논문 2018-13-38

오프라인 우선 정책에 의한 멀티 디바이스의 실시간 데이터 동기화 구현

(An Implementation of Real Time Data Synchronization of Multiple Devices by Offline-first Strategy)

이 대 명, 김 은 후, 주 문 갑*
(Dae-Myoung Lee, Eun-hoo Kim, Moon G. Joo)

Abstract : Offline-first strategy is that it allows data to be saved while offline, and when connected online, data is synchronized to ensure that all devices have the same data. Multi-device is a term that shares data through synchronization on various platforms on Android, ios, etc. First, all of the data is stored in the local repository like SQLite and then on the server via HTTP communication. Then, the synchronization is completed by receiving the changed data from the server and storing it in the local repository at the time of the synchronization, and sending the changes to the server from the client. We proposed and implemented a database structure, APIs, and a illustrative application running on PC and Android phone.

Keywords : Offline-first, Multi device, Synchronization

Ⅰ. 서 론

휴대용 스마트 기기의 발달에 따라 사용자가 기존의 컴퓨터 (인터넷)에서 하던 일들을 2가지 이상의 다른 플랫폼 상의 스마트 기기에서 작업하는 것이 가능하게 되었다. 또한 휴대용 스마트 기기는 특성상 필수적으로 무선 인터넷을 이용하고 있다. 그런데, 무선 인터넷 환경은 24시간 인터넷 연결이 잘 되어 있다고 보장하기가 어렵다.

이러한 시대적 흐름에 맞춰 오프라인 상태에서 저장된 데이터들이 나중에 온라인 연결이 되었을 때, 다른 플랫폼의 여러 가지 기기에서 동기화가 되는 기능이 필요하다. 온라인 상태일 경우에는 변경되는 데이터들이 당연히 실시간으로 다른 여러 디바이스에 동기화되는 기능들이 필요하게 되었다.

기술적으로 오프라인 상태에서 변경된 데이터를 우선으로 데이터를 연동된 다른 장치에서 저장하고 사용하려면, 모든 데이터를 클라이언트 기기에서 접근이 가능해야 하고, 이를 위해 로컬 데이터베이스를 이용해야 한다. 그리고 다른 플랫폼의 여러 디바이스에서 모든 데이터에 접근을 하려면 서버에 모든 데이터가 있어야 하므로 클라이언트에서 데이터를 저장한 후 서버로 데이터를 보내 서버의 데이터베이스와 동기화를 할 수 있도록 해야 한다.

위의 기능들을 이용하는 서비스의 예로서 에버노트, WunderList [1], Todoist [2], Google Calender [3] 등과 같은 유틸리티성 서비스가 있다. 위 서비스들의 가장 큰 특징은, 그림1의 WunderList의 광고에서 볼 수 있듯이, 어떤 플랫폼의 어떠한 기기에서도 서비스를 이용할 수 있다는 점이다. 또한 명시하지는 않지만 오프라인 상태에서도 아무런 제약 없이 서비스를 이용할 수 있으며 인터넷 연결이 되었을 때 자동으로 동기화되어 어느 기기에서든 확인할 수 있다. 이러한 기능들을 제공하기 위해서는 오프라인 우선으로 데이터를 저장하고 다양한 플랫폼의 기기에서 동일한 데이터를 공유할 수 있도록 데이터 동기화 기능을 구현해야 한다.

데이터 동기화 관련하여 학술지 등을 찾아보면, 데이터 동기화 표준인 SyncML 기반의 자료 동기화 [4] 또는 데이터 동기화를 위한 특허 [5] 등을 확인

*Corresponding Author (gabi@pknu.ac.kr)
Received: July. 11 2018, Revised: Oct. 4 2018, Accepted: Oct. 31 2018.
D. Lee, E. Kim, M. Joo: Pukyong National University

 2018 Dec. 13(6) 329-335
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2018.13.6.329

그림 1. 다양한 기기의 데이터 동기화 기능 지원 (WunderList)
Fig. 1 Data synchronization among several devices (WunderList)

할 수 있다. 하지만 오래되어 쓰이지 않는 규격이 되었고, 주 내용은 두 디바이스 간의 데이터 전송 효율에 초점이 맞추어져 있고 멀티 디바이스 환경에서는 잘 고려되지 않았다. 본 논문은 멀티 디바이스 환경과 오프라인 환경 모두를 지원하면서, 데이터 동기화를 좀 더 유연하게 구현하는 것을 목표로 한다.

