카카오페이 SERVER 개발 과제

- 카카오페이 뿌리기 기능 구현하기

지원자명 : 주상혁

작성일자 : 2020년 9월 20일

Contents

1. 요구사항 분석 (1/2)

과제에 도출된 개발에 필요한 요구사항과 제약사항을 점검하고 프로그램 설계, 개발을 위한 기반을 작성함

요구사항ID	업무 구분	요구사항 명	요구사항 상세
REQ-001	시스템 공통	REST API 구현	뿌리기, 받기, 조회 기능을 수행하는 REST API 를 구현
REQ-002		API 의 HTTP Method 정의	API 의 HTTP Method들 (GET | POST | PUT | DEL) 은 자유롭게 선택
REQ-003		사용자 식별값 HTTP Header 설정	요청한 사용자의 식별값 - HTTP Header / "X-USER-ID" / 숫자
REQ-004		대화방 식별값 HTTP Header 설정	요청한 사용자가 속한 대화방의 식별값 - HTTP Header / "X-ROOM-ID" / 문자
REQ-005		다수 서버/인스턴스 기반 설계	다수의 서버, 다수의 인스턴스로 동작가능하도록 어플리케이션 설계
REQ-006		다량의 트래픽 고려 설계	다량의 트래픽에도 무리가 없도록 효율적으로 어플리케이션 설계
REQ-007		개발언어 정의	개발 언어는 Java, kotlin, scala 중 하나로 구현
REQ-008		데이터베이스 정의	모든 데이터베이스 사용가능
REQ-009		응답코드 정의	에러응답, 에러코드는 자유롭게 정의
REQ-010		소스코드 가독성	작성한 어플리케이션 코드가 간결하고 가독성 좋게 작성
REQ-011	테스트 관리	단위테스트 작성	각 기능 및 제약사항에 대한 단위테스트를 반드시 작성
REQ-012	뿌리기 API	Input 정의	뿌릴 금액, 뿌릴 인원을 요청값으로 받음
REQ-013		Output 정의	뿌리기 요청건에 대한 고유 token을 발급하고 응답값으로 내려줌
REQ-014		token 발급 정의	token은 3자리 문자열로 구성되며 예측이 불가능 해야함
REQ-015		뿌리기 분배 정의	뿌릴 금액을 인원수에 맞게 분배하여 저장 해야함 (분배로직은 자유)
REQ-016		잔액체크 기능 제외	모든 사용자는 뿌리기에 충분한 잔액을 보유하고 있다고 가정. 잔액 체크는 제외

1. 요구사항 분석 (2/2)

과제에 도출된 개발에 필요한 요구사항과 제약사항을 점검하고 프로그램 설계, 개발을 위한 기반을 작성함

요구사항ID	업무 구분	요구사항 명	요구사항 상세
REQ-017	받기 API	Input 정의	뿌리기 시 발급된 token을 요청값으로 받음
REQ-018		Output 정의	token에 해당하는 뿌리기의 분배건 하나를 할당하고 그 금액을 응답값으로 내려줌
REQ-019		받기 횟수 정의	뿌리기 당 한 사용자는 한번만 받을 수 있음
REQ-020		받기 유효기간 정의	뿌린 건은 10분간만 유효함
REQ-021		자신 받기 제약사항 정의	자신이 뿌리기한 건은 자신이 받을 수 없음
REQ-022		사용자 받기 제약사항 정의	뿌린기가 호출된 대화방과 동일한 대화방에 속한 사용자만이 받을 수 있음
REQ-023	조회 API	Input 값 정의	뿌리기 시 발급된 token을 요청값으로 받음
REQ-024		Output 값 정의	token에 해당하는 뿌리기 건의 현재 상태를 응답값으로 내려줌 현재상태는 다음의 정보를 포함 - 뿌린 시각 - 뿌린 금액 - 받기 완료된 금액 - 받기 완료된 정보 ([받은 금액, 받은 사용자 아이디] 리스트)
REQ-025		조회 기간 정의	뿌린 건에 대한 조회는 7일 동안 할 수 있음
REQ-026		조회 권한 정의	뿌린 사람 자신만 조회를 할 수 있음

