# Spring Blog 분석 보고서

1팀

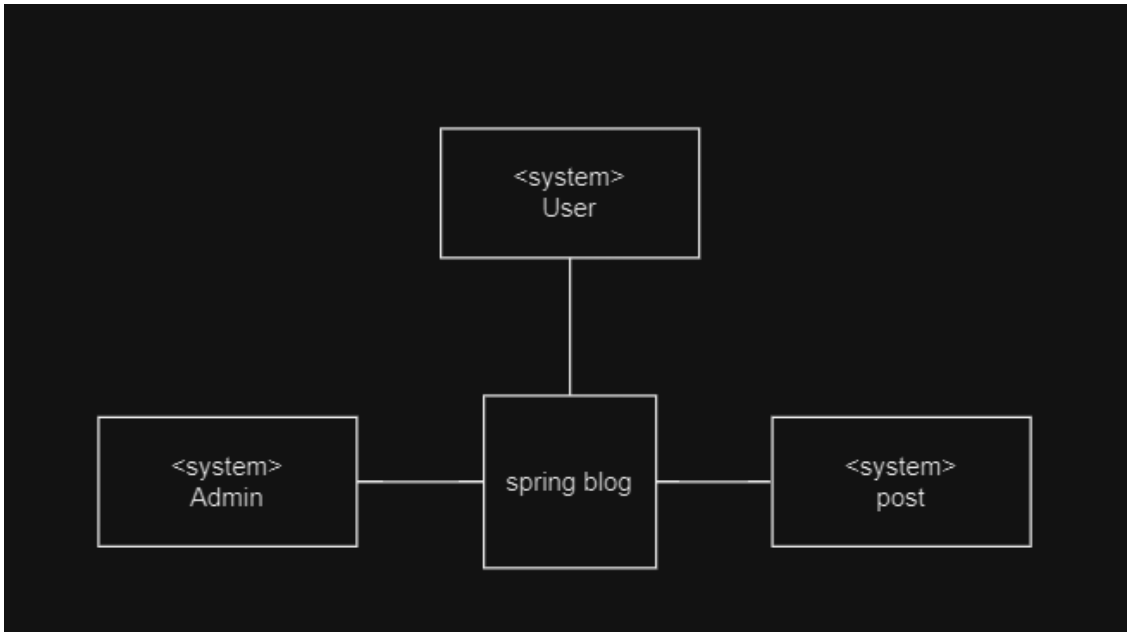
정승원, 박규민, 양수빈, 이나현, 이동준, 조우현

# 목차

# 1. Spring Blog 시스템 모델링

## 1.1 Spring Blog 시스템의 컨텍스트 모델 및 프로세스 모델

컨텍스트 모델은 시스템의 경계 밖에 있는 것을 보여주며 시스템의 운영사항을 명확히 한다. 외부 환경을 시각화 하는데 중점을 두기 때문에 개발 중인 시스템이 실제로 어떻게 사용되는지는 보여주지 않는다. 이러한 컨텍스트 모델은 액티비티 다이어그램을 사용한다.

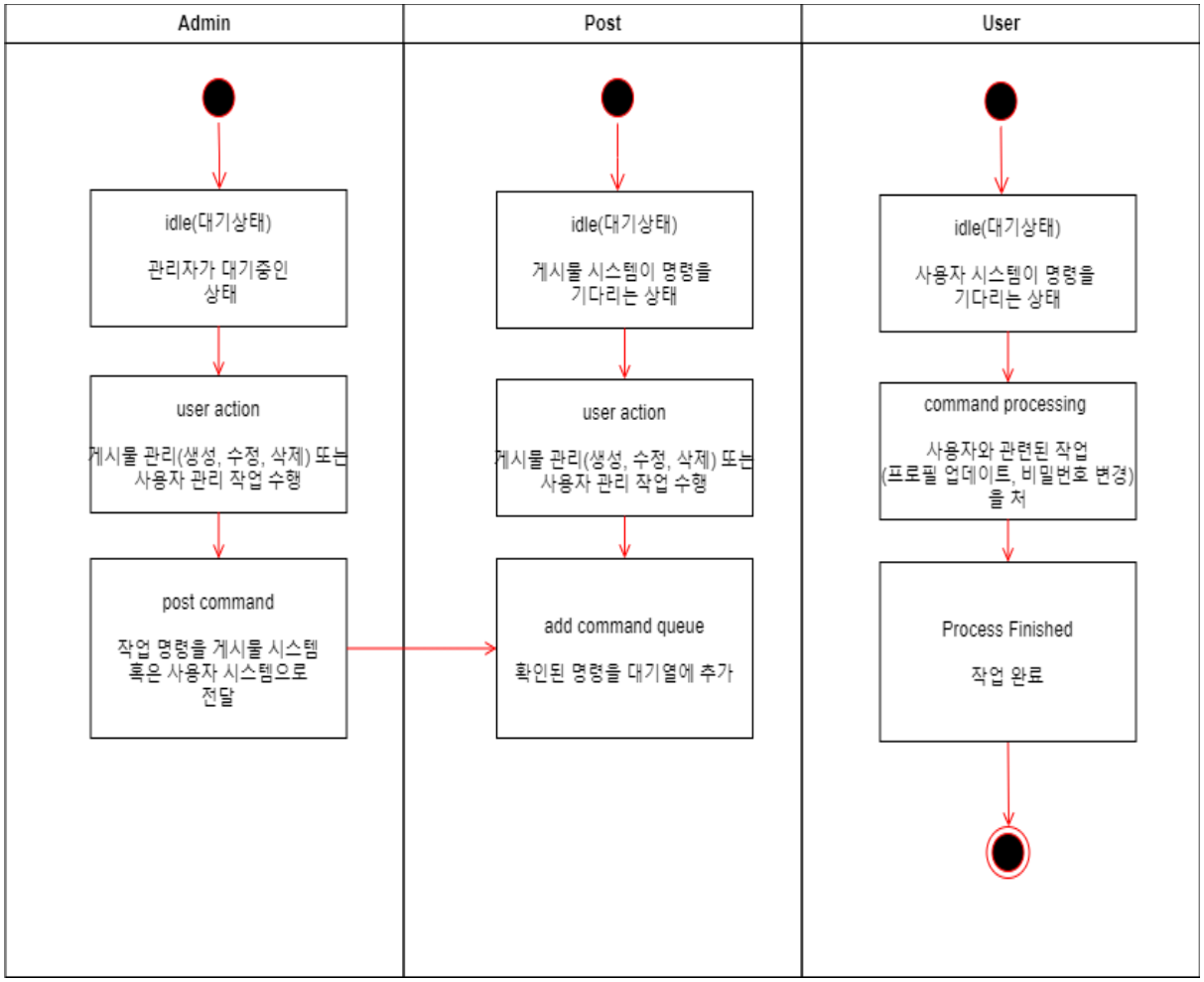



위 다이어그램은, Spring blog 시스템과 그 환경의 다른 시스템들을 보여주는 **컨텍스트 모델**이다. 위와 같은 컨텍스트 모델을 통해 Spring blog 시스템이 사용자(User) 시스템, 관리자(Admin) 시스템, 게시물(post) 시스템과 연결되어 있음을 볼 수 있다.

**AdminController**는 관리자(Admin) 시스템에 관한 코드로, 사이트 설정을 관리하는 기능을 한다.
**UserController**는 사용자(User) 시스템에 관한 코드로, 사용자 관리 및 프로필 수정 기능을 제공한다.
**PostController**는 게시물(Post) 시스템에 관한 코드로, 게시물을 관리하는 CRUD 기능을 제공한다.

이는 시스템에 사용되는 기능들과 함수들을 기술한 코드들을 토대로 각 시스템들이 어떻게 연결되어 있는지 살펴본 결과로, 위와 같이 분석한 코드들과 컨텍스트 모델을 통해 spring blog 시스템이 각각 다른 시스템(User, Admin, Post)들과 어떻게 상호작용하고 있는지에 대해서 구체적으로 알 수 있다.

다음은 각 시스템들이 어떻게 작동되는지에 대해 분석하기 위해 프로세스 모델을 **액티비티 다이어그램**이다. 프로세스 모델은 컨텍스트 모델을 보완하는 모델로, 개발 중인 시스템이 어떻게 사용되고 있는지 보여주며, 외부 시스템도 어떻게 사용되고 있는지 보여준다. 이러한 프로세스 모델은 컨텍스트 모델과 마찬가지로 액티비티 다이어그램을 사용한다. 간단한 프로세스 모델을 다이어그램을 통해 작성해보면 다음과 같다.




위의 프로세스 모델을 통해 Spring Blog의 각 시스템을 살펴보면 다음과 같다.

**관리자(admin) 시스템**

- **대기 상태:** 관리자는 시스템이 새로운 작업 요청을 기다리는 상태
- **사용자 액션 처리:** 관리자는 게시물을 관리하거나 사용자 프로필 정보를 수정하는 등의 작업을 수행
- **명령 전송:** 관리자가 작업을 완료하면 명령이 게시물 시스템으로 전달. 여기에는 게시물 삭제, 수정, 또는 새로운 게시물 추가 요청 등이 포함

**게시물(post) 시스템**

- **대기 상태:** 게시물 시스템은 관리자와 사용자로부터 작업 명령을 수신하기 위해 대기
- **명령 확인:** 관리자가 보낸 명령을 확인하고 작업을 처리합니다. 예를 들어, 새로운 게시물 추가, 기존 게시물 삭제 요청 처리 등이 이루어짐
- **대기열 추가:** 처리할 명령을 대기열에 추가하여 시스템이 순차적으로 처리할 수 있도록 함.

**3. 사용자(user)**

- **대기 상태:** 사용자는 시스템이 응답하기를 기다리는 상태
- **명령 처리:** 사용자의 요청(예: 프로필 업데이트, 비밀번호 변경 등)을 시스템이 처리
- **작업 완료:** 모든 요청이 정상적으로 처리되면 작업이 완료

전반적인 시스템 프로세스의 흐름을 요약하자면 다음과 같다.