논문의 구성은 다음과 같다. 2장에서는 전체적인 시스템 구성과 로직 설계 과정을 다루며, 세부적으로는 클라이언트, 서버 각각의 구체적인 설계와 로직 구현 예시를 다룬다. 3장에서는 실제로 구현하고 적용 및 테스트했던 시나리오와 환경에 대한 내용을 다루고, 마지막으로 결론과 향후 목표에 대한 이야기를 다룬다.

II. 본 론

Todoist 또는 구글캘린더의 동기화 API [6] 등을 살펴 보면, 먼저 전체 동기화를 통해 서버와 클라이언트 상태를 완전히 동기화 하고 동기화 토큰을 얻는다. 이후 토큰을 이용하여 변경 사항들을 받아오면서 지속적으로 업데이트 하는 방식으로 데이터 동기화를 하고 있다.

1. 시스템 구성

전체적으로 동기화 시스템을 그려 보면, 그림2와 그림3과 같이 클라이언트에서 서버의 변경사항 전체를 받아와서 처리를 한 후, 클라이언트의 변경사항을 서버로 업로드하여 동기화를 완료한다.

위 과정과 같이 서버와 클라이언트 간의 데이터를 동기화하기 위해서 데이터베이스에 몇 가지 컬럼을 추가한다. 그리고 그 컬럼들을 이용하여 서버와 클라이언트 각각 동기화를 위한 데이터 처리 로직을 설계하여 구현한다.

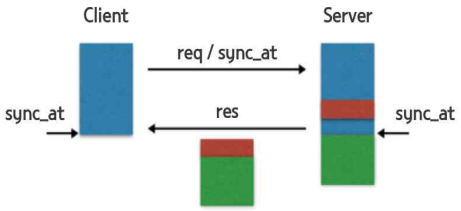

그림 2. sync_at 이후의 서버의 변경 사항 반영
Fig. 2 Modification of changes of the server after sync_at service

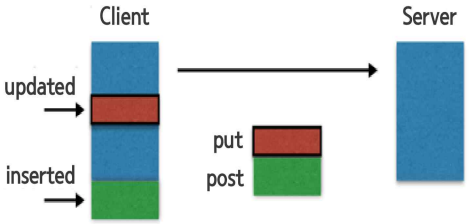

그림 3. 클라이언트의 변경사항 전송
Fig. 3 Sending changes of client to server

2. 동기화를 위한 데이터베이스 구조

동기화가 진행될 데이터베이스의 테이블에는 서비스를 위한 컬럼들 (id, title, content, updated_at 등등) 이외에 두 개의 컬럼 server_id와 dirty_flag를 추가적으로 생성하여 그림4와 같이 구성하였다. 첫 번째로 server_id는 서버 측 데이터베이스에서 관리되는 값이다. 이는 컬럼이 추가됨에 따라 항상 바뀌어 유일한 값을 가진다. 이 값을 이용하여 여러 클라이언트의 데이터가 동기화가 된 적이 있는지 없는지 등등을 확인할 수 있다.

두 번째로 클라이언트의 로컬 저장소에 해당 데이터가 변경 또는 추가 되어 서버와 동기화가 필요한 데이터인지를 체크 할 boolean 값이 필요하다. 이를 dirty_flag라고 표현하였다. 클라이언트에서 데이터가 변경이 되거나 추가가 되면, 해당 데이터의 dirty_flag를 true로 세팅 한다. 동기화가 완료된 후에는 dirty_flag를 false로 바꾸어 동기화가 완료되었다는 표시를 한다.

