사용자인지 확인하기 위해 DB에서 사용자 검색
  const checkUserQuery = `SELECT * FROM users WHERE username = '${username}':`;
  db.query(checkUserQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }

    if (result.length > 0) {
      res.status(400).json({ message: '이미 등록된 사용자 이름이다.' });
    } else {
      // 새로운 사용자 등록
      const insertUserQuery = `INSERT INTO users (username, password) VALUES ('${username}', '${password}')`;
      db.query(insertUserQuery, (err, result) => {
        if (err) {
          res.status(500).json({ message: '서버 오류' });
          return console.log(err);
        }
        res.json({ userId: result.insertId, message: '사용자 등록이 완료되었다.' });
      });
    }
  });
});

// 로그인 API에 대한 POST 요청 처리

```

```

app.post('/HealthMachineShare/users/login', (req, res) => {
  const { username, password } = req.body;

  // 사용자명과 비밀번호를 검증하기 위해 DB에서 사용자 검색
  const loginUserQuery = `SELECT * FROM users WHERE username = '${username}' AND password = '${password}';`;
  db.query(loginUserQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }

    if (result.length > 0) {
      res.json({ userId: result[0].user_id, message: '로그인이 완료되었다.' });
    } else {
      res.status(400).json({ message: '비밀번호가 일치하지 않다.' });
    }
  });
});

// 사용자 목록 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/users', (req, res) => {
  // DB에서 모든 사용자 가져오기
  const getUsersQuery = 'SELECT * FROM users;';
  db.query(getUsersQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    res.json({ users: result });
  });
});

// 머신 목록 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/machines', (req, res) => {
  // 머신 정보와 해당 머신의 브랜드 정보를 조인하여 가져오는 SQL 쿼리
  const sql = `SELECT machines.machine_id ,machines.name AS machine_name, machines.description, machines.image_name AS machine_image,
                brands.name AS brand_name, brands.founding_year, brands.image_name AS brand_image
                FROM machines
                INNER JOIN brands ON machines.brand_id = brands.brand_id;`;

  db.query(sql, (err, rows) => {
    if (err) {
      res.json({ result: 'error' });
      return console.log(err);
    }
    res.json({ machines: rows });
  });
});

```

```

// 브랜드 정보 등록 API에 대한 POST 요청 처리
app.post('/HealthMachineShare/brands', uploadBrandImage.single('image'), (req, res) => {
  const { name, country, founding_year } = req.body;
  const image = req.file.filename;

  // 등록된 브랜드인지 확인하기 위해 DB에서 브랜드 검색
  const checkBrandQuery = `SELECT * FROM brands WHERE name = ?`;
  db.query(checkBrandQuery, [name], (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }

    if (result.length > 0) {
      res.status(400).json({ message: '이미 등록된 브랜드 이름이다.' });
    } else {
      // 새로운 브랜드 등록
      const insertBrandQuery = `INSERT INTO brands (name, country, founding_year, image_name) VALUES (?, ?, ?, ?)`;
      db.query(insertBrandQuery, [name, country, founding_year, image], (err, result) => {
        if (err) {
          res.status(500).json({ message: '서버 오류' });
          return console.log(err);
        }
        res.json({
          brandId: result.insertId,
          message: '브랜드 정보 등록이 완료되었다.',
        });
      });
    }
  });
});

// 브랜드 정보 목록 보기 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/brands', (req, res) => {
  // DB에서 모든 브랜드 가져오기
  const getBrandsQuery = 'SELECT * FROM brands';
  db.query(getBrandsQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }

    res.json({ brands: result });
  });
});

// 머신 정보 등록 API에 대한 POST 요청 처리
app.post(

```

```

'/HealthMachineShare/machines/register',uploadMachineImage.single('image'),
(req, res) => {
  const { brand, name, description } = req.body;
  const image = req.file.filename;

  // 등록된 브랜드의 brand_id 가져오기
  const getBrandIdQuery = 'SELECT brand_id FROM brands WHERE name = ?';
  db.query(getBrandIdQuery, [brand], (err, brandResult) => {
    if (err) {
      console.log(err);
      return res.status(500).json({ message: '서버 오류' });
    }

    const brandId = brandResult[0].brand_id;