1. 관리자가 시스템에 명령을 입력하면 해당 작업이 게시물 시스템으로 전송
2. 게시물 시스템은 명령을 확인한 후 대기열에 추가하여 요청을 처리
3. 사용자는 자신의 요청(예: 비밀번호 변경)이 성공적으로 완료될 때까지 대기
4. 시스템은 관리자 및 사용자 간의 작업 흐름을 조율하며 안정적으로 콘텐츠를 관리

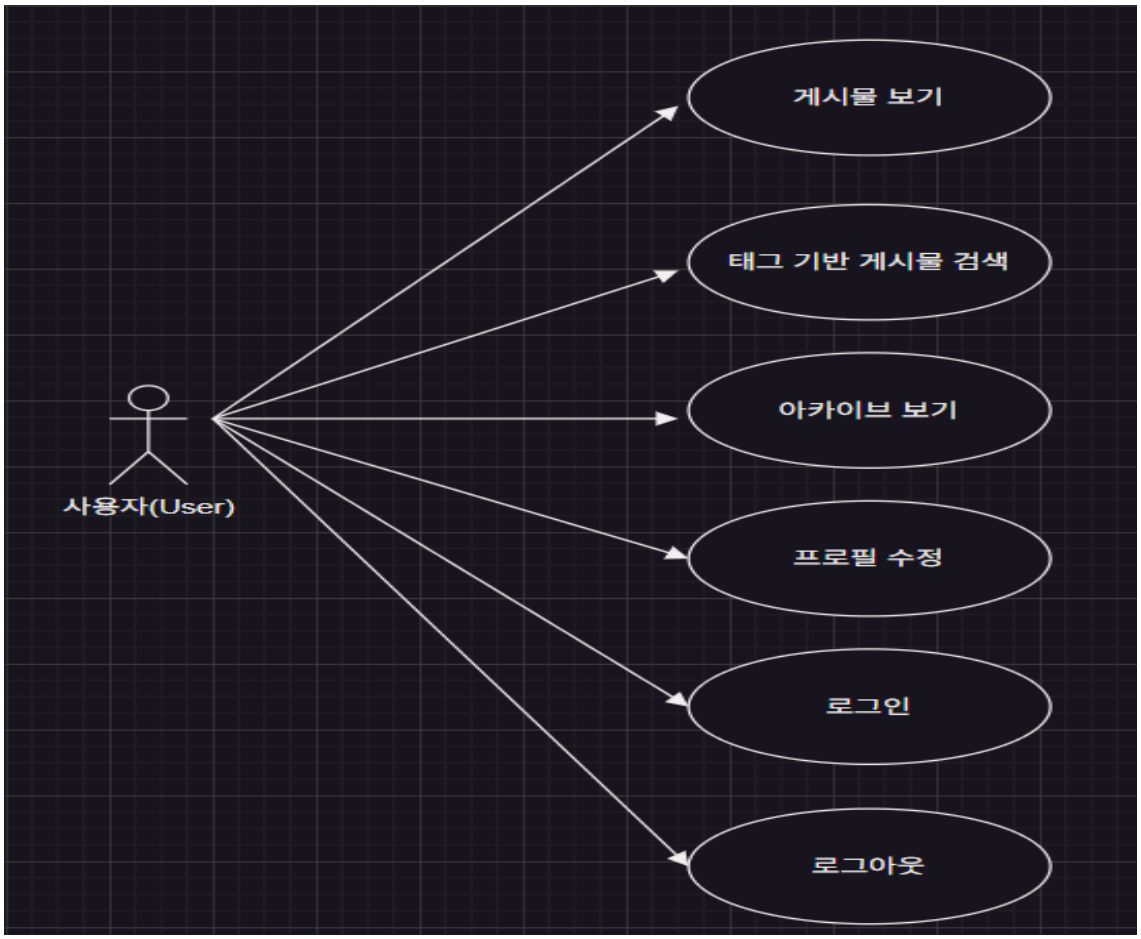
이와 같이, 컨텍스트 모델과 프로세스 모델을 통해 spring blog 시스템이 외부의 컨텍스트나 환경과 어떻게 상호작용하는지에 대해서 알 수 있으며, 연결된 각 시스템이 서로 어떻게 상호작용하는지에 대한 이해를 하는 데 효율적으로 활용할 수 있다.

### 1.2 Spring blog 시스템의 상호작용 모델

상호작용 모델은 사용자 상호작용을 모델링 하거나, 시스템 간의 상호작용을 모델링하는 것이다. 이는 유스케이스 다이어그램과 시퀀스 다이어그램을 사용한다.

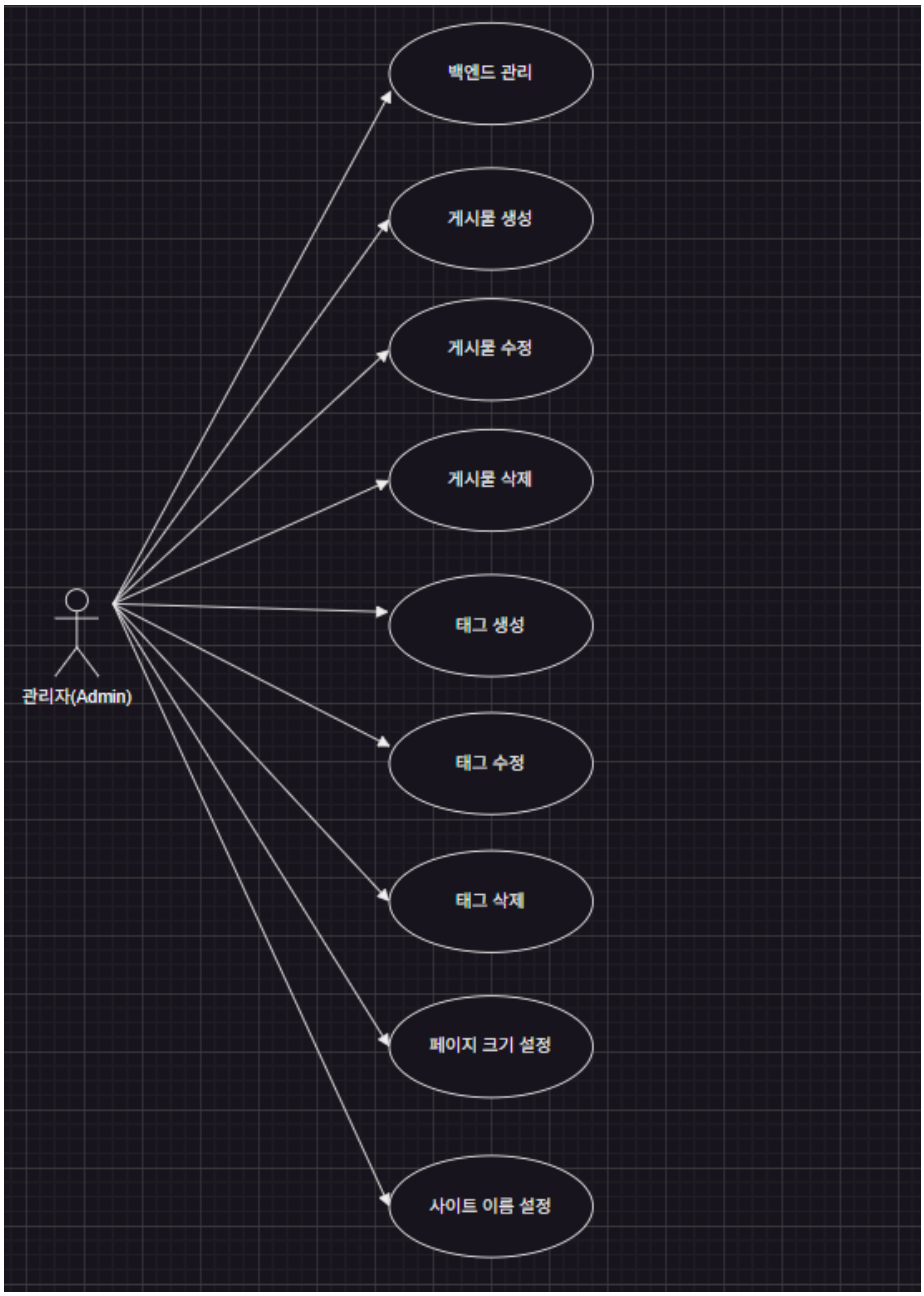
UML 다이어그램은 시스템 문서화 자료로 활용되어 유지보수 시 시스템 구조, 사용자와 시스템 간의 상호작용을 쉽게 파악할 수 있는 장점이 있어, 시스템 모델링에서 유용하게 쓰인다. Spring Blog의 주요 유스케이스 다이어그램은 사용자와 시스템 간의 상호작용을 시각화한다.

다음은 **'사용자' 유스케이스 다이어그램**이다.




주요 액터는 사용자이며, 시스템의 데이터를 탐색하고, 자신의 계정 정보를 관리하는 일반 사용자를 말한다. 즉, 액터인 사용자는 스프링 블로그의 주요 콘텐츠 소비자이다. 사용자 유스케이스는 '게시물 보기(공개된 게시물 열람)', '태그 기반 게시물 탐색(특정 태그를 기준으로 게시물 탐색)', '아카이브 보기(연도별로 정리된 게시물 탐색)', '로그인', '로그아웃','프로필 수정(사용자 계정 정보를 수정)'이 있다. 게시물 조회, 댓글 작성, 게시물 공유 등의 활동을 수행한다.

다음은 **'관리자' 유스케이스 다이어그램**이다.



