J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

RESTful API를 위한 SPA ViewModel 변환

SPA ViewModel Transformation for RESTful API

조 동 일1*
Dong-il Cho

요 약

SPA(Single-Page Application)는 RESTful API와 통신을 위해 데이터의 변환을 필요로 한다. BFF(Backend for Frontend) 패턴은 이 변환을 서버에서 처리하고 있으나 통신 횟수를 증가시키고 개발과 배포를 어렵게 하는 문제가 있다. 본 연구에서는 SPA의 ViewModel과 RESTful API의 모델을 SPA에서 직접 매핑하는 아키텍처를 제안한다. 제안한 아키텍처는 RESTful API의 문서 모델인 OpenAPI 사양을 이용하여 RESTful API 모델과 ViewModel 간의 매핑 모델을 자동 생성한다. SPA의 통신 컴포넌트는 생성된 모델을 이용하여 RESTful API 데이터와 ViewModel을 자동 변환한다. 사례 연구를 통해 기존 BFF 방식과 비교한 결과 제안한 아키텍처는 BFF에 비해 높은 개발 생산성을 보였고 부하테스트 결과 BFF에 비해 약 6 %이상 낮은 서버 CPU 점유율을 기록하였다.

☞ 주제어 : RESTful APIs, Single-Page Application, ViewModel, Model Mapping

ABSTRACT

Single-Page Application(SPA) requires data transformation for communication with RESTful API. The Backend for Frontend(BFF) pattern handles this transformation in the server, but there is some problem that increases the number of communication and makes development and distribution difficult. In this study, we propose an architecture that maps the ViewModel of SPA and the model of RESTful API directly in SPA. The proposed architecture automatically generates a mapping model between the RESTful API model and the ViewModel using the OpenAPI specification, which is the document model of the RESTful API. The data transfer component of SPA automatically converts RESTful API data and ViewModel using the created model. As a result of comparison with the existing BFF method through case study, the proposed architecture showed higher development productivity than BFF, and as a result of load tests, it recorded about 6% lower server CPU occupancy compared to BFF.

☞ keyword : RESTful APIs, Single-Page Application, ViewModel, Model Mapping

1. 서 론

RESTful API를 이용하여 기업의 필수 프로세스를 지원하는 SPA(Single-Page Application)는 단일 서버와 통신하는 방식에서 마이크로서비스 아키텍처 기반에서 여러 서버의 서비스를 사용하는 방식으로 변화하고 있다[1].

RESTful API에서 제공되는 정보가 SPA에서 요구되는 모델형식과 완전히 맞지 않을 수 있기 때문에 RESTful API의 데이터는 SPA에 맞게 변환되어야 한다[1]. 과거 이 변환 프로세스는 SPA 내부에서 수행되었으며 SPA의 복잡성을 증가시키고 민첩성과 변경 가능성을 손상시키는 원인이 되었다. BFF(Backend for Frontend)는 이런 문제를 개선하기 위해 변환 프로세스를 서버에서 처리하여 클라이언트에서는 단일 서비스만 관리하면 되는 편의성을 제공한다[1]. 하지만 BFF는 SPA에 종속적인 변환 프로세스를 서버에서 처리하여 서버와 클라이언트 간의 통신 횟수를 증가시키고 개발을 어렵게 하며 변경에 유연하지 못하다.

본 연구는 OAS(OpenAPI Specification)를 이용하여 SPA에서 RESTful API 모델과 SPA의 ViewModel을 직접 매핑하는 DMM(Direct Model Mapper) 아키텍처를 제안한다. DMM을 위해 본 연구에서는 ViewModel Map의 생성과 통신 컴포넌트의 자동변환을 지원하기 위한 일반화된 ViewModel을 제시하였고, OAS를 이용한 모델 간 매핑 방법을 설계하였다. RESTful API와 통신하는 SPA의 통신 컴포넌트는 ViewModel의 데이터를 매핑 모델을 이용하여 RESTful API 모델로 자동 변환한다.

