KIPS Trans. Softw. and Data Eng.
Vol.8, No.3 pp.97~108 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2019.8.3.97

A Design and Implementation of Integrated Content Management System Based on Microservices Architecture

Kyung Sik Yoon† · Young Han Kim††

ABSTRACT

As digital content items increase, new content services are often added to or integrated among existing content management systems to manage them. For efficient system integration, this paper designed a content management system that combines two existing content management systems based on a microservices architecture. In addition, during the development process, integrated system that existing systems were recycled without disruption to existing systems, integrated efficiently and implemented as scalable systems. It measured the resource usage of this systems and analyzed the differences between features for system integration using traditional middleware.

Keywords : Microservices Architecture, Content Management System, System Integration

마이크로서비스 아키텍처 기반의 통합 콘텐츠 관리 시스템 설계 및 구현

윤 경 식† · 김 영 한††

요 약

디지털 콘텐츠 재화가 증가됨에 따라 이를 관리하기 위한 콘텐츠 관리 시스템에 새로운 콘텐츠 서비스를 추가하거나 기존 콘텐츠 관리 시스템 간에 통합하는 경우가 빈번하게 발생한다. 효율적인 시스템 통합을 위해 본 논문에서는 이 기종으로 구성된 두개의 콘텐츠 관리 시스템을 마이크로서비스 아키텍처 기반으로 통합 콘텐츠 관리 시스템을 설계하여 개발 간 기존 시스템의 중단 없이 재활용이 가능하고, 효율적으로 통합이 가능하며, 확장성을 가진 시스템을 구현하였다. 이를 통해 구현된 시스템의 소요되는 자원 사용량을 측정하고, 기존 미들웨어를 사용한 시스템 통합을 방식 간의 차이점을 분석하였다.

키워드 : 마이크로서비스 아키텍처, 콘텐츠 관리 시스템, 시스템 통합

1. 서 론

세계 디지털 콘텐츠에 대한 재화의 증가는 그에 따른 관련 소프트웨어들이 함께 요구될 것으로 보인다. 세계 디지털 콘텐츠 시장 규모 및 성장률은 2021년까지 지속적인 성장을 전망하고 있다[1]. 이에 기업들이 웹을 기반으로 한 콘텐츠 서비스를 적극적으로 전개하고 있다. 이 흐름에 맞춰 콘텐츠를 관리하고 서비스를 하기 위한 관리 환경을 필요로 하고 있으며, 이러한 목적의 대표적인 시스템이 콘텐츠 관리 시스템이라고 할 수 있다.

콘텐츠 관리 시스템은 특히 웹을 기반으로 한 웹콘텐츠 관리에 매우 활성화되어 있는데, 이러한 시스템을 필요에 따라 직접 구축하거나, 또는 상용 소프트웨어를 기반으로 활용하는 경우가 많다. 이러한 콘텐츠 관리 시스템들의 공통적인 특징은 콘텐츠의 데이터를 저장할 저장소, 어플리케이션 로직을 수행하는 영역과 업무 컴포넌트 및 프론트 서비스 영역으로 구분된 일관성을 가지고 있다. 초창기 콘텐츠 관리 시스템은 시스템과 시스템의 연동보다는 목적에 부합된 시스템을 구축하는 것이 일반적이었다. 이러한 시스템들의 내부 구조는 하나의 목적에 맞는 시스템으로 단일화된 구조인 모놀리딕 구조를 가지고 있다[2].

최근 들어 PC, Mobile, TabletPC와 같은 다양한 콘텐츠 소비 매체들이 등장하면서, 이러한 콘텐츠 관리 시스템도 다양한 멀티채널을 통한 콘텐츠 서비스를 하기 위해서는 각 채널에 맞는 서비스 영역을 해당 어플리케이션의 언어로 개발되

† 정 회 원 : 숭실대학교 IT융합학과 석사과정
†† 종신회원 : 숭실대학교 전자정보공학부 교수
Manuscript Received : September 20, 2018
First Revision : December 14, 2018
Accepted : January 13, 2019
* Corresponding Author : Kim Young Han(younghak@ssu.ac.kr)

어야 하기 때문에 이러한 특성에 맞춰서 콘텐츠 관리 기능을 갖되, 프론트엔드로 서비스되는 영역은 API로만 제공되는 헤드리스(Headless) CMS가 등장하였다[3].

콘텐츠 관리 시스템은 각 기업 조직의 요구와 상황에 맞게 구축되거나, 상용 패키지 솔루션을 도입하여 구축한 경우가 대부분이다. 하지만, 조직의 통합, 분리 등의 이슈로 인해서 기존에 이러한 기반의 구축된 시스템을 통합하기 위해서는 플랫폼, 연동 인터페이스, 프로세스, 통합 범위 등의 여러 고려되어야할 사항이 있다. 그러나 대부분 조직 간 상이한 플랫폼, 상이한 연동 인터페이스로 인해 기술적인 어려움을 해결하기 위해 상용 어플리케이션 통합 솔루션을 도입하거나, 통합된 전체 시스템을 재구축을 한다.

이러한 시스템 재구축 또는 통합 솔루션 도입을 통한 새로운 시스템 구축 방법은 상용 소프트웨어에 대한 라이선스에 대한 비용뿐만 아니라, 새로운 시스템 구축을 위한 시간과 개발에 대한 부담이 발생하게 된다. 때문에 서비스가 증가되거나, 사용자가 늘어나는 데 있어 필요한 물리적인 하드웨어 자원에 대해서 효율적인 관리를 위해서는 설계된 통합 시스템에 대한 재구성이나 재개발이 발생하게 되므로, 본 논문은 이를 효율적인 설계를 통해서 대용량 콘텐츠에 대한 대응이나 콘텐츠 서비스를 증가하는데 있어 제약 없이 확장할 수 있는 방법을 제안한다.

본 논문은 이 기종 간의 콘텐츠 관리 시스템 간의 통합을 목표로 할 때 기존의 방법과 비교하여 기존 사용되는 시스템들을 그대로 재활용하면서 좀 더 빠르게, 효율적인 연동 방식인 마이크로서비스 아키텍처 기반에서 새로운 환경과 새로운 서비스를 구축할 수 있는 통합 콘텐츠 관리 시스템에 대한 설계에 대한 연구이다.

본 연구에서는 이 기종의 콘텐츠 관리 시스템의 중단 없이 상호 통합관리가 가능한 관리 시스템을 구현하는데 있다. 특히 콘텐츠 관리 시스템은 기업 내에서 콘텐츠 관리에 특화된 솔루션을 기반으로 사용되고 있는데, 내/외부 시스템간의 연계를 위해 표준연동 인터페이스를 제공하고 있다. 이 기종 간의 서로 다른 CMS (Content Management System)를 통합하기 위한 기존 방법들을 알아본다.

시스템 통합 방법 중 SOA (Service Oriented Architecture) 기반의 ESB (Enterprise Service Bus)를 활용한 통합방법과 마이크로서비스 아키텍처 기반의 API Gateway를 활용한 통합 방식에 대한 차이를 비교한다. 실제 구현된 시스템 통해서 제안한 시스템의 성능을 실험하고 분석한다.

2. 관련 연구

2.1 CMS 동향

일반적인 콘텐츠 관리 시스템들은 데이터베이스를 중심으로 관리를 위한 영역과 서비스 영역이 모두 단일화 구성되어 있었다. 이런 모놀리딕 구조의 경우, 프론트 서비스를 구현하기 위해서는 콘텐츠 관리 시스템에서 사용된 개발언어를 통해서만 구현이 가능했기 때문에 서비스를 확장하거나, 변경을 하기 위해서는 개발 측면에서는 소규모 수정에 따른 전체 배포 등의 생산성 저하 및 어플리케이션 확장을 위한 제약사항을 가지고 있다[4].

이러한 문제 때문에 멀티채널에 대한 다양한 프론트 서비스가 요구되면서 콘텐츠 관리 시스템에서 Fig. 1과 같이 뷰(View)가 없는 헤드리스(Headless) CMS[3]가 활용되고 있다. 콘텐츠 관리 시스템의 원래 목적인 관리 기능을 가지고 있으면서, 프론트엔드 측에는 API 를 활용하기 위해 제공하는 방식이다. 데이터 관리를 위한 메타데이터 설계 및 모델링을 할 수 있으며, API를 사용하기 위한 인증 수단을 제공한다. 최근 오픈소스 형태로 알려진 헤드리스 CMS로는 graphCMS[5], butterCMS[6], Contentful [7], DirectUs[8], Prismic.io[9] 등이 있다. 그리고, 온-프레미스(On-premises) 기반과 오프-프레미스(Off-Premises) 기반의 SaaS (Software as a Service)서비스 형태 모두를 제공하는 추세이다.

