Best Practice

더 좋은 방법을 고민하는 개발자 **Bepi**입니다

이력

- 학점 연계 인턴십 샵라이브 코리아 백엔드 개발자 (2022.03 ~ 2022.06)
- 스타트업 겨울방학 인턴십 DINO-STUDIO 백엔드 개발자 (2021.12 ~ 2022.02)
- 숭실대학교 창업지원단 소속 온더브릿지 백엔드 개발자(2021. 04 ~ 2021.08)

동아리 및 커뮤니티

- 숭실대학교 개발동아리 유어슈 백엔드 팀원(2021.10~)
- Google Developers Student Club 숭실대 지부 Server/Cloud Member(2021.09~)

학력

- 숭실대학교 IT대학 글로벌미디어학부(2016.03 ~)

백엔드 개발자 오진호

Resume: 링크
Email: ohjinho7@gmail.com
Github: https://github.com/ohjinhokor

ShopLive Korea
(학점 연계 인턴십)

서비스 명 : 샵라이브

- **인턴십 기간**

  2022. 03 ~ 2022. 06

- **서비스 설명**

  - 라이브 영상을 시청하며 실시간으로 물건을 구매할 수 있는 서비스입니다.
  - B to B 서비스로서 고객사의 UI 화면에 플러그인 형식으로 서비스를 제공합니다.

- **사용 기술 스택**

  - Java, SpringBoot, MySQL, gRPC

- **나의 구현 내용**

  ⊙ gRPC를 이용해 고객사에게 제공할 라이브러리(프로토타입) 구현

  ⊙ 비즈니스 로직 작성

  ⊙ mem-cached를 이용한 캐싱

  ⊙ 검색 기능 및 페이지네이션 구현

  ⊙ 리팩토링을 통한 쿼리 시간 단축 및 코드 간소화

○ 고객사에게 제공할 라이브러리 구현(프로토타입)

- 소규모 프로젝트로 재현 : 링크
(https://github.com/ohjinhokor/grpc)

⊙ 라이브러리 상세

- 기존에 제공하던 Public API를 **gRPC**를 이용한 라이브러리로 제공

- 고객사에서 gRPC 클라이언트 부분 코드를 구현하게 하지 않고, 클라이언트 측 코드를 대신 구현하여 jar 파일로 제공

- 이로 인해 고객사는 ip 주소와 port를 몰라도 라이브러리 추가만으로 회사와 통신할 수 있음.

⊙ Rest 방식에서 gRPC로 대체한 이유

1. 속도
2. 고객사의 편의
3. API 호출의 제어권을 가져오기 위함

ex) 제공하는 jar 파일에 캐싱을 적용. 짧은 시간내에 여러 번 동일한 요청을 하는 경우
캐싱된 데이터를 반환하도록 하여 서버에 들어오는 Request를 줄임

○ 비즈니스 기능 추가

- 고객사가 직접 이미지와 비디오를 등록 수정 삭제 및 조회 가능

⊙ 검색 기능 및 페이지네이션

- 검색 기능

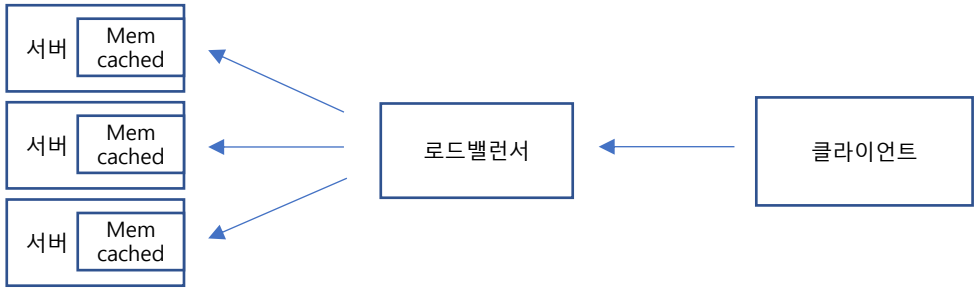
  - 미디어 종류, 업로드 시간, 검색어 등 조건에 따라 검색하는 기능

- 페이지네이션

  - 클라이언트로부터 한 페이지에 보여줄 미디어 파일 갯수를 받아 해당 갯수만큼 데이터를 반환합니다.

