

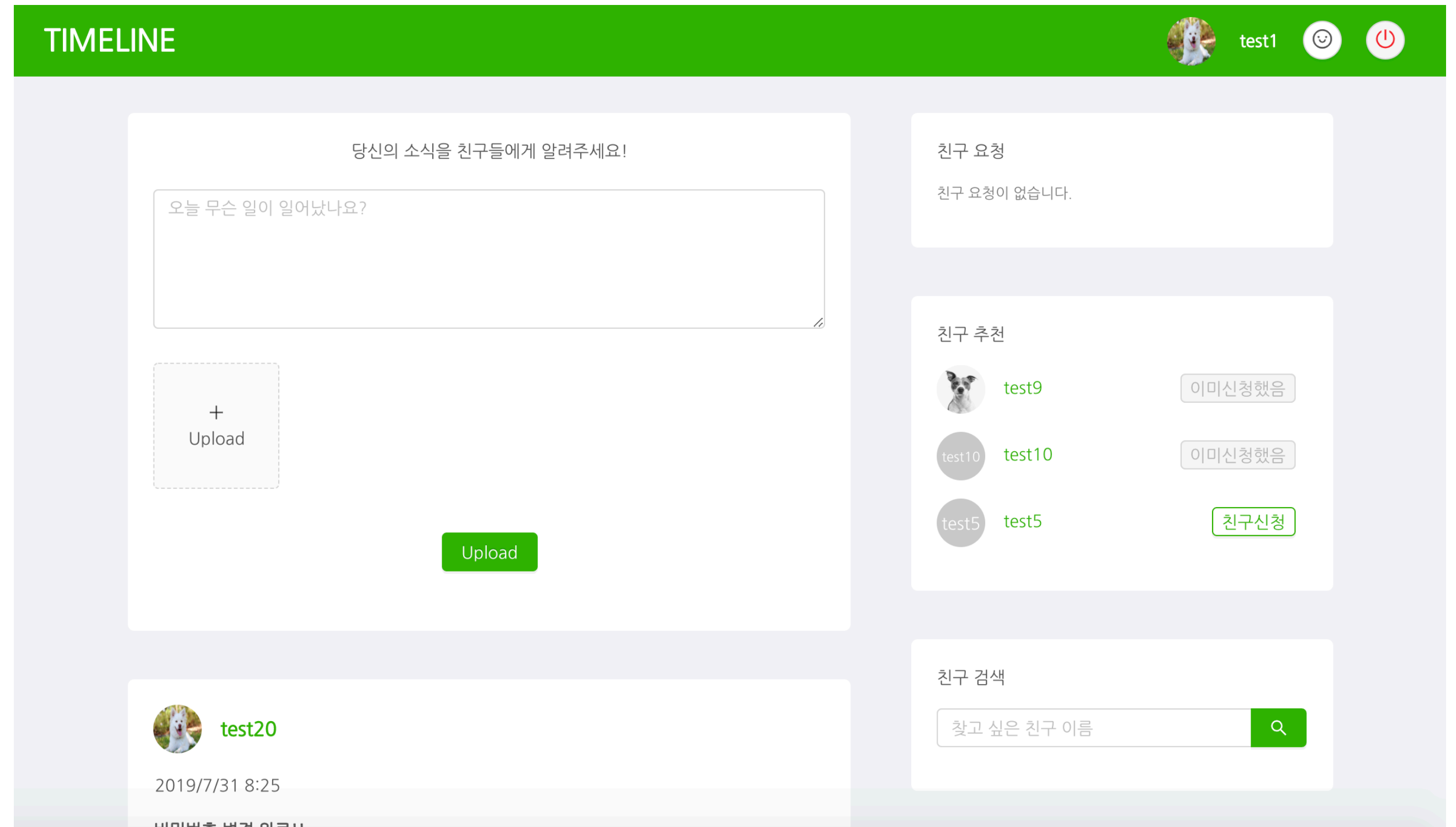
TIMELINE

MADE BY

JUNG YEONGMO

yeongmoj94@gmail.com

DESCRIPTION



친구맺기, 친구검색, 친구추천, 게시물 작성, 친구
들의 게시물 모아보기 등의 기능을 갖춘

SNS 타임라인

HOW TO USE

TIMELINE

[Register Now!](#)

[Or login now!](#)

이메일 주소가 계정 아이디로 사용되며, 비밀번호 및
비밀번호 찾기용 질문과 대답을 함께 입력 후 회원가
입 할 수 있습니다

HOW TO USE

TIMELINE

이메일 주소를 입력 해 주세요

Submit Email Address

Or login now!

TIMELINE

보물 1호는?

Submit Answer

Or login now!

TIMELINE

새로운 비밀번호를 입력해 주세요

Reset Password

Or login now!