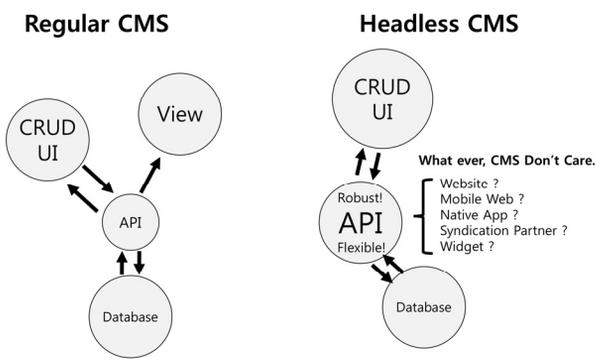

Fig. 1. Difference Between Regular CMS and Headless CMS

이러한 헤드리스 CMS에서는 데이터를 XML이나 JSON형태로 제공하고 있으며, 프레젠테이션 영역에 맞는 개발자가 선호하는 개발언어를 사용하기 때문에 다중 채널이나 다수로 구성된 웹사이트에 서비스로 전달이 용이하다. 기획부터 개발까지 리드타임이 낮고, 낮은 운영비용을 가지고 있는 구조여서 유지보수성이 좋다. 서비스로 제공되는 물리적인 파일은 CDN(Content Delivery Network)과 같은 별도의 서비스로 분리하여 관리할 수 있어, 글로벌 네트워크 서비스에 효율성으로도 장점을 꼽을 수 있다. 하지만, 이러한 구조의 단점은 뷰 영역에 대한 직접적인 처리를 담당하지 않기 때문에 사이트 구축 시에는 개발자의 역할이 필요하며, 마케팅 담당자는 직접 서비스에 필요한 시나리오에 개입하기 어려우므로, 개발자의 의존도가 높다[10].

오픈소스 CMS로 가장 많이 알려진 WordPress[11]의 경우, 모놀리딕 구조이지만, 외부 연동을 위한 Restful API를 활용할 수 있는 플러그인을 공식적으로 제공함으로써 헤드리스 CMS와 동일한 기능을 수행한다. 그 외에 이러한 환경을 고려한 오픈소스 CMS로는 Drupal[12], Neos[13], Django[14] 등이 추가로 Restful API를 제공하고 있다.

2.2 시스템 통합 기술 요소

콘텐츠 관리 시스템을 비롯한 대부분의 솔루션 벤더(Vendor)의 제품이나, 목적에 따라 개발된 업무용 시스템들이 점점 시스템 간의 연동에 대한 이슈가 증가하면서 시스템 통합에 대한 여러 가지 방법들이 존재한다.

어플리케이션 간 연동을 돕는 상용도구 EAI (Enterprise Application Integration), ESB (Enterprise Service Bus) 등을 이용하거나, CDC (Change Database Capture)와 같이 물리적으로 DB를 복제하여 통합하는 방식 등이 있다.

EAI는 기업 내 이 기종 간의 어플리케이션이나, 데이터베이스, 네트워크 등을 통합하는 미들웨어이다. 이것은 가상의 단일 플랫폼 또는 단일 시스템처럼 보이도록 환경을 제공하고 있다. 이는 이 기종 간에 어플리케이션의 연동 인터페이스를 어댑터를 사용하고 있으며, 다른 시스템간의 데이터 포맷 차이로 변환을 데이터 브로커(Data Broker)를 사용해서 변환된 데이터맵(DataMap)을 생성한다. 이렇게 생성된 데이터맵을 어댑터(Adapter)를 통해서 각각의 어플리케이션에 전달하도록 하고 있다. 업무 간 비즈니스 프로세스를 어플리케이션 간 워크플로우를 설계해야 하는 경우에도 활용되고 있다[15].

ESB는 어플리케이션을 비즈니스 서비스 단위로 연결하고 중개하는 메시지 기반의 미들웨어이다. 라우팅 및 메시지 변환 기능을 가지고 있으며, 어플리케이션 간 메시지 브로커를 이용하여 단순형태의 중개 기능을 제공한다. 여기서 ESB는 SOA를 지향하는 기술로써 기존 어플리케이션에 대한 통합 또는 기존 과 새로운 어플리케이션 추가를 통한 시스템 확장을 할 수 있도록 한다[16]. 때문에 어플리케이션 간 상이한 플랫폼이더라도 확장성 있게 연동을 지원할 수 있도록 지원하고 있으며, 비교적 IT 자원이 부족하거나, 낮은 비용의 시스템 통합에서 활용된다[17].

그러나, ESB를 활용한 연동에는 어플리케이션 간의 프로토콜이 각기 다르거나, 주요하게 사용된 프로토콜은 SOAP, WSDL, UDDL이며, 특히 SOAP 프로토콜은 REST와 비교했을 때, 낮은 성능과 높은 비용을 보인다[18-20].

2.3 마이크로서비스 아키텍처

마이크로서비스 아키텍처는 독립적으로 분리된 소규모의 서비스 단위를 연결하여 전체 어플리케이션을 설계하는 아키텍처이다. 이러한 구조는 어플리케이션 측면, 어플리케이션 인프라스트럭처, 인프라스트럭처 패턴으로 구분할 수 있다[21].

이러한 구조는 서비스 분리 단위를 비즈니스 기준 으로 분리하거나, 테스트를 간소화할 수 있는 독립적인 규모로 구성될 수 있는 팀이 서비스 단위로 분리될 수 있다. 그리고 소스에 대한 배포 단위는 호스트 당 다중 서비스 인스턴스, 호스트별 서비스 인스턴스, VM(Virtual Machine)별 인스턴스, 서버리스(Serverless) 배치 플랫폼, 서비스 배포 플랫폼 등으로 배포 단위를 다양하게 하도록 설계할 수 있다. 분리된 서비스 간에는 원격 프로시저 호출로 서비스 간 통신을 RPI(Remote Procedure Invocation)기반 프로토콜 또는 서비스 간 통신에 비동기 메시징을 사용하거나, 도메인 특정 프로토콜을 쓸 수 있다.

외부 클라이언트들의 서비스를 통합하는 인터페이스 제공을 위해서 API gateway를 필수로 사용하며, 프론트엔드를 위한 백엔드를 구성하기 위해 API를 활용할 수 있다. 때문에 API gateway를 통한 통합 및 각 클라이언트에 최적의 API를 제공할 수 있도록 한다.

신뢰성 측면에서는 네트워크 또는 서비스 장애가 다른 서비스에 연속적으로 영향을 주지 않도록 원격 호출에 따른 실패 임계치를 설정하여, 수치에 따라 차단하거나 작동을 재개하여 장애에 대한 파급을 낮추도록 한다. API gateway에서 서비스 호출에 이 패턴을 활용하여 API 호출된 서비스의 신뢰성을 높여준다. API Gateway에서는 프론트엔드/백엔드와 서비스사이에 위치하며, 리다이렉트 또는 포워딩을 통해서 다른 서비스로의 연결을 돕는다. 하나의 API Gateway만을 운용시의 과부하를 피하기 위해서 클라이언트 어플리케이션 기준으로 여러 개로 API Gateway 분리하여 트래픽을 분산시킬 필요가 있을 수 있다. 이 경우 API Gateway의 앞단에 Load Balancer를 활용하여, 요청된 API URL에 따라 적절한 API Gateway를 통해 해당 백엔드 모듈로 서비스가 연결되게 한다.

마이크로서비스 구조에서 데이터 일관성을 유지하면서 쿼리를 구현하는 방법으로 API 컴포지션 (Composition)나, CQRS (Command Query Responsibility Segregation)을 활용한다. 데이터베이스 구조는 서비스 별 데이터베이스를 두거나, 서비스별 데이터베이스를 분리하는 방식을 적용할 수 있다. 보안 측면에서는 서비스 간 요청자의 인증 및 인가를 위해 액세스 토큰(Access Token)을 통해 사용자 정보를 안전하게 저장하는 역할을 한다.

