OP.GG CLONE

LOLify .GG

장성훈 박진우

팀원소개

장성훈(FE / 팀장)　박진우 (BE)

프론트엔드 개발
디자인
프로토타입 제작(카카오 오븐)
화면 구현(뼈대 구현)
[redacted]
유저 [redacted] 구현
리액트
백엔드에서 받은 데이터를 state로 설정해서 state가
변경 됐을 때 적절하게 UI를 변경
모델 구현
컨트롤러 구현

백엔드 개발
기획
기존 OP.GG 웹사이트 기능 분석
차별점 및 기능 추가할 것들
OP.GG, [redacted] 데이터 추출 및
데이터베이스 저장 [redacted]
node.js express 서버 구현
Riot api GET 구현
정제한 데이터 클라이언트에게 전달

프로젝트 동기 및 목표

✓ OP.GG 사이트 클론

✓ 기존 피해량 산정 그래프에 차별점인 '**기여도 피해량**'을 추가

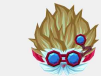
Q 기여도 피해량이란?

A 기여도 피해량 = 총피해량 - (외부 버프로 얻은 증가량) + (나의 버프로 인한 외부의 증가량)

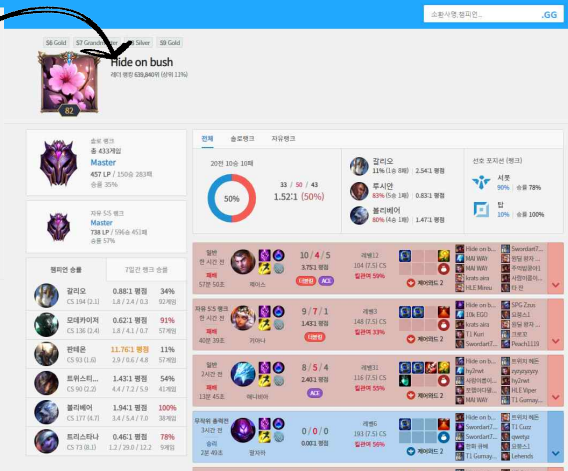
Q 왜 만들게 되었는가?

A 버프 스킬을 주로 이루는 몇몇 챔피언들은 상대적으로 딜 스킬 위주의 챔피언들에 비해
총 피해량이 낮은 편
사실 이는 굉장히 자연스러운 현상이지만 현재 리그오브레전드 관례상 총 피해량이
게임의 기여도를 판단하는 큰 잣대가 되는 경우가 많고, 이에 따라 버프 챔피언들은
상대적으로 총 피해량이 낮아 실질적인 기여도보다 낮은 평가와 대우를 받고 있다고
생각 이를 개선해 보고자 기여도 피해량이란 시스템을 도입하기로 결정.

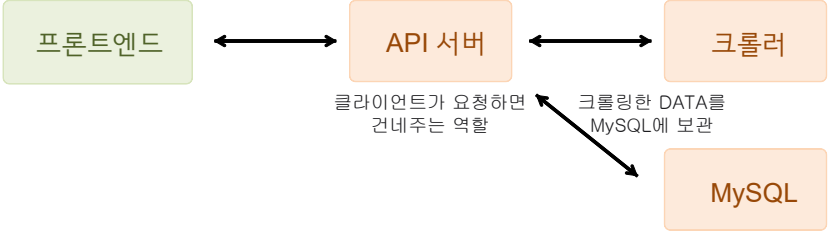
Project overview



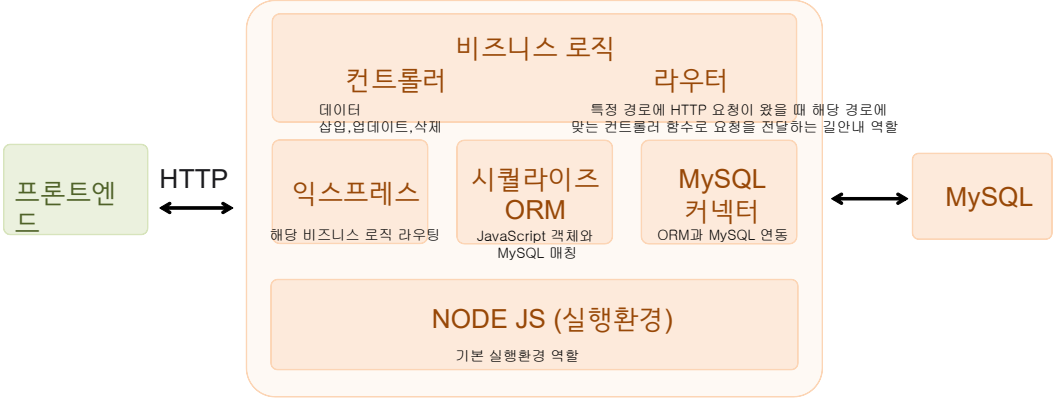
소환사명을 입력해주세요.