    // 새로운 머신 등록
    const insertMachineQuery =
      'INSERT INTO machines (brand_id, name, description, image_name) VALUES (?,
?, ?, ?)';

    db.query(insertMachineQuery, [brandId, name, description, image], (err, result) => {
      if (err) {
        console.log(err);
        return res.status(500).json({ message: '서버 오류' });
      }
      res.json({
        machineId: result.insertId,
        message: '머신 정보 등록이 완료되었다.',
      });
    });
  });
}
);

// 머신 평점 및 리뷰 등록 API에 대한 POST 요청 처리
app.post('/HealthMachineShare/machines/reviews/register/:machineId', (req, res) => {
  const { machineId } = req.params;
  const { userId, rating, review } = req.body;

  // 머신 평점 및 리뷰 등록
  const insertReviewQuery = `INSERT INTO ratings_reviews (machine_id, user_id, rating, review) VALUES (${machineId},
${userId}, ${rating}, '${review}')`;
  db.query(insertReviewQuery, (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    res.json({ message: '평점 및 리뷰가 제출되었다.' });
  });
});

```



```

// 리뷰 및 평점 목록 조회 API에 대한 GET 요청 처리
app.get('/HealthMachineShare/machines/reviews/:machineId', (req, res) => {
  const { machineId } = req.params;
  const sql = `SELECT ratings_reviews.rating_id, machines.name AS machine_name, users.username AS user_name, ratings_reviews.rating, ratings_reviews.review
    FROM ratings_reviews
    LEFT JOIN machines ON ratings_reviews.machine_id = machines.machine_id
    LEFT JOIN users ON ratings_reviews.user_id = users.user_id WHERE machines.machine_id = ?`;

  db.query(sql, [machineId], (err, rows) => {
    if (err) {
      console.log(err);
      return res.json({ result: 'error' });
    }
    res.json({ reviews: rows });
  });
});

// 머신 리뷰 및 평점 삭제 API에 대한 DELETE 요청 처리
app.delete('/HealthMachineShare/machines/reviews/:reviewId', (req, res) => {
  const { reviewId } = req.params;