구현된 시스템의 테스트를 쉽게 구성하기 위해서 호출하는 모든 서비스에 대한 테스트 이중화를 사용하여 격리된 서비스를 테스트하는 방식으로, 서비스 구성 요소 테스트를 적용하거나, 서비스를 사용하는 다른 서비스의 개발자가 작성한 서비스를 위한 테스트 방식인 서비스 통합 테스트를 진행하기도 한다. 그리고 응응프로그램의 운용 관측 측면에는 로그집계, 어플리케이션 프로그램 측정, 사용자 활동 기록, 트레이스(Trace), 예외 추적(Exception Tracking), API 가동 현황 점검(Health Check API), 배포 기록을 확인하는 방법을 사용한다.

3. 설계 및 구현

3.1 설계

본 논문에서는 통합 콘텐츠 관리 시스템을 구현하기 위해 통합 대상인 두개의 콘텐츠 관리 시스템을 임의 선정했다. 하나는 오픈소스 콘텐츠 관리 시스템으로 전 세계적으로 가장 많은 사용을 하고 있는 워드프레스 (WordPress) 이며, 또 다른 하나는 헤드리스 CMS로써, 콘텐츠를 위한 메타데이터 설계가 용이한 다이렉트어스(DirectUs)를 선정했다. 두 개의 시스템 모두 각각 독립적인 환경에서 운영되고 있는 시스템이며, Restful API 프로토콜을 지원하고, 소규모 단위의 서비스 API를 제공하고 있다. 이러한 시스템을 통합하여 콘텐츠 서비스할 수 있는 웹사이트와 새로운 통합 관리 시스템을 구현하는 것을 목표로 한다.

이 구현을 통해서 기대하는 요소는 다음과 같다.

첫째 기존 시스템에 대한 보존 및 재활용을 한다. 통합에 따른 기존 투자비용을 줄이는 방법은 기존 시스템의 재활용이다. 때문에 기존 운영 중인 시스템에 대해서 독립적인 운영 환경을 유지하고, 중단시간을 최소화하여 새로운 시스템을 구현할 수 있도록 한다.

둘째는 통합 콘텐츠 관리 시스템을 통해서 기존 시스템의 운영을 위한 업무를 할 수 있도록 콘텐츠 관리를 위한 기능을 포함할 수 있도록 한다.

셋째는 향후 증가될 서비스에 대해 확장성을 가진 시스템을 구현한다. 때문에 서비스가 추가되거나, 서비스 이용자가 증가하더라도 증설이 가능한 구조이어야 한다. 향후 새로운 어플리케이션 서비스가 추가되더라도, 기존 시스템에 대한 수정을 최소화 하도록 한다. 일관성 있는 인터페이스 설계 구조를 통해서 새로운 시스템이 추가되더라도 같은 프로세스와 연동 규격을 통해서 확장될 수 있도록 한다.

1) 시스템 구성도

목표 시스템은 Fig. 2와 같이 마이크로서비스 아키텍처에서는 여러 시스템간의 연동된 시스템의 백엔드와 프론트엔드를 연결하는 역할을 API Gateway를 통해서 진행된다.

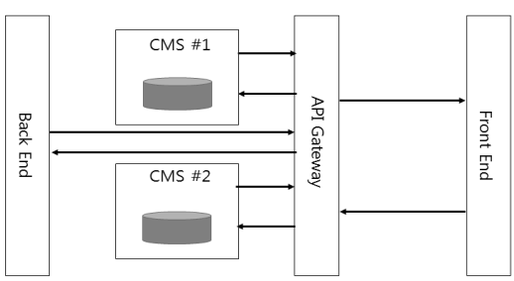

Fig. 2. Target System

API Gateway의 주 역할은 다음과 같다.

- 각 CMS에 요청하고 받은 API 정보를 새로운 경로의 URL로 치환하여 Legacy 시스템으로부터 요청 및 응답 한다.
- API 호출이 여러 호출이 순차적으로 발생하는 요청을 하나의 요청으로 집계(Aggregation) 한다.
- 통합 콘텐츠 관리 시스템에서 관리가 용이한 형태의 API 규격에 맞게 조정(Mediation) 한다.
- CMS에 연결 시, API 요청 시, 인증을 통해서 허용한다. 각각의 서비스 단위로 부터 API를 연결하여 새로운 API를 구성할 때 핵심요소이며, API 설계에서 URL 규칙 및 API 형식에 대한 정의를 API gateway를 통해서 진행된다.

2) 시스템 시퀀스다이어그램

통합 콘텐츠 관리 시스템이 각 시스템의 API를 활용하기 위해서는 인증 이후에 호출하도록 한다. CMS#1 과 CMS#2의 경우도 각 시스템의 아이디와 패스워드로 인증하여 연동하였다. 인증을 통해 API 요청 및 응답은 Fig. 3과 같은 시퀀스다이어그램 기술하였다.

3) API 설계

각 시스템 단에서 요청되는 통계에 대한 분석을 위해 백엔드와 프론트엔드에서 요청되는 API를 분리해서 관리가 필요하여 Fig. 4와 같이 URL 패턴을 설계하였다. 또한 기존 사용중인 CMS에서 호출되는 구분, 그리고 데이터를 쓰거나, 읽거나, 수정하거나, 삭제하기 위한 각각의 호출정보를 API 로 분리하도록 했다.

이를 기준으로 통합 콘텐츠 관리시스템 구현을 위해 설계한 API 정보는 아래와 같다.

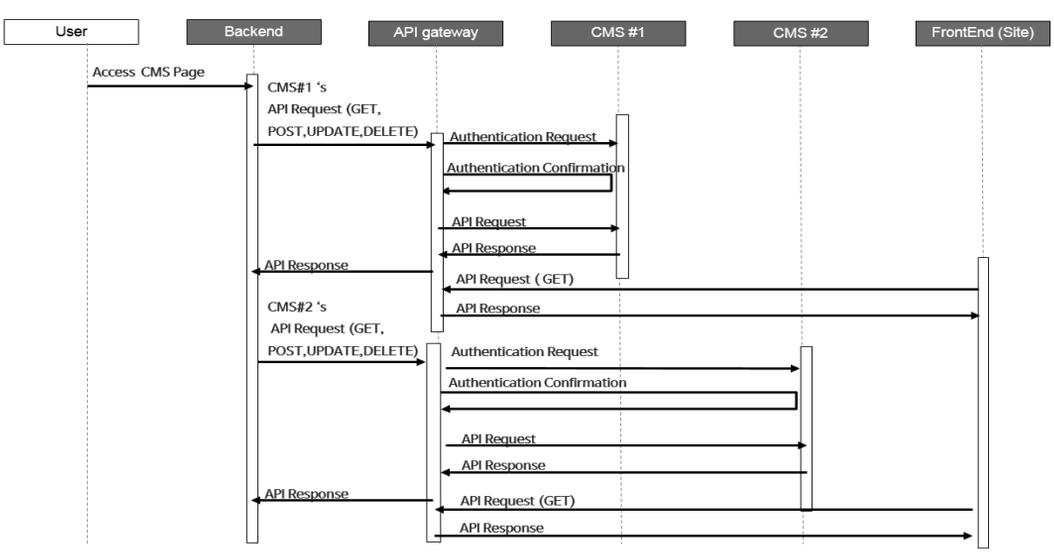

Fig. 3. Sequence Diagram

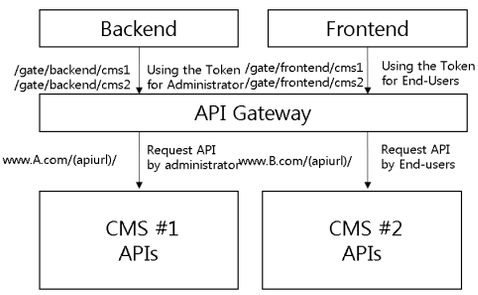

Fig. 4. Design of API URL

a) 콘텐츠 표준 메타데이터 정의

기존 CMS 마다 콘텐츠의 메타데이터를 정의하고 있는 포맷이 다르기 때문에 각 시스템에서 사용하고 있는 콘텐츠에 대한 메타데이터를 통일하기 위한 표준 메타데이터 정의가 필요하여, Table 1과 같이 통합된 메타데이터 필드를 설계하였다.