LOLify 구조


프론트엔드
API 서버
크롤러
클라이언트가 요청하면
건네주는 역할
크롤링한 DATA를
MySQL에 보관
MySQL

API 서버의 구조


비즈니스 로직
컨트롤러
데이터
삽입,업데이트,삭제
라우터
특정 경로에 HTTP 요청이 왔을 때 해당 경로에
맞는 컨트롤러 함수로 요청을 전달하는 길안내 역할
프론트엔드
HTTP
익스프레스
해당 비즈니스 로직 라우팅
시퀄라이즈 ORM
JavaScript 객체와
MySQL 매칭
MySQL 커넥터
ORM과 MySQL 연동
MySQL
NODE JS (실행환경)
기본 실행환경 역할

기여도피해량 그래프 예시화면

23. 11. 07 ~ 23.11. 10

크롤링 결정

Riot 공식 API를 사용해서 데이터를 GET 해오고자 했지만
공식 API 키 발급엔 2주정도 걸림
발급전까지 전적 사이트 크롤링을 통해서 필요한 데이터를 가져오기로 결정.

LOL 전적 검색 사이트를 조사한 결과 OP.GG는 크롤링에 대해 정책적으로는 허용했지만 JavaScript를 통해 동적으로 페이지를 만드는 사이트였기 때문에 정적인 HTML을 크롤링해 오는 작업에 있어 막히는 부분이 있어 FOW.KR로 타겟을 변경.

크롤링 정책을 알기 위해서 URL 뒤에 robots.txt를 입력하면 해당 사이트가 크롤링을 허용하는지 알 수 있음.

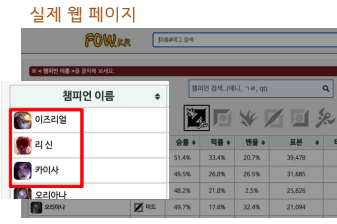

```
User-agent: *
Disallow: /

User-agent: AhrefsBot
Disallow: /

User-agent: Mediapartners-Google
Allow: /

User-agent: Googlebot
User-agent: Bingbot
User-agent: Naverbot
User-agent: Yeti
Allow: /
Disallow: /gb/
```

23. 11. 07 ~ 23.11. 10

크롤링 구현 과정

crawler 폴더를 새로 만들고 HTML 을 Load하기 위해 Axios 라이브러리를 사용.
Load 한 HTML 을 파싱하기 위해 cheerio 라이브러리를 사용.
FOW 사이트에서 챔피언의 승률, 픽률, 밴율, 표본수 데이터를 가져옴.
FOW 사이트에서 데이터를 CSS selector 문법으로 해당 요소를 선택, 자바스크립트 객체로 변환.
HTML 을 Load, Load한 HTML을 파싱, 파싱한 HTML을 자바스크립트 객체로 변환하는 코드를 모아 하나의 모듈로 만듦.

