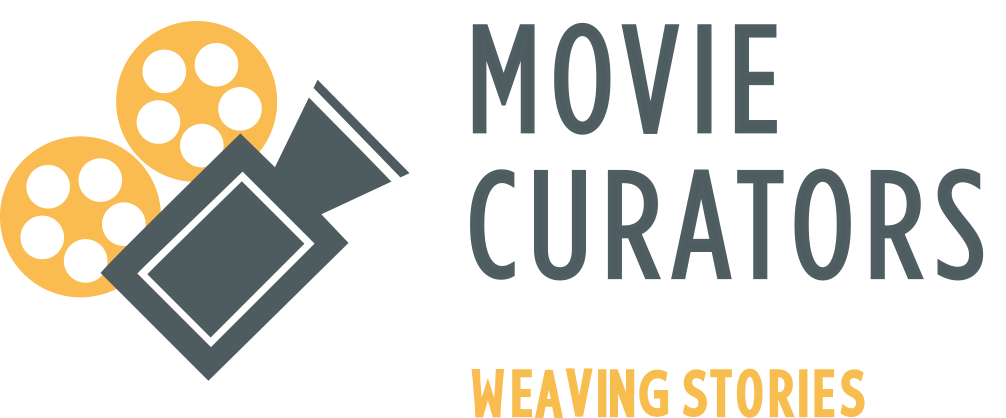
|  |
| --- |
|  |



|  |
| --- |
| **Movie Curators** |
| **시스템 기술서** |
|  |
|  |
|  |
|  |
|  |



[I. 프로젝트 개요 3](#_Toc92124392)

[1. 프로젝트 개요 3](#_Toc92124393)

[2. 프로젝트 배경 3](#_Toc92124394)

[II. 프로젝트 구성 5](#_Toc92124395)

[1. 데이터 전처리/입력 5](#_Toc92124396)

[가) **빅데이터 머신 러닝** 5](#_Toc92124397)

[나)  **인구통계학적 필터링** 6](#_Toc92124398)

[다)  **컨텐츠 기반 필터링** 7](#_Toc92124399)

[2. 백엔드 프레임 워크 8](#_Toc92124400)

[가) **Dependency (구조)** 8](#_Toc92124401)

[나) **Api (RestController)** 10](#_Toc92124402)

[다) **Domain (Entity)** 10](#_Toc92124403)

[라) **Service** 11](#_Toc92124404)

[마) **Repository** 12](#_Toc92124405)

[3. 프론트 엔드 프레임 워크 13](#_Toc92124406)

[가) **Dependency** 13](#_Toc92124407)

[나) **구조** 14](#_Toc92124408)

[4. 배포 (Deployment) 17](#_Toc92124409)

[가) **EC2** 17](#_Toc92124410)

[나) **RDS** 17](#_Toc92124411)

[다) **Google Cloud Storage** 17](#_Toc92124412)

[라) **Netlify** 17](#_Toc92124413)

[III. 프로젝트 주요 Concept 18](#_Toc92124414)

[1. 사용자 경험 극대화 18](#_Toc92124415)

[2. Frameworks (Django vs Spring) 18](#_Toc92124416)

[IV. 정리 19](#_Toc92124417)

[1. 사용 기술 스택 19](#_Toc92124418)

[2. 산출물 19](#_Toc92124419)

**Movie Curators 시스템 기술서**

# 프로젝트 개요

## **프로젝트 개요**

Movie Curatrors는 영화 정보 기반 추천 서비스입니다. 영화를 주제로 하고, 평점을 기반으로 한 커뮤니티 서비스로, 웹 사이트의 형태로 운영됩니다. 사이트에서 영화의 정보와 함께 토론 공간을 제공하면, 유저들은 영화에 대한 정보를 보며 평가를 생성/수정/삭제 할 수 있고, 타 유저들은 생성된 타 유저의 평가에 대하여 댓글을 생성/수정/삭제하거나, 후원하기, 좋아요 등의 기능을 통해 영화에 대한 평가나 평가 작성자를 지원할 수 있습니다.

