

데이터베이스 설계 문서

데이터베이스 선택 과정

NoSQL VS RDBMS

- **데이터 구조의 유연성**
 - NoSQL은 Schema-less로 설계되어 데이터의 구조 변경이 자주 발생하거나, 각 문서(데이터)의 구조가 달라질 수 있는 프로젝트에 적합하다.
 - 유지컬과 전시회 데이터는 각기 다른 필드를 가질 수 있으며, 내용이 유동적이다. 이러한 특성 때문에 **정형화된 테이블**을 요구하는 RDBMS보다는 **JSON 기반의 문서 저장 방식**이 더 적합하다.
- **확장성(Scalability)**
 - NoSQL은 수평적 확장(Horizontal Scaling)이 용이하다. 프로젝트가 성장하여 데이터를 대량으로 저장해야 하거나 읽기/쓰기 요청이 폭발적으로 증가하는 경우에도 손쉽게 클러스터 노드를 확장할 수 있다.
 - 특히 **서비스**에서 사용량이 증가하면 많은 읽기/쓰기 작업이 발생할 가능성이 크기 때문에, NoSQL의 장점인 확장성이 요구된다.
- **LLM RAG와의 적합성**
 - RAG(Retrieval-Augmented Generation) 시스템에서는 문서 기반 데이터 검색과 관련된 작업이 빈번히 발생한다.
 - MongoDB와 같은 NoSQL은 JSON 문서 형태로 데이터를 저장하므로, 벡터 검색과 결합하여 효율적인 검색 작업을 수행할 수 있다. 특히 RAG에서 데이터를 다룰 때 MongoDB와 같은 **문서 기반 데이터 저장소**가 훨씬 유리하다.

MongoDB를 선택한 이유

- **MongoDB의 문서 기반 데이터 모델**
 - MongoDB는 데이터를 **JSON-like BSON(Binary JSON)** 문서로 저장하므로 LLM에서 다루기 적합한 **구조화된 텍스트 데이터**와 잘 맞는다.
- **벡터 검색 지원(Vector Search)**
 - MongoDB Atlas는 내장된 **벡터 검색(Vector Search)** 기능을 지원한다. 이를 통해 임베딩된 벡터 데이터를 효율적으로 저장하고 검색할 수 있으며, 추천 알고리즘 및 유사도 검색 작업이 용이하다.
- **확장성 및 관리 용이성**
 - MongoDB는 AWS DynamoDB와 유사하게 클라우드 기반으로 작동하므로 수평적 확장이 가능하다.

- 하지만 MongoDB는 보다 세밀한 쿼리와 복잡한 검색 작업이 가능하다. 특히 복합 인덱싱이나 텍스트 검색, 벡터 검색과 같은 고급 기능이 있어 RAG 시스템과의 적합성이 뛰어나다.
- **팀의 학습 곡선 고려**
 - MongoDB는 직관적인 쿼리 언어(MQL: MongoDB Query Language)를 사용하며, 팀원들이 RDBMS의 SQL 경험이 있다면 상대적으로 쉽게 적응할 수 있다.
 - DynamoDB는 키-값 기반 접근 방식을 사용하며, 복잡한 쿼리를 구현하기 위해 추가적인 학습이 필요하다. 이로 인해 학습 곡선이 가파를 수 있다.
- **JSON 문서 처리 및 통합**
 - MongoDB는 LLM 모델과 쉽게 통합할 수 있도록 설계되었으며, JSON 데이터를 직접 저장하고 처리하는 데 강점을 가진다. 이는 LLM RAG가 JSON 형태의 데이터를 주로 다룬다는 점에서 MongoDB가 더 유리하다.