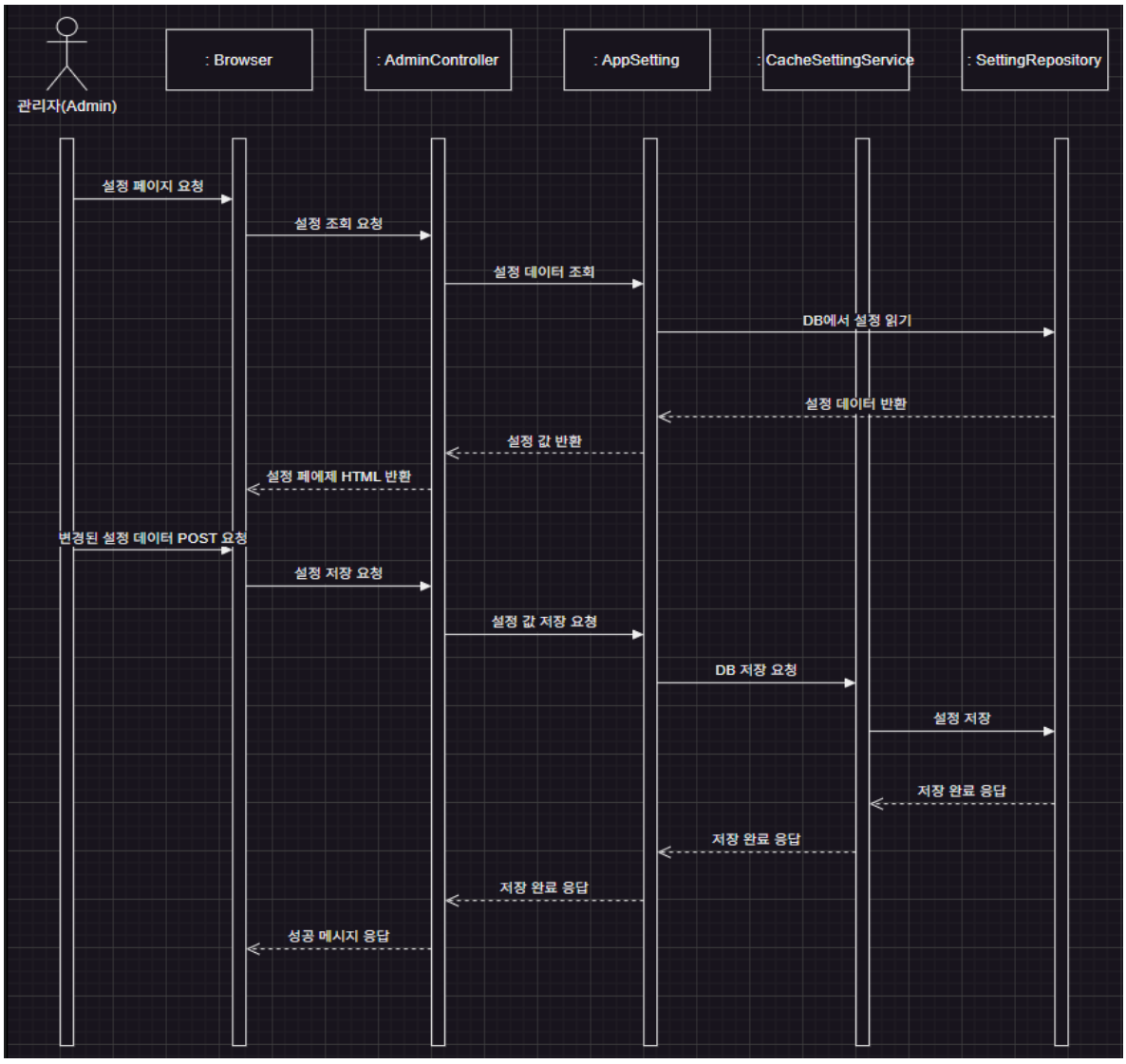

관리자(Admin)를 주요 액터로 한 유스케이스 다이어그램이다. 관리자는 스프링 블로그 시스템의 전반적인 운영과 설정을 담당하며, 다양한 관리 작업을 수행한다.

관리자는 게시물 관리, 태그 관리, 사이트 설정과 같은 기능을 수행한다. 구체적으로 게시물에 대해 생성, 수정, 삭제를 수행하며, 태그 또한 생성, 수정, 삭제가 가능하다. 또한, 사이트의 외형적 설정을 담당하여 페이지 크기와 사이트 이름을 조정할 수 있다. 이 모든 작업은 관리자 계정을 통해 이루어지며, 시스템의 백엔드 관리 역시 포함된다.

다음은 spring blog의 **시퀀스 다이어그램**이다. 시퀀스 다이어그램은 기능별 요청 흐름을 상세히 표현하여 컨트롤러, 서비스, 저장소 계층 간의 상호작용을 모델링하는 데 효과적이다. 동작모델 또한 시퀀스 다이어그램으로 표현하기 때문에 상호작용모델이자, 동작모델의 데이터 주도 모델링이다.

Spring blog의 전반적인 흐름을 '관리자 시나리오'와 '사용자 시나리오'로 구분하여 살펴보았다. 관리자 시나리오는 첫 번째, '관리자가 블로그를 설정하는 것', 두 번째, '관리자가 게시물을 추가', 세 번째, '관리자가 게시물을 수정하는 것', 마지막으로 '관리자가 게시물을 삭제하는 것' 순으로 전개된다.

먼저, **'관리자가 블로그를 설정하는 것'에 관한 시나리오이다.** 액터인 관리자(Admin)는 전체적인 블로그 설정(사이트 이름, 슬로건, 페이지 크기)을 변경할 수 있다. 시퀀스 다이어그램 속 객체로는 Browser, AdminController, AppSetting, CacheSettingService, SettingRepository 등등이 있다. 다음은 이를 토대로 작성한 spring blog의 **시퀀스 다이어그램**이다.



시퀀스 다이어그램의 전반적인 흐름은 다음과 같다.

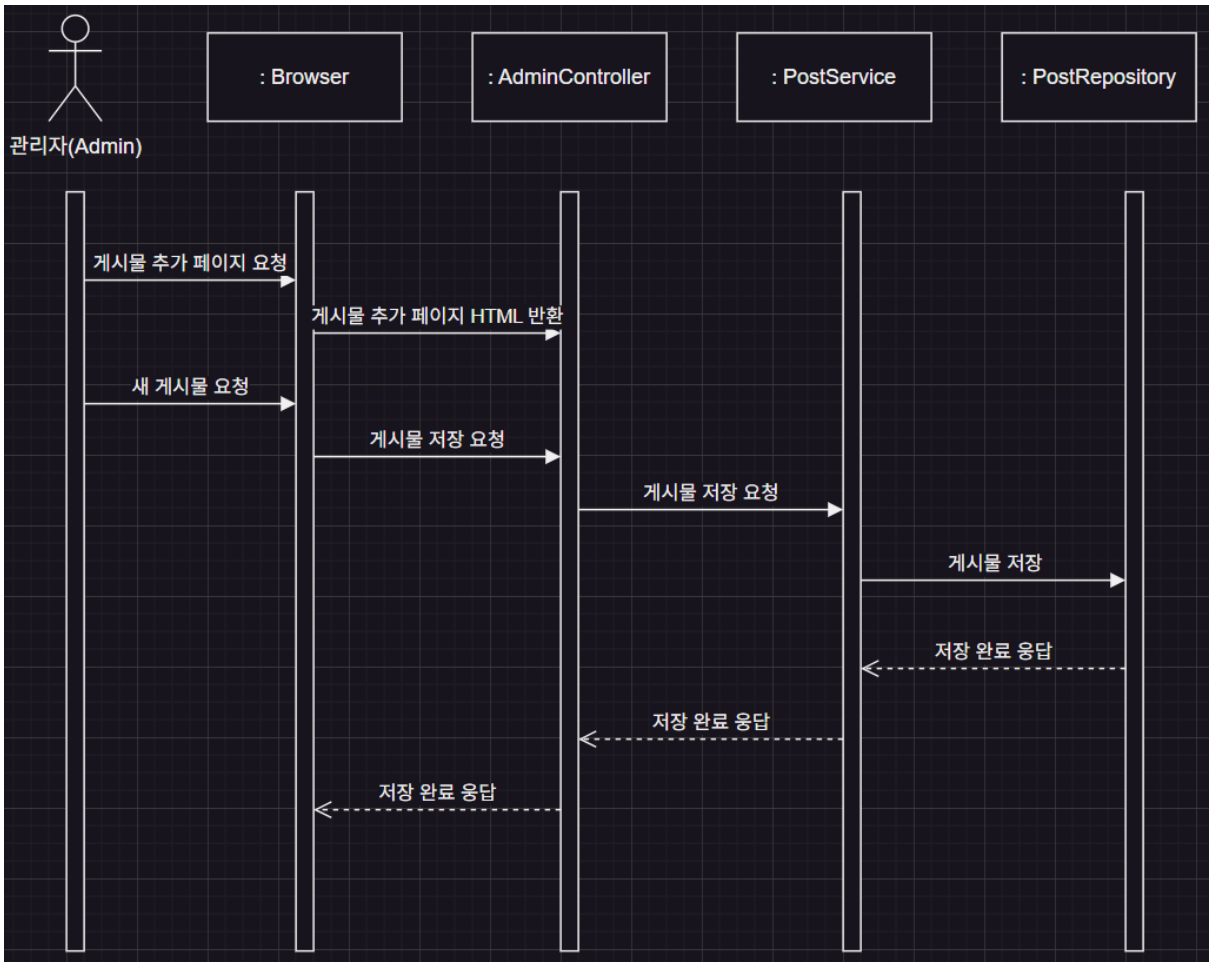
**설정 페이지 요청**

- Admin → Browser: 설정 페이지 요청
- Browser → AdminController: 설정 조회 요청
- AdminController → AppSetting: 설정 데이터 조회
- AppSetting → SettingRepository: DB에서 설정 읽기
- SettingRepository → AppSetting: 설정 데이터 반환
- AppSetting → AdminController: 설정 값 반환
- AdminController → Browser: 설정 페이지 HTML 반환
-

**변경된 설정 데이터 전송**

- Admin → Browser: 변경된 설정 데이터 POST 요청
- Browser → AdminController: 설정 저장 요청
- AdminController → AppSetting: 설정 값 저장 요청
- AppSetting → CacheSettingService: DB 저장 요청
- CacheSettingService → SettingRepository: 설정 저장
- SettingRepository → CacheSettingService: 저장 완료 응답
- CacheSettingService → AppSetting: 저장 완료 응답
- AppSetting → AdminController: 저장 완료 응답
- AdminController → Browser: 성공 메시지 응답
- CacheSettingService → AppSetting: 저장 완료 응답
- AppSetting → AdminController: 저장 완료 응답
- AdminController → Browser: 성공 메시지 응답

다음은 **'관리자가 게시물을 추가하는 경우'의 시퀀스 다이어그램**이다.




전반적인 흐름은 '게시물 추가 페이지 요청'과 '게시물 저장' 으로 구분하여 살펴볼 수 있다.