  // 리뷰 삭제
  const deleteReviewQuery = `DELETE FROM ratings_reviews WHERE rating_id = ?`;
  db.query(deleteReviewQuery, [reviewId], (err, result) => {
    if (err) {
      res.status(500).json({ message: '서버 오류' });
      return console.log(err);
    }
    if (result.affectedRows === 0) {
      res.status(404).json({ message: '존재하지 않는 리뷰이다.' });
    } else {
      res.json({ message: '리뷰가 삭제되었다.' });
    }
  });
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});

```

- MySQL 데이터베이스와 연결하기 위해 dbconf 파일을 import하고, db.connect()를 사용하여 연결을 수행한다.
- CORS(Cross-Origin Resource Sharing)를 허용하기 위해 cors 미들웨어를 사용한다.
- 요청 본문의 JSON 데이터를 파싱하기 위해 bodyParser 미들웨어를 사용한다.
- 이 외에 설명은 위에서 자세하기 기술하였으므로 생략하겠다.

5. 결 론

- 개발 내용 및 실험 결과 요약

이번 프로젝트에서는 Express를 사용하여 헬스 머신 정보를 관리하는 서버를 개발했다. 주요 기능으로는 사용자 등록, 로그인, 브랜드 및 머신 정보 등록, 목록 조회, 머신 리뷰 및 평점 등록, 목록 조회, 리뷰 및 평점 삭제 등이 있다. 이를 위해 MySQL 데이터베이스와 연동하여 데이터를 저장하고 조회하는 기능을 구현했다. 파일 업로드는 multer 미들웨어를 활용하여 사용자에게 입력받은 이미지를 서버에 저장하고 데이터베이스에는 파일 이름을 저장하는 방식으로 구현했다.

- 향후 개선 과제

- 인증과 보안 강화: 현재는 사용자의 비밀번호가 평문으로 저장되고 있으며, 인증 절차도 단순하다. 보안을 강화하기 위해 비밀번호를 해시하여 저장하고, 인증 과정에 JWT(JSON Web Token) 또는 세션을 활용하는 등의 방법을 도입할 수 있다.
- 입력값 검증 및 예외 처리: 현재 코드에서는 입력값에 대한 검증과 예외 처리가 제한적이다. 사용자의 잘못된 입력이나 예외 상황에 대한 적절한 처리를 추가하여 안정성을 높일 수 있다.
- 데이터베이스 성능 최적화: 대량의 데이터가 쌓이면 데이터베이스 성능에 영향을 줄 수 있다. 쿼리의 실행 계획을 분석하고 인덱스를 적절하게 설정하여 데이터베이스 성능을 최적화할 수 있다.
- 코드 구조 개선: 현재 코드는 하나의 파일에 모든 API 핸들러와 라우팅이 구현되어 있다. 코드의 재사용성과 유지보수성을 높이기 위해 모듈화를 고려하여 코드를 여러 파일로 분리할 수 있다. 각 기능과 역할에 따라 모듈을 나누고, 코드의 가독성과 확장성을 고려하여 리팩토링할 수 있다.
- 테스트 추가: 현재 코드에서는 테스트 코드가 제공되지 않았다. 향후 개선 과제로는 단위 테스트, 통합 테스트 등을 추가하여 코드의 정확성과 안정성을 검증할 수 있다. Jest, Mocha, Supertest 등의 테스트 프레임워크를 사용하여 테스트 코드를 작성하고 실행할 수 있다.

- 기술적, 사회적, 경제적 파급 효과 및 기대 효과

- 헬스 머신 정보 공유 플랫폼 개발: Health Machine Share는 헬스 머신의 정보, 사용자들의 경험 및 평가를 공유하는 웹 어플리케이션이다. 이 프로젝트를 깃허브를 통해 오픈소스로 제공한다면 개발자들은 해당 코드를 기반으로 실제 웹 어플리케이션 개발 경험을 쌓을 수 있으며, 웹 개발 기술과 프레임워크에 대한 이해도를 향상시킬 수 있다.
- 헬스 머신 사용자들의 경험 공유: Health Machine Share를 통해 사용자들은 헬스 머신에 대한 경험과 평가를 공유할 수 있다. 이를 통해 다른 사용자들은 헬스 머신의 성능, 기능, 사용자 편의성 등에 대한 정보를 얻을 수 있으며, 헬스 머신 선택에 도움을 받을 수 있다. 또한, 사용자들 간의 상호작용을 통해 헬스 관련 지식을 공유하고 커뮤니티를 형성할 수 있다.