[Figure 1] [www.moviecurators.com](http://www.moviecurators.com)의 홈 화면

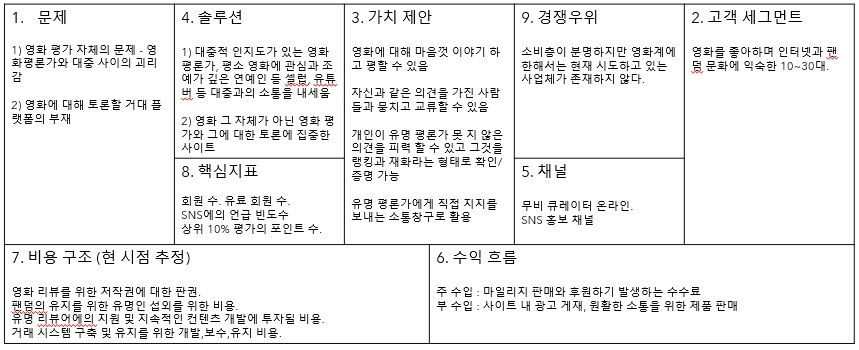


## **프로젝트 배경**

해당 프로젝트에서는 댓글, [좋아요], [후원하기] 등이 핵심 키워드 중 하나로, 이러한 활동이 활발히 성립할 수 있는 환경을 구축하는 것이 중요합니다. 그러나 네이버, 다음으로 대표되는 블로그나 카페 등, 대다수의 커뮤니티만 보아도, 일반적인 사용자에게 타인의 게시글에 [후원하기]는 물론 [좋아요]를 할 만한 문화가 정착되어 있기는커녕, 동기조차 거의 없다는 결론에 이르렀고, 이는 프로젝트의 근간을 흔들 만한 중요한 문제라고 생각하였습니다.

따라서 Movie Curatrors는 사용자에게 서비스를 구축하고 제공함에 앞서, 사용자의 동기를 채울 만한 가치를 파악하고 제공하기 위하여, 비즈니스 모델을 추가하고 린 캔버스로 분석함으로써, 해당 커뮤니티를 누가 이용할지에 대한 의문에 추가적인 설득력을 부여하고, 단순한 프로젝트로서의 Hello World를 넘어 Real World를 지향할 수 있었습니다.

[Figure 2] 비즈니스 모델 린 캔버스



그 결과 Movie Curators를 구축하기 위하여 다음과 같은 결론을 내렸습니다.

결론 ①

* 영화 그 자체에 중점을 두기 보다는 영화에 대한 평가를 중심으로 한 커뮤니티

결론 ②

* 후원하기, 좋아요 기능등을 그저 구현할 뿐이 아닌 유저들의 커뮤니티를 지원하기 위한 점수, 랭킹, 보상제도로 활용하여 동기를 부여

해당 결론을 바탕으로 한 프로젝트를 추구함으로써, 저희 Movie Curators는 모든 유저가 영화를 추천하고 평가자가 될 수 있는 동시에, 평가나 평가자를 추천하고 평가를 추천하고 지원할 수 있는 후원자의 입장이 될 수 있게 구축하였고, 이러한 특징들을 바탕으로 타인의 평가를 지원할 수 있는 분위기와 동기를 부여할 것입니다.

# 프로젝트 구성

## **데이터 전처리/입력**

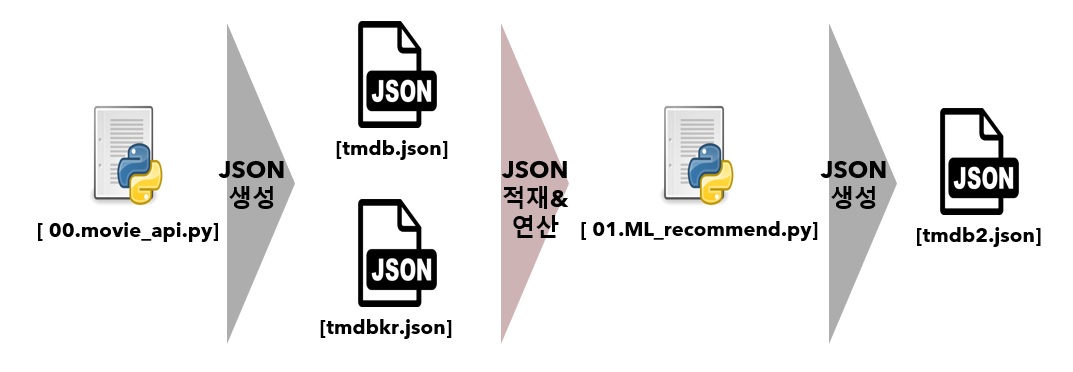
### **빅데이터 머신 러닝**

Movie Curators는 기본적인 인기순, 최신순, 평점순과 판타지, 서부, 로맨스 등 17종의 장르별 구분을 제공하는 것은 물론, Database에 존재하는 Record를 기반으로 한 영화 추천 시스템을 제공합니다.