Contents

2. 문제해결 전략

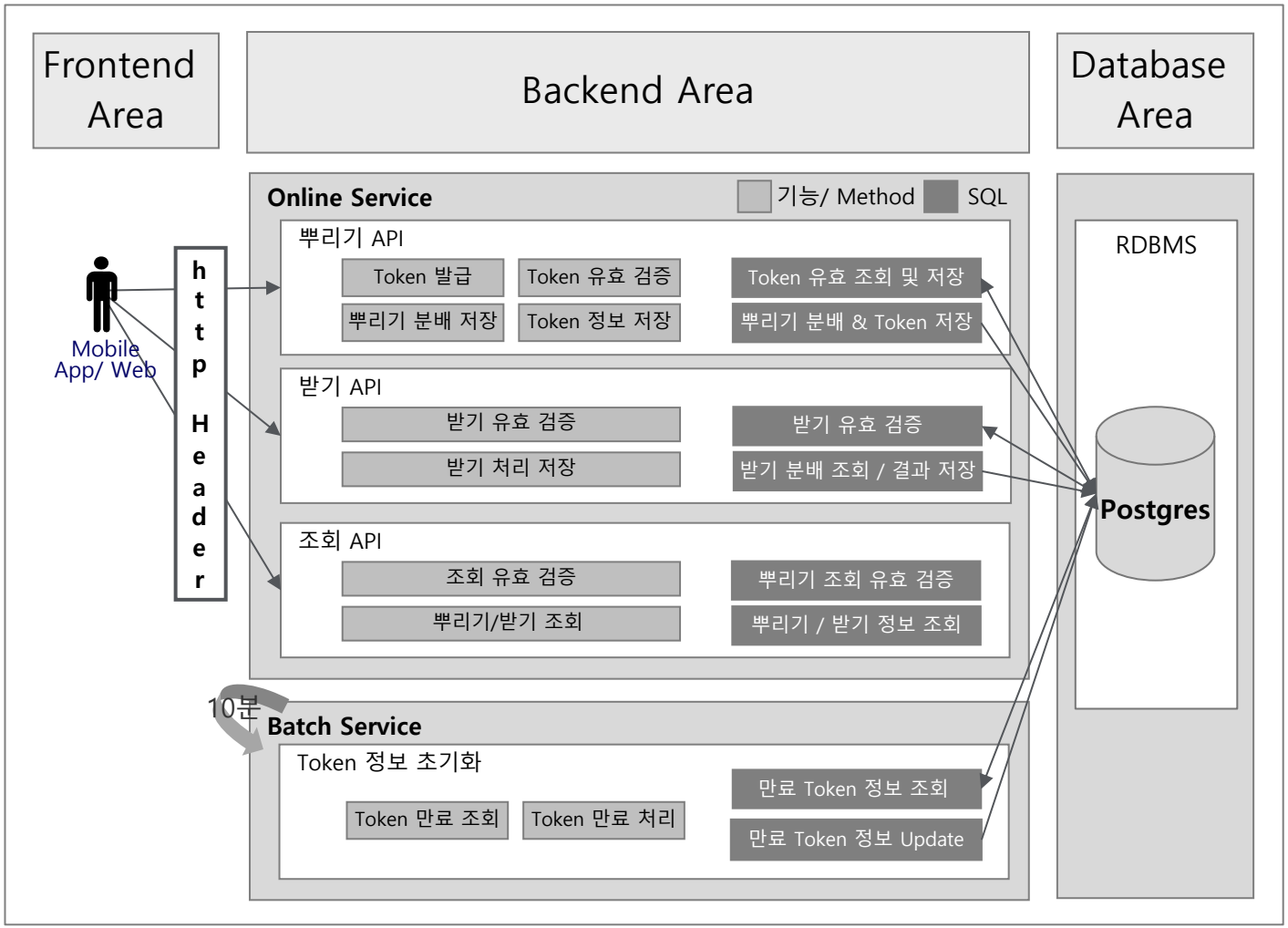
고유 식별자 토큰을 발행 및 관리하는 프로세스 및 DB 설계, 다중 서버 및 대량 트래픽 환경에서 데이터의 정합성과 응답/처리 속도를 보장하기 위한 어플리케이션 구성함