RESTful API 모델과 SPA의 ViewModel 간의 변환을 개발영역에서 분리하고 RESTful API 모델을 그대로 사용할 수 있는 DMM의 특징은 BFF에 비하여 다음과 같은 이

1 Division of R&D, TOMATOSYSTEM, Seoul, 06104, Korea.
* Corresponding author (chodongil@yahoo.co.kr)
[Received 20 October 2022, Reviewed 08 December 2022(R2 01 February 2023), Accepted 15 February 2023]

점을 제공한다.

- 개발 복잡성을 감소시킴.
- 개발 생산성을 증대시킴.
- 변경에 유연한 구조를 제공.
- RESTful API Server와 SPA 간의 통신 횟수를 감소시켜 서버의 부하를 줄임.

본 연구에서는 DMM을 설계 및 구현하였고 사례연구를 통해 기존 BFF 방식으로 구현된 애플리케이션과 비교하였다.

2. 관련 연구

2.1 ViewModel

Web Frontend 개발에서 최근 인기 있는 디자인 패턴은 MVVM(Model-View-ViewModel)이다[2]. MVVM은 주로 MVC 패턴을 기반으로 하며 실제 MVC와는 ViewModel의 존재 여부로 구분된다[3]. ViewModel은 UI(User Interface)의 상태 즉 위젯에 표시되는 것과 밀접하게 관련된 데이터를 유지하며 모델에 포함된 데이터의 일부를 반영할 수 있다[2]. ViewModel은 공용속성 및 명령을 노출하는 뷰의 추상화 개념으로 MVC 패턴의 컨트롤러 또는 MVP(Model-View-Presenter) 패턴의 Presenter 대신 ViewModel의 View와 바인딩된 속성 간의 통신을 자동화하는 바인더가 있다[4].

ViewModel을 사용하면 View의 데이터를 관리하는 계층과 View 계층을 분리하고 View의 동작에 따라 변경된 데이터를 ViewModel에 쉽게 반영할 수 있으며 이와 관련해서는 이미 다양한 연구가 진행되었고 활용되고 있다.

ViewModel은 View에 종속적인 데이터 모델로 RESTful API에서 수신한 데이터와 일치하기 않기 때문에 SPA에서 수신한 데이터는 ViewModel로 변환되어야 하는데, 일반적인 사용 사례에 대한 제한 없이 API 요청 후 자동 변환 규칙을 모델링하는 것은 불가능하기 때문에 개발자가 직접 변환 규칙을 지정해야 한다[2].

2.2 Backend for Frontend(BFF)

BFF는 클라이언트의 요청에 대한 게이트웨이 역할을 하는 마이크로서비스이다[5]. [6]에서 소개된 BFF 패턴은 마이크로서비스에서 클라이언트 유형의 요구사항에 맞게 조정할 수 있는 고유성을 허용하는 채널별 서비스 인터페이스를 표현하는 방법을 제공한다. BFF는 클라이언트를 위한 단일 API 역할을 하는 BFF를 빌드하고 클라이언트 유형에 따라 맞춤화된 API가 있는 다양한 유형의 클라이언트에 대해 서로 다른 BFF를 구현하여 적용한다.

BFF는 모바일 애플리케이션 또는 SPA와 같이 동적 Frontend가 있는 마이크로서비스 아키텍처를 구축하는데 핵심적인 부분으로 Frontend에 필요한 필터링 및 변환과정을 포함한다[7].

예를 들어 SPA는 화면 흐름을 나타내는데 필요한 특정 정보의 집합을 반환하기 위해 개별 비즈니스 마이크로서비스에 대한 여러 호출의 오케스트레이션을 요구하거나 반대로 한 호출 결과의 필터링을 요구할 수 있다.

BFF에서는 대부분의 경우 클라이언트 애플리케이션과 BFF를 빌드하는 책임이 동일한 팀에 있다[7]. 따라서 Frontend 개발자가 Backend 개발의 일부에도 참여해야 하며 개발 및 테스트를 위해 종단의 코드를 동시에 제어하고 동기화해야 하기 때문에 개발과 배포를 어렵게 한다.