회원 가입 시 입력했던 비밀번호 찾기용 질문
을 통해 비밀번호를 초기화 할 수 있습니다

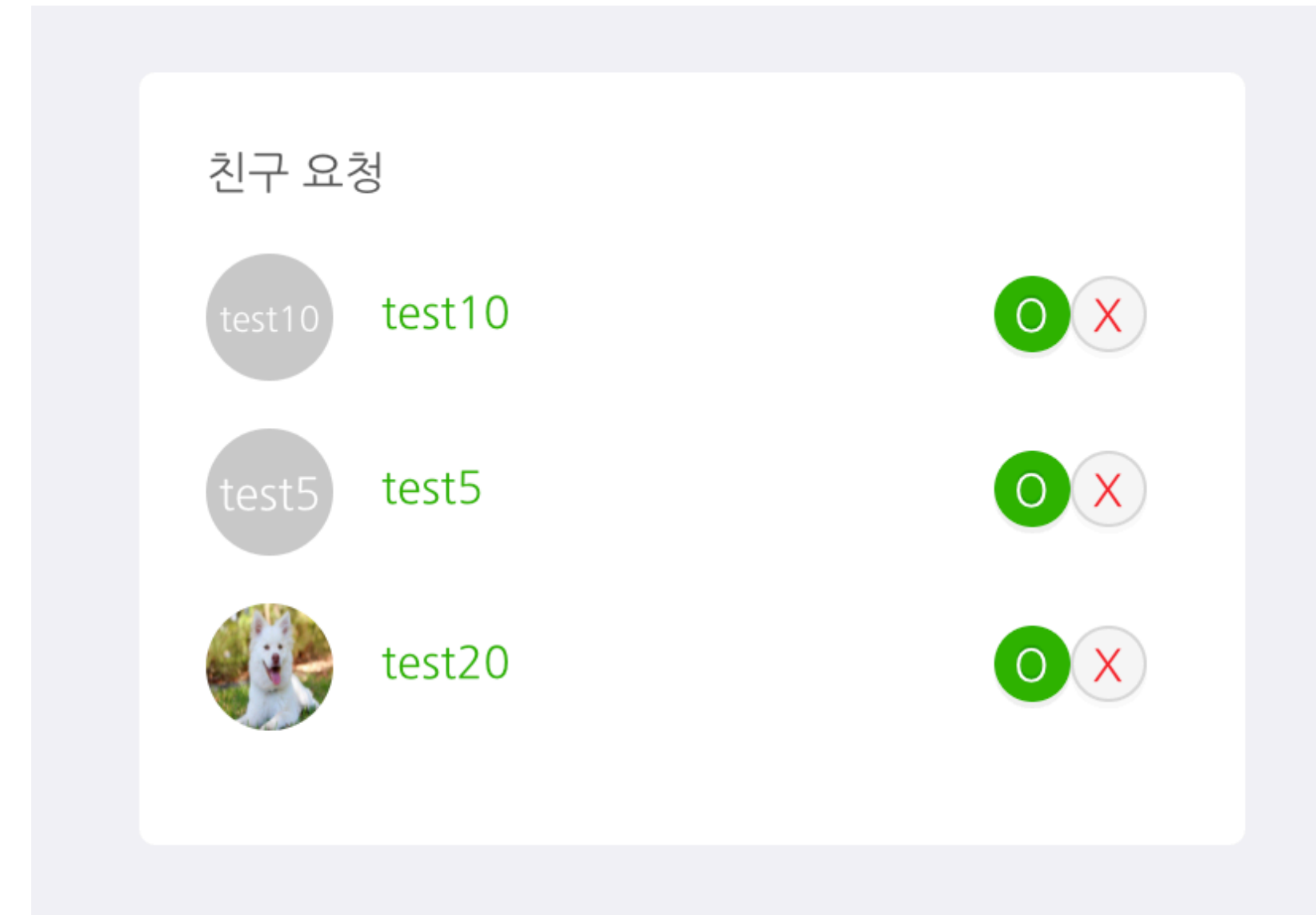
HOW TO USE

TIMELINE

[Forgot password](#)[Or register now!](#)