Table 1. Definition of Integrated Content Metadata Field

Integrated Metadata	CMS#1 Metadata	CMS#2 Metadata
id (Integer)	id (integer)	row-id (integer)
status (string)	status (String)	status (string)
title (string)	title (string)	title (string)
content (Text)	content (Text)	content (Text)
author (String)	author (Integer)	author (String)
date (Date)	date (Date)	date (Date)
category (String)	categories (String)	table-name (String)

b) 백엔드영역을 위한 API 설계

API Gateway를 통해서 백엔드에서만 사용되는 API를 구분하기 위해 'backend' 경로를 생성하였다. 그리고 호출되는 기존 시스템을 구분하기 위해 다음처럼 /gate/backend/cms1/, /gate/backend/cms2/ 각각 cms1과 cms2로 구분하였다.

향후 새로운 서비스가 추가될 경우, 시스템의 넘버를 증가시켜 cms(n)으로 확장할 수 있도록 한다. Backend에서 API는 관리 시스템의 용도이기 때문에 데이터를 관리하기 위하여 생성, 읽기, 갱신, 삭제 가 가능해야한다. 따라서, 아래와 같이 기능별로 사용할 요소는 posts, view, write, delete, update를 정의한다.

./posts/, ./write/, ./delete/, ./update/ 와 같이 구분된 URL을 통해서 API 동작에 대한 쉽게 이해를 할 수 있도록 하위 경로를 구성하며 전체 경로는 다음과 같다.

/gate/backend/{CMS1|CMS2}/{posts|view|write|delete|update}/

c) 프론트엔드 영역을 위한 API 설계

API Gateway를 통해서 프론트 영역에서만 사용되는 API를 구분하기 위해 프론트엔드 경로를 생성하였다. 데이터 호출이 가장 많을 것으로 보이고, 별도의 캐시(Cache) 설정을 위한 API를 생성하고, 갱신 주기 등을 설정하여, 좀 더 효율적으로 운영할 수 있도록 한다. 그리고 호출되는 기존 시스템 구분을 위해 다음과 같이 /gate/frontend/cms1/, /gate/frontend/cms2/를 각각 cms1과 cms2로 구분하였다.

프론트 서비스에서는 단방향 서비스 위주의 배치로 인해 보기를 위한 API만 허용하며, 다음과 같은 ./posts/, ./view/로 하위 경로를 구성하며 전체 경로는 다음과 같다.

/gate/frontend/{CMS1|CMS2}/{posts|view}/

3.2 구현

설계한 내용을 기반으로 통합 콘텐츠 관리 시스템을 구현하였다. 구현을 위해서는 Fig. 5와 같이 적용되었으며, 기존 CMS#1, 기존 CMS #2, API Gateway, 백엔드, 프론트엔드 영역으로 구분된다.

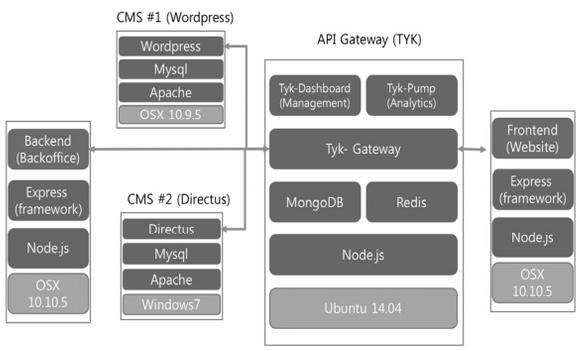

Fig. 5. System Architecture

기존 CMS의 경우, 각각 다른 도메인을 가지고 있기 때문에 타 도메인 간 데이터 교환을 가능하도록 하는 크로스도메인 처리를 위한 설정을 해주도록 한다. 백엔드와 프론트엔드는 node.js기반에 익스프레스 프레임워크(Express framework)를 사용하였다. 통합 관리자를 위한 계정 정보는 API Gateway의 계정정보를 활용하거나, 자체 로그인 정보를 생성하여 관리하도록 할 수 있는데, 구현은 익스프레스 세션(Express-session)을 활용한 자체 로그인으로 구현하였다.

1) 기존 CMS 정보

우선, 연동을 위한 두개의 기존 CMS의 시스템의 정보는 다음과 같다.

첫 번째 CMS#1인 워드프레스(Wordpress)는 콘텐츠 관리 시스템으로 대표적인 오픈소스 솔루션이다. 외부연동을 위한 인터페이스로 Restful API지원을 위해 추가 플러그인(WP REST API)[22]을 제공하고 있으며, API 인증을 위한 Basic Auth를 사용할 수 있다.

두 번째 CMS#2 는 다이렉트어스(DirectUs)[8]로 오픈소스 헤드리스 CMS 중의 하나로 PHP기반에 Mysql 환경에서 운용된다. 콘텐츠 메타데이터 설계 기능이 있어, 콘텐츠 통합을 위한 설계를 별도로 할 수 있으며, 프론트엔드에서 사용할 수 있는 API 인증을 위한 방식으로 Basic Auth와 Bearer Token을 사용할 수 있다.

2) API Gateway

본 논문에서는 마이크로서비스 아키텍처에서 연동을 위한 미들웨어로 TYK (API Gateway)[23]를 사용하여 각 Legacy 시스템과의 연동을 구현하였다.

TYK는 클라우드 서비스 및 분산에 대비한 클러스터링(Clustering)을 지원하여, 향후 증설 및 대용량에 대한 대응이 가능하다.

a) API Gateway를 이용한 API 생성

API 리스트를 Table 2와 같이 생성하였다. 프론트엔드 영역과 백엔드 영역을 구분하고, 동작의 목적에 따라 API를 분리하여 구성하여 관리되도록 하였다.

Table 2. API List

Type	Target	API Name	Action
Backend	CMS1	Back CMS1	List,Create,Update,Delete
		Back CMS1 view	Read
		Back CMS1 tables	Table Information
	CMS2	Back CMS2	List,Create,Update,Delete
		Back CMS2 view	Read
		Back CMS2 tables	Table Information
Frontend	CMS1	Front CMS1	List
		Front CMS1 view	Read
	CMS2	Front CMS2	List
		Front CMS2 view	Read

b) API Gateway를 이용한 조정

Fig. 6과 같은 절차로 API 포맷변경을 위한 조정(Mediation)을 하여, 통합 콘텐츠 관리 시스템의 백엔드와 웹사이트에서 사용할 API 의 형식을 통일하였다. 이를 통해서 통합 인터페이스에서 활용할 수 있도록 변경 하였으며, 이러한 포맷 조정을 통해 향후 Rest API를 지원하는 다른 API 서비스도 추가될 수 있다.

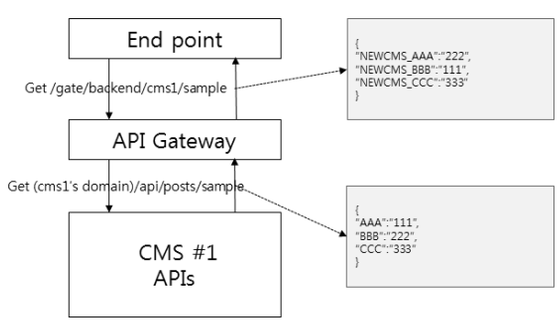

Fig. 6. Example of Mediation Processing for API Formatting

여기서 사용된 변환 템플릿을 위한 스크립트는 golang[24]을 사용하였으며, Fig. 7은 실제 포맷을 변경한 실제 변환 템플릿 중 List를 구성하기 위한 API 포맷을 변환 코드이다. 위의 코드는 CMS#1에 대한 코드이며, 아래의 코드는 CMS#2에 대한 코드이다. 실제 변환된 코드는 Fig. 8처럼 동일한 형식으로 변환된다.

```
[{{range $index, $element := .}}
  {{if $index}}
    ,{
      "id" : "{{.id}}"
      ,"date" : "{{.date}}"
      ,"author" : "admin"
      ,"active" : "{{.status}}"
      {{range $subindex, $subelement := .title}}
      ,"title": "{{$subelement}}"
      {{end}}
    }
  {{else}}
    {
      "id" : "{{.id}}"
      ,"date" : "{{.date}}"
      ,"author" : "admin"
      ,"active" : "{{.status}}"
      {{range $subindex, $subelement := .title}}
      ,"title": "{{$subelement}}"
      {{end}}
    }
  {{end}}
{{end}}]
```

```
[{{range $index, $element := .data}}
  {{if $index}}
    ,{
      "id" : "{{.id}}"
      ,"date" : "{{.date}}"
      ,"status" : "{{.status}}"
      ,"author" : "{{.author}}"
      ,"title": "{{.title}}"
    }
  {{else}}
    {
      "id" : "{{.id}}"
      ,"date" : "{{.date}}"
      ,"status" : "{{.status}}"
      ,"author" : "{{.author}}"
      ,"title": "{{.title}}"
    }
  {{end}}
{{end}}]
```