구분	내용
토큰	▪ 예측 불가능하게 구성하기 위한 난수 발생 (시퀀스 사용 불가) ▪ 3자리 문자열 구성으로(238328개) 고유 식별자 역할을 위해서 대화방 별 토큰 구성 ▪ 동시 다발적인 요청에서도 중복없이 고유성을 유지하도록 해당 대화방은 Lock설정 ▪ 뿌리기 유효기간 7일 이후 건과의 중복은 허용하도록 구성
보안, 성능 및 데이터 정합성	▪ 분배 시점에 대화방 참여 인원 정보를 저장함으로써 보안 및 정합성 확보 ▪ 유효기간 10분 후 상태코드(타임아웃)를 변경하기 위한 배치 프로그램 구성 ▪ 금액의 정합성 유지를 위해 뿌리기, 받기, 타임아웃 시점에 입출금 API 호출 ▪ 받기가 진행중인 경우 할당 건이 중복되지 않도록 해당 대화방은 Lock설정 ▪ 추후 금액의 흐름 파악 혹은 잔액 대사가 가능하도록 필요한 정보를 남길수 있도록 테이블 설계 진행

Contents

3. 시스템 구성 특징

Online은 Rest API로 구성, 빠른 응답속도를 위해 PostgreSQL를 선택해 Cluster확장 가능한 기반을 만들고, Batch를 통해 데이터 정합성을 보장하도록 구성함



구성 특징

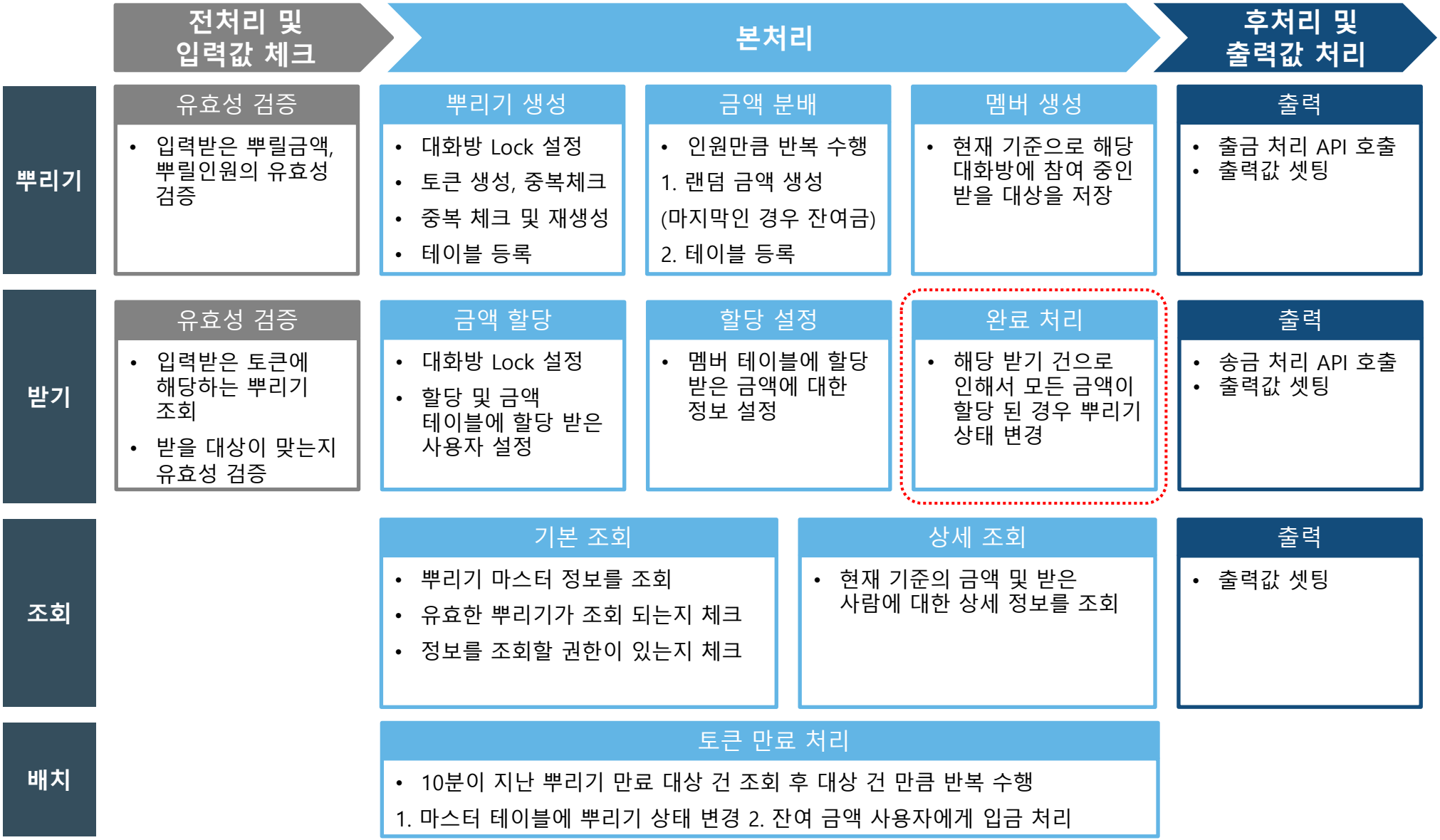
1. Frontend에서 대화방 및 사용자 식별값을 Http Header담아 Rest API 호출
2. 뿌리기 API에서 발급한 Token의 유효 검증은 안정성확보를 위해 Row Lock Select로 조회하고 1초 Wait 설정
3. 뿌리기 분배결과, 받기 결과, 뿌리기/받기 조회는 RDBMS에 저장된 데이터를 조회
4. 받기 처리는 동시 처리를 보장하고 동일 데이터 접근 제어를 위해 DB Lock처리하고 1초 Wait 설정 (대용량 처리를 위해 1초 이상 Wait를 Fail 처리)
5. Batch를 통해 유효기간이 만료된 Token을 RDBMS에 상태코드 Update