영화 추천 시스템은 메인 페이지 화면에서 제공할 영화를 추천하는 인구통계학적 필터링과 각각의 영화에 맞춘 컨텐츠 기반 필터링 2가지로 나뉩니다.

* **파일 생성**

[Figure 3] DB 적재 파일 생성

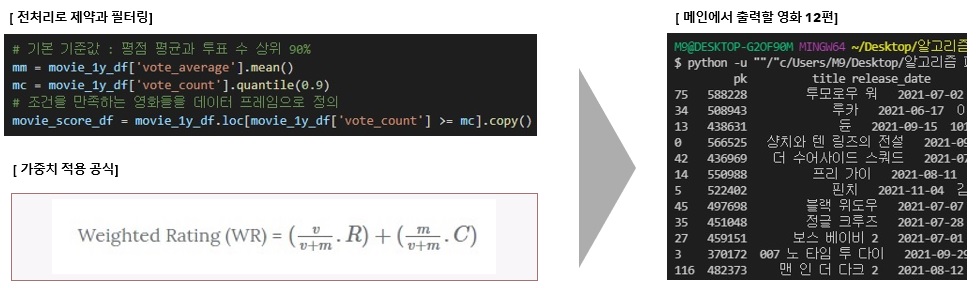


실제 빅데이터 연산을 API 요청시 처리하면 막대한 연산이 필요하기에 그대로 적용한다면 실용성이 떨어질 수 있지만, 미리 파일화 하여 연산하고 DB에 적재해두면 그 연산양을 획기적으로 줄일 수 있었습니다. (행렬 연산O(N^2)-> DB값 참조 O(1))

또한, Movie Curators는 빅데이터 분류와 분석을 위한 라이브러리가 많은 파이썬을 활용, DB에 적재할 Record를 생성하는 프로그램을 만듦으로써, 프로그램 실행 두 번만으로 데이터 전처리와 적재가 가능하게 하였습니다.

### **인구통계학적 필터링**

[Figure 4] 인구통계학적 필터링



메인 페이지에 추천할 영화를 선정하는 방식은 인구 통계학적 필터링(Demographic Filtering)입니다. 인구 통계학적 필터링이란 더 인기 있고 비평가들의 찬사를 받은 영화가 일반 관객에게 더 좋아질 확률이 더 높다. 라는 기본적인 생각을 적용한 방식입니다.

또한, 가중치 방식과 체인 필터링을 적용한 데이터 전처리 방식을 적용 가능하여 뛰어난 데이터 정제가 가능합니다.

* **가중치 방식**

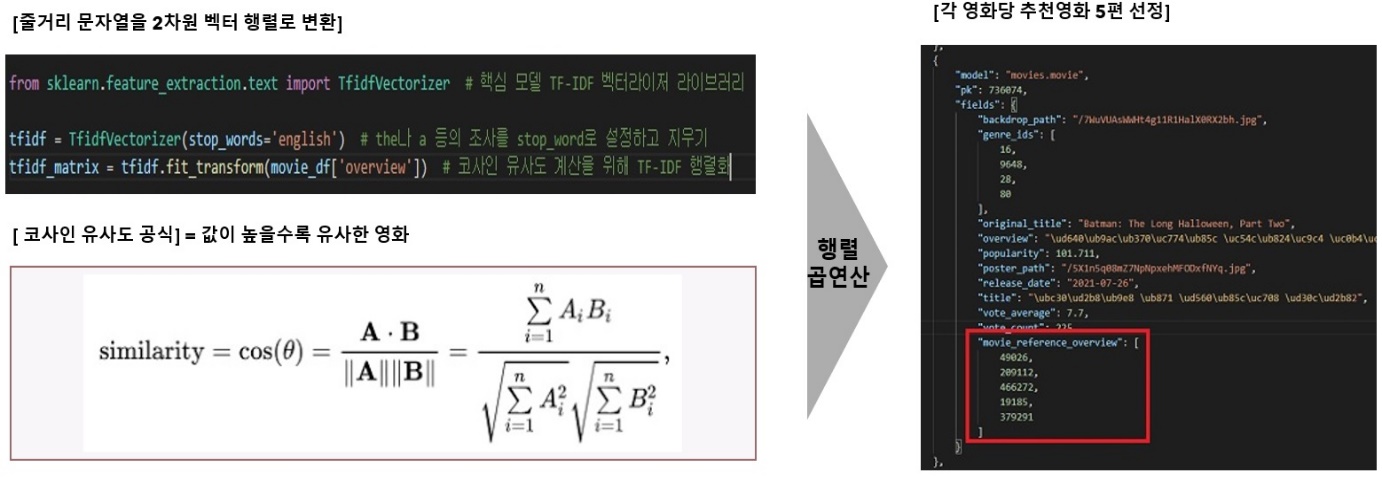
단순 평점 계산시 놓칠 수 있는 부분을 가중치 방식을 통하여, 보완할 수 있습니다. 예를 들어 30명이 투표한 평점 7.7 영화보다 100명이 투표한 평점 7.5 영화를 더 높게 평가할 수 있습니다.