Fig. 7. Template Code for Content List API

이 과정을 통해서 통합 메타데이터에서 정의한 기준에 맞게 형식을 변경하도록 하며, 이 규칙은 설계에 따라 메타데이터를 추가하거나, 변경함으로써 메타데이터의 포맷이나, 데이터의 규칙을 변형하도록 한다. 다만, 서비스 API 에서는 메타데이터에 대한 필드 설계가 가능해야 하는 전제가 필요하다.

```
// 20180412202424
// http://10.211.55.7:8080/gate/backend/cms2/third/posts/

[
    {
      "id": 16,
      "date": "2018-03-28T09:32:29-04:00"
      "status": "publish",
      "author": "admin",
      "title": "test"
    },
    {
      "id": "11",
      "date": "2018-03-26T05:27:56-04:00"
      "status": "publish",
      "author": "choonzang",
      "title": "홈페이지 업데이트"
    },
    {
      "id": "14",
      "date": "2018-03-26T09:32:44-04:00"
      "status": "publish",
      "author": "admin",
      "title": "test"
```

```
// 20180417200713
// http://10.211.55.7:8080/gate/backend/cms1/1/posts/

[
  {
    "id": "1308",
    "date": "2018-03-29T00:05:13",
    "author": "admin",
    "active": "publish",
    "title": "testtse"
  },
  {
    "id": "1307",
    "date": "2018-03-28T23:50:50",
    "author": "admin",
    "active": "publish",
    "title": "test"
  },
  {
    "id": "1298",
    "date": "2018-03-28T01:28:03",
    "author": "admin",
    "active": "publish",
```

Fig. 8. Result of Mediation for List API

c) API Gateway를 이용한 인증

API Gateway를 통해 기존 CMS 측에 Request를 보낼 때, 인증이 필요하다. CMS#1(Wordpress)에서는 Basic Auth를 CMS#2(DirectUs)에서는 Bearer Token을 통해서 인증하도록 하였다. 때문에 요청이 발생하는 API 호출에서는 헤더에 해당 정보를 통해서 호출되도록 하여 인증, 권한 설정을 통해 시스템 간의 API 사용에 대한 권한을 설정하였다.

3) 백엔드 구성

통합 콘텐츠 관리를 위한 통합 인터페이스를 별도의 서버 환경을 구축하여 구현하였다.

백엔드 시스템 구성 환경은 다음과 같다.

- OS : MAC OSX 10.10.5
- node.js 8.9.3
- express(framework) 4.15.5 , express-session 추가

통합 콘텐츠 관리를 위한 백엔드 영역을 Fig. 9와 같이 구현하였다. 관리자 화면을 통해 인증 및 인가된 사용자가 로그인 페이지, 콘텐츠의 리스트, 상세보기, 등록, 수정, 삭제 할 수 있다.

기존 CMS에서 새로운 콘텐츠 등록을 위해 새로운 테이블을 생성하였을 때, 생성된 테이블 정보를 참조하여 테이블 별 데이터의 리스트를 볼 수 있도록 구성한다.

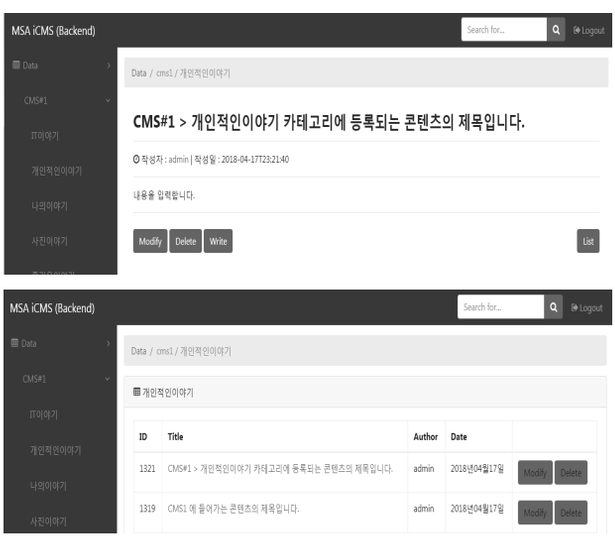

Fig. 9. Implemented Backend System Screens

4) 프론트엔드 구성

통합 콘텐츠 관리 시스템을 이용한 웹사이트를 구축하였으며, Fig. 10에서는 CMS#1과 CMS#2의 콘텐츠를 통합된 리스트 제공된 화면이다.

프론트엔드 시스템 구성 환경은 다음과 같다.

- OS : MAC OSX 10.10.5
- node.js 8.9.3
- express(framework) 4.15.5

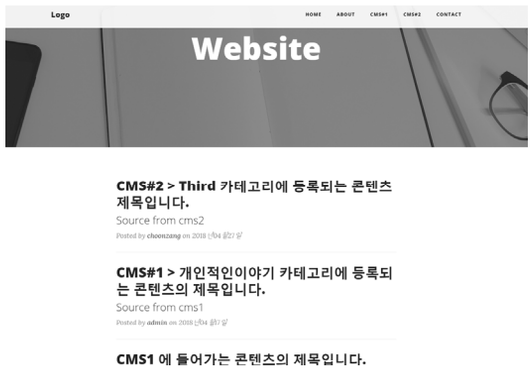

Fig. 10. Implemented Frontend Website Screen

구현된 시스템은 Fig. 11과 같은 구조로 정리된다.

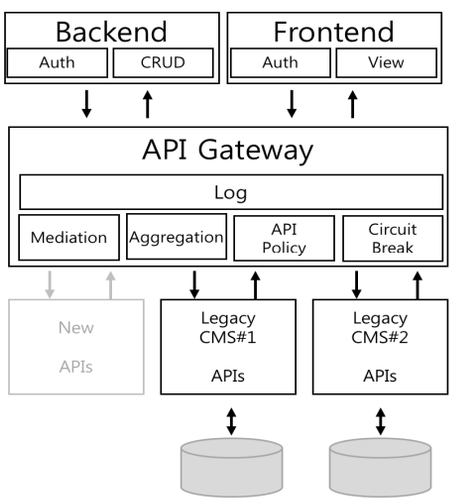

Fig. 11. Implemented System Architecture as a Result

기존 CMS들은 축소된 API Service들로 활용되고, API Gateway를 기반으로 모든 엔드 포인트(End-point)를 일원화하여 통합관리시스템에서 제어된다. 각각의 API는 독립적인 환경으로 각 API 별로 DB 역시 분리되어있으며, 각 서비스에 대한 API 서비스 환경이나, 관리 규칙 등을 서비스 API 단위로 구성된다. 새로운 서비스를 API로 연결하기 위해서는 API Gateway를 통해서 제어 및 변환하고, 서비스 운영을 위한 정책을 결정할 수 있으므로, 이러한 구조는 마이크로서비스 아키텍처의 일반적인 특징으로 볼 수 있다.

4. 실험 및 분석

4.1 테스트 환경 및 실험

본 논문에서는 통합 콘텐츠 관리 시스템의 동작 성능 분석을 위해서 테스트 시나리오를 정의하고, 자원 사용량 측정을 실험하였다.

구현 시스템에 대한 측정을 위해서 키바나(Kibana), 엘라스틱서치(Elastic Search), 메트릭비트(Metricbeat)[25]의 구

성을 통해서 사용하였다. 측정이 필요한 백엔드 구현 시스템 측에 메트릭비트의 클라이언트를 설치하고, 시스템의 자원에 대한 사용에 대한 정보를 엘라스틱서치로 저장하였다. 이때 메트릭비트에서 수집되는 정보 중에 시스템 자원 사용에 대한 측정에 필요한 정보로 전달되도록 대시보드(Dashboard)를 위한 플러그인을 추가로 설치한다. 그리고 Fig. 12와 같이 키바나는 수집된 정보를 대시보드 또는 지정 차트를 선정해서 자세한 정보를 제공한다.