이메일과 비밀번호의 조합으로
로그인 할 수 있습니다

HOW TO USE



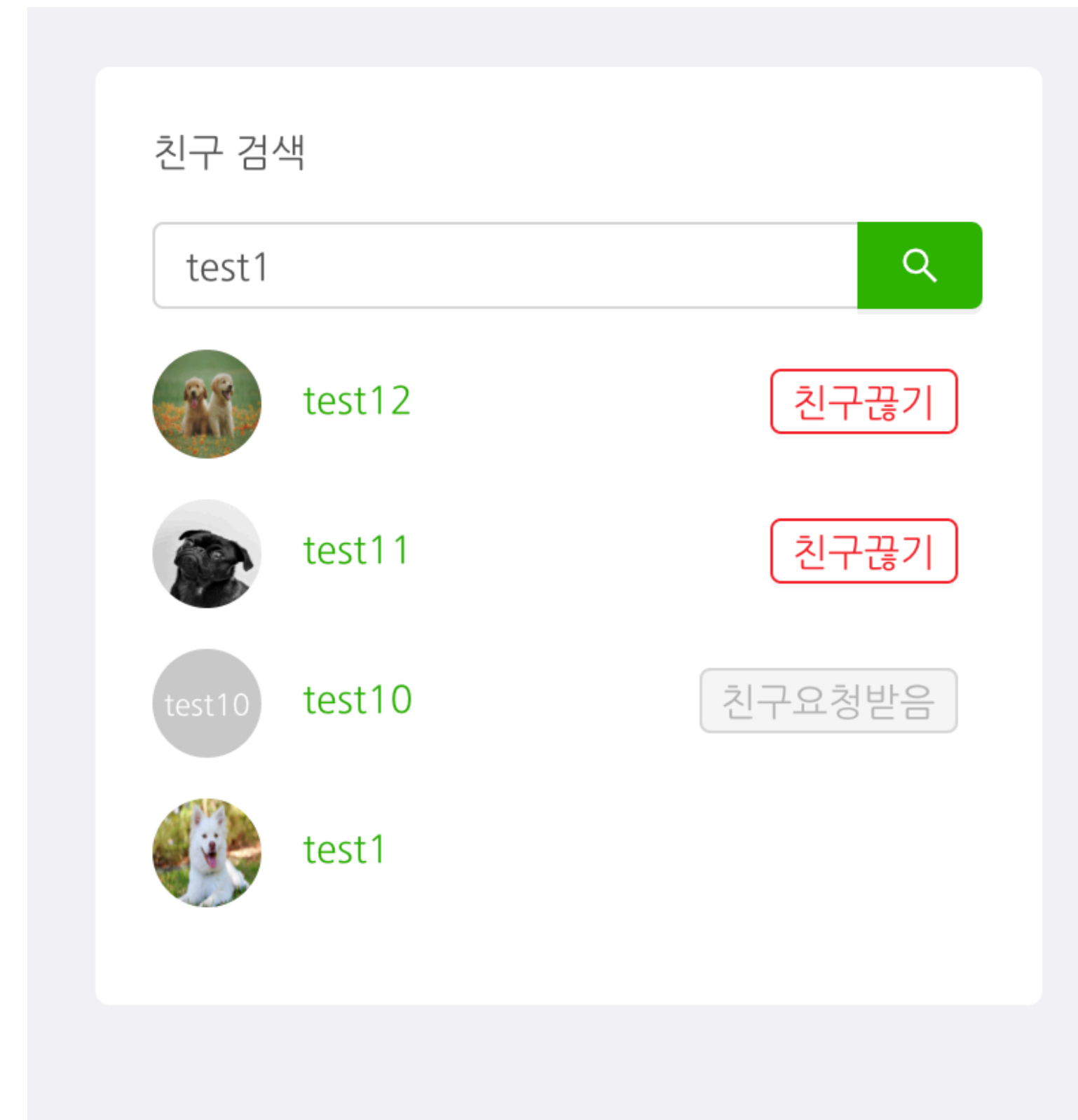
나에게 들어온 친구 요청들을 수락
하거나, 거절할 수 있습니다

HOW TO USE



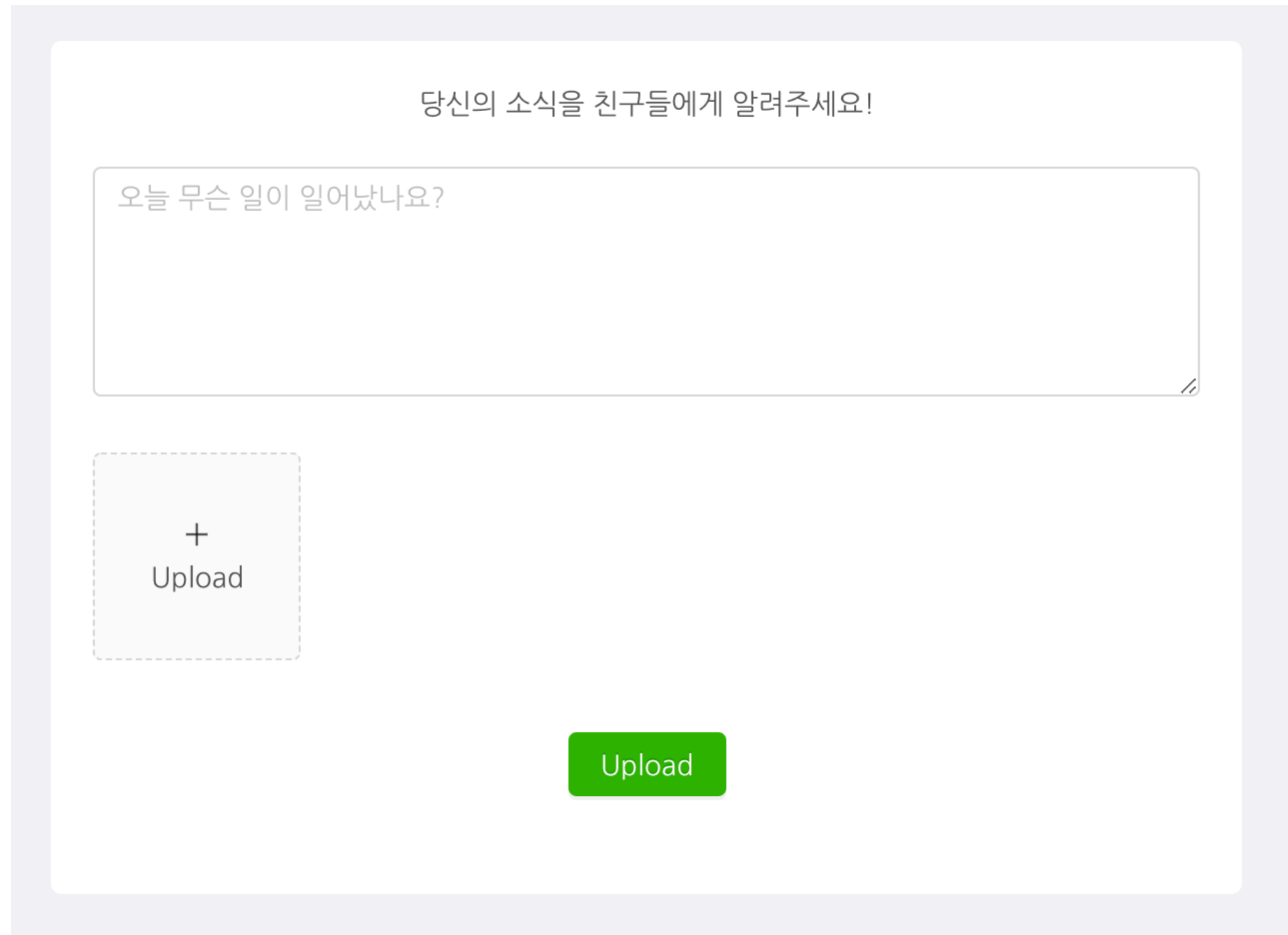
아직 나와 친구가 아닌 회원들 중에서
친구 추천을 받을 수 있습니다

HOW TO USE



친구들의 이름을 통해 회원 검색이
가능합니다

HOW TO USE



당신의 소식을 친구들에게 알려주세요!

오늘 무슨 일이 일어났나요?