특징	MongoDB	CouchDB
데이터 모델	BSON(Binary JSON) 기반 문서 저장.	JSON 문서로 데이터 저장.
데이터 액세스	MongoDB Query Language(MQL)을 사용해 복잡한 쿼리와 집계 지원.	MapReduce를 통한 쿼리 처리. 복잡한 쿼리는 구현이 어려움.
확장성(Scalability)	**수평적 확장(Horizontal Scaling)**이 기본 제공됨. 클러스터 구성과 샤딩(sharding)이 용이함.	**수직적 확장(Vertical Scaling)**에 적합. 분산형 환경에서는 복제(replication)에 초점.
인덱싱	다양한 복합 인덱스와 텍스트 인덱스, 벡터 검색 지원.	기본적인 인덱스만 지원.
데이터 동기화	샤딩 및 클러스터 노드 간 동기화.	Master-Master 복제를 사용해 분산형 데이터베이스 구축에 적합.
벡터 검색 지원	MongoDB Atlas에서 벡터 검색(Vector Search) 기본 제공.	벡터 검색을 직접 지원하지 않으며, 외부 시스템과의 통합 필요.
트랜잭션 지원	멀티 도큐먼트 ACID 트랜잭션 지원.	단일 문서 수준의 ACID 트랜잭션 지원.
성능	고속 읽기/쓰기 성능, 대규모 데이터 처리 및 복잡한 검색 작업에 적합.	안정적인 읽기 성능. 쓰기 성능은 비교적 느림.
복제 및 분산 처리	기본적인 클러스터 구성 및 복제 가능. 분산형 쿼리와 샤드 기반 데이터 처리 에 최적화됨.	다중 Master 복제를 통해 분산 데이터 동기화 에 강점.
적합한 활용 사례	대규모 데이터 처리, 실시간 분석, 복잡한 검색 쿼리, 벡터 검색.	분산 데이터 동기화가 필요한 애플리케이션 (e.g., 모바일, 오프라인 앱).

커뮤니티 및 지원	대규모 커뮤니티, MongoDB Atlas의 상업적 지원 제공.	커뮤니티가 상대적으로 작으며, 상업적 지원은 제한적.
구현 및 개발 난이도	사용이 쉬운 쿼리 언어(MQL)와 풍부한 문서화 제공.	MapReduce 기반의 쿼리로 인해 복잡한 쿼리 구현 난이도가 높음.

AWS DynamoDB 대신 MongoDB Atlas를 선택한 이유

- **복잡한 쿼리 처리 능력**
 - DynamoDB는 **키-값 및 테이블** 기반으로 동작하며, 복잡한 쿼리 처리에 제약이 있다. 반면, MongoDB는 **복합 인덱스, 집계 파이프라인 (Aggregation Pipeline)** 등 복잡한 쿼리를 더 유연하게 처리할 수 있다.
 - 추천 알고리즘에서 필요한 필터링, 정렬, 검색 작업을 MongoDB에서 더 쉽게 구현 가능하다.
- **벡터 검색**
 - MongoDB Atlas는 **벡터 검색** 기능을 제공하여 임베딩 데이터를 효율적으로 저장하고 유사도를 계산할 수 있다. 반면 DynamoDB는 기본적으로 벡터 검색을 지원하지 않으며, 이를 위해 별도의 외부 시스템을 연동해야 한다.
- **클러스터 관리 및 확장성**
 - MongoDB Atlas는 AWS 환경과 완벽히 통합되며, 관리형 클러스터를 통해 복제본 세트 구성, 백업, 확장 등을 쉽게 처리할 수 있다. DynamoDB도 관리형 서비스를 제공하지만, MongoDB는 더 다양한 설정 옵션과 커스터마이징이 가능하다.
- **팀의 친숙도**
 - MongoDB는 NoSQL 학습에 적합하며, MQL을 사용하여 DynamoDB보다 직관적으로 작업을 수행할 수 있다. DynamoDB는 독특한 설계 방식과 제한된 쿼리 언어 때문에 팀의 학습 곡선을 고려할 때 적합하지 않을 수 있다.
- **비용 효율성**
 - AWS DynamoDB는 읽기/쓰기 작업량에 따라 비용이 증가하며, 정교한 작업이 많을수록 비용이 크게 증가할 수 있다. MongoDB Atlas는 작업량이 많더라도 효율적인 비용 구조를 제공하며, 필요한 기능만 활성화하여 최적화할 수 있다.