2.3 OpenAPI Specification(OAS)

현대 RESTful API는 일반적으로 OAS를 사용하여 수행할 수 있는 정확한 작업과 입력 및 출력 매개변수의 형식을 명시적으로 설명하고 문서화한다[8, 9]. OAS는 소스코드 또는 문서에 접근하거나 네트워크 트래픽 검사를 통해 접근하지 않고도 인간과 컴퓨터가 서비스 기능을 발견하고 이해할 수 있도록 하는 RESTful API에 대한 표준 인터페이스를 명세한다. 적절하게 명세되면 소비자는 최소한의 로직 구현으로 원격서비스를 이해하고 상호 작용할 수 있다[10].

OAS를 이용한 도구들은 주어진 인터페이스 파일에서 코드, 문서 및 테스트 사례를 생성할 수 있고 다양한 연구에서 동적 개체, 정적 디렉터리, 예제 코드, 임의 값의 지식기반 사용 및 테스트 모델 생성 등에 활용할 수 있음을 제시하였다[8, 9, 11, 12, 13, 14].

본 연구에서는 OAS를 이용하여 SPA ViewModel과 RESTful API의 서비스 모델의 매핑모델을 자동 생성한다.

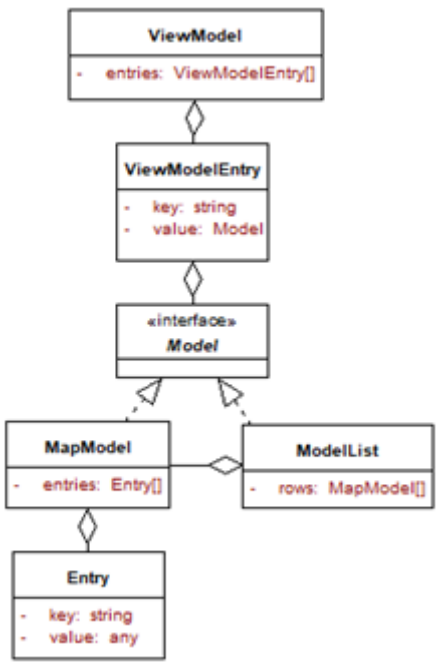
3. Direct Model Mapper

3.1 ViewModel의 일반화

SPA에서 ViewModel은 UI에 종속적인 데이터 모델로

정의된다. 자유형식의 ViewModel은 예측이 어렵고 공통적인 기능구현에 많은 제약을 준다.

UI를 표현하기 위한 Model은 key:value 형식의 Map 구조와 Map이 반복되는 배열 구조가 일반적이다. 이 두 형식으로 ViewModel을 일반화하면 RESTful API Model과 ViewModel 간의 형식 매핑 구조를 보다 단순하게 표준화할 수 있다.



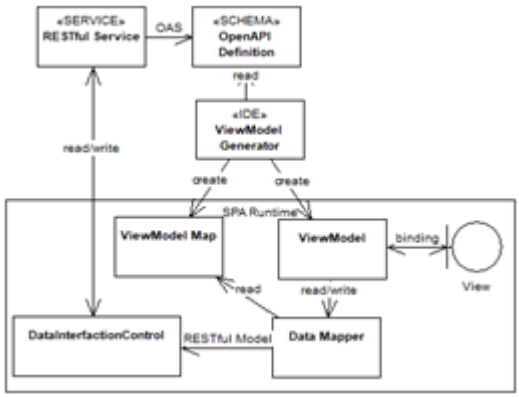
(그림 1) ViewModel Class Diagram
(Figure 1) ViewModel Class Diagram

본 연구에서는 그림 1과 같이 ViewModel을 MapModel과 ModelList의 집합으로 정의한다. MapModel은 key:value 쌍의 값을 가지는 Map 구조의 Model이고 여러 행을 가지는 ModelList는 MapModel을 배열로 관리한다. 이들은 Model 인터페이스를 구현하고 있으며 각각의 Model은 ViewModel 객체에 key 값으로 색인할 수 있도록 저장된다. SPA에서는 ViewModel에서 key 값으로 원하는 Model 객체에 접근할 수 있다.

3.2 아키텍처 개요

본 연구에서는 RESTful API는 OAS에 맞춰 명세 되어 있다고 가정한다. 일반적으로 Java 언어에서는 소스코드에 Annotation을 추가하여 명세 되고 Swagger는 이 Annotation을 읽어 문서화한다.