⊙ 인메모리 데이터베이스(Memcached)를 이용한 캐시

- 방송이 시작되면 방송 종료시까지 같은 요청에 대한 반환 response가 동일합니다.
- 따라서 데이터 정합성을 위한 별도의 캐시서버가 필요하지 않습니다
- 예시) 방송이 시작한 후에는 판매 중인 상품이 변동되지 않습니다. 따라서 판매 상품을 반환해주는 로직은 항상 같은 데이터를 반환합니다.



○ 리팩토링(코드 중복 제거)

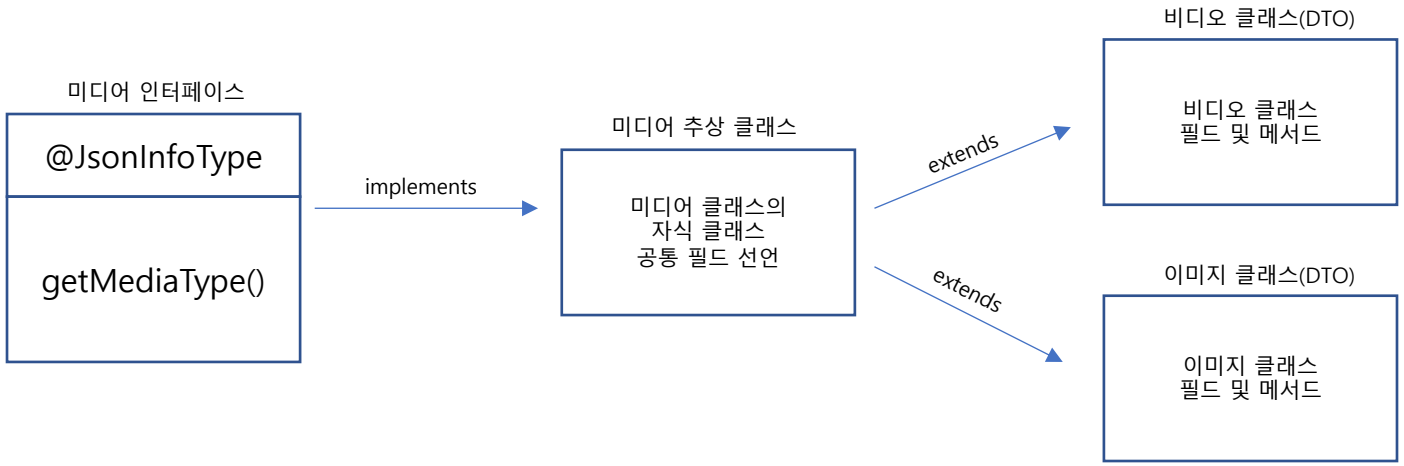
⊙ 상황

- 비디오 클래스와 이미지 클래스는 미디어 클래스를 상속합니다.

- 비디오 클래스와 이미지 클래스는 다른 필드를 가지고 있으므로 클라이언트의 요청에 대해 다른 DTO와 메서드를 사용합니다. 따라서 비슷한 코드가 여러 번 작성됩니다.

- 미디어의 종류가 추가될 때마다 코드 중복이 발생합니다.

⊙ 인터페이스와 추상클래스를 이용한 코드 중복 해결 및 확장성 증가



1. JsonInfoType 어노테이션을 이용하여 메시지 컨버터가 요청에 알맞은 구체 클래스로 데이터를 매핑합니다.

2. getMediaType 메서드를 이용하여 코드 레벨에서 인자로 들어온 클래스의 타입을 확인합니다. 이를 통해 어떤 구체 클래스의 객체로 역직렬화 되었는지 알 수 있습니다. 이를 이용하여 다음 서비스 로직을 실행합니다.

3. 이를 통해 컨트롤러와 서비스에서 하나의 메서드로 모든 미디어 타입에 대한 요청에 응답할 수 있습니다.

DINO STUDIO
(스타트업 인턴십)

프로젝트 명 : 하이엔드(High-End)

○ 프로젝트 참여 기간

2021. 12. 28 ~ 2022. 02. 28

○ 프로젝트 개요

- 남녀의 만남을 주선하는 소개팅 어플입니다.
- 매일 새로운 이성을 추천합니다.
- 설정에 따라 지인에게 내 정보가 노출되지 않도록 할 수 있습니다.