실제 웹 페이지



23. 11. 13 ~ 23.11. 17

FOW.KR 크롤링

FOW.KR 사이트는 정적인 HTML이 <페이지 소스 보기>에서 나타났기 때문에 크롤링을 정책적으로는 허용하지 않았으나, 크롤링은 가능.

```
</div><div style="position:relative; width:70px; height:64px; display:inline-bloc
    <A HREF='/stats/Rumble'>
    <IMG SRC='//z.fow.kr/champ/68_64.png' width=64>
    </A>
    <DIV style="position:absolute; width:62px; height:14px; bottom:0px; left:2px;
        <b>럼블</b>
    </DIV>
</div><div style="position:relative; width:70px; height:64px; display:inline-bloc
    <A HREF='/stats/Rell'>
    <IMG SRC='//z.fow.kr/champ/526_64.png' width=64>
    </A>
    <DIV style="position:absolute; width:62px; height:14px; bottom:0px; left:2px;
        <b>첼</b>
    </DIV>
</div><div style="position:relative; width:70px; height:64px; display:inline-bloc
    <A HREF='/stats/Rengar'>
    <IMG SRC='//z.fow.kr/champ/107_64.png' width=64>
    </A>
    <DIV style="position:absolute; width:62px; height:14px; bottom:0px; left:2px;
        <b>렝가</b>
    </DIV>
</div><div style="position:relative; width:70px; height:64px; display:inline-bloc
    <A HREF='/stats/Vi'>
    <IMG SRC='//z.fow.kr/champ/254_64.png' width=64>
    </A>
    <DIV style="position:absolute; width:62px; height:14px; bottom:0px; left:2px;
        <b>바이</b>
    </DIV>
</div><div style="position:relative; width:70px; height:64px; display:inline-bloc
    <A HREF='/stats/Blitzcrank'>
    <IMG SRC='//z.fow.kr/champ/53_64.png' width=64>
    </A>
```

23. 11. 13 ~ 23.11. 17

크롤러 구현에 어려웠던 점

크롤링한 데이터를 DB 테이블에 맞게 공백을 제거하고 줄바꿈을 제거하는 등의 처리과정에서 막힘이 생김.

크롤링한 결과 데이터

데이터를 처리하는 과정이 중요하다는 것을 느낌.

23. 11. 20 ~ 23.11. 24

크롤링한 데이터를 넣을 DB 구현

크롤링한 데이터를 DB에 넣기 위해서 DB와 TABLE을 만듦.
Sequelize 라는 ORM 을 사용해서 DB와 TABLE을 만듦.
TABLE을 만들기 위해서 Model을 작성.
Champion, stat, item TABLE을 만듦.
테이블에 들어갈 데이터들의 자료형과 primary key를 만듦.
테이블을 만든 후 ORM 과 MySQL을 연결하는 코드를 만듦.
우측 사진처럼 ORM 을 사용해서 JavaScript 코드를 짰는데 DB의 TABLE이 완성된 것을 확인.

테이블 모델 코드

```
const { DataTypes } = require('sequelize');

module.exports = (sequelize) => {
  return sequelize.define(
    'Stat',
    {
      id: {
        autoIncrease: false,
        type: DataTypes.INTEGER.UNSIGNED,
        allowNull: false,
        primaryKey: true,
      },
      level: {
        type: DataTypes.INTEGER.UNSIGNED,
        allowNull: false,
      },
```

23. 11. 20 ~ 23.11. 24

크롤링한 데이터를 넣을 DB 구현

ORM 을 사용하면 TABLE을 MySQL workbench 로 직접 TABLE을 다루지 않아도 간편하게 TABLE을 조작할 수 있다는 것을 알게 됨.
크롤링한 데이터를 TABLE에 넣기 위해서는 TABLE Model을 만들때 크롤링한 데이터의 자료형, 데이터의 용량을 맞춰서 설계해야 하는 것을 배움.

모델이 데이터베이스의 테이블이 된 모습

```
Database changed
mysql> show tables;
+--------------------+
| Tables_in_lolify |
+--------------------+
| Champion           |
| Item               |
| Stat               |
| Summary            |
+--------------------+
4 rows in set (0.00 sec)

mysql> desc champion;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int unsigned | NO   | PRI | NULL    | auto_increment |
| championName | char(10)     | NO   |     | NULL    |                |
| runes        | char(10)     | NO   |     | NULL    |                |
| skill        | char(10)     | NO   |     | NULL    |                |
| spell        | char(10)     | NO   |     | NULL    |                |
| item         | char(10)     | YES  |     | NULL    |                |
| stat         | int unsigned | NO   |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

23. 11. 27 ~ 23.12. 01

DB를 조작하는 컨트롤러 코드를 제작

DB를 조작하기 위해서 컨트롤러 코드를 제작.
클라이언트에서 특정 url 로 GET 요청이 들어왔을 때,
DB에서 해야하는 일들을 ORM 으로 제작하는 코드.