그림 2와 같이 ViewModel Generator는 이렇게 문서화된 명세를 읽어 ViewModel Map과 ViewModel을 생성한다. ViewModel은 View를 통해 표현되는 데이터를 저장한 객체이고 ViewModel Map은 RESTful API에 전달할 요청 및 응답 데이터와 ViewModel 간의 연결 정보를 정의한다.



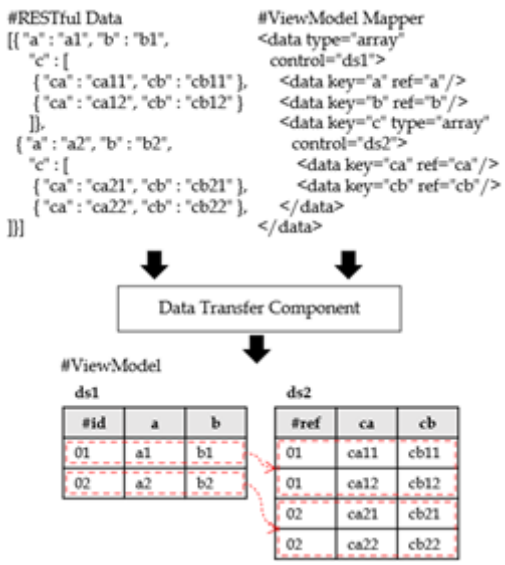
(그림 2) DMM 개요
(Figure 2) DMM Overview

View는 ViewModel의 데이터를 표현하고 필요에 따라 RESTful API와 통신을 요청하는 이벤트를 발생시킨다. 서버 통신 이벤트가 발생하면 DataMapper는 ViewModel Map과 ViewModel을 조회하여 서버로 전송할 요청데이터를 생성한 후 RESTful API 서버로 전송하고 서버에서 받은 응답데이터는 ViewModel Map의 매핑정보를 이용하여 ViewModel에 반영한다.

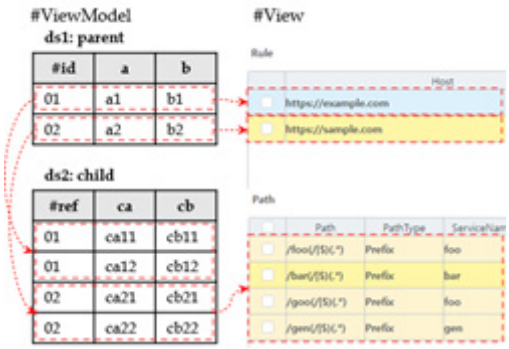
3.3 모델 매핑

그림 3과 같이 ViewModel Map은 RESTful API의 요청/응답 데이터와 동일한 계층구조를 가진다. 각 노드의 계층관계는 통신 데이터의 계층관계와 매핑되며 ViewModel의 키 값과 MapModel 내의 키 값이 정의된다.

계층을 가지는 배열은 Parent-Child 관계를 가지는 ModelList로 표현된다. 그림 4와 같이 Child ModelList는 Parent의 모든 행의 하위 데이터를 모두 저장한다. Child ModelList의 행은 Parent ModelList의 종속관계 행과 참조키를 저장하여 관계를 유지한다. Child ModelList는 Parent 행을 조건으로 필터링하여 종속관계 행 목록을 노출시킬 수 있다.



(그림 3) 모델 매핑 프로세스
(Figure 3) Model Mapping Process



(그림 4) ViewModel 계층관계
(Figure 4) ViewModel hierachical relationship

다음은 Child ModelList의 데이터 그룹을 전환하는 예시 코드이다.

```
grid1.addEventListener("selection-change", (e) => {
vm2.switchDataGroup(e.selectedRow.rowId);
grid2.redraw();
});
```

코드에서는 ViewModel vm2를 선택된 상위행의 rowId로 데이터 그룹을 전환한 후 vm2와 바인딩된 View인 grid2를 새로 그려 현재 노출해야 하는 데이터를 보여준다. View에서 데이터를 추가 또는 제거했을 때 현재 활성화된 데이터 그룹의 데이터가 추가되거나 제거된다.