○ 기술 스택

- Java, SpringBoot, Jpa, Querydsl, MySQL
- EC2, RDS, S3, CloudFront
- Github Action, Code Deploy, S3, Nginx

○ 나의 구현 내용

⊙ 프로젝트 초기 인프라 구축 – EC2, RDS, S3, CloudFront

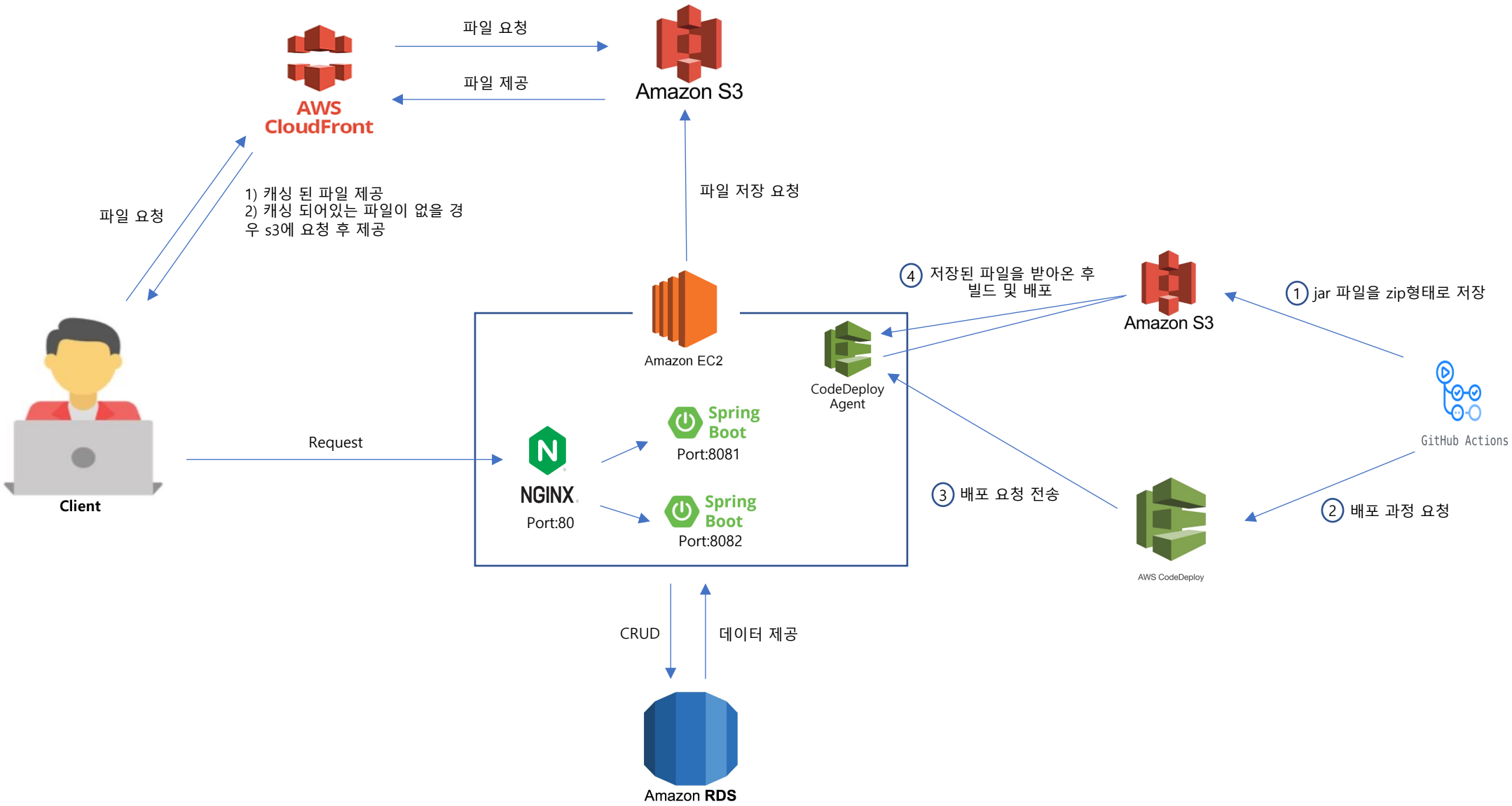
⊙ 비즈니스 로직 작성

⊙ Spring WebClient를 이용한 외부 API와의 통신

⊙ CI/CD와 무중단 배포 - Github Action, Code Deploy, S3, Nginx

⊙ 스웨거를 이용한 API 문서 자동화

○ 서버 구조도


파일 요청
파일 제공
AWS CloudFront
Amazon S3
파일 요청
1) 캐싱 된 파일 제공
2) 캐싱 되어있는 파일이 없을 경우 s3에 요청 후 제공
파일 저장 요청
4 저장된 파일을 받아온 후 빌드 및 배포
1 jar 파일을 zip형태로 저장
Amazon EC2
CodeDeploy Agent
Amazon S3
GitHub Actions
Client
Request
NGINX
Port:80
Spring Boot
Port:8081
Spring Boot
Port:8082
3 배포 요청 전송
2 배포 과정 요청
AWS CodeDeploy
CRUD
데이터 제공
Amazon RDS

ㅇ 아는 사람 피하기 기능

- 매핑 테이블을 이용하여 비즈니스 요구사항을 풀어냈습니다.

- ‘아는 사람 피하기’ 기능을 사용할 경우 핸드폰에 저장되어 있는 지인은 추천 되지 않으며 나의 정보를 지인에게 노출시키지 않을 수 있습니다.

- 비즈니스 로직 대부분은 아는 사람 피하기 기능과 관련되어 동작합니다. 따라서 **속도를 빠르게 하기 위한 설계를** 하였습니다.