* 제출한 프로그램은 개발 과제에 목적이 있어 DB를 별도로 구성하지 않고 프로젝트내의 Embedded PostgreSQL을 적용함.

Contents

4. 프로그램 흐름도

과제의 요구사항이 충족하도록 뿌리기, 받기, 조회, 배치 프로그램의 프로그램 흐름을 설계함

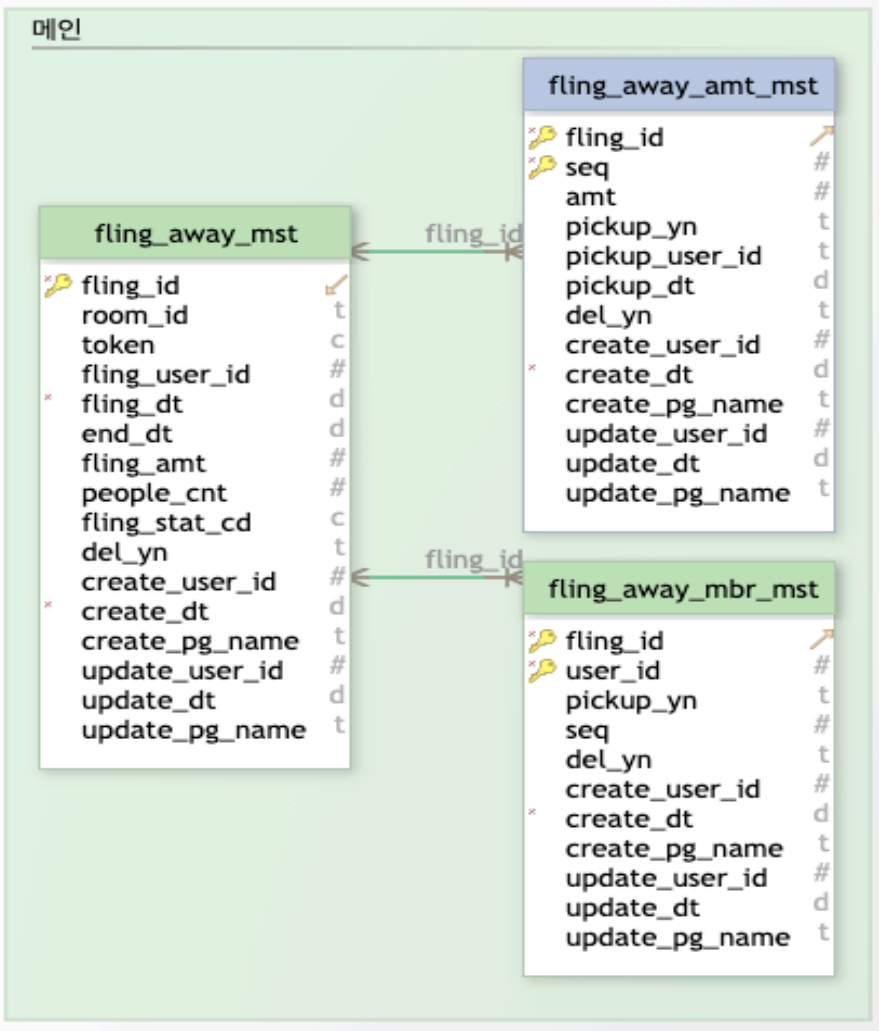


Contents

5. 데이터베이스 설계

과제의 요구사항이 충족하도록 데이터베이스를 설계하고, 테스트를 수행하기 위한 테이블을 설계함

< 데이터베이스 물리 모델 >



* 테이블 물리모델로 논리모델을 별도 첨부