그리고 별도로 클라이언트에서는 동기화 완료 시점을 저장해야 하고, 이는 각 디바이스에 sync_at이라는 이름으로 저장한다. 클라이언트에서는 이 시점을 기준으로 서버에 변경된 데이터를 요청하여 동기화를 진행한다.

그림4. 데이터베이스 구조 예시
Fig 4. Database structure

```
[GET] http://host.io/api/v1/sync_local?
(query) sync_at : long
```

그림 5. 서버의 변경사항을 받기위한 API
Fig. 5 API for receiving changes of server

3. 동기화 기본 전체 로직 [7]

일반적으로 동기화는 클라이언트 측면에서 해당 서비스를 이용하면서 로컬 저장소에 데이터가 저장 혹은 업데이트가 되면 서버로 동기화 요청을 하면서 진행이 된다. 이때 요청이 클라이언트가 HTTP 통신으로 서버에 변경사항을 모두 전송해달라는 요청이다. 그리고 서버에서 변경 사항을 내려 주기 위해서는 기준이 있어야 하므로, HTTP 요청의 파라미터로 time stamp를 함께 보낸다. 이 time stamp는 클라이언트가 동기화를 완료한 시점이고, 서버는 그 시점 이후에 변경된 데이터들을 모두 전송한다. 이 파라미터의 이름은 sync_at으로 명명했다.

그림5는 해당 API의 예시이며, GET 요청의 query로 sync_at에 time stamp를 넣어서 요청한다.

클라이언트에서는 위 API의 응답으로 받아온 데이터들을 모두 로컬 저장소에 저장 또는 업데이트를 하면 된다. 이 과정은 앞으로 설계한 로직에 따라서 진행이 된다. 최초 동기화 요청 시에는 sync_at의 time stamp 값이 0 일 것이고, 서버의 모든 데이터가 전송된다.

로컬의 저장소 동기화가 모두 끝나면, 이제 클라이언트에서 변경된 데이터들을 서버 쪽으로 모두 전송한다. 이때 dirty_flag를 통해 변경된 사항들을 확인한다.

이 과정까지 끝나면 동기화 완료 시점으로 디바이스에 저장된 sync_at값을 현재 time stamp로 수정해준다.

전체 동기화 진행시 주의해야 할 사항으로, 여러 테이블을 가진 데이터베이스 동기화 설계시 테이블 간의 관계를 생각해서 상의 모델에서 하위 모델 순서로 진행하여야 한다.

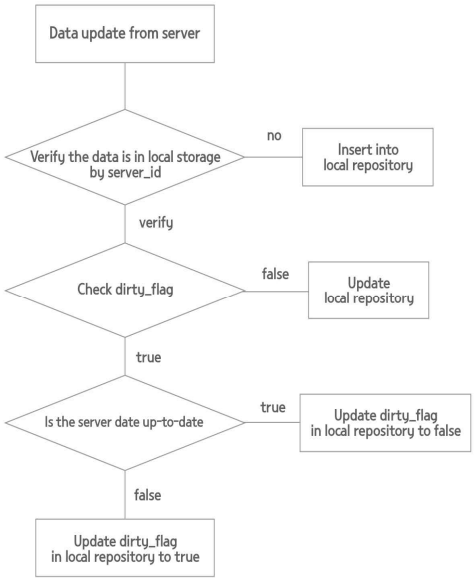

그림 6. 로컬 저장소 동기화 과정 블록다이어그램
Fig. 6 Block diagram of synchronization of local repository

4. 클라이언트 동기화

그림6은 클라이언트 동기화 과정을 블록다이어그램으로 도식화한 것이다. 동기화를 위한 첫 요청으로 클라이언트의 동기화 완료 시점 이후에 변경된 모든 데이터들을 서버로부터 받는다. 이 데이터들을 모두 클라이언트의 로컬 저장소에 동기화 시킨다.

우선 서버의 모든 데이터에는 각각의 고유한 server_id값들이 있다. 이 값을 이용하여 로컬 저장소에 해당 server_id를 가지고 있는 데이터가 있는지 체크한다. 이 값을 가진 데이터가 로컬 저장소에 있다는 말은 해당 데이터가 서버와 동기화가 한 번이라도 되었다는 말이고, 반대로 없다면 서버에서 받아온 이 데이터는 다른 디바이스에서 추가되어 서버에 동기화가 되어 있다가 해당 디바이스에 동기화가 되어가는 과정임을 의미한다.