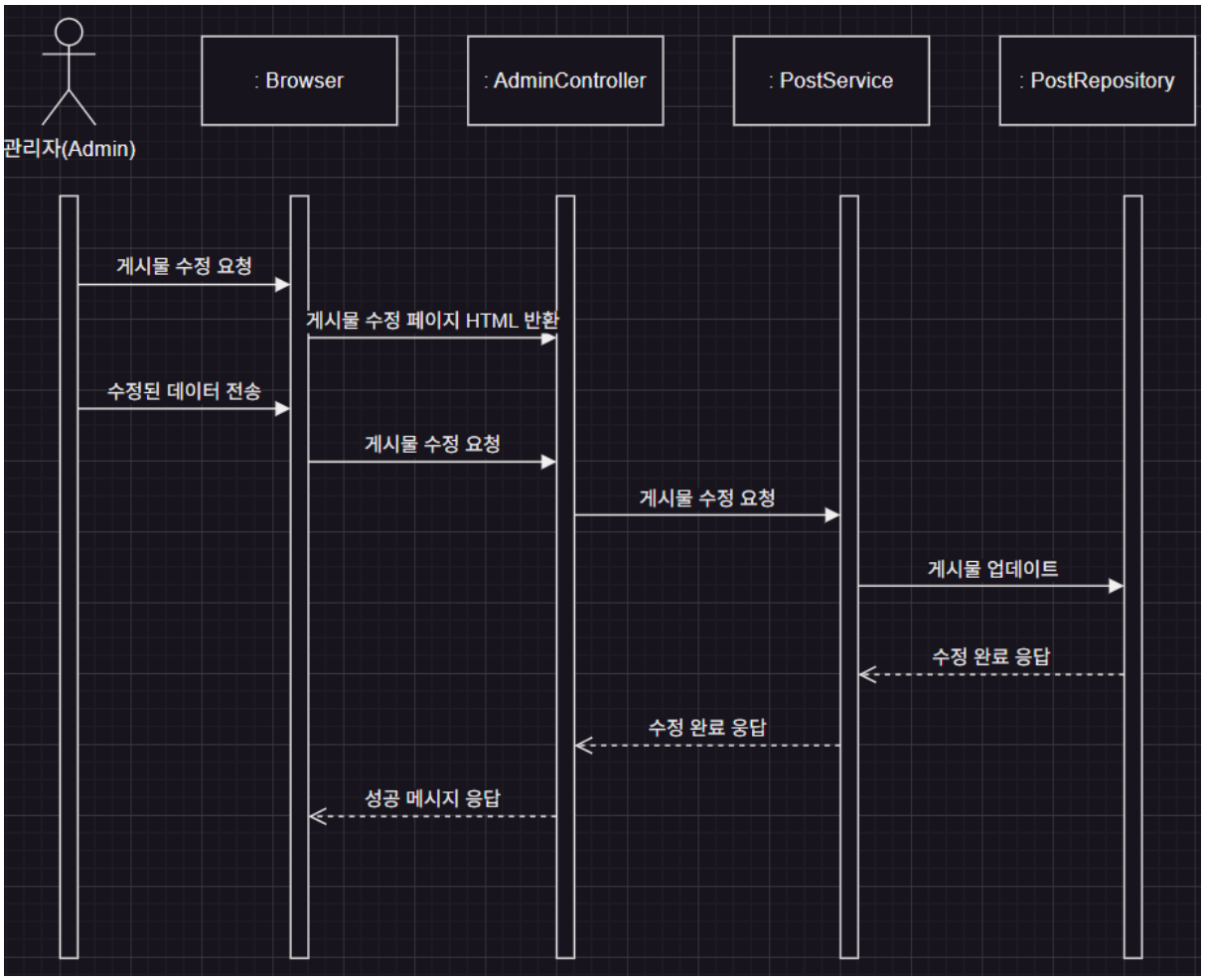
**게시물 추가 페이지 요청**

- Admin → Browser: 게시물 추가 페이지 요청
- Browser → AdminController: 게시물 추가 페이지 HTML 반환

**게시물 저장**

- Admin → Browser: 새 게시물 생성 요청
- Browser → AdminController: 게시물 저장 요청
- AdminController → PostService: 게시물 저장 요청
- PostService → PostRepository: 게시물 저장
- PostRepository → PostService: 저장 완료 응답
- PostService → AdminController: 저장 완료 응답
- AdminController → Browser: 성공 메시지 응답

다음은**'관리자가 게시물을 수정하는 경우'**의 시퀀스 다이어그램이다.




전반적인 흐름은 '게시물 수정 페이지 요청'과 '게시물 수정 저장' 으로 구분하여 살펴볼 수 있다.

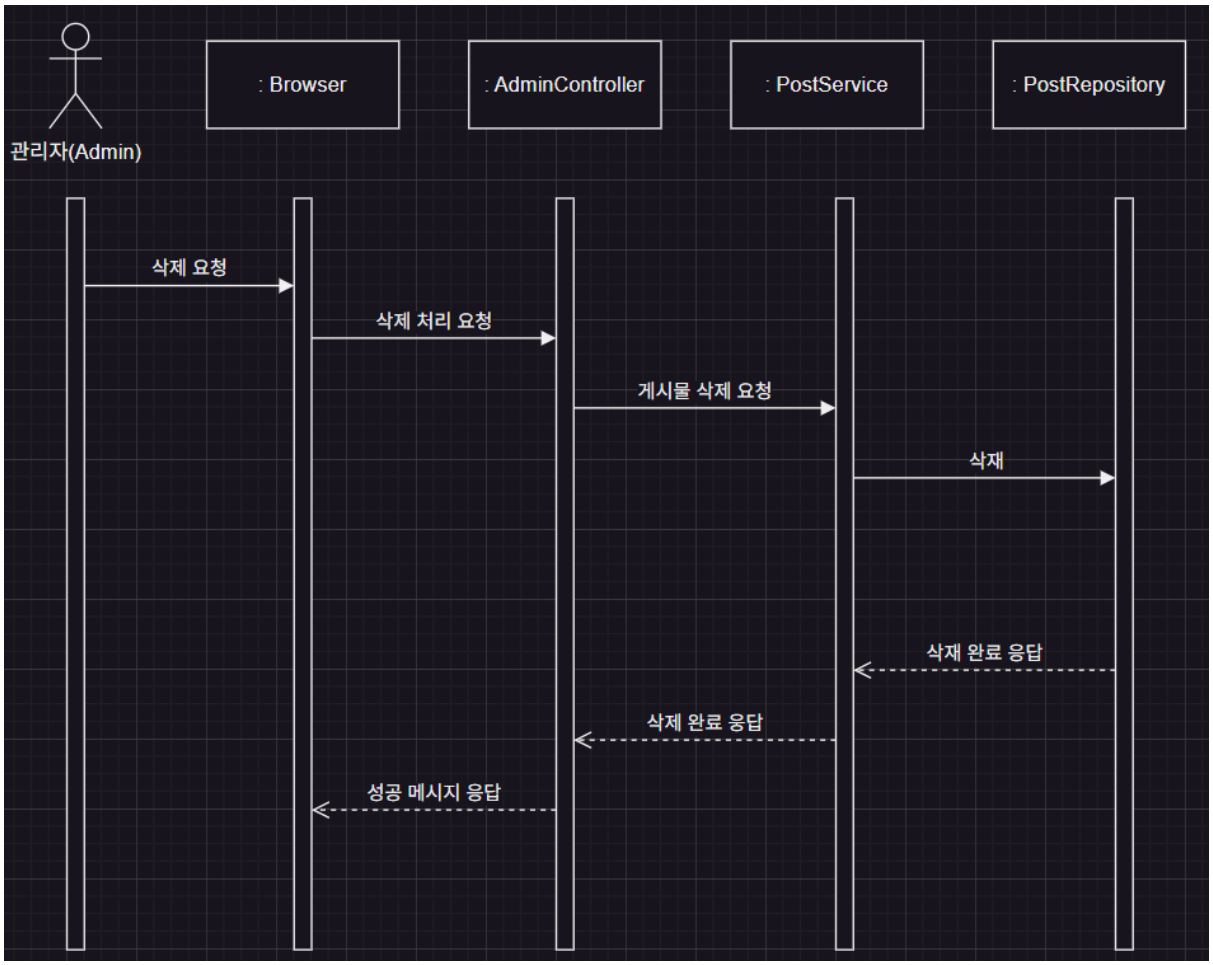
**게시물 수정 페이지 요청**

- Admin → Browser: 게시물 수정 요청
- Browser → AdminController: 게시물 수정 페이지 HTML 반환

**게시물 수정 저장**

- Admin → Browser: 수정된 데이터 전송
- Browser → AdminController: 게시물 수정 요청
- AdminController → PostService: 게시물 수정 요청
- PostService → PostRepository: 게시물 업데이트
- PostRepository → PostService: 수정 완료 응답
- PostService → AdminController: 수정 완료 응답
- AdminController → Browser: 성공 메시지 응답

**'관리자가 게시물을 삭제하는 경우'**의 시퀀스 다이어그램은 다음과 같다.



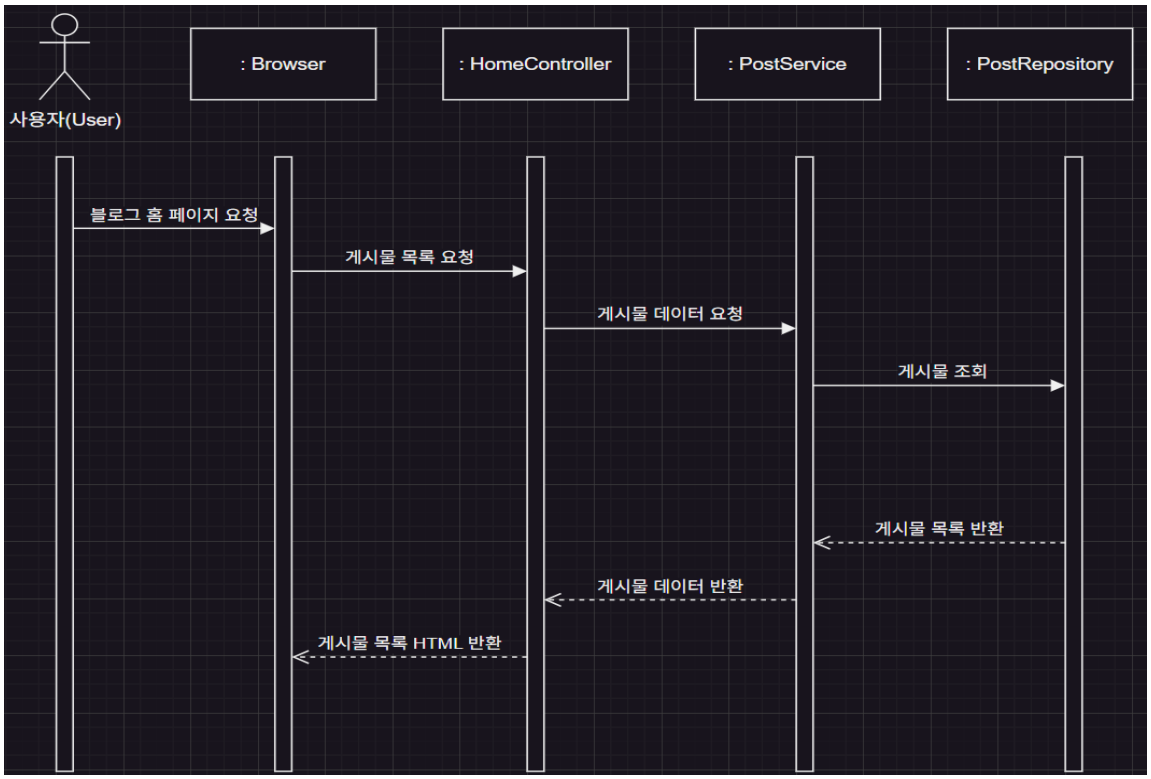

게시물 삭제에 관한 시퀀스 다이어그램은 다음과 같은 흐름으로 전개된다.