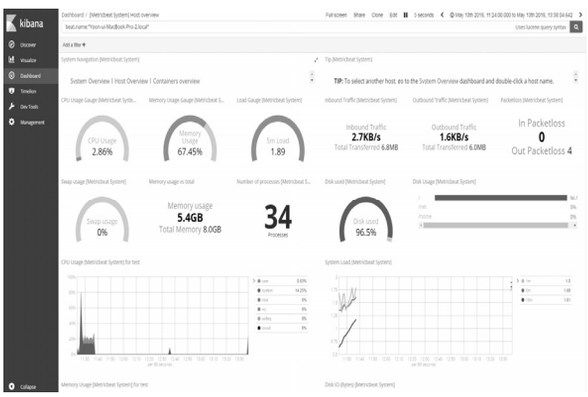

Fig. 12. System Resource Measuring Using Kibana

테스트할 통합 콘텐츠 관리 시스템의 환경 정보는 Table 3과 같다.

Table 3. Environment for Test System

Hardware	MacBook Pro (Retina, Mid 2012)
OS	OS X Yosemite 10.10.5
CPU	Intel core i7-3732QM 2.6GHz
Memory	8GB 1600 MHz DDR3
Graphic	Intel HD graphics 4000 1024MB
Disk	HDD 500GB

본 시스템 내에 VM(Virtual Machine) 환경 구성을 통해서 별도의 OS(Ubuntu 14.04) 에 API Gateway 시스템을 구성되었으며, 그 환경은 Table 4와 같다.

Table 4. Environment for API Gateway

OS	Ubuntu 14.04 TLS 64bit
CPU	Intel core i7-3732QM 2.6GHz x 2
Memory	2GB 1600 MHz DDR3
Disk	66.4GB

백엔드와 프론트엔드 서버의 환경은 MAC 내에서 별도의 인스턴스를 구성하여, 각각 node.js를 이용한 서비스를 가동하도록 구성되어, OSX 에서 terminal 을 통해 기동하도록 구성되어있다. 구현한 시스템에 대한 시스템 운용 간 자원 사용량에 대한 측정을 위해 실험을 진행하였다. 기존 연동 대상의 API 호출에 따른 기존 CMS들은 연동을 위해 별도로 개발된 영역이 아니므로, 자원 사용의 측정범위에서 제외하였다.

실험에 대한 시나리오는 다음과 같다.

① 서비스를 모두 중지된 상태에서의 기본 OS 구동에 따른 CPU와 메모리 사용량을 2분 간 측정한다.

② API Gateway 실행을 위해 우분투(Ubuntu) OS 실행 및 API Gateway 서비스 실행 후, CPU와 메모리 사용량을 3분 간 측정한다.

③ 백엔드 서비스와 프론트엔드 서비스 실행 후, CPU와 Memory 사용량을 2분 간 측정한다.

④ 백엔드 서버에 접속하여 로그인, 콘텐츠 등록, 수정, 삭제, 보기 기능 수행 간 CPU와 Memory 사용량을 3분 간 측정한다.

⑤ 프론트엔드 서버에 접속하여 콘텐츠 보기 실행 간 CPU와 Memory 사용량을 2분 간 측정한다.

총 10분에 걸쳐 각 단계별 진행에 대한 기록을 모니터링하여 운용에 따른 성능을 기록하여 구현 시스템에 대한 자원 사용량을 측정하였다.

4.2 시스템 운용 성능분석

실험을 통해서 리소스 자원에 대한 사용 성능을 분석하였다. Fig. 13, Fig. 14에 따르면 API Gateway구동을 위해 OS를 실행하는데, CPU 사용률은 26.9%를 보였으며, 이어서, API Gateway 서비스 구동을 위해 5.6%로 상승한 것을 확인하였다. 구동 전 평균 CPU 사용량은 약 0.5%이었으며, 거의 사용이 없는 상태여서 API Gateway 서비스 구동 이후에 약 3.2% 사용량이 증가된 것을 확인하였다. 그러나 전체 시스템 대비 증가 수치는 비교적 낮은 사용률을 보이고 있다. 메모리의 경우, 평소 3.9GB 사용 중, 시스템 구동 및 사용 간 5.1GB로 약 14.3% (1.2GB) 증가된 수치를 보였다. 이는 OS 구동에 따른 메모리 사용률로 OS 가동 이후에, API Gateway 서비스 실행을 통한 증가 수치는 4.9GB에서 5.1GB 로 약 200MB의 사용량을 보였다. node.js 기동에 대해서는 당시의 일시적인 CPU 증가를 제외하고는 Memory에서는 거의 증가된 범위를 확인할 수 없었다. 때문에 서버 기동에 따른 메모리 사용은 큰 비중을 차지하지 않으며, Cache양의 부분적인 증가로 볼 때, 복잡하고 다수의 페이지 구성이 아닌 백엔드 환경에서는 Memory의 추가적인 증가보다는 캐시(Cache)의 증가만 발생할 것으로 예상된다.

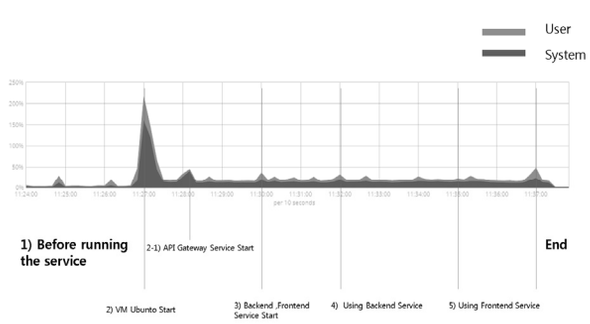

Fig. 13. Measurement Chart of System CPU Usage

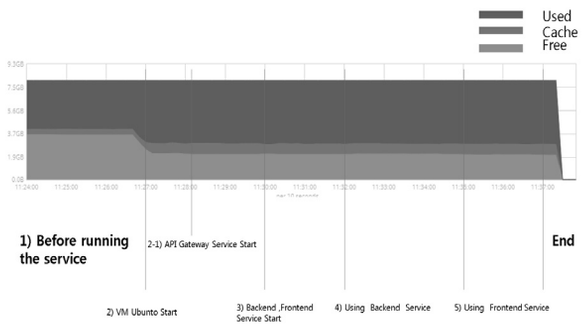

Fig. 14. Measurement Chart of System Memory Usage

Table 5와 Table 6의 결과로 볼 때, 구현된 시스템의 서비스 실행 과정에서의 CPU와 메모리 사용량은 평균 3.2% , OS 구동을 제외한 메모리 사용량은 약 200MB로 측정되어 비교적 낮은 자원 사용을 통해 구현된 시스템임을 확인하였다.

Table 5. Measurement of System CPU Usage per Minute

Time	user	system	Usage	Step
11:24:00	2.6%	2.6%	0.7%	1) Before running the service
11:25:00	16.3%	12.0%	0.4%	
11:26:00	11.9%	5.6%	0.5%	
11:27:00	54.4%	160.4%	26.9%	2) VM OS Start
11:28:00	6.5%	40.9%	5.6%	2-1) API Gateway start
11:29:00	12.1%	12.5%	2.0%	
11:30:00	18.0%	18.1%	4.5%	3) Server Start
11:31:00	10.3%	12.8%	2.0%	
11:32:00	12.9%	18.6%	3.9%	4) Using Backend
11:33:00	12.7%	13.8%	2.3%	
11:34:00	11.6%	15.4%	3.4%	
11:35:00	7.6%	14.4%	2.8%	5)Using Frontend
11:36:00	3.6%	12.4%	2.0%	
11:37:00	12.4%	17.4%	5.9%	

Table 6. Measurement of System Memory Usage per Minute

Time	Used	Cache	Usage	Step
11:24:00	3.9GB	437.3MB	48.60%	1) Before running the Service
11:25:00	3.9GB	443.7MB	48.60%	
11:26:00	3.9GB	443.7MB	48.70%	
11:27:00	4.9GB	572.7MB	61.80%	2) VM OS start
11:28:00	5.1GB	827.9MB	63.20%	2-1) API Gateway start
11:29:00	5GB	857.3MB	63.10%	
11:30:00	5.1GB	800.5MB	63.60%	3) Server Start
11:31:00	5.1GB	818.3MB	63.50%	
11:32:00	5.1GB	823MB	63.40%	4) using backend
11:33:00	5.1GB	817.2MB	63.20%	
11:34:00	5.1GB	817.6MB	63.40%	
11:35:00	5.1GB	852.5MB	63.40%	5) using frontend
11:36:00	5.1GB	852MB	63.50%	
11:37:00	5.1GB	896MB	63.40%	

4.3 구현 시스템 분석

본 논문에서 구현된 통합 콘텐츠 관리 시스템은 CMS#1, CMS#2, 백엔드 영역, 프론트엔드 영역, API Gateway 로 구성된다. 기존 CMS들은 기존 서비스를 그대로 유지하고 있는 상태에 변경이 없기 때문에 기존 상태에 대한 서비스 유지를 그대로 준수한다.