* **데이터 전처리 체인 필터링**

데이터 전처리에 있어 필요한 여러가지 전제를 체인 필터링 방식을 이용하여 병렬적으로 처리할 수 있습니다. 6개월 이내의 영화여야 한다. 평균 이상의 평점을 보유하여야 한다. 투표치 상위 90% 영화여야 한다. 같은 데이터 전처리를 제한간 간섭 없이 알아보기 쉽게 처리할 수 있고 체인 필터링을 이용하여 제한의 추가와 제거를 매우 자유롭고 쉽게 구성하였습니다.

### **컨텐츠 기반 필터링**

[Figure 5] 줄거리 기반(Plot description based) 추천



각각의 영화 추천에는 줄거리 기반 추천(Plot description based Recommender)을 사용하였습니다.

줄거리 설명('overview' 컬럼) 기반으로 모든 영화에 대한 각각의 유사성 점수를 계산하고 유사성 점수를 기반으로 영화를 추천하는 방식입니다.

* **TF-IDF 벡터라이저**

데이터분석 라이브러리 사이킷-런(Scikit-learn)에서 제공하는 모듈입니다. 영화마다 줄거리의 길이가 다르기 때문에 데이터 분석에는 줄거리에 나온 단어 수가 아닌 단어의 빈도로 계산할 필요가 있는데 이에 유용한 모듈입니다.

또한, 벡터라이저에서 자체적으로 제공하는 기능을 이용하여 자주 쓰이지만 데이터 분석에는 의미를 가지지 않는 a나 the등의 관사를 제거할 수 있습니다.

* **코사인 유사성 점수**

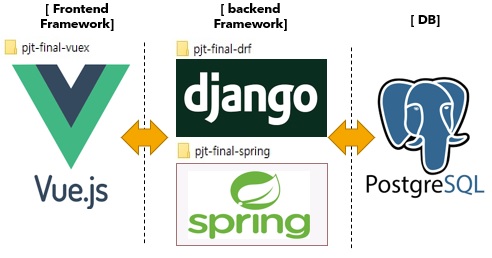
유사성 점수를 계산하는 데에는 계산이 빠르고 상대적으로 단순한 코사인 유사성 점수를 사용하였습니다. 벡터라이저 된 숫자들을 내적하는 것으로 간단하게 구할 수 있습니다. 코사인 유사성 점수가 높을수록 줄거리 상 그 영화와 높은 연관을 가진 영화입니다.

해당 연산을 위해 영화의 개수만큼의 길이를 가지는 정사각행렬간의 계산이 필요하지만, 미리 계산하여 DB에 적재함으로써 API 수행시 그 연산을 수행하지 않게 하였습니다.

## **백엔드 프레임 워크**

유저경험(이하, UX)를 고려하여, Movie Curators는 단일 페이지 어플리케이션(SPA)으로 구축되었고, 그에 맞춘 환경을 제공하기 위하여 백 서버는 REST API로 구축되었습니다.

[Figure 6] 백-프론트-DB 구조 간략화



비교를 위하여 같은 request를 받고 같은 response를 반환하는 로직을 각각 Spring framework와 Django로 구현하였고, 이에 대한 자세한 내용은 ’02. Movie Curators 모형 기술서’를 참고하시길 바랍니다.

현, 시스템 기술서에서는 Spring Framework를 기준으로 간략한 구조 설명이 진행됩니다.

### **Dependency (구조)**

해당 프로젝트를 수행하는데 있어서 Spring에서 사용한 Dependency입니다.

* **Spring Web**

전반적인 Rest Controller 구현 및 작업환경을 구축하였습니다.

* **Spring Security**

Spring Security를 통하여 회원가입을 포함한 회원관리, 비밀번호 암호화, 해쉬 기능, jwt유효 검사 등 필터 및 인터셉터 수행, cors, csrf 이슈 해결 등 보안을 포함한 인증, 권한을 관리하였습니다.

* **JWT**

JSON Web Token을 활용하고 Spring Security와 연계하여, 안전하고 편하게 회원 정보 교환을 수행하였습니다.