**게시물 삭제**

- Admin → Browser: 삭제 요청
- Browser → AdminController: 삭제 처리 요청
- AdminController → PostService: 게시물 삭제 요청
- PostService → PostRepository: 삭제
- PostRepository → PostService: 삭제 완료 응답
- PostService → AdminController: 삭제 완료 응답
- AdminController → Browser → 성공 메시지 응답

다음은 **사용자(user) 시나리오에 기반하여 시퀀스 다이어그램**이다. 사용자 시나리오의 액터는 사용자(User)이며, 객체는 Browser, HomeController, PostService, PostRepository 등 등이 있다. 전반적인 사용자 시나리오는 '게시물 목록 조회', '특정 게시물 상세 조회', '사용자가 태그 목록 요청', '태그 검색 요청'으로 구분하여 살펴볼 수 있다.

**'게시물 목록 조회' 시퀀스 다이어그램**

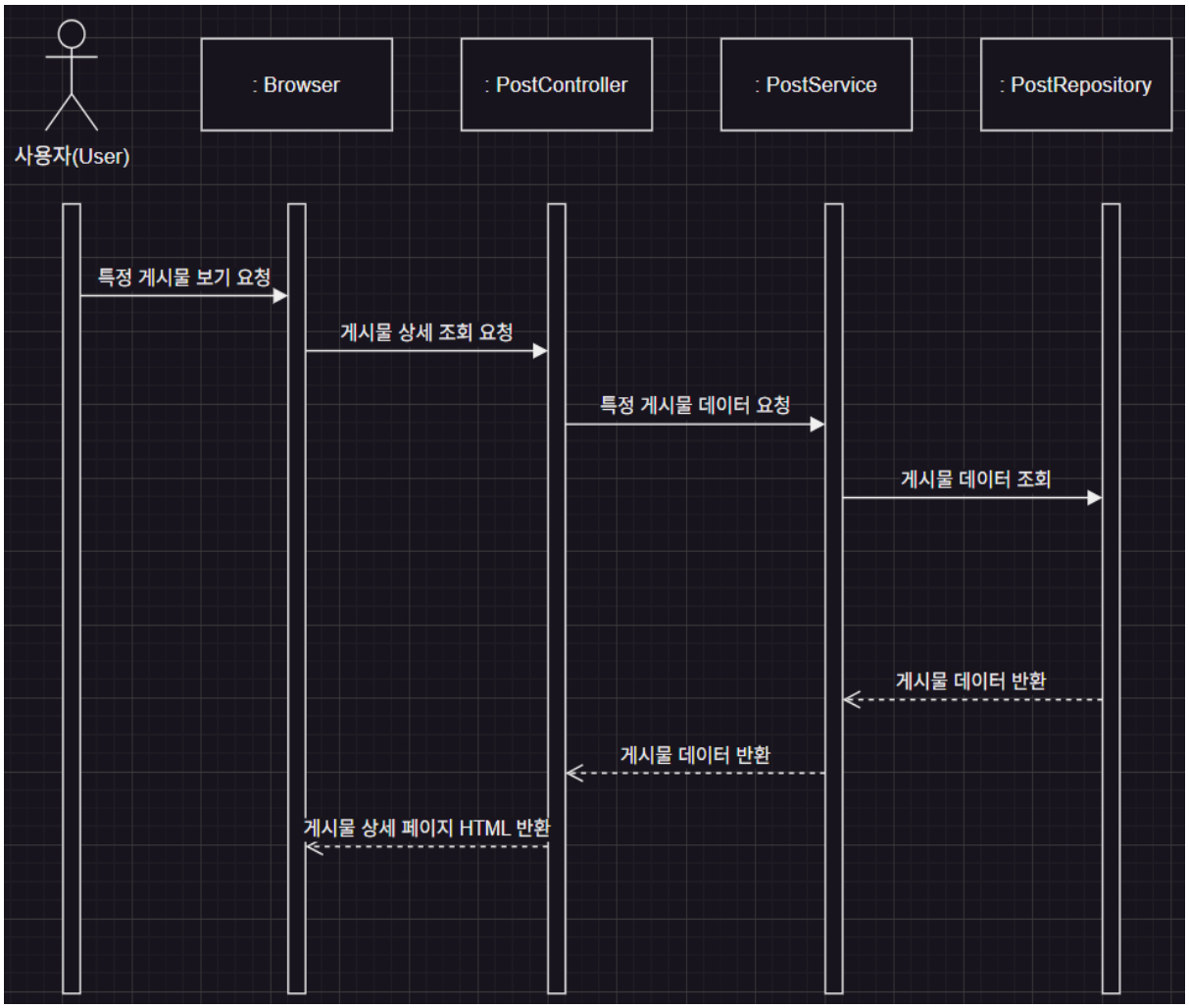



'게시물 목록 조회'와 관련한 시퀀스 다이어그램은 다음과 같은 흐름으로 전개된다.

**게시물 목록 조회**

- User → Browser: 블로그 홈 페이지 요청
- Browser → HomeController: 게시물 목록 요청
- HomeController → PostService: 게시물 데이터 요청
- PostService → PostRepository: 게시물 조회
- PostRepository → PostService: 게시물 목록 반환
- PostService → HomeController: 게시물 데이터 반환
- HomeController → Browser: 게시물 목록 HTML 반환

**‘특정 게시물 상세 조회’ 시퀀스 다이어그램**

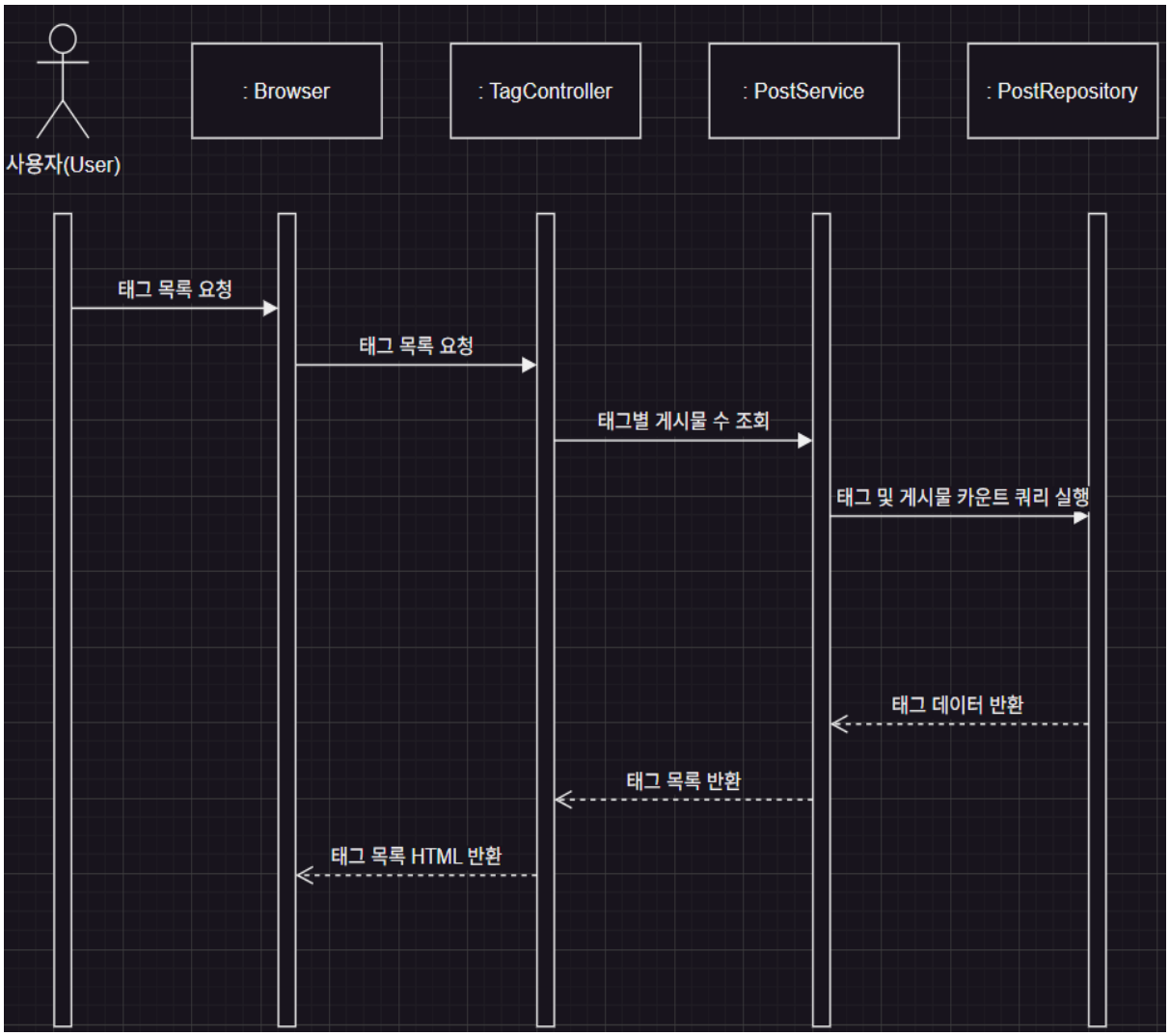



‘특정 게시물 상세 조회’와 관련한 시퀀스 다이어그램은 다음과 같은 흐름으로 전개된다.