백엔드 영역은 통합 인터페이스를 통해서 두 개의 기존 CMS의 콘텐츠를 모두 공통된 UI에서 관리할 수 있는 환경을 구현하였다. 이미 구축된 새로운 CMS 또는 새로운 서비스 API가 추가될 때, 콘텐츠를 선택할 수 있는 메뉴들이 추가되면서 선택된 메뉴에 따른 콘텐츠 리스트를 등록, 수정, 삭제, 보기를 통해 관리되도록 유지한다.

프론트엔드 영역은 역시 node.js를 사용한다. 각각 다른 CMS에서 호출된 데이터들을 취합하여 하나의 리스트로 부여주거나, CMS 별로 또는 CMS의 하위 테이블 리스트 별로 보여주도록 구현되었다. 프론트엔드는 서비스 기획에 따라 서비스 화면을 자주 변경해야 하므로 API를 이용한 뷰(View)를 구성하는데 있어, 프론트엔드 개발자의 비중이 높아질 것으로 보인다.

API Gateway는 백엔드와 프론트엔드의 API 를 분리하여 생성하였다. 백엔드는 서비스 요소에 따라 상세보기와 리스트를 분리하거나, 기존 CMS로부터 추가로 필요한 정보, 예를 들면 테이블 리스트 등을 가져오기 위해 API 를 추가로 생성한다. 이 구성은 Fig. 15와 같이 API 호출에 대한 로그 기록을 분리하고, 호출 빈도, 오류 빈도, 데이터 갱신 빈도 등을 고려하여, API 별로 캐시(Cache)를 적용할 수 있으므로 운영 간의 API 응답 효율을 높일 수 있다.

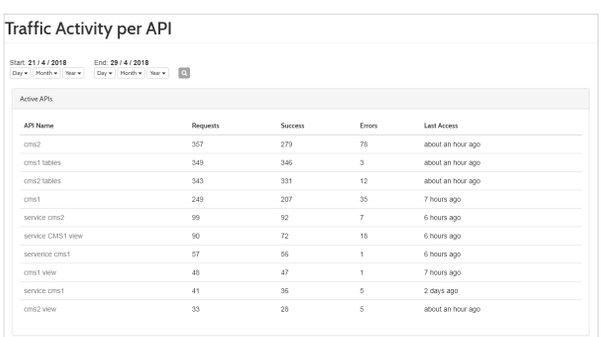

Fig. 15. Call and Error Statistic Per API

이 시스템에 다른 서비스를 추가될 경우, 서비스 단위의 서버의 Rest API에 대한 지원 및 연동을 위해 인증 수단이 필요하다. 그 다음은 아래와 같은 절차를 통해서 새로운 서비스를 추가할 수 있다.

① 추가될 서비스의 API 분석
② 인증 처리
③ API 포맷 또는 메시지 변경
④ API URL 생성
⑤ 통합 콘텐츠관리시스템에 추가되는 API 에 대한 반영

위의 순서를 통해서 새로운 서비스가 추가될 수 있다.

1) 기존 방식과의 비교분석

제안 시스템의 비교대상은 SOA의 미들웨어인 ESB의 역할이 마이크로서비스아키텍처의 API Gateway에서의 역할과 데이터를 연계하도록 하는 기능적 측면에서 매우 유사하다. 때문에 ESB를 이용한 통합 방식하고 구현된 시스템과 어떠한 차이가 있는지 비교하였다.

Table 7은 ESB와 API Gateway를 활용한 제안 시스템 간의 상세 비교 분석한 내용이다.

Table 7. Comparison ESB based SOA with Proposed System

Type	ESB based SOA	Proposed system
Interworking Method	Based on interworking using adapters to various protocols, interworking between common platforms	Based on interworking using lightweight protocol (REST/HTTP) between various platforms
Integration System Environment	Requires additional application server	Requires node.js and modules without additional application server
I/O Process	Multi-thread method with more overhead	Single thread method using event looping
Legacy system modified scope	Legacy system needs modification according to requirements.	Interworking without legacy system modification
Interworking extensibility	Within application server capacity	No Limit within REST API Interface

먼저, 가장 큰 차이로 ESB는 대부분 상용 소프트웨어로서 기업에서 라이선스에 대한 비용 부담이 있으며, 반드시 어플리케이션 서버를 활용해야 하는 반면, 제안 시스템은 오픈소스 또는 서비스 API Gateway를 이용한 구현이며, 통합 관리 시스템에 대한 플랫폼의 제약을 받지 않는다. 그리고, ESB에서는 데이터를 연동하기 위해 대표적으로 SOAP 프로토콜을 이용한 XML 포맷의 웹서비스를 가장 많이 활용되고 있다. 제안 시스템에서 사용된 프로토콜은 REST의 JSON 타입을 활용하고 있으므로, 동일 로직 간에 두 개의 인터페이스를 통해서 송수신 데이터의 RTT(Round Trip Time)를 측정했을 때, Fig. 16과 같은 차이가 발생한다. SOAP-XML은 기본적으로 주고받는 메시지들에 대한 오버헤드가 크기 때문에 실제 데이터에서 네트워크를 통해 주고 받는 데이터 사이즈가 Rest API 비해 크고, 지연시간도 3배~4배 길다[18-20].

ESB의 경우, 특정 연동 프로토콜의 종류가 다양하여, 연동 타깃 시스템이 지원 하는 어댑터를 사용해야 하며, 벤더가 이를 지원하지 않을 때는 개발에 관련된 비용이 증가할 수 있다. 또한 반대로 제안 시스템의 경우 연동 타깃 시스템이 Rest API를 지원하지 않을 경우, 이를 개발해야 하는 단점이 있다. 최근 연동 규격에는 HTTP나 Rest 방식을 선호하여 장기적으로는 제안 시스템 제약이 줄어들 것으로 예상한다.

연동 API를 이용한 방식은 구현된 시스템은 개발 과정에서 시스템 중단 없이 현재 시스템 상태에서 통합되었다. 때문에 개발 기간 동안 발생되는 시스템 중단으로 인한 영향을 받지 않고 진행할 수 있었다. 더불어 기존 운영자 측면에서는 개발된 통합 관리 시스템이나 기존 시스템 모두 병행하여 운영이 가능하다.

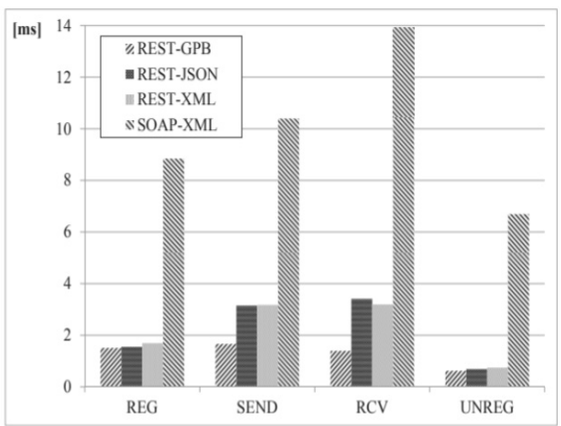

Fig. 16. REST and SOAP Response Time Measurements [20]

2) 향후 발전 방향

구현된 시스템은 모든 데이터의 호출을 API Gateway를 통해서 주고받는다. 이때 API Gateway의 Request를 받을 때, 요청 쿼터(Quota)를 제한할 경우, 연결된 하위 서비스에 대한 보호 및 성능을 유지할 수 있도록 한다. 때문에 과도한 요청이 발생에 대한 대비를 통해서 기존 시스템 영향을 최소화 할 수 있다. 또한, API 호출 제한을 통해 콘텐츠를 제공하는 API를 활용한 유료서비스 모델에서 해당 API에 대한 요청 당 비용을 정산하는 구조의 확장도 가능하다.