* **Validation**

대상 정보가 유효한지 파악하고 관련 오류를 제어하고, message\_xx.yml을 이용하여 국제화 메시지를 구현하였습니다.

* **Spring Data JPA**

Spring에서 제공하는 ORM인 Hibernate를 사용하여 DB에 종속되지 않고, 객체 중심의 개발을 할 수 있었고, Spring Data를 통하여 생산성은 물론 유지/보수 등의 관리도 하기 편한 환경을 구축하였습니다.

또한 Fetch Join 기능을 활용하여 현업에서 발생할 수 있는 ‘1+N’ 이슈 등을 사전에 해결하고 효율적인 쿼리셋을 작성할 수 있었습니다.

* **Lombok**

코드 자체의 가독성을 올리고, Getter나 setter을 많이 활용하는 DTO나 의존성 주입이 필요한 클래스들에게 RequiredArgsConstructor 등의 기능을 제공하여 코드를 간략하게 짤 수 있었습니다.

* **PostgreSQL Driver**

오픈소스인 PostgreSQL Database를 사용하였습니다.(버전 11.6)

* **AWS (EC2 + RDS)**

EC2 서비스를 이용하여 Amazon Linux 플랫폼에서 서버를 구축하고, RDS 서버를 이용하여 DB를 적재하였습니다.

* **Google Cloud Storage**

프로필 이미지 등 static한 이미지를 저장하기 위하여 Google Cloud Platform에서 제공하는 Google Cloud Storage를 이용하였습니다.

* **Kakao Pay**

Kakao API로 제공되는 카카오 결제 시스템을 이용하여, 프리미엄 회원 등의 상품을 판매하고 결제할 수 있게 하였습니다. 결제는 프론트에서 진행되지만 그 내역은 토큰으로 전달되어 백 서버를 거쳐 DB에 저장합니다.

### **Api (RestController)**

Controller로 요청 주소에 따른 Mapping을 시행합니다. Request 요청과 함께 넘겨 받은 Data와 함께 필요할 경우 json web token에서도 Data를 추출하여 지정된 Service를 호출합니다.

* **UserApiController**

회원가입, 로그인, 프로필 수정, 회원 조회, 마일리지 충전, 후원하기 등 회원과 관련된 요청을 받고 응답합니다. 로그인시, JWT 토큰을 발급합니다.

* **HomeApiController**

메인 페이지 진입시 필요한 요청을 받고 응답합니다.

* **MovieApiController**

영화 정보 조회, 영화와 관련된 평가 등 영화와 관련된 요청을 받고 응답합니다.

* **ArticleApiController**

평가 조회, 생성, 삭제, 수정, 좋아요, 평가 포인트 갱신 등 평가와 관련된 요청을 받고 응답합니다.

* **CommentApiController**

댓글 조회, 생성, 삭제, 수정 등 댓글과 관련된 요청을 받고 응답합니다.

* **KakaoApiController**

결제 관련 요청을 받고 응답합니다.

### **Domain (Entity)**

로직을 수행할 때 필요한 클래스들의 집합입니다. JPA를 이용하여, 자동으로 관련 테이블이 생성되며, DB에 종속되지 않은 객체 중심의 클래스 개발이 가능해집니다.

| **Entity** | **Table명** | **설명** |
| --- | --- | --- |
| User | accounts\_user | 유저 정보 관리 |
| Curator | accounts\_curator | 유저와 유저 간 다대다 관계 중간 테이블 |
| Movie | movies\_movie | 영화 정보 관리 |
| Genre | movies\_genre | 장르 정보 관리 |
| MovieGenre | movies\_movie\_genre\_ids | 영화와 장르 간 다대다 관계 중간 테이블 |
| Article | movies\_article | 평가 정보 관리 |
| Likes | movies\_likes | 유저와 평가 간 다대다 관계 중간 테이블 |
| Comment | movies\_comment | 댓글 정보 관리 |

[Table 1] Entity 리스트

### **Service**

Controller에서 호출되어 필요한 Repository나 다른 Service를 주입 받아 필요한 정보를 가져와서 관련 로직을 수행합니다.

각각의 Service가 대체 불가능하고 확장성을 고려할 필요 없을 정도로 1:1로 대응되어 OCP원칙을 지켰을 때 얻을 수 있는 이득이 거의 없었기 때문에 Impl 인터페이스를 만들고 구현체로 구현하는 관습적인 방식을 사용하지 않았습니다.