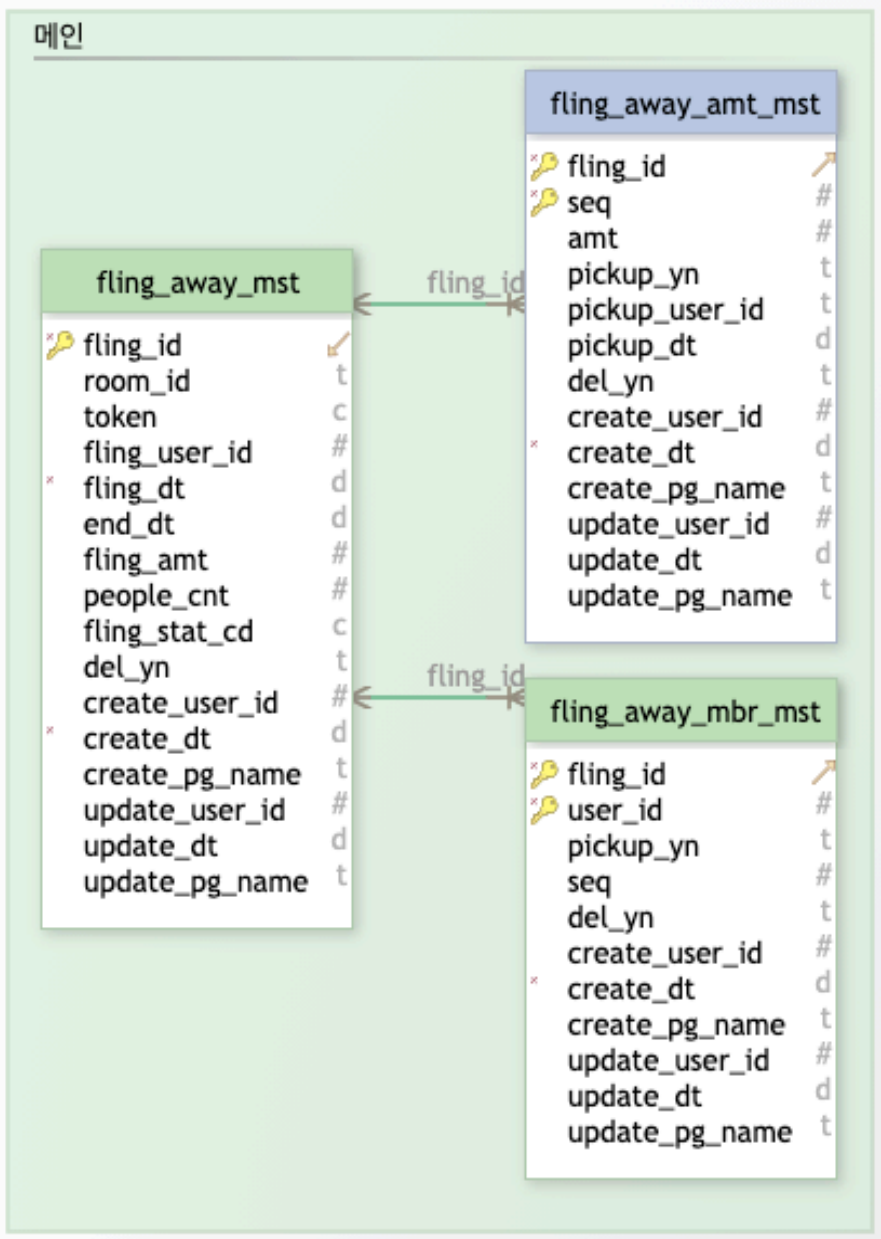
< 시스템 공통 컬럼 >

del_yn	varchar(1) DEFAULT 'N'::character varying	삭제여부
create_user_id	integer	생성자 ID
create_dt	timestamptz DEFAULT CURRENT_TIMESTAMP	생성일시
create_pg_name	varchar(100)	생성 프로그램 명
update_user_id	integer	수정자 ID
update_dt	timestamptz	수정일시
update_pg_name	varchar(100)	수정 프로그램 명