**특정 게시물 상세 조회**

- User → Browser: 특정 게시물 보기 요청
- Browser → PostController: 게시물 상세 조회 요청
- PostController → PostService: 특정 게시물 데이터 요청
- PostService → PostRepository: 게시물 데이터 조회
- PostRepository → PostService: 게시물 데이터 반환
- PostService → PostController: 게시물 데이터 반환
- PostController → Browser: 게시물 상세 페이지 HTML 반환

**'사용자가 태그 목록 요청을 한 경우'의 시퀀스 다이어그램**

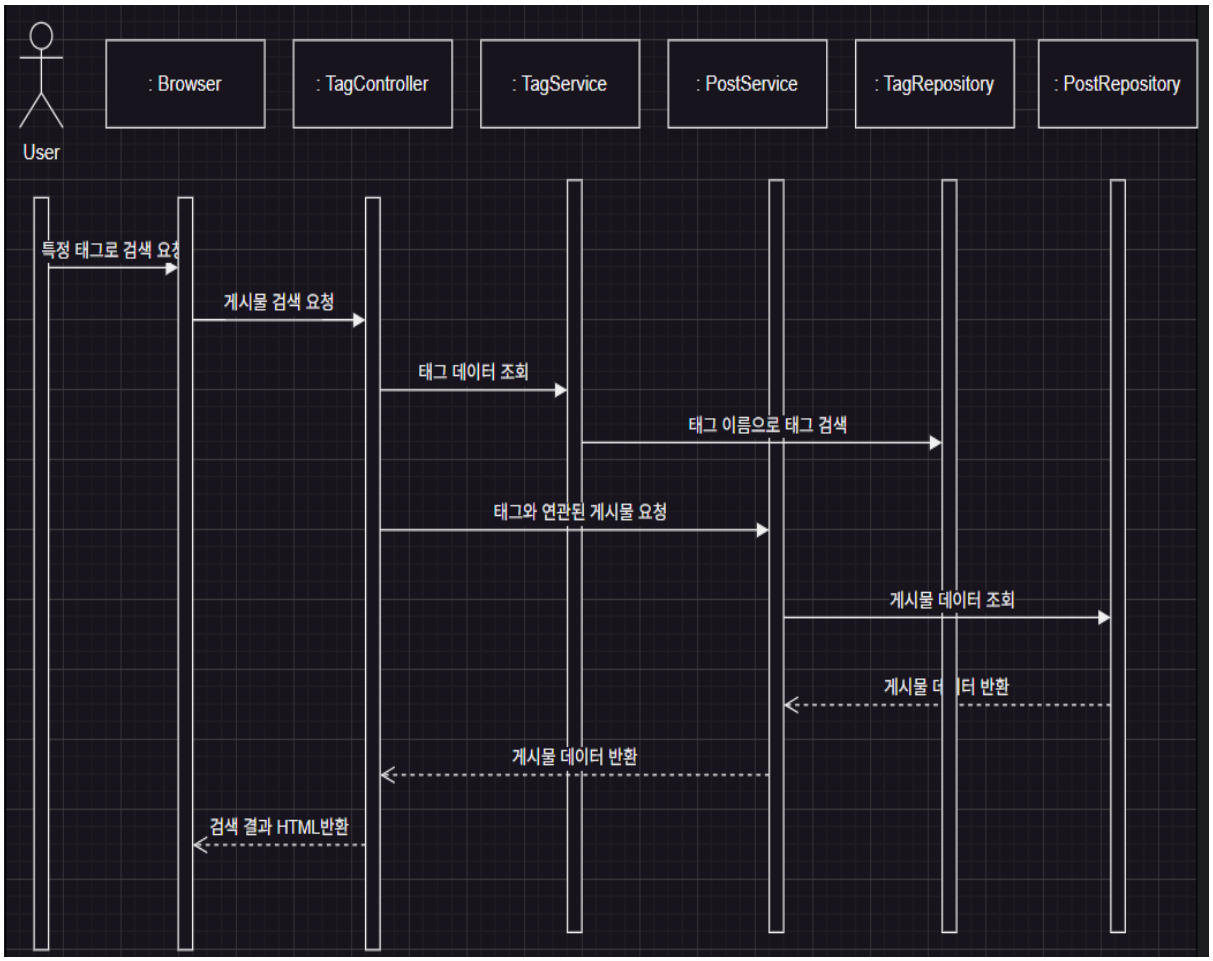



'사용자가 태그 목록을 요청한 경우'와 관련한 시퀀스 다이어그램은 다음과 같은 흐름으로 전개된다.

**사용자가 태그 목록 요청**

- User → Browser: 태그 목록 요청
- Browser → TagController: 태그 목록 요청
- TagController → PostService: 태그별 게시물 수 조회
- PostService → PostRepository: 태그 및 게시물 카운트 쿼리 실행
- PostRepository → PostService: 태그 데이터 반환
- PostService → TagController: 태그 목록 반환
- TagController → Browser: 태그 목록 HTML 반환

마지막으로, **'사용자가 태그 검색을 요청한 경우'**에 대한 시퀀스 다이어그램이다.




'사용자가 태그 검색을 요청한 경우'와 관련한 시퀀스 다이어그램은 다음과 같은 흐름으로 전개된다.

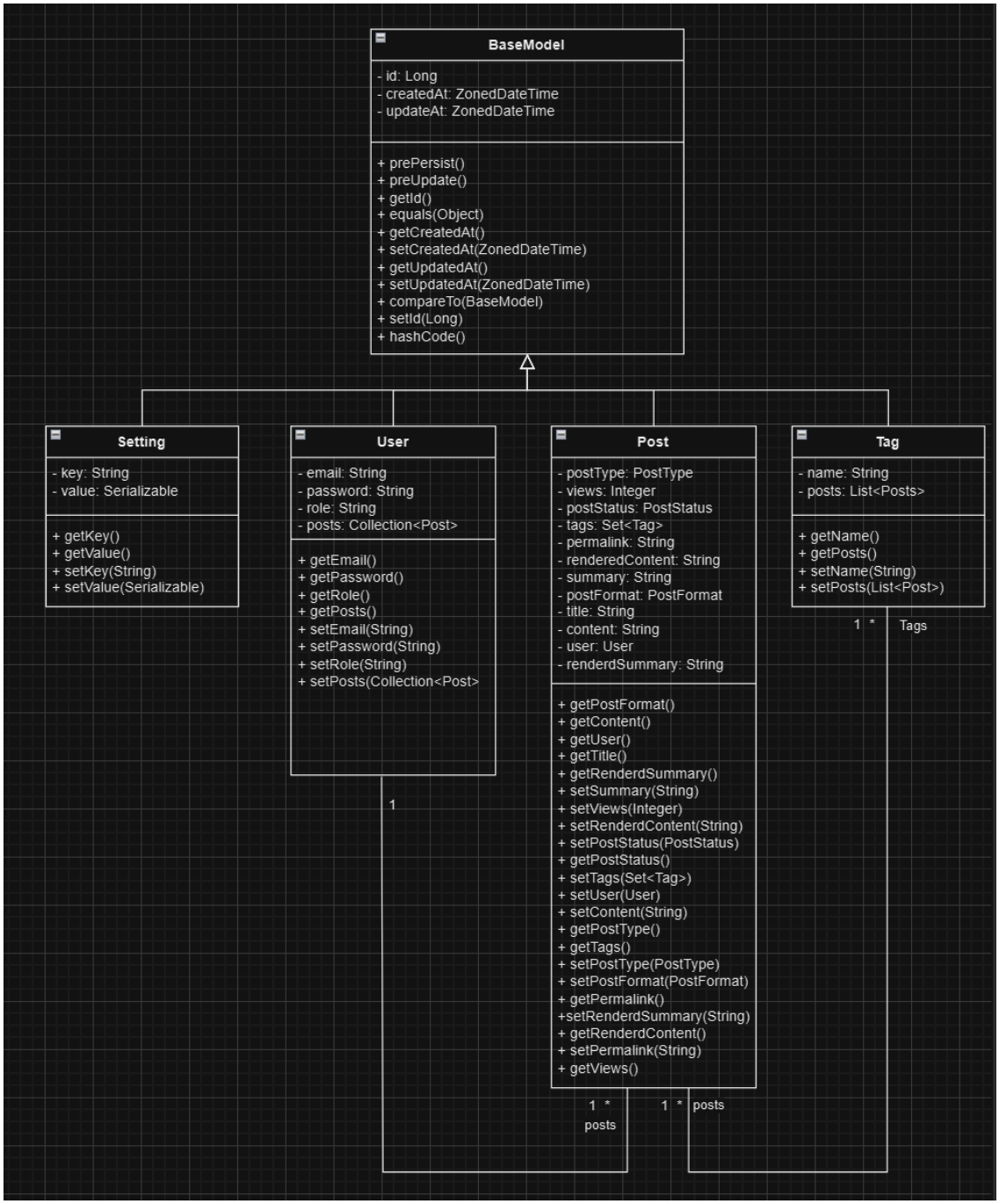
**태그 검색 요청**

- User → Browser: 특정 태그로 검색 요청
- Browser → TagController: 게시물 검색 요청
- TagController → TagService: 태그 데이터 조회
- TagService → TagRepository: 태그 이름으로 태그 검색
- TagController → PostService: 태그와 연관된 게시물 요청
- PostService → PostRepository: 게시물 데이터 조회
- PostRepository → PostService: 게시물 데이터 반환
- PostService → TagController: 게시물 데이터 반환
- TagController → Browser: 검색 결과 HTML 반환

## 1.3 Spring blog 시스템의 구조모델

구조모델은 해당 시스템을 구성하는 구성요소와 그 관계의 관점에서 시스템의 구성을 보여준다. 이러한 구조모델은 클래스 다이어그램을 사용한다.

다음은 구조 모델의 관점에서 SpringBlog 시스템의 객체 클래스들과 그들 간의 연관을 모델링한 클래스 다이어그램이다.




이 클래스 다이어그램은 Spring Blog시스템을 구성하는 BaseModel 클래스를 기반으로 한 주요 클래스인 User, Post, Tag, Setting 클래스와 이들 간의 관계를 보여준다.

**BaseModel 클래스는** 공통 속성과 동작을 포함한 추상 클래스로 모든 다른 클래스(User, Post, Tag, Setting)가 이를 상속 Id와 생성 및 수정 날짜 정보를 가지는 created_at, updated_at의 속성을 가진다.

**User 클래스**는 사용자를 관리하는 클래스로 사용자 정보인 username, email, password와 사용자가 작성한 게시물 목록 posts를 속성으로 가진다.

getPosts()/setPosts(Collection<Post> posts) 메서드로 사용자와 연결된 게시글 목록을 가져오거나 설정하고, 이메일, 비밀번호, 역할 정보를 받아 사용자 객체를 생성하는 메서드를 가진다.

User는 여러 개의 Post와 일대다(1:N) 관계를 가진다. 즉, 한 명의 사용자가 여러 게시물을 작성할 수 있다.

**Post 클래스**는 사용자가 작성한 게시물을 나타내는 클래스로 게시물의 제목, 내용인 title과 content 속성, 게시물 생성 및 수정 날짜와 시간을 나타내는 createdAt, updatedAt 속성과 게시물 작성한 사용자 Id를 나타내는 userId, 게시물에 연결된 태그 목록을 가지는 tags를 속성으로 가진다.

게시물의 제목, 내용, 작성자 정보, 게시물의 정보 등을 조회 및 설정하는 메서드를 가진다.

Post는 여러 개의 Tag와 다대다(M:N) 관계를 가진다. 즉, 하나의 게시물은 여러 태그를 가질 수 있으며, 하나의 태그는 여러 게시물에 사용될 수 있다.