+
Upload

Upload

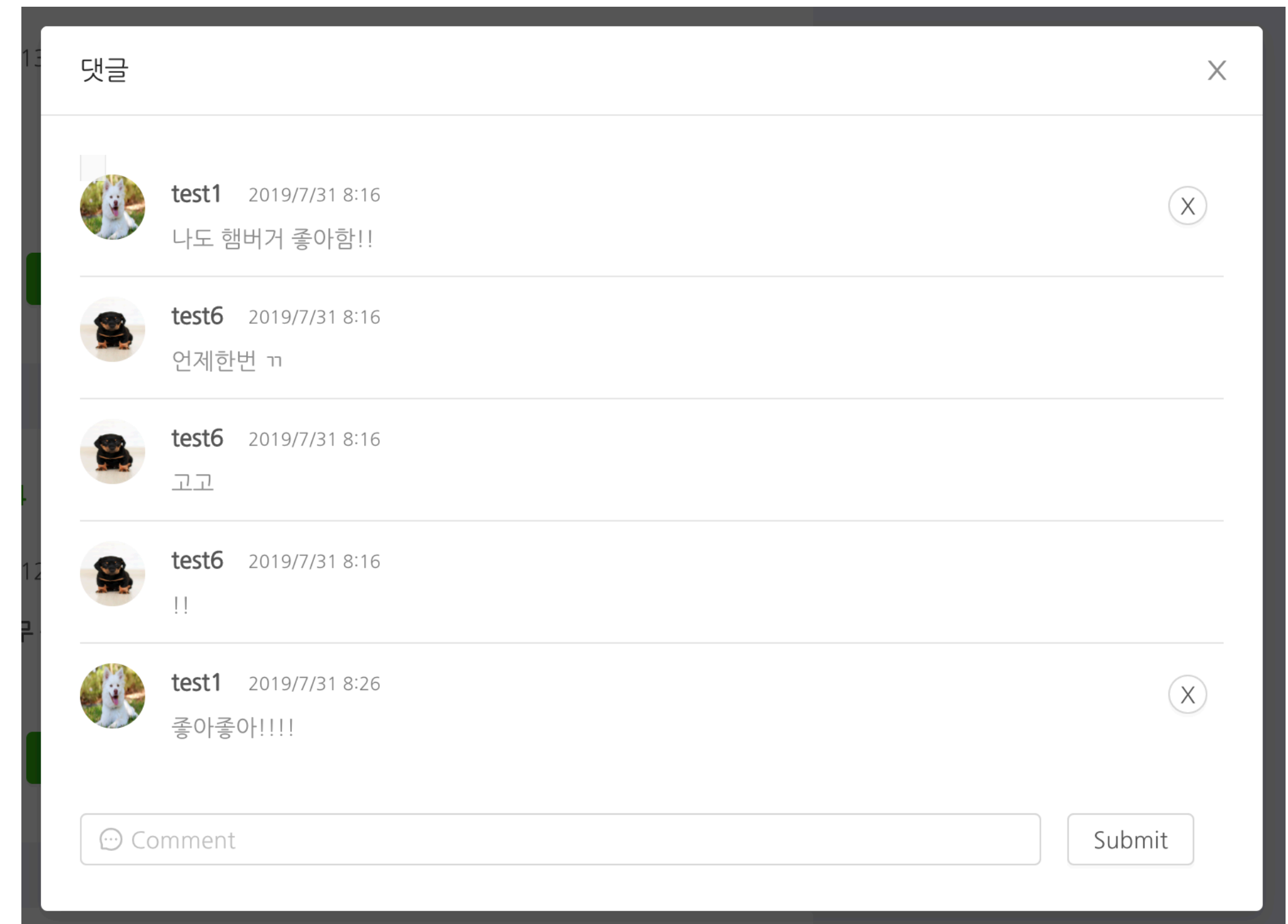
사진을 첨부 해 친구들에게 나의 소식을
공유 할 수 있습니다

HOW TO USE



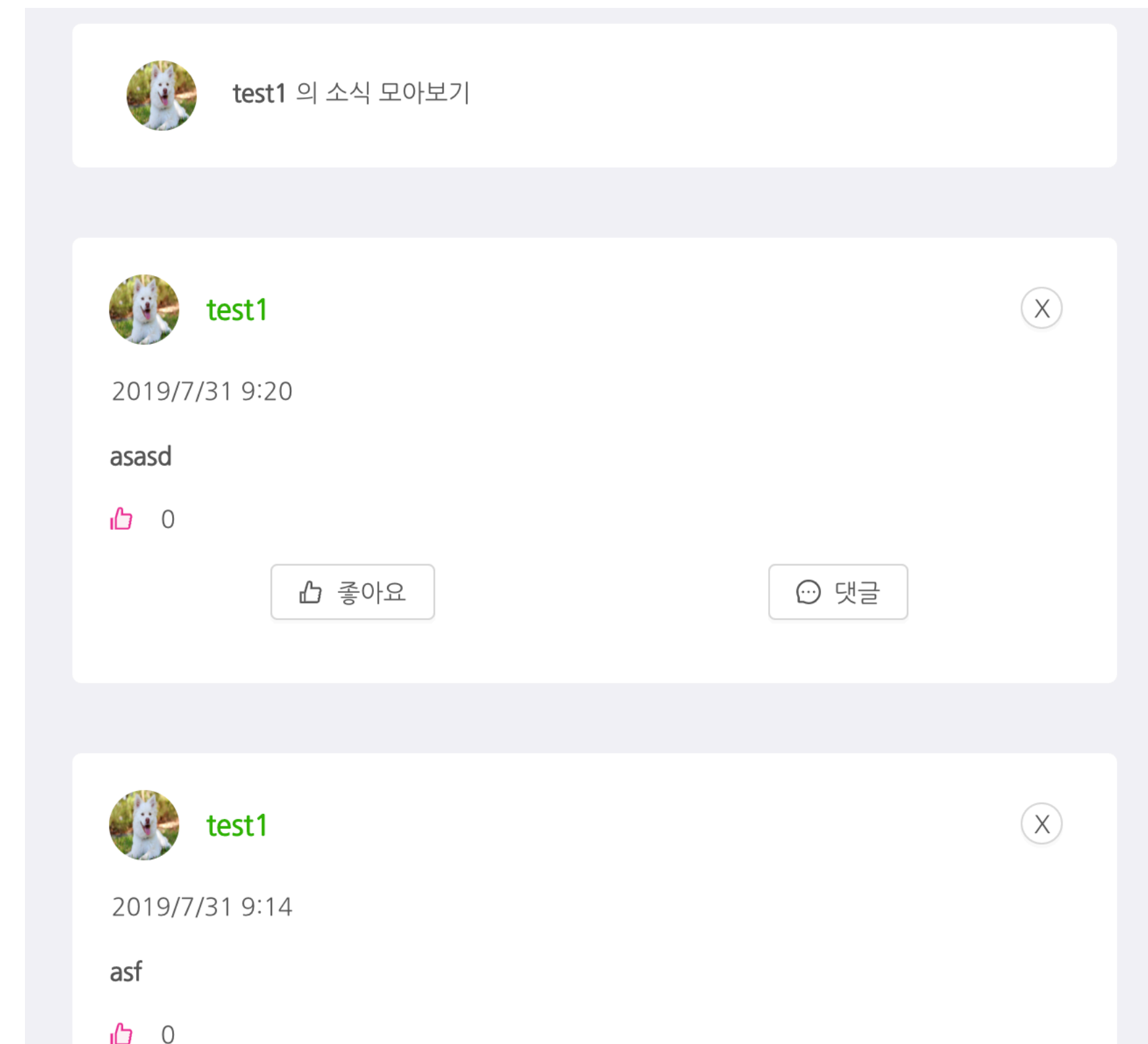
친구들의 소식들을 시간순으로 모아 보고, 반응(좋아요, 댓글)을 남길 수 있습니다 (타임라인)