프로젝트와의 적합성 고려


- **LLM RAG와의 결합**
 - MongoDB의 벡터 검색 및 문서 기반 데이터 저장소는 RAG 시스템에서 요구하는 데이터를 효율적으로 처리한다. 특히, Musical 및 Exhibition 데이터를 기반으로 **유사도 검색**과 **추천 알고리즘**을 효과적으로 구현할 수 있다.

- 유연한 데이터 모델링
 - MongoDB의 스키마리스 설계는 뮤지컬과 전시회 데이터가 갖는 다양성과 확장성을 지원하며, 변경사항이 있을 경우에도 쉽게 구조를 조정할 수 있다.
- 빠른 개발 주기 지원
 - 프로젝트 기간이 2개월로 짧은 점을 고려했을 때, MongoDB는 데이터 모델링과 배포가 빠르며, 관리형 클러스터를 제공해 개발자들이 데이터 설계와 검색에 집중할 수 있다.
- 클라우드 인프라와의 통합
 - MongoDB Atlas는 AWS와 원활히 통합되며, 클러스터 관리, 데이터 보안, 백업 등 고급 기능을 제공하여 안정적인 클라우드 환경을 유지할 수 있다.

ERD 설계


- NoSQL DB


Musical		
 _id	objectId	NN
M_title	string	NN
M_context	string	NN
M_poster	string	NN
M_price	int	NN
M_place	string	NN
M_date	string	NN
M_link	string	NN
M_cast[]	string	NN
M_story	string	NN
M_genre	string	NN
M_editor	string	NN
M_runtime	string	NN
M_day[]	string	NN
M_time[]	string	NN
M_status	string	NN
M_license	string	NN

Exhibition		
 _id	objectId	NN
E_title	string	NN
E_context	string	NN
E_poster	string	NN
E_price	int	NN
E_place	string	NN
E_date	string	NN
E_link	string	NN
E_artists[]	string	NN
E_ticketcast	string	NN

Suggestion		
 _id	objectId	NN
S_title	string	NN
S_content	string	NN

- Vector DB

Musical			...
 _id	objectId	NN	
M_text	string	NN	
M_embedding[]	double	NN	
M_original_id	objectId	NN	

Exhibition			...
 _id	objectId	NN	
E_text	string	NN	
E_embedding[]	double	NN	
E_original_id	objectId	NN	

JSON 구조

Main DB

Musical

```
{
  "_id": "ObjectId",
  "M_title": "string",
  "M_context": "string",
  "M_poster": "string",
  "M_price": "number",
  "M_place": "string",
  "M_date": "string",
  "M_link": "string",
  "M_cast": ["string"],
  "M_story": "string",
  "M_genre": "string",
  "M_editor": "string",
  "M_runtime": "string",
  "M_day": ["string"],
  "M_time": ["string"],
  "M_status": "string",
  "M_license": "string"
}
```

Exhibition

```
{
  "_id": "ObjectId",
  "E_title": "string",
  "E_context": "string",
  "E_poster": "string",
  "E_price": "number",
  "E_place": "string",
  "E_date": "string",
  "E_link": "string",
  "E_artists": ["string"],
  "E_ticketcast": "string"
}
```

Suggestion

```
{
  "_id": "ObjectId",
  "S_title": "string",
  "S_content": "string"
}
```

Vector DB

Musical

```
{
  "_id": "ObjectId",
  "M_text": "string",
  "M_embedding": ["number"],
  "M_original_id": "ObjectId"
}
```

Exhibition

```
{
  "_id": "ObjectId",
  "E_text": "string",
  "E_embedding": ["number"],
  "E_original_id": "ObjectId"
}
```