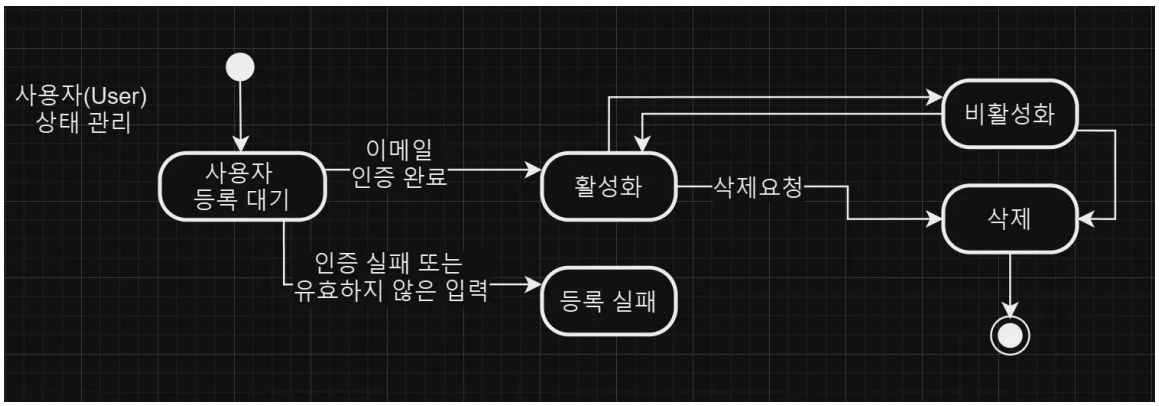
**Tag 클래스**는 게시물에 대한 태그를 관리하는 클래스로 태그 이름 name과 해당 태그가 연결된 게시물 목록인 posts를 속성으로 가진다.

태그의 이름과 해당 태그가 연결된 게시물 목록을 조회 및 설정하는 메서드를 가진다.

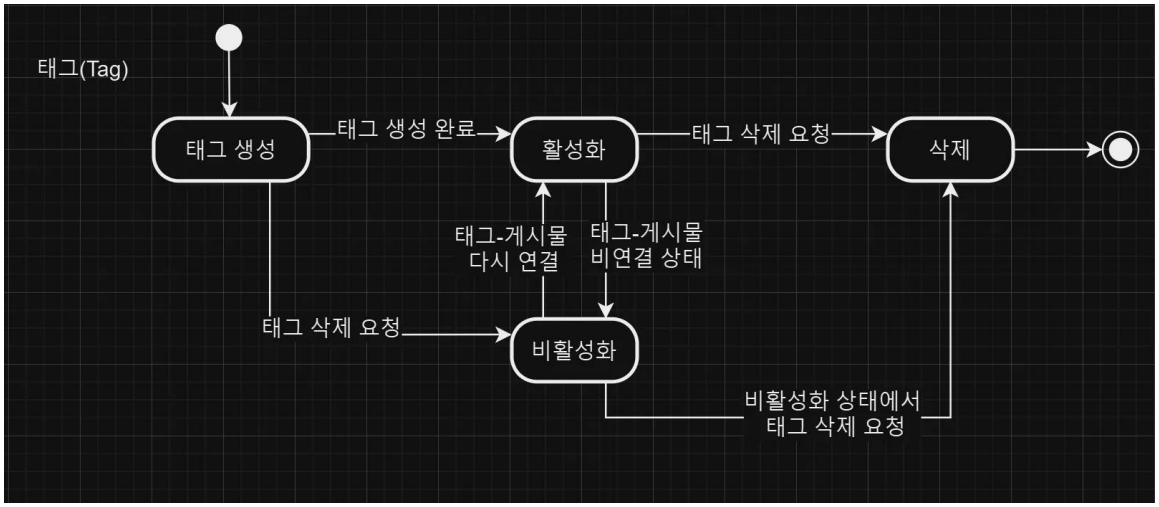
Tag는 여러 개의 Post와 다대다(M:N) 관계를 형성한다.

**Setting 클래스**는 시스템의 설정 정보를 관리하는 클래스로 설정 항목의 키와 값을 갖는 key, value를 속성으로 가진다.

## 1.4 Spring blog 시스템의 동작모델




사용자 상태 관리는 사용자 등록 대기에서 시작하며, 이메일 인증 완료 시 활성화 상태로 전환된다. 인증 실패나 유효하지 않은 입력이 있을 경우 등록 실패 상태로 종료되며, 활성화된 사용자는 삭제 요청에 따라 삭제 상태로 이동할 수 있다. 활성화 상태의 사용자는 관리자 설정에 의해 비활성화 상태로 전환될 수 있으며, 필요 시 다시 활성화될 수 있다.




태그 관리는 태그 생성 상태에서 시작하며, 생성이 완료되면 활성화 상태가 된다. 태그는 게시물과 연결이 끊기거나 요청에 의해 비활성화 상태로 전환될 수 있으며, 삭제 요청 시 삭제 상태로 이동되어 종료된다.

## 2. Spring Blog 시스템의 Architecture design

Spring blog를 분석한 결과 **Layered Architecture, MVC Pattern, Repository Architecture,** 그리고 **Transactional Architecture**이 사용되고 있음을 알 수 있었다.

### 2.1 Layered Architecture

이를 바탕으로 시스템의 구조를 다음과 같이 계층별로 정리할 수 있었다.

Presentation Layer (Controller + View)

├─── Admin/ Post/ User/ Tag / Home Controller

├─── templates (HTML 템플릿, Thymeleaf/Jade4j)

Service Layer (Business Logic)

├─── Post / User / Tag Service

├─── AppSetting

├─── CacheSettingService

Data Access Layer (Repository)

├─── Post / User / Tag / SettingRepository

Domain Layer (Entity + DTO)

├─── Entity |─── Post ─── User ─── Tag ─── Setting

├─── Support Classes |─── Post Format / Status / Type

├─── DTO |─── Post / Settings / User From

Infrastructure Layer (Security, Caching, Utilities)

├─── SecurityConfig

├─── CacheConfiguration

├─── Ehcache (ehcache.xml)

├─── Utilities |─── DTOUtil ─── Markdown ─── PygmentsService ─── ViewHelper

Spring Blog는 이러한 layered Architecture 설계를 통해 각 계층이 특정 역할과 책임을 명확히 분리하여 관리하고 있다. 계층별 역할은 다음과 같다.

1. Presentation Layer (컨트롤러 + 뷰)

: Controller + View

├─── AdminController

├─── PostController

├─── UserController

├─── TagController

├─── HomeController

Controller들은 요청을 수신하고 적절한 서비스를 호출하며, 데이터를 뷰로 전달한다

View는 Thymeleaf, Jade4j를 사용하여 HTML page를 생성, 사용자와 상호작용하며 요청 데이터를 표시하고 수집한다.

2. Service Layer (비즈니스 로직)

├─── PostService

├─── UserService

├─── TagService

├─── AppSetting (애플리케이션 설정 관리)

├─── CacheSettingService (캐싱 처리)

Presentation Layer에서 받은 요청을 처리하고 비즈니스 로직을 실행하며, 결과를 반환한다. 또한 데이터 가공 및 캐싱 처리 등 중간 역할을 수행한다.

3. Data Access Layer

├─── PostRepository

├─── UserRepository

├─── TagRepository

├─── SettingRepository

데이터베이스와 상호작용하며 JPA를 생성해 엔티티를 관리한다.

4. Domain Layer

<<Entity>>

├── Post ├── User ├── Tag ├── Setting

엔티티가 포함되며 DB 테이블에 매핑되는 클래스들이 존재한다.

<<support class>>

├── PostFormat (HTML/Markdown 구분)

├── PostStatus (Draft/Published 구분)

├── PostType (Page/Post 구분)

Enum 및 기타 도메인 관련 지원 클래스들이 존재한다.

<<DTO>>

├── PostForm ├── SettingsForm ├── UserForm

컨트롤러에서 Data를 전달하거나 받을 때 사용한다.

Domain Layer는 데이터의 표현 및 상태를 정의하고, 비즈니스 로직과 데이터베이스 간의 데이터 흐름을 관리한다.

5. InfraStructure Layer

├── Security

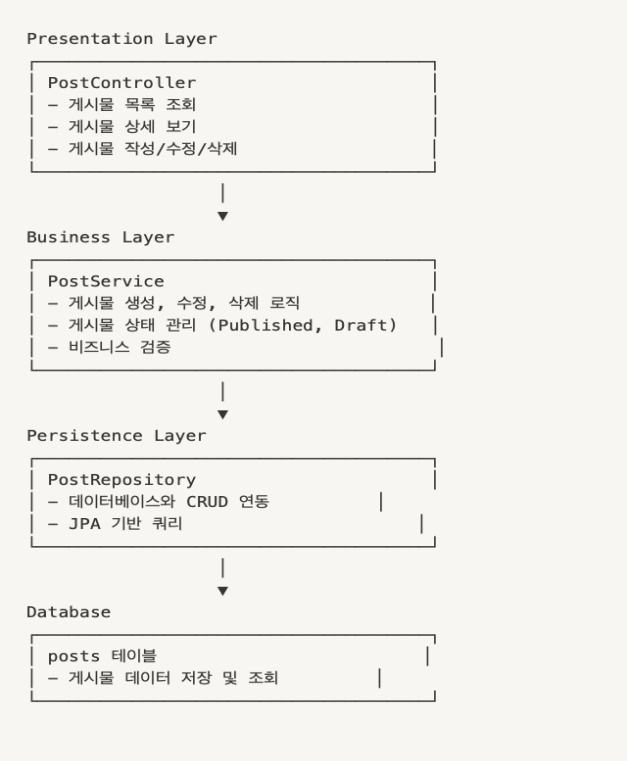
├── Caching

├── Utilities ── DTOUtil ── Markdown ── PygmentsService ── ViewHelper

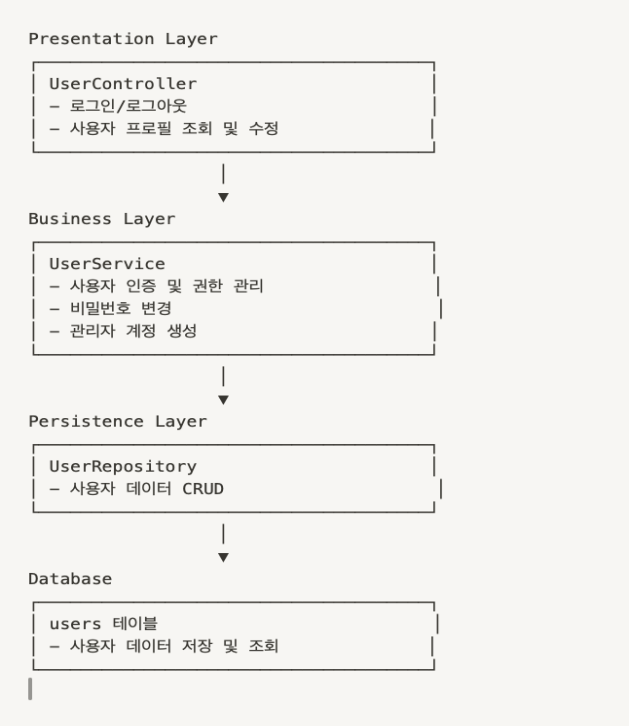
인증, 캐싱 및 공통 유틸리티 작업을 수행, 시스템의 비기능적 요구사항을 지원한다.