HOW TO USE



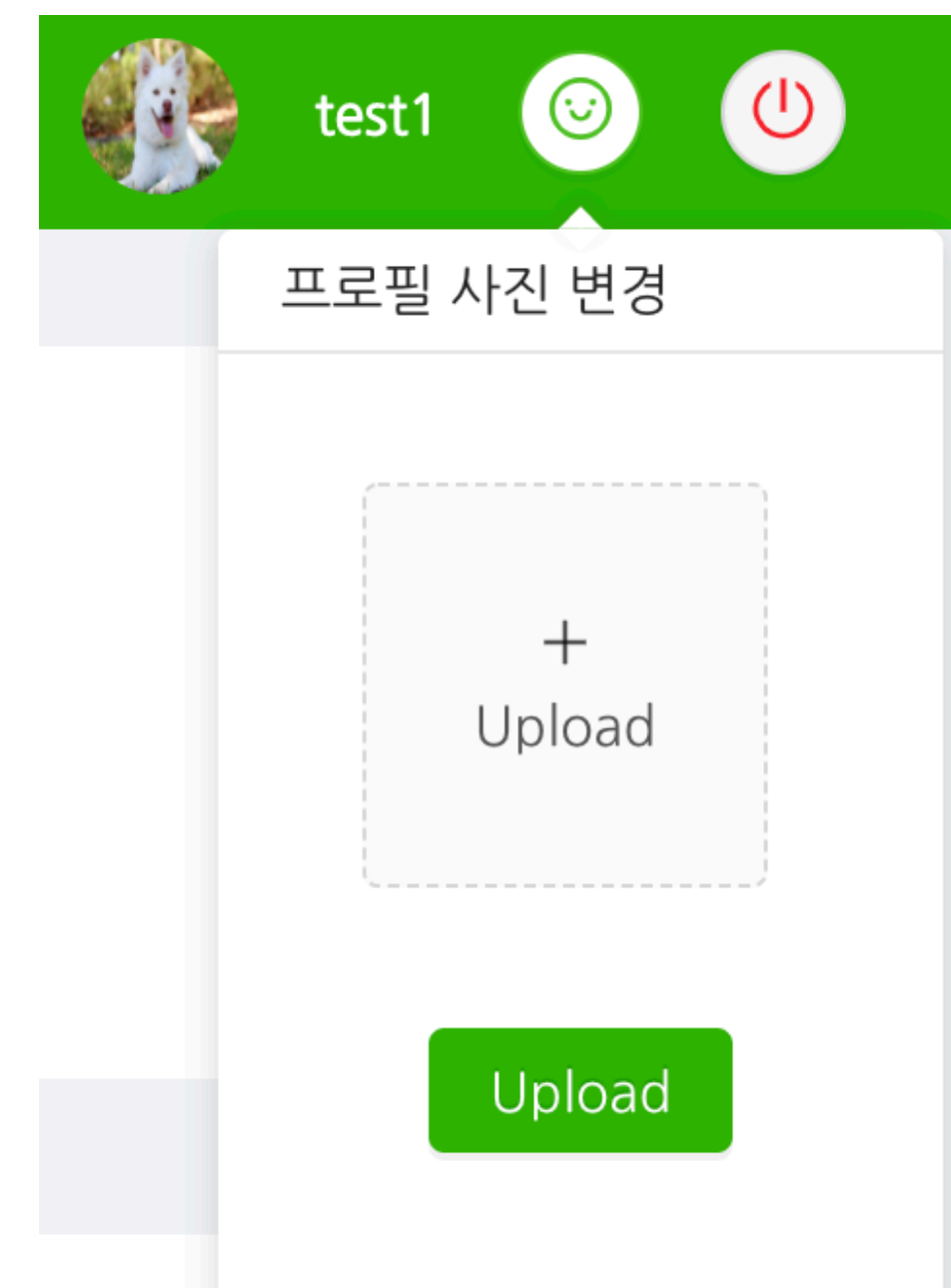
친구들과 댓글을 통해 대화를 나눌 수 있습니다

HOW TO USE



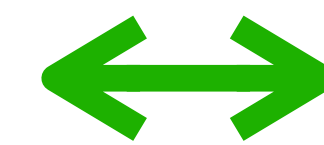
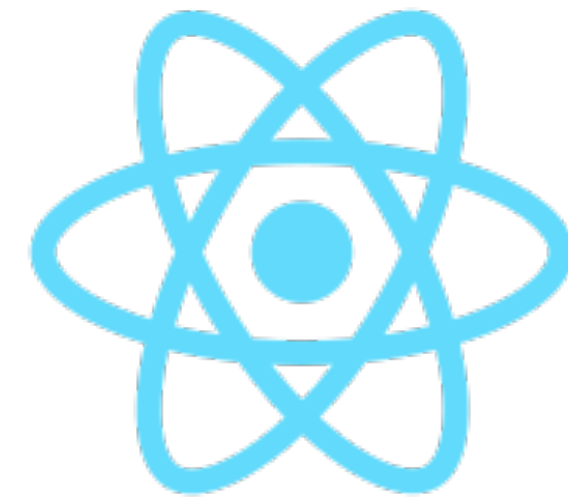
원하는 친구(혹은 나)의 소식만을 모아 볼 수
있습니다

HOW TO USE



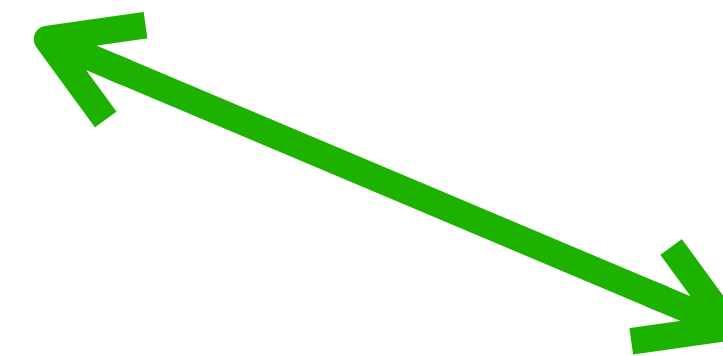
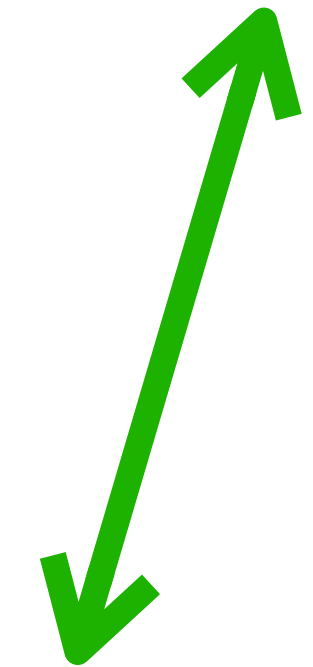
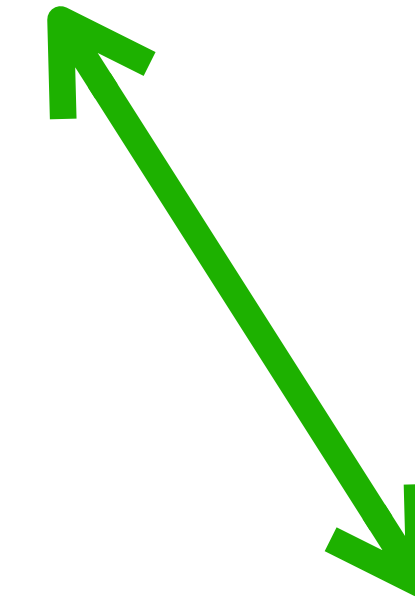
상단 바를 통해 프로필 사진 변경이 가능합니다

DETAIL



MySQL[®]