⊙ 시나리오 예시 (순서)

1) 유저 A는 유저 B의 번호를 가지고 있고, 유저 B는 유저 A의 번호를 모릅니다. 이 때 유저 A가 아는 사람 피하기를 신청합니다.

2) 시간이 지나 유저 B가 아는 사람 피하기 기능을 신청합니다.

3) 이 과정 이후 유저 A가 아는 사람 피하기 기능을 취소하더라도 유저 A와 유저 B는 서로 추천되어서는 안됩니다. B는 여전히 아는 사람 피하기 기능을 사용하고 있고, 비록 유저 B의 핸드폰에 유저 A의 번호가 없더라도 이전 과정을 통해서 두 사람이 아는 사이임을 알 수 있기 때문입니다.

⊙ 지인 등록

지인 등록은 두가지 상황에서 동작합니다.

1) 이미 가입한 유저가 아는 사람 피하기 기능을 요청할 때
2) 새로운 유저가 가입할 때

1번 상황에서 지인의 번호 목록을 통해 기존에 가입한 지인들을 파악하고,
2번 상황마다 미리 등록해놓은 번호를 이용하여 지인 여부를 판단한다면 모든 상황을 대처할 수 있습니다.

○ 아는 사람 피하기 기능

⊙ 데이터 베이스 설계

1) users 테이블에 기능 사용 여부 컬럼 추가

- '아는 사람 피하기 기능' 사용 여부를 알려주는 컬럼이 추가됩니다.

2) acquaintance_phone_numbers 테이블

- user 핸드폰에 등록된 번호 목록을 저장하는 테이블입니다.
- user 테이블과 1:N 관계입니다.

3) acquaintance_relations 테이블 (매핑 테이블)

- 아는 사람 관계를 미리 저장하여 로직 수행 시간을 단축하기 위한 테이블입니다.
- 두 user가 아는 사이임을 나타냅니다.
- user 대 user 의 N:N 관계를 풀어냈습니다.