4. 적용 및 평가

본 연구에서는 제안한 아키텍처를 검증하기 위해 Kubernetes SPA 구축에 적용하여 검증하였다.

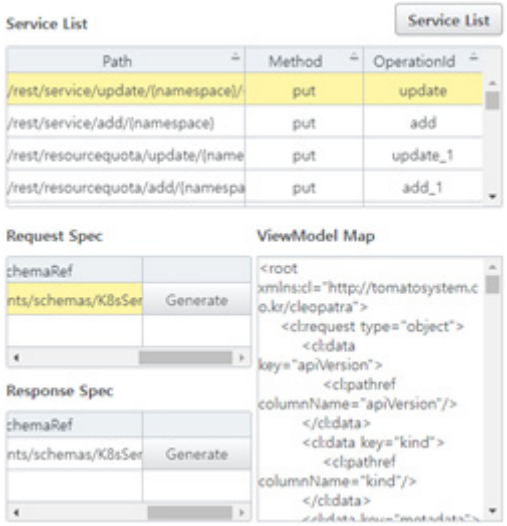
4.1 적용 사례

사례 프로젝트의 서버는 Java 언어로 구축되었으며 구성환경은 다음과 같다.

○ Kunernetes
- Ubuntu linux 20.04.4
- Kubernetes API V1
- Docker 20.10.17

○ RESTful Server:
- Kubernetes java client 15.0.1
- spring boot 2.7.3
- swagger-ui 4.14.0
- springdoc-openapi-ui 1.6.11

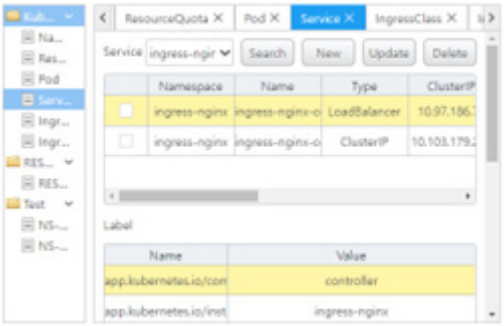
○ SPA 구축환경
- ECMAScript6
- SPA ui-framework 자체 개발



(그림 5) ViewModel 생성 마법사
(Figure 5) ViewModel generation wizard

그림 5는 ViewModel 생성 마법사의 WEB UI이다. 이 UI을 통해 OAS을 공급하는 Swagger 서버에 접속하여 생성하고자 하는 서비스 목록을 조회하고 원하는 서비스의 요청 또는 응답 항목의 "Generate" 버튼을 클릭해 ViewModel Map을 생성할 수 있다.

DMM은 이 Map 데이터를 이용하여 ViewModel을 생성하거나 기존 ViewModel을 새로 매핑된 정보를 반영하여 갱신한다. 완성된 매핑정보는 데이터 전송모듈에서 ViewModel과 RESTful API 모델 간의 데이터 자동 변환에 활용되고 ViewModel은 View의 각종 UI요소에 바인딩된다.



(그림 6) Kubernetes Service SPA 사례 연구
(Figure 6) Kubernetes Service SPA case study

그림 6은 DMM을 이용해 구현한 Kubernetes Service SPA 이다. 화면 좌측에는 Kubernetes의 각종 서비스에 접근할 수 있는 메뉴가 있고 이 메뉴를 통해 각 서비스 화면으로 진입할 수 있다. 서비스 화면에서는 방대한 Kubernetes 서비스 정보 중 사용자에게 의미있는 요소만 식별하여 화면에 표현하고 그 외의 정보는 숨긴다. 추가 정보를 노출해야 할 경우 이미 SPA에서 관련 정보를 ViewModel로 가지고 있기 때문에 표현할 방법만 결정되면 화면에 추가할 수 있다.

4.2 평가

본 연구에서는 DMM으로 구현한 Kubernetes Service SPA를 BFF 방식으로 구현하여 개발편의성과 운영성능을 상호 비교하였다.