< 테스트 기초 데이터 테이블 >

[Backup] 데이터베이스 물리 모델 및 상세설명

< 데이터베이스 물리 모델 >



< 뿌리기 테이블 : fling_away_mst >

Idx	Field Name	Data Type	Description
*🔑↙	fling_id	varchar(12) DEFAULT ('F'::text || lpad((nextval('fling_id_seq'::regclass))::text, 11, '0'::text))	뿌리기 ID
	room_id	varchar(12)	대화방 ID
	token	char(3)	토큰
	fling_user_id	integer	뿌린사람
*	fling_dt	timestamptz DEFAULT CURRENT_TIMESTAMP	뿌린일시
	end_dt	timestamptz	종료일시
	fling_amt	numeric(12,0)	뿌린금액
	people_cnt	integer	총받을사람수
	fling_stat_cd	char(1) DEFAULT '1'::bpchar	뿌리기상태코드

< 뿌리기 금액 테이블 : fling_away_amt_mst >

Idx	Field Name	Data Type	Description
*🔑↗	fling_id	varchar(12)	뿌리기 ID
*🔑	seq	integer	순번
	amt	numeric(12,0)	금액
	pickup_yn	varchar(1) DEFAULT 'N'::character varying	받기여부
	pickup_user_id	varchar(100)	받은사람
	pickup_dt	timestamptz	받은일시

< 뿌리기 멤버 테이블 : fling_away_mbr_mst >

Idx	Field Name	Data Type	Description
*🔑↗	fling_id	varchar(12)	뿌리기 ID
*🔑	user_id	integer	사용자 ID
	pickup_yn	varchar(1) DEFAULT 'N'::character varying	받기여부
	seq	integer	순번

Contents

6. 기능구현 (1/3) – 뿌리기

뿌리기 API 수행 후 각 테이블 데이터 확인 (amt = 3000, cnt = 3)

뿌리기 테이블 : fling_away_mst

	1
fling_id	F00000000001
room_id	R00000000001
token	k5d
fling_user_id	10
fling_dt	2020-09-19 13:38:00.877751
end_dt	2020-09-19 13:48:00.877751
fling_amt	3000
people_cnt	3
fling_stat_cd	1
del_yn	N
create_user_id	10
create_dt	2020-09-19 13:38:00.877751
create_pg_name	FlingService.flingAway
update_user_id	<null>
update_dt	<null>
update_pg_name	<null>

뿌리기 금액 테이블 : fling_away_amt_mst

	1	2	3
fling_id	F00000000001	F00000000001	F00000000001
seq	1	2	3
amt	527	1574	899
pickup_yn	N	N	N
pickup_user_id	<null>	<null>	<null>
pickup_dt	<null>	<null>	<null>
del_yn	N	N	N
create_user_id	10	10	10
create_dt	2020-09-19 13:38:00.877751	2020-09-19 13:38:00.877751	2020-09-19 13:38:00.877751
create_pg_name	FlingService.flingAway	FlingService.flingAway	FlingService.flingAway
update_user_id	<null>	<null>	<null>
update_dt	<null>	<null>	<null>
update_pg_name	<null>	<null>	<null>

뿌리기 멤버 테이블 : fling_away_mbr_mst

	1	2	3	4
fling_id	F00000000001	F00000000001	F00000000001	F00000000001
user_id	11	12	13	14
pickup_yn	N	N	N	N
seq	<null>	<null>	<null>	<null>
del_yn	N	N	N	N
create_user_id	10	10	10	10
create_dt	2020-09-19 13:38:00.877751	2020-09-19 13:38:00.877751	2020-09-19 13:38:00.877751	2020-09-19 13:38:00
create_pg_name	FlingService.flingAway	FlingService.flingAway	FlingService.flingAway	FlingService.flingA
update_user_id	<null>	<null>	<null>	<null>
update_dt	<null>	<null>	<null>	<null>
update_pg_name	<null>	<null>	<null>	<null>

6. 기능구현 (2/3) – 받기

받기 API 수행 후 각 테이블 데이터 확인 (user_id = 11)

뿌리기 금액 테이블 :
fling_away_amt_mst

뿌리기 멤버 테이블 :
fling_away_mbr_mst

6. 기능구현 (3/3) – 배치

배치 수행 후 뿌리기 만료 처리 각 테이블 확인

뿌리기 테이블 :
fling_away_mst

변경 전

	1
fling_id	F00000000001
room_id	R00000000001
token	k5d
fling_user_id	10
fling_dt	2020-09-19 13:38:00.877751
end_dt	2020-09-19 13:48:00.877751
fling_amt	3000
people_cnt	3
fling_stat_cd	1
del_yn	N
create_user_id	10
create_dt	2020-09-19 13:38:00.877751
create_pg_name	FlingService.flingAway
update_user_id	<null>
update_dt	<null>
update_pg_name	<null>