Or any RDBMS



spring
boot

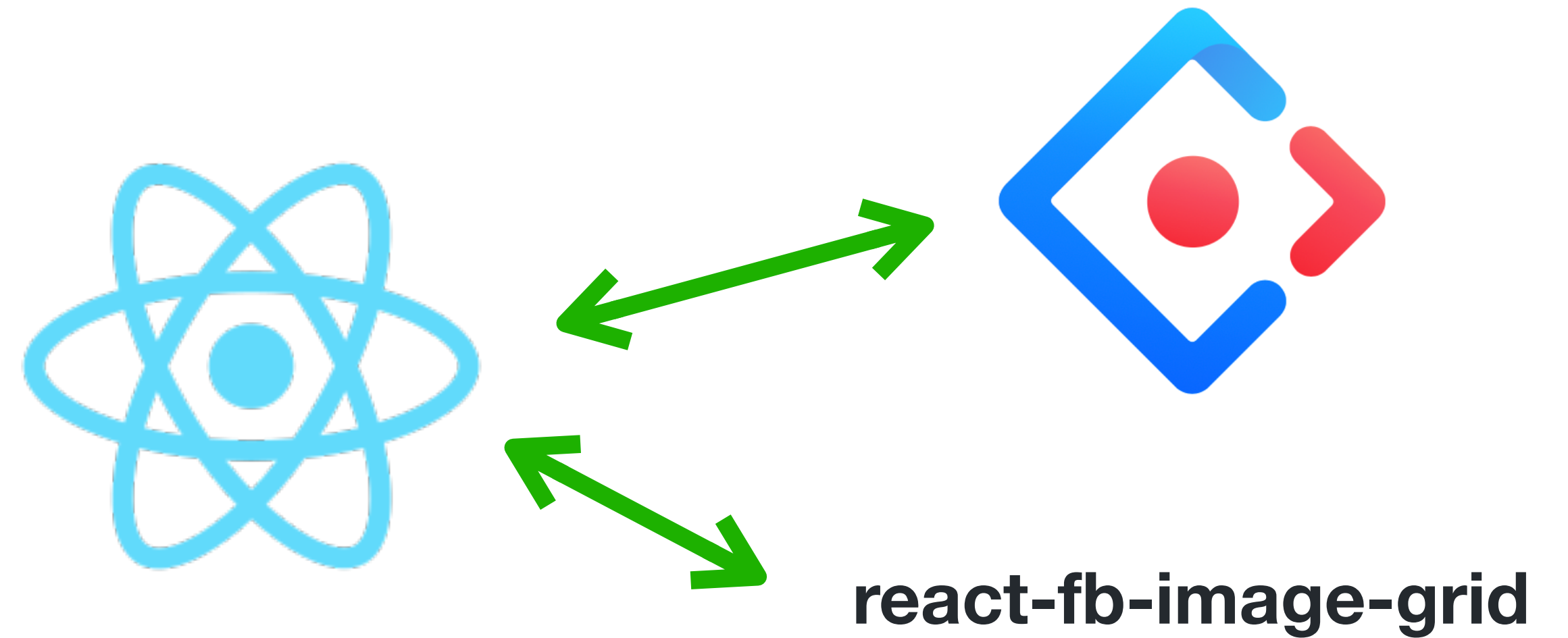
프론트엔드 React

백엔드 Spring boot

그래프 데이터베이스 S2Graph

관계형 데이터베이스 MySQL

DETAIL



UI 프레임워크로는 Ant Design을 사용했으며 사진의 정렬을 위해 react-fb-image-grid 라이브러리를 사용 하였습니다.

GRAPH DATABASE

나의 소식, 친구들의 소식을 관계형 데이터베이스를 이용해 저장하게 될 경우,

친구들의 소식을 모으는 방법 중 한가지 방법은, 메모리 내에 각 회원들 별로 저장소를 할당해 주고, 자신의 친구들이 소식을 업데이트 할 때마다 (게시글을 업로드 할 때마다) 게시글들을 저장소에 추가 해주는 방식이 있습니다.

해당 방법은 회원 수가 많아질수록 필요한 저장소 (혹은 메모리)의 부담이 커지는 단점이 있습니다.

GRAPH DATABASE

또 다른 방법 중 한가지 방법은, 회원이 자신의 친구들의 소식을 모아 보는 기능을 요청할 경우에 (자신의 타임라인을 방문하고자 하는 경우에) 친구들의 소식을 실제로 모아서 응답해주는 방식으로 볼 수 있는데,

정규화 되어 있는 데이터베이스에서 친구들의 소식을 얻고자 한다면, 최소 두 번의 join 연산을 필요로 하는데 (나 -> 친구들 -> 글 목록), 친구들의 수가 많아지고, 글의 수가 많아짐에 따라 아주 큰 부담이 될 수 있습니다.

GRAPH DATABASE

이 문제점들을 모두 해결해 주는 것이 그래프 데이터베이스입니다.

그래프 데이터베이스는, 자료구조의 그래프와 같이 정점과, 간선으로 데이터를 관리하는 데이터베이스로써, 질의를 통해 어느 한 정을 시작으로 하는 BFS 탐색을 수행 할 수 있습니다.

따라서, 전체 데이터가 아닌 내가 필요로 하는 데이터만을 대상으로 탐색을 수행하기 때문에, 관계형 데이터베이스의 join보다 부담이 훨씬 덜합니다.

GRAPH DATABASE

S2Graph에서는, 정점의 정의를 Column이라고 하며, 간선의 정의를 Label이라고 합니다.

회원이라는 Column의 실제 객체인 정점 A와 정점B를 친구라는 Label의 실제 객체인 간선으로 연결해 줌으로써 'A와 B는 친구관계이다'를 표현할 수 있습니다.



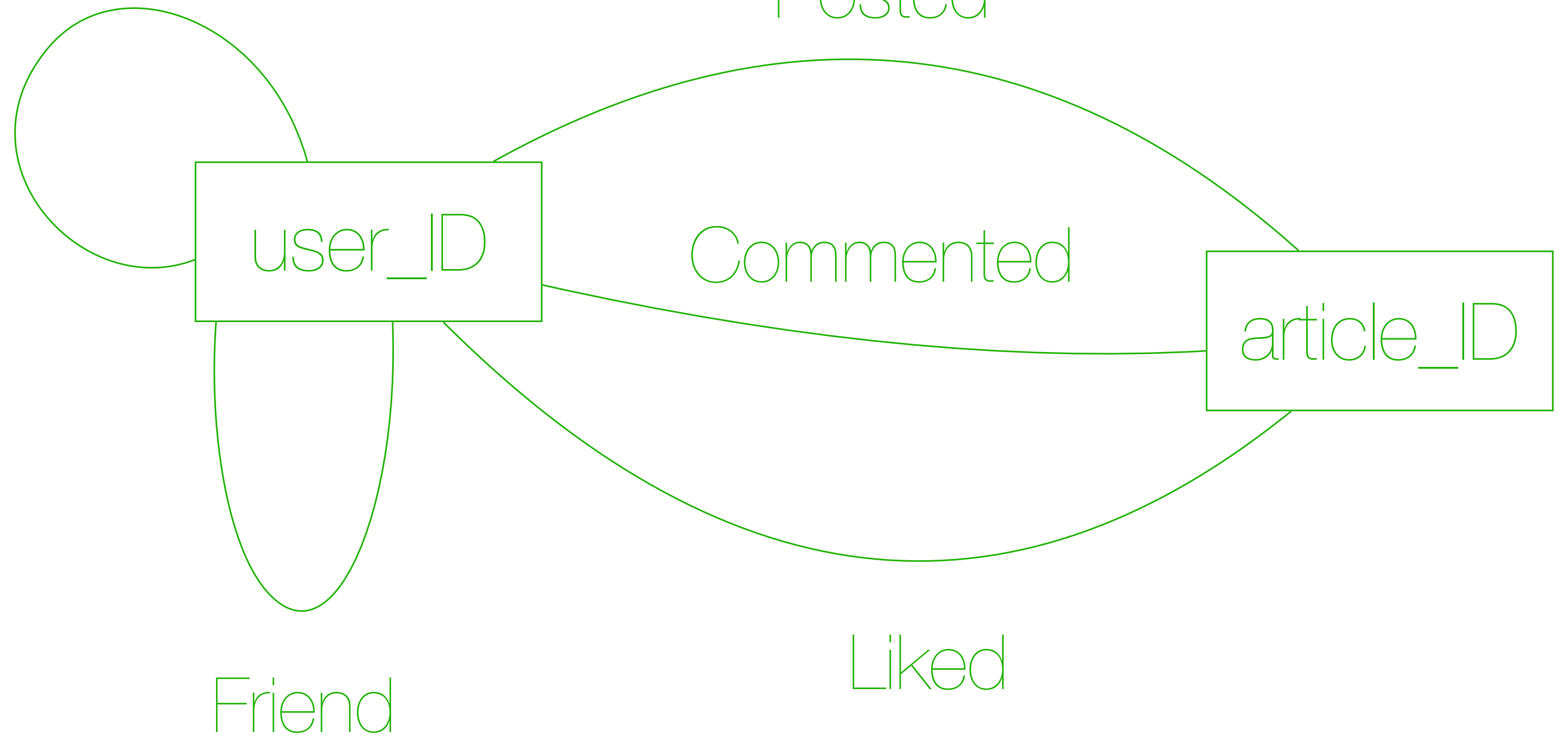
GRAPH DATABASE

본 어플리케이션에서 사용 된 데이터 모델 입니다.