트랜잭션을 Repository가 아닌 Service에서 일관성 있게 처리하였습니다. 이로 인해 DB에 갱신되지 않는 함수들을 전부 readonly로 처리할 수 있게 되어 처리속도와 효율성을 증가시킬 수 있었습니다.

* **UserService**

회원가입, 중복 검증, 마일리지 충전, 후원하기, 프로필 갱신 등 유저와 관련된 로직을 수행합니다.

* **JwtTokenProvider**

Json web token 발급, 유효성 확인, 정보 추출 등의 로직을 수행합니다.

* **MovieService**

영화 관련 정보 조회 로직을 수행합니다. DB에 영향을 주는 로직이 없습니다.

* **ArticleService**

평가 생성, 수정, 삭제, 좋아요 등 평가와 관련된 로직을 수행합니다.

* **CommentService**

댓글 생성, 수정, 삭제 등 댓글과 관련된 로직을 수행합니다.

* **FileService**

Google Cloud Storage에 올릴 이미지 파일을 변환하고, 적재하는 로직을 수행합니다.

* **KakaoPayService**

카카오 페이 결제 로직을 수행합니다.

### **Repository**

DB에 직접적으로 접근하고 관련 로직을 실행하는 저장소입니다. Spring Data를 이용하여 JPA Repository를 상속받은 인터페이스 형태로 구축함으로써, ORM을 적용한 쿼리를 복잡한 절차 없이 실행할 수 있습니다.

페치 조인을 사용하여, 현업에서 발생할 수 있는 ‘1+N 쿼리’ 문제를 해결하였습니다.

| **Repository 명** | **연계 Domain** | **설명** |
| --- | --- | --- |
| UserRepository | User | 유저 정보 관리 |
| CuratorRepository | Curator | 유저와 유저 간 다대다 관계 중간 테이블 |
| MovieRepository | Movie | 영화 정보 관리 |
| ArticleRepository | Article | 평가 정보 관리 |
| LikesRepository | Likes | 유저와 평가 간 다대다 관계 중간 테이블 |
| CommentRepository | Comment | 댓글 정보 관리 |

[Table 2] Repository 리스트

## **프론트 엔드 프레임 워크**

유저경험(이하, UX)를 고려하여, Movie Curators는 단일 페이지 어플리케이션(SPA)으로 구축되었고, 그에 맞춘 환경을 제공하기 위하여 프론트 서버는 Vue를 활용하였습니다.

### **Dependency**

Vue.js를 기반으로 확장하며 구축하였습니다.

* **Vue**

Vue-router로 히스토리 모드를 적용한 라우터를 설정하였고, Vuetify, Sweetalert2 등의 라이브러리를 설치하여 각종 컴포넌트들을 꾸미고, alert 창과 modal창을 구현하였습니다.

* **Vuex**

공용으로 사용하는 Data를 store에 담고, 함수를 저장하여 관리하였습니다. 그 외에도, 메인 페이지 등 자주 방문하는 곳의 정보를 저장하여, 로딩 속도를 빠르게 하였습니다.

* **Bootstrap**

부트스트랩을 이용하여 디자인하였습니다.