변경 후

	1
fling_id	F00000000001
room_id	R00000000001
token	k5d
fling_user_id	10
fling_dt	2020-09-19 13:38:00.877751
end_dt	2020-09-19 13:48:00.877751
fling_amt	3000
people_cnt	3
fling_stat_cd	3
del_yn	N
create_user_id	10
create_dt	2020-09-19 13:38:00.877751
create_pg_name	FlingService.flingAway
update_user_id	99999999
update_dt	2020-09-19 13:48:29.772055
update_pg_name	FlingTaskService.expireToken

< 배치 실행 결과 >

```
2020-09-19 22:48:29.757  INFO 63930 --- [   scheduling-1] com.fling.task.FlingTaskService          : 토큰 만료 배치] 시작
2020-09-19 22:48:29.773  INFO 63930 --- [   scheduling-1] com.fling.task.FlingTaskService          : 토큰 만료 배치] 사용자 송금 처리 API 호출 부분.
2020-09-19 22:48:29.774  INFO 63930 --- [   scheduling-1] com.fling.task.FlingTaskService          : 토큰 만료 배치] userId=10, amt=2473
2020-09-19 22:48:29.774  INFO 63930 --- [   scheduling-1] com.fling.task.FlingTaskService          : 토큰 만료 배치] 사용자 송금 처리 API 호출 부분.
2020-09-19 22:48:29.774  INFO 63930 --- [   scheduling-1] com.fling.task.FlingTaskService          : 토큰 만료 배치] 종료
```

Contents

7. 단위테스트 (1/4)

테스트 방식 1 - http 파일, 정상 응답 및 Validation 검증

```
###############################################################
### 1.기본동작확인
###############################################################
### 뿌리기
POST http://localhost:5000/fling
Content-Type: application/json
X-USER-ID: 10
X-ROOM-ID: R00000000001

{
  "amt": 3000,
  "cnt": 3
}
> {%
    client.test("뿌리기 정상 응답", function() {
        client.assert(response.status === 200, response.status + " 오류응답발생")
    })
    client.test("뿌리기 결과 토큰 확인", function() {
        var token = response.body.token
        client.assert(token !== undefined && token.length === 3, token + " 토큰오류발생")
    })

    client.global.set("rToken", response.body.token);
%}
```

```
HTTP Requests                                  802 ms
  http://localhost:5000/fling                   63 ms
    Response                                     0 ms
    Post-processor                              50 ms
    Tests                                       13 ms
      뿌리기 정상 응답                            9 ms
      뿌리기 결과 토큰 확인                       4 ms
  http://localhost:5000/fling                   45 ms
    Response                                     0 ms
    Post-processor                              32 ms
    Tests                                       13 ms
      받기 정상 응답                              9 ms
      받기 결과 금액 확인                         4 ms
  http://localhost:5000/fling                   44 ms
    Response                                     0 ms
    Post-processor                              31 ms
    Tests                                       13 ms
      받기 정상 응답                              9 ms
      받기 결과 금액 확인                         4 ms
```

7. 단위테스트 (2/4)