따라서 로컬 저장소에 해당 server_id의 데이터가 없다면 해당 데이터를 그대로 insert하여 준다.

그리고 server_id를 가진 데이터가 로컬 저장소에 있다면 해당 데이터의 dirty_flag값을 체크해야 한다. 이 때, dirty_flag가 true라는 말은 클라이언트가 해당 데이터를 수정 한 것이기 때문에 다른 디바이스의 데이터와 충돌되는 상황인 것이다. 이때

```
[POST] http://host.io/api/v1/sync_server
body : {
        "post" : {
            "table" : {
                "local_id" : 2,
                "name" : "insert .. "
                ...
            },
            ...
        },
        "patch" : {
            "table" : {
                "local_id" : 3,
                "server_id" : 11,
                "name" : "update ,, ",
                ...
            },
            ...
        }
    }
```

그림 7. 클라이언트의 변경사항을 전달하는 API
Fig. 7 API to send the changes of clients

```
private var timeGap: Long = 0

fun setTimeGap(localTime: Long, serverTimestamp: Long): Long {
    return if (1L == localTime / 1000000000) {
        timeGap = serverTimestamp - localTime
        timeGap
    }
    else {
        0
    }
}

fun syncUpdatedAt(localUpdatedAt: Long): Long {
    return localUpdatedAt + timeGap
}
```

그림 8. 서버-클라이언트간의 시간 차이 제거 코드
Fig. 8 Example code to eliminate the time difference between server and client

는 미리 정해둔 충돌 상황 일 때의 정책에 맞춰서 데이터를 동기화 한다. 정책이 서버의 데이터로 로컬의 데이터를 업데이트해주는 정책 이라면 로컬의 데이터를 업데이트 해주고, dirty_flag를 false로 바꾸어 서버와 동기화가 완료되었다는 것을 표시해준다.

위의 로직으로 서버에서 받은 모든 데이터들을 하나씩 동기화 하면 된다.

5. 서버 동기화

위의 클라이언트 동기화 과정을 모두 거치면 서버의 변경사항들이 로컬의 저장소에 모두 반영이 된 것이다. 그리고 남은 과정은 로컬 저장소의 변경사항을 모두 서버로 반영해주는 것이다. 로컬 저장소의 변경된 데이터들을 모아 서버로 전송을 해야하는데, 이때는 dirty_flag값을 이용한다, 로컬 저장소의 데이터들 중에 dirty_flag값이 true인 값들을 모두 찾아와서 서버로 보낼 HTTP POST요청의 body를 만들어준다.

Body는 경량의 데이터 교환형식인 JSON [8] 형식을 이용하여 만든다. 위에서 찾아온 dirty_flag가 true인 데이터들의 server_id가 저장되어 있는지를 체크하여 각각 분리하여 JSON을 구성하여 보내도록 한다. 그림7은 해당 API의 body JSON 형식의 API를 나타낸다.

Server_id가 존재하지 않는 경우는 서버에 동기화가 된 적이 없고 서버에는 없는 데이터라는 의미이다. 따라서 해당 데이터들은 서버에서 모두 insert 해주면 되는 데이터들이다. 이 데이터들을 JSON의 post라는 key값으로 묶어서 구성한다.

server_id가 존재하는 경우는 동기화가 한번 이상은 진행되었던 데이터이고, 서버에 데이터가 있다는 말이다. 따라서 해당 데이터는 서버에서 해당 server_id의 데이터를 찾아서 update를 해야하는 데이터들이다. 이 데이터들은 JSON의 patch라는 key값으로 묶어서 구성한다.

서버에서는 API를 통해 받아온 데이터들을 post / patch 에 맞추어 서버의 데이터베이스에 insert 혹은 update를 해준다.