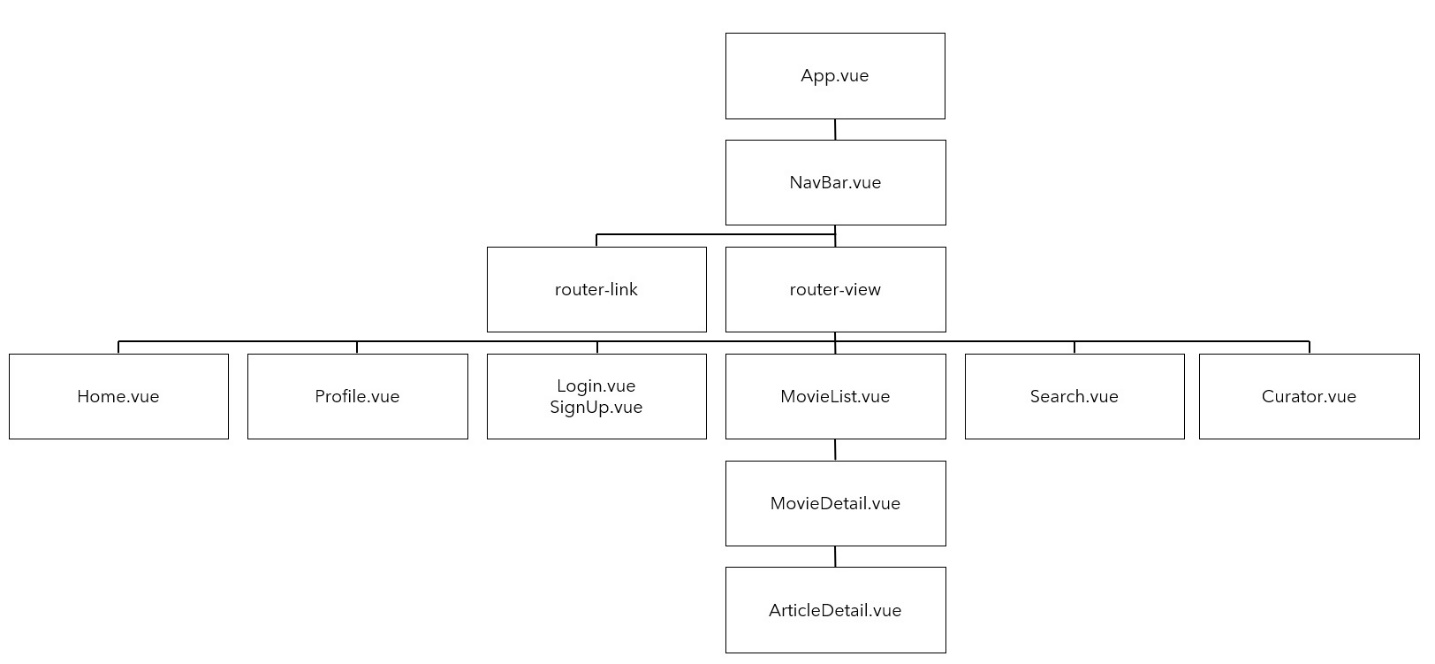
* **OAUTH**

구글 OAUTH를 이용하여, 손님이어도 임시적으로 서비스를 이용할 수 있는 인증 Access 토큰을 발급하였습니다.

* **KakaoPay**

KakaoPay 결제 API와 직접 연결하여, 결제를 신청하고 지정 된 콜백함수로 그 결과와 토큰을 받아서, 결제 내역을 백 서버에 전달하였습니다.

### **구조**

프론트 서버의 상위 구조를 중심으로 설명하겠습니다.

[Figure 7] 컴포넌트 구조 (상위)

* **NavBar.vue**

Fixed Top 되어 항상 표시되며, 어떤 상황에서도 세 번 클릭 안에 원하는 곳으로 갈 수 있게 해주는 UX를 제공하기 위해 필수적인 컴포넌트입니다. 손님 / 구글 로그인 / 회원 등 사용자 상태에 따라서 로그인 / 회원가입 / 로그아웃 등의 표현이 달라집니다.

* **Home.vue**

Movie Curators 웹 사이트에 진입하게 되면 제일 먼저 마주하게 되는 메인 페이지를 관리하는 상위 컴포넌트입니다. MovieReferenceTitle, MovieReferenceCurator, MovieReferenceExtra를 하위 컴포넌트로 가집니다.

영화에 대한 평가마다 각각 좋아요, 후원금액 등 내부 평가를 기준으로 계산한 포인트를 부여하며, 분기 중 많은 지원을 받은 평가들은 홈페이지 상단에 위치하게 되어 다른 유저들에게 자주 노출되게 됩니다.

* **MovieList.vue**

영화를 분류하여 인기순, 최신순, 평점순으로 제공하거나, 판타지, 서부, 로맨스 등 19종의 장르별로 구분하여 제공하는 상위 컴포넌트입니다.

* **Search.vue**

영화 검색을 돕는 상위 컴포넌트입니다. 영어, 한국어, 숫자, 띄어쓰기, 대소문자 이슈에 모두 대응합니다. 해당 기능을 이용하기 위해서는 최소한 구글 로그인을 통한 인증이 필요합니다.

* **Curator.vue**

유저 검색을 돕는 상위 컴포넌트입니다. Curator이란 영화를 평가/추천해 다른 사람들이 흥미로운 영화를 찾을 수 있도록 돕는 모든 유저들을 일컫는 단어입니다.

유저 검색 시, 프리미엄 유저 여부와 함께, 해당 유저가 적은 평가 중 인기있는 평가가 최대 3개 까지 표시됩니다. 해당 기능을 이용하기 위해서는 최소한 구글 로그인을 통한 인증이 필요합니다

* **Signup.vue**

Movie Curators는 진입은 쉽지만, 제공된 모든 서비스를 즐기기 위해서는 결국 회원가입을 해야만 하는 구조로 설계하였습니다. 유저 경험을 고려하여 회원가입시 축하메시지와 함께 자동으로 로그인하게 하며 홈으로 보내게 설정하였습니다.