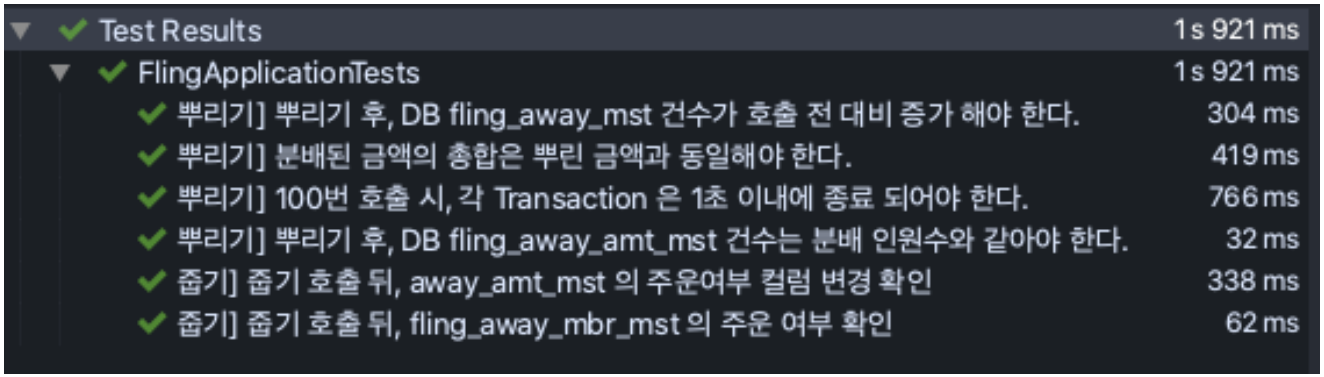
테스트 방식 2 - jUnit 통한 DB 정합성 검증

```java
@Test
@DisplayName("뿌리기] 뿌리기 후, DB fling_away_mst 건수가 호출 전 대비 증가 해야 한다.")
void flingAwayTest1() {

    // given
    int userId = 10;
    String roomId = "R00000000001";
    long amt = 1000;
    int cnt = 5;

    // when
    int cntBf = mapper.selectCntFlingAwayMst();
    service.flingAway(userId, roomId, amt, cnt);
    int cntAf = mapper.selectCntFlingAwayMst();

    // then
    log.info("bf={}, af={}", cntBf, cntAf);
    assertThat(cntAf).isGreaterThan(cntBf);
```



7. 단위테스트 (3/4)

항목	대화방ID	사용자ID	금액	인원수	결과
[뿌릴금액 체크] 0원 유효하지 않은 뿌릴금액이 입력 되었습니다.	R00000000001	10	0	3	"timestamp": "2020-09-19T13:55:04.920+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "유효하지 않은 뿌릴금액이 입력되었습니다.", "path": "/fling"
[뿌릴금액 체크] 1조 유효하지 않은 뿌릴금액이 입력되었습니다.	R00000000001	10	10000000000	3	"timestamp": "2020-09-19T13:57:24.135+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "유효하지 않은 뿌릴금액이 입력되었습니다.", "path": "/fling"
[뿌릴인원 체크] 유효하지 않은 뿌릴인원이 입력되었습니다.	R00000000001	10	100000	0	"timestamp": "2020-09-19T14:03:47.397+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "유효하지 않은 뿌릴인원이 입력되었습니다.", "path": "/fling"
[뿌릴금액인원 체크] 인원수 보다 적은 금액이 입력되었습니다.	R00000000001	10	1	2	"timestamp": "2020-09-19T14:04:15.383+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "인원수 보다 적은 금액이 입력되었습니다.", "path": "/fling"
[받기] 요청하신 뿌리기가 존재하지 않습니다.	R00000000003	30			"timestamp": "2020-09-19T14:05:20.582+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "요청하신 뿌리기가 존재하지 않습니다.", "path": "/fling"
[받기] 뿌린 사람은 받기가 불가능합니다.	R00000000003	20			"timestamp": "2020-09-19T14:06:36.203+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "뿌린 사람은 받기가 불가능합니다.", "path": "/fling"

7. 단위테스트 (4/4)

항목	대화방ID	사용자ID	금액	인원수	결과
[받기] 뿌리기 마감이 완료되었습니다.	R00000000001	20			"timestamp": "2020-09-19T14:07:22.825+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "뿌리기 마감이 완료되었습니다.", "path": "/fling"
[받기] 해당뿌리기를 받을 수 있는 사용가 아닙니다.	R00000000002	19			"timestamp": "2020-09-19T14:08:09.778+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "해당뿌리기를 받을 수 있는 사용가 아닙니다.", "path": "/fling"
[받기] 이미 받은 사용자입니다. 받기는 한번만 가능합니다.	R00000000002	21			"timestamp": "2020-09-19T14:09:39.092+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "이미 받은 사용자입니다. 받기는 한번만 가능합니다.", "path": "/fling"
[조회] 뿌린 본인만 조회가 가능합니다	R00000000002	21			"timestamp": "2020-09-19T15:48:33.855+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "뿌린 본인만 조회가 가능합니다.", "path": "/fling"
[조회] 7일이 지난 뿌리기는 조회가 불가능합니다.	R00000000001	10			"timestamp": "2020-09-19T15:49:23.254+00:00", "status": 400, "error": "Bad Request", <1 internal call> "message": "7일이 지난 뿌리기는 조회가 불가능합니다.", "path": "/fling"

END.