```
const { Champion } = require('../models');

async function getAll(req, res) {
  const result = await Champion.findAll();
  res.status(200).json({ result });
}

module.exports = {
  getAll,
  insertOrUpdate,
  remove,
};
```

TABLE과 컨트롤러 코드는 제작했지만,
TABLE에 데이터를 넣는 구간에서 막힘.
즉, 크롤링하는 모듈과 TABLE 코드 모듈이 2개가
하나로 연결되지 않고 두개가 따로 노는 문제가 발생

데이터의 자료형이나 이름, 용량도 신경써야 한다는
점도 느끼는 계기가 됨.

23. 12. 04 ~ 23.12. 08

Riot 공식 api 키 발급,
크롤링으로 얻기 힘든 데이터

실시간 매치 정보와 같이 갱신 기간이 비교적 짧은 데이터는
FOW도 자주 바뀌기 때문에 크롤링으로 얻기 힘들었음.

프로젝트 시작 후 2주 정도 지났을 무렵 Riot 공식 API 키가
발급 됐고, 크롤링으로 얻을수 없었던 데이터를 추가로
받아오기로 결정.

23. 12. 04 ~ 23.12. 08

어려웠던 점

프로젝트 기간 도중에 소환사 이름 대신 “암호화된 소환사이름(puuid)“을 사용한다는 소환사이름 폐지 정책이 실행되었고, 특정 소환사가 특정 시간에 플레이한 게임의 정보를 가져오기 위해선 기존에 방식에 puuid를 추가로 얻고 게임 id를 가져와야 하는데, response.body 가 request.header 에 추가되어야 하다보니 이해하는데 오래걸렸다.

배운 점

이 과정을 통해 여러가지의 GET 요청에 대해 알게 됨.

에러와 문제 정리

CORS 에러

서버에서 연결한 포트번호로 클라이언트가 접속했을 때 CORS 에러를 경험.
CORS란 Cross Origin Resource Sharing의 약자로 직역하면, 교차되는 출처 자원들의 공유.
풀이하자면 CORS는 한 도메인 또는 Origin의 웹 페이지가 다른 도메인 (도메인 간 요청)을 가진 리소스에 액세스 할 수 있게하는 보안 메커니즘.
  - 오류가 생긴 이유
프론트(클라이언트) : localhost:3000
서버 : localhost:4001
 ㄴㄴOrigin(오리진) 은 host와 port 를 포함한 데이터의 출처를 의미.
→ 지금 클라이언트와 서버의 포트가 다르기 때문에 오리진이 다르다고 판단됐기 때문에 CORS 에러가 발생.

해결방법

- npm 에서 `cors` 라는 라이브러리를 install 해서 import
- `const cors = require('cors');`
- `app.use(cors());` // 해당 코드는 모든 오리진에 대해서 cors 를 허용한다는 뜻.
- 위 코드를 통해서 cors 에러를 해결.

Custom Hooks

게임 데이터를 다루는 프로젝트의 특성상 이미지 파일,
데이터파일 등 다양한 파일들을 서버에서 클라이언트로
전달해야 했음.
그래서 로직은 같았지만, 파일 이름이 달라서 비슷한 코드를
여러 번 입력해야 했던 비효율이 있었음.
코드로 예를 들면,
'http://localhost:4001/api/itemImg/1001'
코드에서 itemImg가 아니라 championImg로
바뀌면 위의 코드를 전부 다시 쳐야 하는 상황이
발생, 비효율적이라고 생각하여 찾아본 결과
커스텀 훅이란 걸 알게 되었고 학습 후 적용.

느낀점

커스텀 훅을 통해서 같은 코드를 여러 번 쳐야하는 수고를
덜었던 것처럼 효율적으로 코드를 작성할 수 있는 방법을
찾았던 갚진 경험이었음.

```
// app.js
import {useState} from 'react';

const app = () => {
const [itemImgSrc, setItemImgSrc] = useState('');

useEffect(()=> {
const fetchItemImg = async () => {
      try {
        const response = await axios.get(
          'http://localhost:4001/api/itemImg/1001',
          {
            responseType: 'arraybuffer',
          }
        );
        const imgBlob = new Blob([response.data], {
          type: response.headers['content-type'],
        });
        const imgUrl = URL.createObjectURL(imgBlob);
        setItemImgSrc(imgUrl);
      } catch (error) {
        console.error('아이템을 불러오는 도중에 에러발생!', error);
      }
    };
    fetchItemImg();
  };
}

return(
  //
)
export defualt app;
```


Thank
you