여기에서 post로 묶여온 데이터들을 서버 저장소에 insert 하고 나면 server_id가 만들어지는데, 그 값들을 모아서 응답으로 클라이언트에 전송한다. 클라이언트에서는 받은 server_id를 다시 로컬 저장소에 해당 데이터를 찾아 server_id를 update 한다.

이 과정들이 완료되는 시점이 동기화가 모두 완료되는 시점이고, 이 시점을 time stamp로 클라이언트 기기의 sync_at에 저장한다.

6. 충돌 회피 전략

충돌 상황에서의 회피 전략으로 크게 3가지가 있다. 로컬의 데이터가 항상 선택되도록 하거나, 서버에서 받은 데이터가 항상 선택되도록 하거나, updated_at 을 비교하여 최근에 수정된 데이터가 선택되도록 하는 것이다. 정책의 선택은 각각의 서비스의 성격에 맞도록 선택하여 구현하면 된다. 만약에 본 논문에서 사용하는 바와 같이 updated_at을 비교하는 오프라인 우선 정책의 경우에는 주의사항이 한 가지가 있는데, 클라이언트의 디바이스마

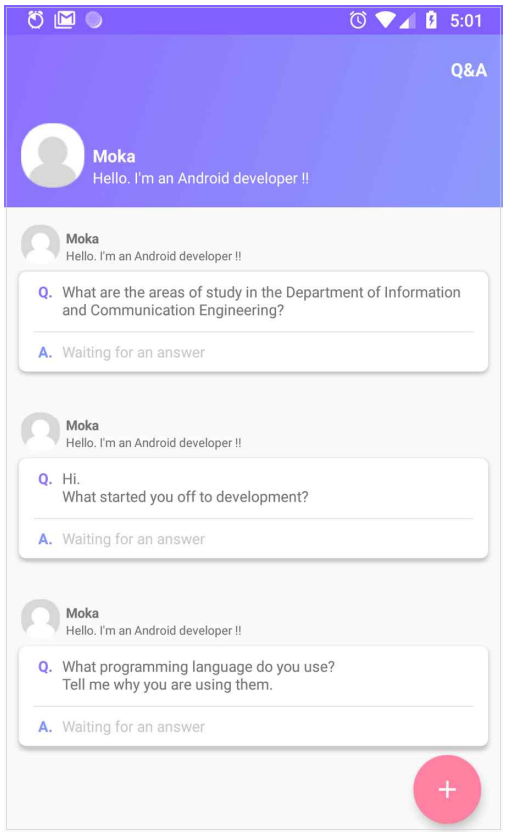

그림 9. 테스트를 위한 앱 구현 (질문 리스트 화면)
Fig. 9 Implementation of a test application

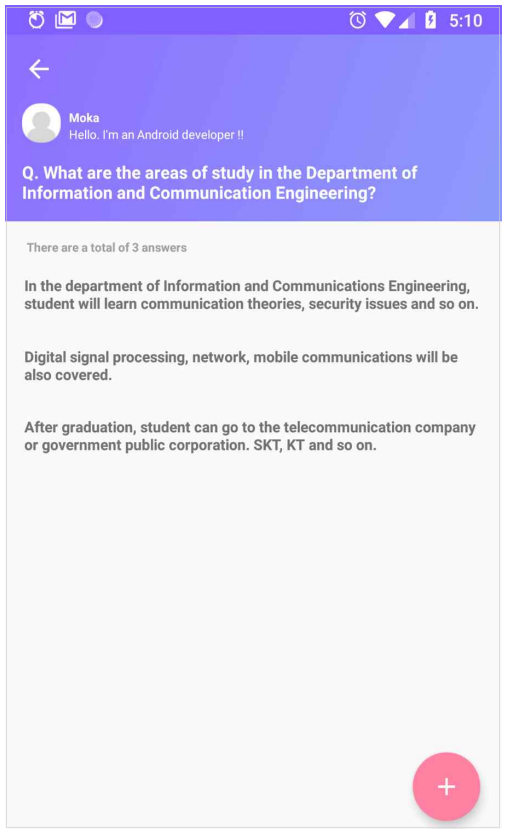

그림 10. 테스트를 위한 앱 구현 (상세 답변 화면)
Fig. 10 Implementation of a test application

다 time zone이 다를 수가 있으므로 서버와 클라이언트의 시간 차이를 고려하여 비교할 수 있도록 그림8과 같이 구현한다.