(정점은 사각형, 간선은 실선)

Friend Request

Posted



GRAPH DATABASE

user_ID Column

회원 한명을 나타내는 객체입니다. 회원의 고유 ID 번호만이 정점으로 저장되고, 나머지 회원의 정보들은 RDB에 저장 됩니다. 따라서, 로그인 혹은 회원가입 등의 회원 정보에 대한 접근은 모두 RDBMS를 통해 처리가 되며, 그래프 데이터베이스에는 ID만이 저장됩니다.

GRAPH DATABASE

article_ID Column

게시글(소식) 하나를 나타내는 객체입니다. 마찬가지로 이 정점에는 소식의 고유 ID번호만이 저장됩니다. 하지만 게시글의 다른 정보들은 RDB에 저장되지 않습니다.

GRAPH DATABASE

Posted Label

user_ID 정점에서 나와서 article_ID 정점으로 들어가는 간선입니다. user_ID에 해당하는 회원이 article_ID에 해당하는 소식을 작성 했다는 의미이며, 이 간선에 글 작성의 모든 정보가 들어 있게 됩니다. 간선에는 props라고 하는 추가 정보를 저장 할 수가 있는데, 여기에 글 내용, 사진 등의 정보들이 저장됩니다.

따라서, 소식 열람을 위해선 이 간선을 탐색하면 됩니다.

GRAPH DATABASE

Commented Label

user_ID 정점에서 나와서 article_ID 정점으로 들어가는 간선입니다. user_ID에 해당하는 회원이 article_ID에 해당하는 소식에 댓글을 작성 했다는 의미이며, 또한 props를 통해 댓글의 내용이 저장됩니다.

해당 소식에 작성된 댓글들을 열람하고 싶으면,
이 간선을 탐색하면 됩니다.

GRAPH DATABASE

Liked Label

user_ID 정점에서 나와서 article_ID 정점으로 들어가는 간선입니다. user_ID에 해당하는 회원이 article_ID에 해당하는 소식에 좋아요를 눌렀다는 의미입니다. 따로 저장되는 추가 정보는 없습니다

해당 소식에 좋아요를 눌렀는지, 혹은 누른 사람이 몇명인지를 확인 하려면 이 간선을 탐색하면 됩니다.

GRAPH DATABASE

Friend Label

user_ID 정점에서 나와서 user_ID 정점으로 들어가는 무방향 간선입니다. user_ID에 해당하는 회원이 다른 user_ID에 해당하는 회원과 친구 사이라는 의미를 갖고 있는 간선입니다.

친구들의 소식을 불러오게 될 때, 본 간선을 먼저 탐색 하게 됩니다.

GRAPH DATABASE

FriendRequest Label

user_ID 정점에서 나와서 user_ID 정점으로 들어가는 방향 간선입니다. user_ID에 해당하는 회원이 다른 user_ID에 해당하는 회원에게 친구 요청을 보냈다는 의미를 갖고 있습니다.

친구 요청이 들어왔는지 알기 위해선, 본인의 user_ID 정점으로 들어오는 간선이 있는지 탐색하면 됩니다.

GRAPH DATABASE

질의

그래프 데이터베이스를 이용한 질의에는 insert, get, check, delete 등이 있습니다.

insert는 말그대로 정점 및 간선들을 추가하는 질의입니다. 예를 들어 Post label에 새로운 간선을 추가해줘 새로운 글을 등록해 주었다는 것을 표현, 저장 해 줄 수 있습니다. 본 프로젝트에서는 양 끝 정점을 회원 혹은 소식의 id로 하는 간선들만 등록해줄 것입니다.

Check는 해당 간선이 존재하는지를 얻는 질의입니다. 예를 들어 Liked label의 간선의 존재 유무를 통해 몇명이 해당 글에 좋아요를 눌렀는지, 혹은 해당 회원이 좋아요를 눌렀는지를 확인 할 수 있습니다.

GRAPH DATABASE

질의

delete는 존재하는 간선을 삭제해주는 연산으로써, friend label의 간선을 삭제해 주어 친구 관계를 끊어줄 수 있고, posted label의 간선을 삭제해 주어 소식의 삭제를 할 수 있습니다.

GRAPH DATABASE

질의

get은 타임라인을 구성하는 가장 중요한 질의라고 할 수 있는데, 질의의 양식은 다음과 같습니다.

일단 시작할 정점을 정합니다. 그 다음으로, 여러 step에 따라 탐색하고자 하는 간선들을 지정해 주게 되고, 이를 통해 BFS 탐색을 하게 됩니다.

예를들어 a정점에서 시작하고 step으로 b간선과 c간선을 지정했다면, a정점 -> b간선 -> b'정점 으로 이루어지는 간선 끝의 정점 집합 b'에서 출발하는 c간선의 집합이 질의의 결과가 됩니다.

따라서, a회원을 시작으로, friend간선 -> posted 간선 순으로 탐색 하면, 다른 필요 없는 데이터들의 탐색은 전혀 필요 없이 원하는 타임라인의 결과를 얻을 수 있습니다.

CODE



패키지 구조입니다.

Configuration : 스프링 설정 클래스들이 있는 패키지입니다.

Entity : JPA를 위한 entity 클래스 및 그래프 데이터베이스에 props로 저장 될 형식들로 이루어진 클래스들이 있는 패키지입니다.

CODE

Forresponse : 데이터에 저장된 전체 자료들과 프론트엔드에서 필요로 하는 자료들의 구조에는 차이가 있습니다. 이를 맞춰주기 위한 중간단계의 클래스들이 있는 패키지 입니다.

graphEntity : 그래프 데이터베이스의 질의를 위해 질의의양식을 맞춰주기 위한 클래스들이 존재하는 패키지 입니다. 질의는 json 형식으로 http요청을 통해 이루어집니다.

Interceptor : 프론트엔드에서 요청을 보낼 때, 권한 체크를 위한 인터셉터가 존재하는 패키지 입니다.

Repository : JPA repository interface가 존재하는 패키지 입니다.