또한, 아이디와 비밀번호만 입력하면 간단히 가입되지만 정보가 필요해지면 추가로 요청하는 방식인 2단계 가입구조를 채택하였습니다.

* **Login.vue**

로그인을 돕는 상위 컴포넌트입니다. 로그인에 실패할 경우, Sweetalert 라이브러리를 활용하여, 각각의 상황에 맞는 경고 메시지와 함께 modal을 띄워주게 설정하였습니다.

경고 메시지는 백서버의 글로벌 메시지 기능을 활용하여 로케일 언어가 한국어 외에 영어일 경우에도 대응할 수 있게 설정하였습니다.

* **MovieDetail.vue**

선택한 영화와 그 영화에 달린 평가를 볼 수 있는 상위 컴포넌트입니다. 또한 빅데이터 머신러닝 프로세스를 기반으로 선택한 영화와 비슷한 내용의 영화를 추천해줍니다.

* **ArticleDetail.vue**

선택한 평가 글의 상세 내용을 보고 수정하거나 삭제할 수 있습니다. 또한, 해당 평가에 댓글을 작성하거나, [좋아요], [후원하기] 기능을 통하여 평가를 지원할 수 있습니다.

작성된 댓글은 댓글 작성자 외에도 평가 작성자가 삭제할 수 있으며, 후원하기 기능을 통해 더 많은 마일리지를 사용한 후원 댓글이 우선적으로 표기됩니다.

* **Profile.vue**

로그인 한 사용자의 기본적인 사항을 보고 수정할 수 있는 컴포넌트입니다. 프로필 이미지, 닉네임, 자기소개등의 수정이 가능하며, 마일리지 충전버튼을 통해 후원에 필요한 마일리지를 충전할 수 있습니다.

또한, 액자 기능을 제공하여 사용자가 [좋아요]한 평가나 후원한 큐레이터 들을 관람하고 나열할 수 있습니다.

* **NotFound.vue**

올바르지 않은 접근이나 없는 페이지로 넘어갈 경우 보이는 컴포넌트입니다. 이 컴포넌트에 접근할 경우, 안내 메시지를 담은 Modal과 함께 지정된 시간 후 메인 페이지로 넘어갑니다.

## **배포 (Deployment)**

프론트와 백을 각각 Netlify와 AWS, Google Cloud Platform 환경에서 배포했습니다.

### **EC2**

Amazon Linux 환경에서 구축하였습니다. NGINX와 환경변수 Setting, 로드밸런서 등을 설정하여 배포하였습니다.

### **RDS**

PostgreSQL DB를 설치하고 필요 테이블과 Record들을 적재하였습니다.

### **Google Cloud Storage**

Google Cloud Storage 서비스의 Bucket과 Blob을 이용하여 프로필 이미지 등 웹 사이트에서 사용하는 이미지를 static하게 제공하였습니다.

### **Netlify**

Vue에서 build한 dist 폴더를 이용하여 프론트 서버를 배포하고, 도메인을 설정하였습니다.

# 프로젝트 주요 Concept

Movie Curators는 두 가지 주요 컨셉을 기반으로 구축되었습니다. 첫 번째는 유저에게 동기를 심어주고, 그 동기에 부응할 수 있는 큐레이팅 서비스를 제공하여 사용자 경험 극대화하는 것이며 두 번째는 같은 로직을 갖는 백 서버를 Django와 Spring 두 가지 Framework로 구현하여 두 Framework를 비교하고 차이를 학습하는 것입니다.

## **사용자 경험 극대화**

동기 부여를 위해 도입된 시장성 조사와 비즈니스 모델은 ‘어떤’ 유저가 ‘왜’ 이 사이트를 이용할 것인지에 대한 명확한 이유를 제공해줍니다.

웹 사이트를 단일 페이지 어플리케이션(SPA)으로 구축하여, ajax로 동적 데이터를 요청하는 비동기 모델을 통해 반응성을 대폭 향상 시켰습니다.

클릭 세 번으로 모든 곳에 갈 수 있고, 모든 것을 볼 수 있게 구조를 작성하였고, 회원 가입, 경고문, 좋아요, 결제 완료 등 모든 사용자의 행위에 interaction을 부여하여 충족감을 더했습니다.

## **Frameworks (Django vs Spring)**

[Figure 8] Django vs Spring 소개 타이틀

# 정리

## **사용 기술 스택**

[Figure 9] 사용 기술 스택

## **산출물**

Movie Curators는 이하와 같은 산출물들로 구성되었습니다.