7. 동기화 타이밍 & 실시간 동기화

위의 과정을 모두 구현하면 기본적인 오프라인 우선의 동기화 기능이 모두 구현된다. 마지막으로 고려해야 할 부분이 동기화를 할 타이밍이다. 언제 어떠한 주기로 동기화를 할지 잘 조율하여 서버의 부하도 적게 주면서, 사용자가 실시간으로 잘 동기화가 된다고 느낄 수 있도록 해야 한다.

첫 번째로는 서버에서 변경 사항이 누적되어 동기화가 필요하다는 것을 클라이언트에게 알려줬을 때이다. 서버 쪽에서 클라이언트로 이벤트를 주기 위해서는 모바일 푸시를 이용하거나 socket 통신 등을 이용하여 구현해 준다.

두 번째로는 클라이언트에서 로컬 저장소에 insert / update 가 일어났을 때이다. 이때 이미 동기화가 진행 중일 때는 동기화 요청이 반복되지 않도록 구현한다.

Ⅲ. 구현 및 적용

본 기술을 활용하여 실제 서비스할 수 있는 예로서, 세미나 또는 컨퍼런스에서 이용할 수 있는 Q&A 서비스에 동기화를 활용하는 시나리오를 가정하여 구현하였다. 질문 리스트를 보는 화면 그림9와, 하나의 질문에 대한 답변 리스트를 볼 수 있는 그림10과 같이 구현하였다.

세미나에 참석한 사람이 휴대폰 앱을 통해 질문사항을 올리면 해당 질문은 세미나에 참석한 모든 사람의 휴대폰 앱에서 동기화되어 확인할 수 있어야 한다. 또한 강연자의 앱 또는 웹에 질문들이 보이고, 강연자가 해당 질문들에 답변할 수 있어야 하며, 등록한 답변 또한 동기화가 되어 모든 사람들에게 보여야 한다.

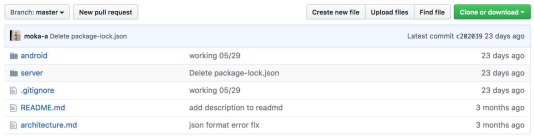

그림 11. 소스 코드의 공개

Fig. 11 Release version of source Code

그림 12. 테스트 환경 구성도

Fig. 12 Configuration of test scenario

그런데 사람들이 많이 모이는 장소의 인터넷 환경은 항상 원활하지 않을 수도 있다. 그럼에도 불구하고 인터넷이 잠시 연결이 되지 않은 경우, 세미나 참여자가 질문을 올리면 본인의 휴대폰에서는 바로 잘 등록이 되는 것처럼 보여야 한다. 오프라인 상황이기 때문에 이 질문이 바로 모든 사람들에게 동기화되어 보이는 것은 아니지만, 인터넷 연결이 다시 원활히 되면, 직접 재등록하는 과정이 필요 없이 자동으로 동기화되어 모든 사람들에게 보여야 하는 것이다.

위 과정을 테스트하기 위해서, 클라이언트 어플리케이션과 서버 어플리케이션을 그림11과 같이 각각 구현하여 코드 공개 저장소인 GitHub에 업로드하였다 [9].

클라이언트 앱은 안드로이드를 이용하여 구현하였다. 로컬 저장소를 위하여 Realm [10] 을 이용하고 서버와의 통신을 위해 HTTP 모듈인 Retrofit [11] 을 활용하였다.

Realm은 모바일환경에서 빠르고 쉽게 사용할 수 있는 데이터베이스이고, Retrofit은 HTTP 통신을 쉽게 구현할 수 있도록 인터페이스를 제공하는 라이브러리 이다.