Service : MVC 패턴의 service 클래스의 패키지 입니다.

Web : MVC 패턴의 controller 클래스의 패키지 입니다.

CODE

```
public class HttpFactory {  
  
    private HttpClientHttpRequestFactory factory;  
  
    private final String serviceName = "timeline";  
  
    private final String graphUrl = "http://localhost:9000/";  
  
    private final String insertEdgeUrl = graphUrl + "graphs/edges/insertWithWait";  
  
    private final String checkEdgeUrl = graphUrl + "graphs/checkEdges";  
  
    private final String getEdgesUrl = graphUrl + "graphs/getEdges";  
  
    private final String deleteEdgeUrl = graphUrl + "graphs/edges/deleteWithWait";  
  
    private final String deleteAllEdgeUrl = graphUrl + "graphs/edges/deleteAllWithWait";  
  
    private final String userIDColumn = "user_id";  
  
    private final String articleIDColumn = "article_id";  
  
    private final String postedLabel = "posted3";  
  
    private final String commentedLabel = "commented2";  
  
    private final String likedLabel = "liked2";  
  
    private final String friendLabel = "friend2";  
  
    private final String friendRequestLabel = "friendRequest";  
  
    private final long maximumTimeStamp = 9999999999999L;  
  
    private long primaryID;  
}
```

graphEntity 클래스의 HttpFactory 클래스 입니다. 백엔드 서버에서 S2Graph로는 http 통신을 통해 하게 되는데, 연결에 필요한 uri들이 이 클래스에서 변수로 관리 됩니다.

CODE

```
public class GetEdgeEntity implements RequestEdgeEntity{

    private LinkedList<Map<String, Object>> srcVertices;

    private LinkedList<Map<String, LinkedList<GetEdgeStep>>> steps;

    public GetEdgeEntity(){
        srcVertices = new LinkedList<Map<String, Object>>();
        srcVertices.add(new HashMap<String, Object>());
        steps = new LinkedList<Map<String, LinkedList<GetEdgeStep>>>();
    }

    public void setSrcVerticesServiceName(String serviceName) {
        srcVertices.get(0).put("serviceName", serviceName);
    }

    public void setSrcVerticesColumnName(String columnName) {
        srcVertices.get(0).put("columnName", columnName);
    }

    public void setSrcVerticesID(long id) {
        srcVertices.get(0).put("id", id);
    }

    public void setNextStep(GetEdgeStep nextStep) {
        LinkedList<GetEdgeStep> nextStepList = new LinkedList<GetEdgeStep>();
        nextStepList.add(nextStep);
        Map<String, LinkedList<GetEdgeStep>> step = new HashMap<String, LinkedList<GetEdgeStep>>();
        step.put("step", nextStepList);
        steps.addLast(step);
    }
}
```

Get 질의를 위한 양식들은 이 GetEdgeEntity 클래스로 손쉽게 작성 할 수 있습니다.

CODE

```
public class InsertEdgeEntity implements RequestEdgeEntity{  
    private long timestamp;  
    private long from;  
    private long to;  
    private String label;  
    private Props props;  
}
```

Insert 질의에 필요한 props들은 InsertEdgeEntity 클래스를
통해 손쉽게 작성 할 수 있습니다.

CODE

```
@RestController
public class PhotoController {

    //파일이 저장될 경로
    static final String PATH="./photos/";

    /*
    * 실질적으로 파일을 저장 해주는 메소드
    * 유저의 아이디 + 날짜 + 랜덤 값의 조합으로 파일이름을 만든 뒤, 이 이름으로 지정된 경로에 저장한다
    * 저장이 완료된 후 저장 된 파일이름을 반환
    * 반환된 파일이름을 프론트에서 받아서, 다른 정보들과 함께 다시 요청을 보낸다.
    * (해당 이미지의 파일명을 DB에 저장해서 관리하게 되는데,
    * 이미지 업로드와 DB에 해당 파일명 저장은 별개)
    */
    public String uploadPhoto(MultipartFile file, int userID) throws IOException {
        String originalFileName = file.getOriginalFilename(); //확장자를 얻기 위해업로드 될 파일의 원래 이름을 받아온다.

        //마지막 인덱스부터 하나씩 앞으로 뺄기며 처음 나오는 . 의 위치를 알아낸다. 이 뒤로 있는 문자열이 확장자가 된다.
        int indexOfDot = originalFileName.length()-1;
        while (originalFileName.charAt(indexOfDot)!='.')
            indexOfDot--;

        //업로드 된 시간을 형식에 맞춰서 파일이름에 사용한다.
        Date d = new Date();
        SimpleDateFormat formattedDate = new SimpleDateFormat("yyyyMMddHHmmssSSS");
        String formattedDateString = formattedDate.format(d);

        //파일이름 결정 = 유저 아이디 + 날짜 + 랜덤 + 확장자
        String fileName = userID + formattedDateString + (int)(Math.random()*100)+
            originalFileName.substring(indexOfDot);

        //파일을 경로에 저장한다.
        byte[] data = file.getBytes();
        FileOutputStream stream = new FileOutputStream(PATH+fileName);
        stream.write(data);
        stream.close();

        return fileName; //파일이름 반환
    }
}
```

사진을 업로드 할 때는 사진은 원하는 경로에 파일로 저장한 뒤에, 데이터베이스에는 파일명만 저장합니다. 요청에 따라 프론트로 파일명만을 전송한 뒤, 프론트에서 다시 사진만을 요청하게 됩니다.

```
export const login = ({ user, token }) => {
  localStorage.setItem('USER', JSON.stringify(user))
  localStorage.setItem('token', token)
}

export const getUser = () => {
  const user = localStorage.getItem('USER')
  try {
    return JSON.parse(user)
  } catch (e) {
    return null
  }
}
```

```
export const logout = () => {
  localStorage.removeItem('USER')
  localStorage.removeItem('token')
}

export const getToken = () => {
  try {
    return localStorage.getItem('token')
  } catch (e) {
    //이거 토큰없으므로 처리 해 주기
    return null
  }
}
```

CODE

프론트엔드에서 서버로 요청을 보내게 될 때에, 권한 체크를 하게 되는데, JWT를 이용해 권한 체크를 하게 됩니다. 로그인에 성공하면, 토큰을 response body에 담아 전송하게 됩니다. 프론트에서는 이를 받아 로컬스토리지에 저장해 놓습니다. 다음 요청부터는, 저장해 두었던 토큰을 요청 헤더에 담아 요청하게 되고, 백엔드 인터셉터에서 이 토큰의 유효성 검사를 하게 됩니다.

VIDEO

CLICK!

GITHUB

CLICK!

DEMO PAGE

CLICK!

LICENSE

MIT

RESOURCE

[SPRING BOOT](#)

[REACT](#)

[S2GRAPH](#)

[ANT DESIGN](#)

[REACT-FB-IMAGE-GRID](#)

THANK YOU!
AND HAVE A NICE DAY!