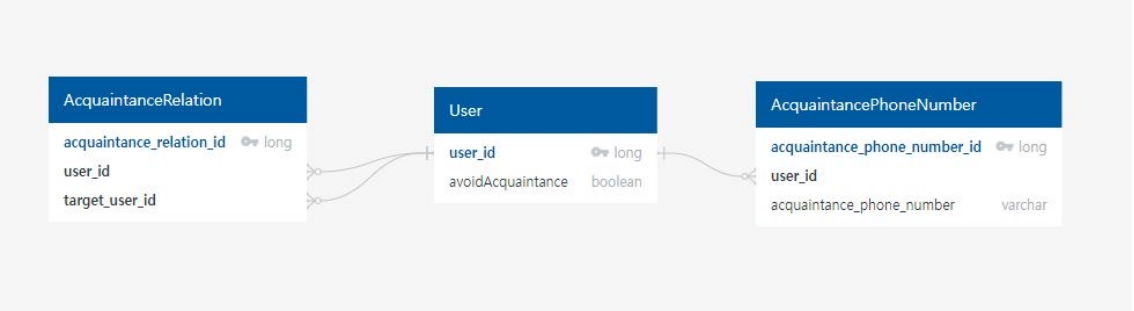
⊙ 지인 목록을 가져오는 경우 필요한 검색 조건 최소화

- 하나의 지인 관계를 하나의 row로 표현 하는 방법 대신 **두개의 row로 표현하여 검색 속도를 빠르게 하였습니다.**

user A와 user B가 지인이라면 user A의 pk를 user_id로 user B의 pk를 target_user_id로 하는 row를 만든 후 반대로 user B의 pk를 user_id로 user A의 pk를 target_user_id로 하는 row를 추가.

- **검색 조건이 되는 column을 하나의 컬럼**(user_id)으로 한 후 **indexing**을 하는 방법을 선택했습니다.

⊙ 데이터 베이스 예시



○ 아는 사람 피하기 기능

⊙ DB 인덱싱을 통한 속도 향상

- acquaintance_relations 테이블의 user_id 컬럼에 index를 사용하여 서비스의 전체적인 로직 시간을 단축하였습니다.

⊙ 인덱싱을 선택한 이유

1) 개발적 관점

- 이성 추천, 게시물 리스트 조회 등 대부분의 비즈니스 로직에서 지인 여부를 확인해야 했고 그에 따라 **select문을 사용하는 경우가 update, delete, insert를 사용하는 경우보다 압도적으로 빈번**했기 때문입니다.

2) UX적 관점

- 사용자가 지인 번호를 등록 또는 삭제하는 경우 시간이 더 소요되는 것에 대해 불편함을 느끼는 것보다, 제공하는 서비스가 전반적으로 빠르게 동작하는 것에 대해 더 큰 만족감을 느낀다고 판단하였습니다.

○ CICD, 무중단 배포

⊙ CICD - github action, code deploy, s3

⊙ Nginx를 이용한 무중단 배포 (블루그린 방식)

○ 로그인

⊙ JWT, Argument Resolver 사용

1. 프로젝트 초기였기에 확장이 용이한 jwt를 선택했습니다.
2. Refresh Token을 사용하여 자동 로그인 기능을 구현하였습니다.
3. 반복되는 코드를 줄이기 위해 Spring에서 제공하는 Argument Resolver를 사용하였습니다.

Side Project (소규모 프로젝트)

프로젝트 명 : Emotimer

웹소켓을 이용한 다중기기 동시 타이머

- **프로젝트 기간**

  2022. 05 ~ 2022. 07

- **프로젝트 개요**

  - 동일 계정으로 접속한 모든 기기에서 동시에 시작하는 타이머입니다.
  - 주간, 월별, 연도별 타이머 사용 통계를 보여줍니다.

- **깃허브** 링크

  URL : https://github.com/gdsc-ssu/emotimer

- **기술 스택**

  - Java, SpringBoot, Jpa, MySQL, QueryDsl
  - web socket (STOMP)