서버 앱은 NodeJs [12]를 활용하여 구현하였다. 서버에서의 데이터베이스는 MariaDB를 이용했다. 그리고 원활한 테스트 환경을 위해 서버 앱을 클라우드 환경인 AWS EC2에서 서비스하도록 올려두었고, 데이터베이스 또한 AWS RDS를 이용하여 서버 환경을 구축하였다.

그림12은 구현된 어플리케이션들의 구성을 나타낸 것이다.

Ⅳ. 결론 및 향후 목표

본 논문에서는 오프라인 우선의 실시간 데이터베이스 동기화의 구현에 대하여 다루고 이를 구현한 방법에 대하여 다루었다. 위 설계 과정에 따라 코드를 구현하여 테스트 한 결과 동시에 다양한 디바이스에서 오프라인 우선 정책에 의한 실시간 데이터가 동기화되는 것을 확인하였다.

본 논문에서 구현된 기술은 다른 앱의 도움을 받거나 제공하는 기능만을 사용하는 것이 아니라, 직접 동기화 로직을 관리하기 때문에 특정 모델이 동기화가 될 때, 원하는 로직을 추가하거나, 특별한 정책을 가지며 동기화할 수 있도록 추후 구현하는 것도 가능하다.

더욱 다양해지는 스마트 기기들 속에서 해당 기술이 많은 서비스에서 필요로 하고 있다. 하지만 이를 구현하는 것이 그렇게 쉽지만은 않고 많은 인력이 필요하다. 향후 목표는 해당 설계를 가진 코드를 각각의 서비스의 데이터베이스 스키마에 맞도록 자동으로 생성하는 것이다. 그렇게 하여 이러한 기술적인 부분에서의 리소스를 최대한 줄이고, 서비스의 본질적인 기능 개발에 리소스를 투자할 수 있도록 도움을 주는 것이다.

References

[1] Avaliable on : https://www.wunderlist.com/ko/

[2] Avaliable on : https://ko.todoist.com/

[3] Avaliable on : https://calendar.google.com/

[4] D.J. Jang, K.H. Park, H.T. Ju, “Development of Data Synchronization Client Based on SyncML,” Jounal of KIISE Trans. Computing Practices, Vol. 11, No. 4, pp. 257-367, 2005.(in Korean)

[5] Darin C. Glatt, Beaverton, “System and Method of Data Synchronization Between Devices,“ United States Patent. No. 7,962,575, 2011.

[6] Avaliable on : https://developers.google.com/calendar/v3/sync

[7] Avaliable on : http://havrl.blogspot.com/2013/08/synchronization-algorithm-for.html
[8] Nurzhan Nurseitov, Michale Paulson, Randall Reynolds, Clemente Izurieta, “Comparison of JSON and XML Data Interchange Formats: a Case Study,” Proceedings of CAINE, No. 9, pp. 157-162, 2009.
[9] Avaliable on : https://github.com/moka-a/Offline-first-multi-device-real-time-data-synchronize
[10] Avaliable on : https://realm.io/kr/products/realm-database
[11] Avaliable on : http://square.github.io/retrofit/
[12] Stefan Tilkov, Steve Vinoski, “Node.js: Using Javascript to Build High-performance Network Programs,” Proceedings of IEEE internet computing, Vol. 14, No. 6, pp. 80-83, 2010.

Dae-Myoung Lee (이 대 명)

He is currently working toward B.S. degree at Pukyong National University, Korea. He is in the course of SW Maestro 9th. His research interests include factory automation and mobile application system.
Email: aud1220a@gmail.com

Eun-Hoo Kim (김 은 후)

He is currently working toward B.S. degree at Pukyong National University, Korea. His research interests include factory automation and embedded system.
Email: wsv0131@naver.com

Moon-Gab Joo (주 문 갑)

He received his B.S. in Electricity and Electronic Engineering from POSTECH, Korea, in 1992. He received his M.S. in Information Communication Engineering from POSTECH in 1994. He received his Ph.D. in Electronic Computer Engineering from POSTECH in 2001. Since 2003, he has been a professor at Pukyoung National University, Korea. His research interests include intelligent control and factory automation.
Email : gabi@pknu.ac.kr