- 헬스 머신 업체와의 협업 기회: Health Machine Share는 헬스 머신 업체들과의 협업 기회를 제공할 수 있다. 업체들은 사용자들의 평가와 피드백을 통해 제품의 개선 방향성을 파악하고 마케팅에 활용할 수 있다. 또한, 프로그램을 통해 머신의 인기도와 수요를 파악할 수 있어 신제품 개발 및 기획에 도움을 받을 수 있다.
- 헬스 머신 선택 및 활용 비용 절감: Health Machine Share를 통해 사용자들은 다양한 헬스 머신의 정보와 평가를 비교 분석할 수 있다. 이를 통해 헬스 머신 선택에 필요한 시간과 비용을 절감

할 수 있으며, 불필요한 헬스 머신 구매를 예방할 수 있다.

- 중고 헬스 머신 시장 활성화: 사용자들이 헬스 머신의 정보와 평가를 공유하고 구매 결정을 내릴 수 있게 되면 중고 헬스 머신 시장이 활성화될 수 있다. 이는 중고 헬스 머신 판매자들과 구매자들 간의 거래를 촉진시키며, 경제적인 이점을 제공할 수 있다.

6. 자체 분석과 평가

- 수행 과정과 결과에 대한 개인적인 생각 및 평가

이번 프로젝트를 수행하면서 많은 것을 배웠고 성취감을 느꼈다. 코드를 작성하고 테스트하는 과정에서 개발이 진행되는 것을 경험하며 기능을 구현하는 즐거움을 느낄 수 있었다. 또한, 프론트엔드와 백엔드 간의 통신, 데이터베이스와의 상호작용 등 다양한 기술을 사용하여 실제 서비스를 구현하는 과정에서 특히 재밌었던 것 같다. 지금까지 해왔던 과제는 실무와 연관되어있지 않아서 크게 흥미를 느끼지 못했었는데 이번 프로젝트를 통해 실무에 가까운 경험을 쌓을 수 있었고, 개발자로서의 역량을 향상시킬 수 있었다. 앞으로도 이러한 프로젝트를 혼자 진행해보며 경험을 쌓고 좋은 개발자가 되고싶다.

프로젝트의 결과에 대해서는 학기중에 다른 수업도 들으며 수행했음에도 나름 만족스러운 성과를 얻었다고 생각하고 있다. 하지만 시간의 제약 때문에 기능을 완벽하게 기능하지 못하였고 보완해야할 부분이 많다. 해당 기능들은 방학때 구현해보고싶은 욕심은 있다.

- 설계서에 작성했지만 구현하지 못한 부분

교수님께서 시간이 없다는 점을 강조하셔서 설계서를 작성하면서 구현을 시작하였기 때문에 설계서에 작성했지만 구현하지 못한 부분은 없다. 그래서 추가로 보완해야할 부분을 기술하겠다.

- 로그인 기능 구현 : JWT를 이용한 로그인 기능
- 검색 및 필터링 기능 : 사용자들이 특정 브랜드, 머신 종류 또는 특정 기능을 검색하고 필터링하여 원하는 정보를 빠르게 찾을 수 있는 기능
- 관리자 기능: 관리자용 대시보드를 만들어서 사용자 계정 관리, 머신 및 브랜드 정보 수정, 리뷰 및 평점 관리 등을 관리자가 효율적으로 수행할 수 있는 기능

- 과제를 수행하면서 어려웠던 점과 그 문제를 해결한 방법

가장 어려웠던 점은 서버와의 데이터 통신과 데이터베이스 연동이었다. 스프링 프레임 워크는 다뤄보았으나 express.js는 처음이라 API 요청과 응답 처리, SQL 쿼리 작성에 익숙하지 않아 많은 시간을 소요했다. 이를 해결하기 위해 공식 문서와 온라인 자료를 참고하여 express.js에 대한 이해를 높였다. 또한, 에러 메시지와 로깅을 통해 발생한 문제를 파악하고 디버깅하는 과정을 통해 오류를 해결했다.

프로젝트 기간 내에 모든 기능을 완벽하게 구현하지 못한 점은 아쉽다. 시간이 부족하거나 기술적인 한계 등의 이유로 일부 기능은 생략하거나 단순화된 형태로 구현했다. 가장 아쉬운 부분은 인증과 보안 방식이다. 시간이 더 있었다면 비밀번호를 해시로 저장 보안을 강화하고 JWT를 사용하여서 로그인 기능을 보완해보고 싶다. 그래도 우선순위를 정하고 핵심 기능을 우선 구현함으로써 주요 요구사항을 충족시킨 것 같아 만족스럽다.

프로그램을 테스트하면서 CORS 에러가 발생했는데 해당 에러를 해결하는 것도 어려웠다. CORS에러에 대해서 간단히 설명하자면 CORS 정책 때문에, 리소스를 가져오는게 블락(금지) 되었기 때문에 해당 파일을 접근하지 못하는 에러이다. 해당 에러를 해결하기 위해 온라인 자료를 참고하여 해결하였다.

7. 부록

- 개발 일정

- 2023/05/24 구현 시작
- 2023/05/28 프로그램 초기 단계 완성
- 2023/06/15 프로그램 구현 완료
- 2023/06/11 최종 보고서 초안 작성
- 2023/06/18 최종 보고서 기술 완료

참고문헌

- [1] Nodejs 테스트 코드 <https://seungjuitmemo.tistory.com/269>
- [2] CORS 에러 해결 <https://i5i5.tistory.com/935>