Spring blog에서 동작하는 기능을 **Layer Architecture**로 살펴본다면 다음과 같이 볼 수 있다.

1. 게시물 관리




2. 사용자 관리

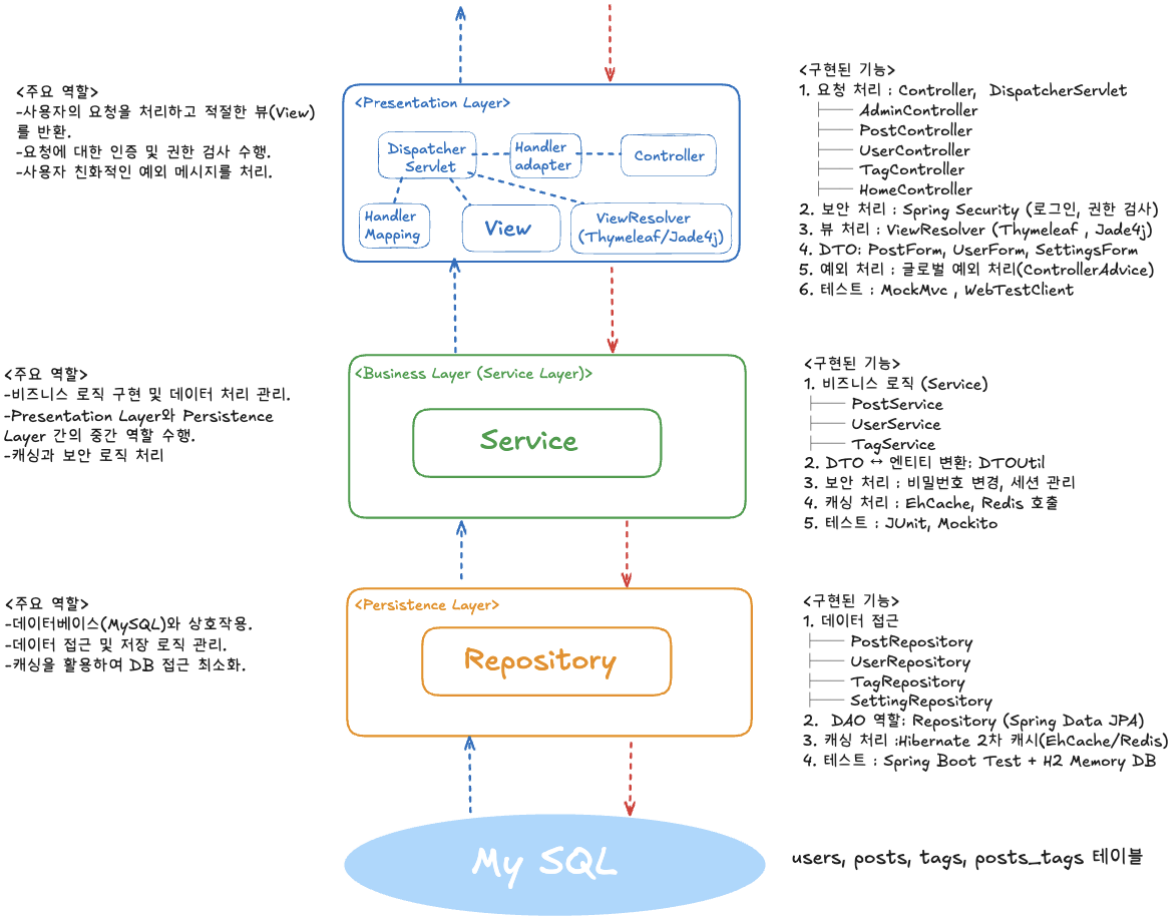


## 2.2 MVC Pattern

이처럼 코드를 분석하면서 Spring blog는 MVC 패턴도 사용하고 있음을 알 수 있었다.

먼저, 컨트롤러 역할이 존재했다. 프로젝트에서 `Controller` 클래스들이 HTTP 요청을 처리하고, 서비스 계층, 데이터 계층과 상호작용하고 있었다. 예를 들어 `PostController` 가 사용자 요청을 받아 데이터 처리 후 뷰로 전달하는 것이다. 다음으로는, 모델 역할의 명확성이다. 프로젝트의 엔티티 클래스(`Post`, `User` 등)가 데이터와 비즈니스 로직을 표현하고 있었다. 이 클래스들은 데이터베이스와 매핑되어 데이터의 상태를 관리했다. 그리고, 뷰 역할의 존재였다. 프로젝트에서 HTML, Thymeleaf, 또는 기타 템플릿 엔진을 사용해 데이터를 사용자에게 표시하는 코드가 있었다. 예를 들어, HTML 파일이 모델 데이터를 기반으로 UI를 렌더링. 마지막으로 코드내에서 명확한 상호작용의 흐름을 볼 수 있었다. 사용자 → Controller → Service → Model(Data) → View로 이어지는 흐름이 코드에서 관찰되었다.

이를 토대로, Spring Blog를 MVC 패턴과 Layer 계층으로 나눠 시각적으로 표현하자면 다음과 같다.



### 2.3 Repository Architecture

Spring Blog는 Layered Architecture와 MVC 패턴이 주요 구성 요소로 작동하는 동시에, 저장소 아키텍처(Repository Architecture)와 트랜잭션 처리 시스템도 시스템 설계의 중요한 요소로 활용되고 있다. 저장소 아키텍처는 대량의 데이터를 다루는 시스템에서 특히 유용하며, Spring Blog의 데이터 처리 방식에서 이를 확인할 수 있다.

저장소 아키텍처는, 데이터를 다루는 중앙 저장소를 중심으로 구성된 설계 방식이다. 한 컴포넌트에서 생성된 Data가 다른 컴포넌트에서 사용되는 응용분야에 적합하며, 컴포넌트들이 독립적이고 다른 컴포넌트들의 존재를 알 필요가 없다. 또한 옵서버 패턴과 유사하게 한 컴포넌트에 의한 변경이 모든 컴포넌트로 전파된다.

Spring Blog에서는 Repository 계층이 데이터의 CRUD 작업을 수행하며, 데이터의 중심적인 역할을 담당한다. 이를 통해 Service 계층은 Repository 계층을 호출하여 간접적으로 데이터에 접근한다.

```
public Post getPost(Long postId) {
    Post post = postRepository.findOne(postId); // 중앙 저장소에 데이터 요청
    return post;
}
```

PostService는 Data를 직접 다루지 않고, PostRepository를 호출해 Data에 접근한다. PostRepository라는 저장소에서 Data의 CRUD 작업을 수행하며 이를 모두 컨트롤한다.

```
// PostService
Post post = postRepository.findOne(postId);
// TagService
Tag tag = tagRepository.findByName(tagName);
```

Service 계층은 Service 계층끼리 서로 상호작용하는 것이 아니라 각자의 Repository를 통해 Data를 관리한다. 또한 Service 계층은 데이터 저장소에 의존하며, 다른 서비스와 Data를 직접 교환하지 않으며 서비스 간 독립성을 지킨다.

또한, Spring blog내에서 기사에 댓글을 달거나 좋아요를 누르면 이 알림이 타 유저에게도 넘어감을 알 수 있어 데이터의 변경이 모든 컴포넌트로 전파됨을 알 수 있다. 이와 같은 이유들로 인하여, Spring blog가 Repository Architecture를 사용하고 있음을 알 수 있었다.

### 2.4 Transictional Architecture

마지막으로, Spring blog는 트랜잭션 처리 시스템을 가진다. 트랜잭션 처리 시스템이란, DB에 있는 정보에 대한 사용자 요청이나 DB 갱신을 위한 요청 처리를 위해 설계된 Architecture이다. 이 아키텍처에서 한 트랜잭션 내의 모든 작업들은 DB 변경이 영구적으로 이뤄지기 전에 완료되어야 한다.

트랜잭션 처리 시스템은 사용자가 서비스 요청을 비동기적으로 처리하는 대화형 시스템의 일종이다. 사용자가 입출력 처리 컴포넌트를 통해 요청을 전달하면, 시스템은 이를 애플리케이션의 특화된 비즈니스 로직으로 처리한 후 데이터베이스(DB) 관리 시스템에 작업을 위임하여 트랜잭션 처리를 완료한다. 이러한 시스템은 요청 처리의 원자성(Atomicity), 일관성(Consistency), 격리성(Isolation), 지속성(Durability)을 보장하는 것이 특징이다.

Spring Blog 프로젝트에서는 **@Transactional** 어노테이션을 사용하여 트랜잭션 처리 시스템을 명시적으로 적용하고 있다.

예를 들어, 다음과 같이 CacheSettingService 클래스에서 이를 확인할 수 있다.

```
package com.raysmond.blog.services;

@Service
@Slf4j
@Transactional
public class CacheSettingService implements SettingService {
    private SettingRepository settingRepository;
    ...
```

위 코드에서 @Transactional 어노테이션을 통해 해당 클래스의 메서드들이 트랜잭션 환경에서 실행됨을 명시하고 있다. 이는 특정 메서드에서 데이터의 일관성과 무결성을 유지하며 작업을 수행할 수 있도록 지원한다. Spring Blog 프로젝트는 모든 클래스가 아닌, 특정 작업에만 트랜잭션을 적용하여, 관련된 작업을 원자적으로 수행할 수 있도록 설계하였다.

다음은 CacheSettingService 클래스의 get메서드 구현 예시이다.

```
@Override
    @Cacheable(value = "settingCache", key = "#key")
    public Serializable get(String key) {
        Setting setting = settingRepository.findByKey(key);
        Serializable value = null;
        try {
            value = setting == null ? null : setting.getValue();
        } catch (Exception ex) {
            log.info("Cannot deserialize setting value with key = " + key);
        }

        log.info("Get setting " + key + " from database. Value = " + value);

        return value;
    }
```

위 코드에서 GET 메서드가 호출되는 동안 다른 사용자가 데이터를 변경하는 경우를 방지하기 위해 작업이 트랜잭션 범위 내에서 원자적으로 수행된다. 이를 통해 데이터의 무결성을 보장받을 수 있다. 또한, 데이터 변환 중 오류가 발생시 예외를 처리해 시스템의 안전성을 유지한다.