구현 결과 BFF 방식은 SPA에 기능이 추가되거나 변경될 때 마다 서버의 모델 변환모듈을 변경해주어야 했다. 사례에서는 BFF를 Java로 구현하여 SPA 개발자가 ECMA Script와 Java 두 가지 언어를 제어해야 했고 변경이 발생하면 서버를 재기동해야 했다. [1] 연구에서와 같이 BFF를 NodeJS 기반으로 구현하면 SPA 개발과 같은 ECMA Script를 이용할 수 있어 여러 언어를 동시에 사용해야 하는 문제는 완화할 수 있으나 [1] 에서도 언급된 바와 같이 변환 과정은 자동화 할 수 없기 때문에 SPA 개발자가 서버 코드에도 개입해야 하는 문제는 여전히 남게 된다.

DMM은 이런 변환 규칙이 SPA 개발과 분리된 ViewModel Mapper에 지정되고 별도의 컴파일이나 서버 재구동 없이 SPA 개발 및 변경과 생명주기를 같이 하기 때문에 변경에 유연하며 개발 편의성이 증대되고 개발생산성에 유리하다는 것을 확인할 수 있었다.

BFF와 DMM 간의 운영 성능을 비교하기 위해 단순 조회 화면에 대해 부하 테스트를 진행하였다. 테스트 시나리오는 다음과 같다.

○ BFF
1) 화면 진입시 Kubernetes Namespace 목록 조회
2) 조회된 Namespace 중 특정 Namespace의 Service 목록 조회
3) 조회된 Service 중 특정 Service의 세부 내역 조회

○ DMM
1) 화면 진입시 Kubernetes Namespace 목록 조회
2) 조회된 Namespace 중 특정 Namespace의 세부내역을 포함하는 Service 목록 조회

DMM은 Service 목록 조회 시 Service의 상세정보를 포함하여 제공하는 Kubernetes API를 그대로 사용할 수 있기 때문에 BFF와 달리 한 서비스 호출로 Service의 모든 정보를 획득하여 활용할 수 있다. 반면 BFF는 화면에서 필요한 정보를 서버에서 가공하여 제공하기 때문에 동일한 서비스 제공에 상세정보를 조회하는 시나리오가 추가되었다.

부하테스트 환경은 다음과 같다.

○ Kubernetes API & RESTful API Server
- Intel(R) i7-8550U / 16GB RAM
- Window10 – WSL2 Ubuntu 22.04 LTS(Processor 4, Memory 8GB)
- Kubernetes 1.22, Docker 19.03
- Kubernetes java client 15.0.1
- Java SDK 17

○ Test Client
- Intel(R) Core(TM) i7-8550U / 16GB RAM
- Windows10
- JMeter 5.4.3

부하는 10개의 Thread, Ramp-up Period를 1초로 설정하고 위 시나리오를 100회 반복하면서 서버의 CPU 사용량을 모니터링 했다.

(표 1) CPU 시간(%)
(Table 1) Total CPU Time(%)

	BFF	DMM
Max	31 %	28 %
Min	25 %	17 %
Avg	28.75 %	22.33 %

모니터링 결과 표 1과 같이 BFF는 평균 28.75 %의 CPU 점유율을 기록한 반면 DMM은 22.33 %를 기록하여 DMM의 CPU 점유율이 약 6 % 적음을 확인할 수 있었다.

전체 테스트 시간은 BFF의 경우 약 4초 소요된 반면 DMM은 2초 이내에 테스트가 종료되었고 이는 DMM의 시나리오가 상대적으로 적어 그 만큼 서버에 부하를 적게 준다는 것을 확인할 수 있다.

5. 결 론

SPA는 현대적인 웹 애플리케이션 개발의 최신 흐름으로 다양한 RESTful API와 연동하여 서비스를 제공하고 있는데 RESTful API에서 제공하는 요청 및 응답데이터는 SPA의 요구사항과 맞지 않는 문제가 있다.

이 문제를 해결하기 위해 BFF는 서버에서 SPA에 맞게 데이터를 변환하는 방식을 제시한다. 이런 BFF의 방식은 Frontend 개발자가 개발 및 테스트를 위해 서버와 클라이언트의 코드를 동시에 제어하고 동기화해야 하기 때문에 개발과 배포를 어렵게 하는 문제점이 있다.