지속적으로 요청이 증가가 발생할 경우, API Gateway 의 클러스터링을 통해 부하를 분산하고, API Gateway에서 제공하는 응답 캐시를 활용하여, 반복적인 요청에 대한 서비스 API 영역까지의 요청을 감소시킴으로써 증가되는 트래픽에 대응할 수 있도록 한다. 또는 API Gateway에서 메시지호출 패턴(Message Exchange Pattern)을 통해서 동기 또는 비동기 호출의 패턴을 변경할 수 있도록 하여, 비동기 호출 받은 데이터 저장 요청 등을 받으면, 요청 시 응답신호를 보내고, 큐(Queue)와 연동하여 메시지를 임시로 저장하고 있다가 서비스 API 측의 성능이 요청을 받을 수 있는 상태가 되었을 때, 요청을 처리하도록 하여 호출 패턴으로 변경하여 성능에 대해 대응한다.

마이크로서비스 아키텍처에서는 특정 API 호출이 하위 서비스에 영향을 최소화시키기 위해서 Circuit Breaker의 역할이 필요하다. 때문에 Fig. 17과 같이 Webhook 를 이용하여 API 호출에 따른 이벤트 케이스를 정해놓고, 상황이 발생했을 경우에 대응한다.

쿼터가 종료되거나, Rate 범위를 초과하거나, 키가 종료되는 등 이벤트가 발생하는 이슈에 대해서 대체 서비스를 제공하거나, 연계 서비스 호출을 차단하거나 정책에 따라 다양한 대응을 설정할 수 있으므로, 서비스 제공자에게 문제가 장애에 따른 안정적인 대체 서비스를 제공할 수 있다.

Fig. 17. Webhook for API Event

5. 결 론

본 논문에서는 이 기종 간에 독립적인 콘텐츠 관리 시스템을 작은 서비스 단위로 구분하고, 서비스 간 연계하기 위한 API Gateway를 이용한 연계를 통해서 통합 콘텐츠 관리 시스템을 구현하였다. 이러한 방식은 시스템을 통합하는 여러 방식 중에 하나이지만, 새로운 어플리케이션을 추가할 경우에 대한 기존 시스템의 재활용을 통해서, 새로운 서비스 추가가 가능한 것을 확인하였다. 또한 통합 운영 환경을 통해서 각각의 서비스를 관리함으로써 통합 콘텐츠 관리 시스템으로서의 역할이 가능했다. 더불어 실험을 통해 구현된 시스템에서 요구되는 낮은 리소스 사용량은 하드웨어에 대한 부담비율을 낮추기 때문에, 대용량 서비스로 전환하더라도 비용적인 측면의 장점을 기대할 수 있다.

이번 연구를 통해서 마이크로서비스 아키텍처 기반에서는 API 서비스를 제공할 수 있다면, 통합 콘텐츠 관리 시스템을 통해서 관리가 가능하고, 통합을 위한 간단한 규칙을 통해서 새로운 서비스를 추가해나갈 수 있는 것을 확인하였다.

향후, 이 시스템에 대한 고도화를 고려한다면, WebHook를 활용한 연결된 서비스 시스템에 대한 보호 장치나 서비스 간 문제가 발생하였을 때에 장애 대응 처리, API 쿼터 제한을 통해서 유료 콘텐츠 서비스 모델로의 적용하거나, API Gateway의 클러스터링을 통해 대규모 서비스 확장이 가능한 모델로 확대해 보거나, 효율적인 통합 운영을 위한 통합 메타데이터 모델링에 대한 효과적인 표준안을 연구하는 것으로 전개가 필요할 것으로 생각한다.

References

[1] “International Digital Content Market Research 2017.12,” in *National IT Industry Promotion Agency*, pp.12, 2017.

[2] Chris Richardson, Monolithic Architecture pattern [Internet], http://microservices.io/patterns/monolithic.html.

[3] Chris Coyier, “What is a Headless CMS?” [Internet], https://css- tricks.com/what-is-a-headless-cms/.

[4] Buzachis Aris, Microservices vs Monolithic architectures [Internet], https://blog.buzachis-aris.com/2014/12/microservices-vs-monolithic-architectures/.

[5] graphCMS, GraphCMS - The GraphQL Headless CMS [Internet], https://graphcms.com/.

[6] butterCMS, ButterCMS: Headless CMS and Content API [Internet], https://buttercms.com/.

[7] Contentful, Contentful: Content Infrastructure for Digital Teams [Internet], https://www.contentful.com/.

[8] DirectUs, Directus: Open-Source Headless CMS and API [Internet], https://getdirectus.com/.

[9] Prismic.io, Prismic: Headless API CMS for both developers and marketers [Internet], https://prismic.io/.

[10] Disadvantages of Headless [Internet], https://en.wikipedia.org/wiki/Headless_CMS.

[11] Wordpress, WordPress.com: Create a free website or blog [Internet], https://wordpress.com.

[12] Drupal, Drupal - Open Source CMS | Drupal.org [Internet], https://www.drupal.org/.

[13] Neos, Neos CMS [Internet], https://www.neos.io/.

[14] Django, Django: The Web framework for perfectionists with deadlines [Internet], https://www.djangoproject.com/.

[15] Jang Seok Lee, Jeong Ki Hong, and Jeong Gwon Jee, “The strategy and approach of EAI for improving business agility,” *Communications of the Korean Institute of Information Scientists and Engineers*, Vol.22, No.7, pp.13-21, 2004.

[16] Yong Deok Kim, “A Design of Secure Key Exchange Protocol and Framework for SOA based ESB Environment,” Ph.D. dissertation, University of Soongsil, Seoul, Korea, pp.11-14, 2013.

[17] Goel and Anurag, “Enterprise integration EAI vs. SOA vs. ESB,” *Infosys Technologies White Paper 87*, 2006

[18] Juris Tihomirovs and Jānis Grabis, “Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics,” *Information Technology and Management Science*, Vol.19, Issue1, pp.92-97, 2016.

[19] Fatna Belqasmi, Jagdeep Singh, Suhib Bani melhem, and Roch H. Glitho, “SOAP-Based Web Services vs. RESTful Web Services for Multimedia Conferencing Applications: A Case Study,” *IEEE Internet Computing*, 2012.

[20] T. Aihkisalo and T. Paaso, “Latencies of Service Invocation and Processing of the REST and SOAP Web Service Interfaces,” *2012 IEEE 8th World Congress on Services*, Honolulu, HI, USA, pp.100-107. 2012.

[21] Chris Richardson, A pattern language for microservices [Internet], http://microservices.io/patterns/index.html.

[22] Wordpress REST API v2 Documentation [Internet], https://v2.wp-api.org.

[23] Tyk Open Source API Gateway, API Management Platform, Developer Portal and Analytics - Tyk - Tyk API Gateway and API Management [Internet], https://tyk.io.

[24] goLang, Go Web Examples: Templates [Internet], https://gowebexamples.com/templates/.

[25] Metricbeat, Metricbeat: Lightweight Shipper for Metrics | Elastic [Internet], https://www.elastic.co/kr/products/beats/metricbeat.

윤 경 식

https://orcid.org/0000-0001-8790-4747

e-mail : choonzang@gmail.com

2016년 한양사이버대학교 컴퓨터공학부 (학사)

2017년~현 재 숭실대학교 IT융합학과 석사과정

관심분야 : 콘텐츠관리시스템(CMS), 프론트엔드 개발, 데이터 분석

김 영 한

https://orcid.org/0000-0002-1066-4818

e-mail : younghak@ssu.ac.kr

1984년 서울대학교 전자공학과(공학사)

1986년 한국과학기술원 전기 및 전자공학 (공학석사)

1990년 한국과학기술원 전기 및 전자공학 (공학박사)

1994년~현 재 숭실대학교 전자정보공학부 교수

관심분야 : 모바일 네트워킹, 엣지 클라우드 시스템, ICN 및 센서 네트워킹