- **나의 구현 내용**

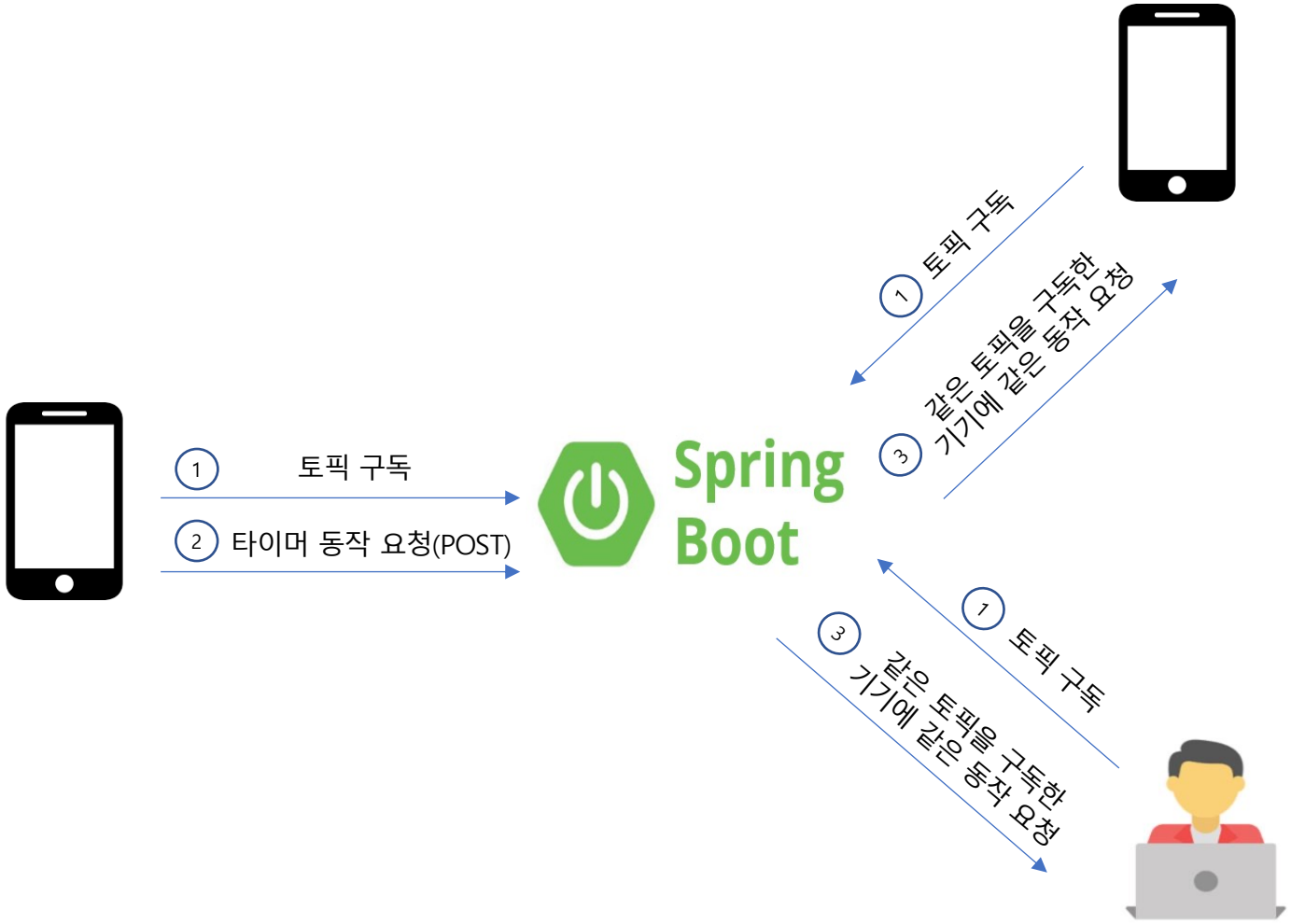
  ⊙ 웹소켓을 이용해 연결된 모든 기기에게 타이머 동작 전송

  ⊙ 타이머 히스토리 저장 후 통계 반환

○ 타이머 동작

- 같은 계정으로 접속한 모든 기기의 타이머가 동시에 동작

- **웹소켓(STOMP) 사용**



ㅇ 타이머 히스토리 자동 저장

- Timer(자바 제공 Util)클래스와 TimerTask 클래스를 사용하여 비동기/Non-Blocking 방식 활용

ㅇ 리팩토링(코드 중복 제거)

⊙ 상황

- 타이머 히스토리의 통계를 연도별, 월별, 주간별로 제공
- 쿼리 결과를 리스트로 반환
- 연도별, 월별, 주간별로 제공하는 통계 데이터의 형식이 동일함

⊙ 제네릭 메서드와 추상클래스를 이용한 코드 중복 해결

1. 반환하는 공통 데이터를 추상 클래스의 필드로 선언
2. 연도별, 월별, 주간별 데이터를 보여주는 클래스는 추상 클래스를 상속
3. 제네릭 메서드를 이용하여 반환 데이터 리스트를 만드는 메서드를 하나로 통일