본 연구에서는 RESTful API 모델을 SPA의 ViewModel과 별도의 데이터변환 없이 직접 매핑하는 DMM을 제안하였다. DMM은 개발툴에서 OAS를 이용해 RESTful API 모델과 SPA의 ViewModel 간의 매핑정보를 SPA 개발과정에서 정의한다. SPA 개발자는 생성된 ViewModel을 이용하여 모델 간의 매핑을 고려하지 않고 SPA를 개발할 수 있다. RESTful API 서버와 통신은 SPA 통신 컴포넌트가 매핑정보와 ViewModel을 이용하여 RESTful API의 사양에 맞게 데이터를 변환하고 응답데이터를 해석하여 ViewModel에 매핑한다.

사례 연구 결과 제안한 DMM은 BFF에 비해 높은 개발 생산성과 변경 유연성을 제공하였다. 운영 성능 비교에서 제안한 DMM은 BFF에 비해 약 6 % 낮은 CPU 점유율을 기록하였고 더 적은 수의 서버 호출을 요구하여 서버 부담을 감소시키는 것을 확인할 수 있었다.

참고문헌(Reference)

[1] HM Adrian, N Scholz, and F Matthes. "A Model-driven Approach for Generating RESTful Web Services in Single-Page Applications.", Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, pp.480-487, 2018. https://doi.org/10.5220/0006608204800487

[2] J Freeman, J Järvi and G Foust. "HotDrink: a library for web user interfaces." ACM SIGPLAN Notices Vol 48, No.3, pp.80-83, 2012. https://doi.org/10.1145/2480361.2371413

[3] J Munro, "Knockout. js: building dynamic client-side web applications", O'Reilly Media, Inc., pp.3, 2014.

[4] Model – view – viewmodel https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

[5] A Pavlenko, N Askarbekuly, S Megha and M Mazzara, "Micro-frontends: application of microservices to web front-ends", Journal of Internet Services and Information Security (JISIS), Vol.10, No.2, pp. 49-66, 2020. http://doi.org/10.22667/JISIS.2020.05.31.049

[6] S Newman, "Building Microservices", pp.41-47, O'Reilly Media, 2015.

[7] K Brown and B Woolf, "Implementation patterns for microservices architectures", Proceedings of the 23rd Conference on Pattern Languages of ProgramsOctober, No.7, pp.1-35, 2016. https://dl.acm.org/doi/10.5555/3158161.3158170

[8] V Atlidakis, P Godefroid and M Polishchuk. "RESTler: Stateful REST API Fuzzing", 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp.748-758, 2019. https://doi.org/10.1109/ICSE.2019.00083

[9] E Viglianisi, M Dallago and M Ceccato, "RESTTESTGEN: Automated Black-Box Testing of RESTful APIs", 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp.142-152, 2020. https://doi.org/10.1109/ICST46399.2020.00024

[10] https://swagger.io/specification/

[11] H Wu, L Xu, X Niu and C Nie, "Combinatorial Testing of RESTful APIs." ACM/IEEE International Conference on Software Engineering (ICSE), pp.426-437, 2022. https://doi.org/10.1145/3510003.3510151

[12] JC Alonso, A Martin-Lopez, S Segura, JM García and A Ruiz-Cortés, "Automated generation of realistic test inputs for web APIs", Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.1666-1668, 2021. https://doi.org/10.1145/3468264.3473491

[13] P Godefroid, BY Huang and M Polishchuk, "Intelligent REST API data fuzzing", Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.725-736, 2020. https://doi.org/10.1145/3368089.3409719

[14] S Karlsson, A Čaušević and D Sundmark, "QuickREST: Property-based test generation of OpenAPI-described RESTful APIs.", 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp.131-141, 2020. https://doi.org/10.1109/ICST46399.2020.00023

◐ 저 자 소 개 ◑

조 동 일(Dong-il Cho)
2003년 수원대학교 기계공학과(공학사)
2008년 숭실대학교 정보과학 대학원 소프트웨어공학과(공학석사)
2012년 숭실대학교 대학원 컴퓨터학과(공학박사)
2003년~현재 (주)토마토시스템 기술연구소 수석연구원
관심분야 : 웹 애플리케이션 아키텍처, 소프트웨어 아키텍처, 미들웨어, WEB UI/UX, etc.
E-mail : chodongil@yahoo.co.kr