

# React SPA를 활용한 경기도 사고 유형별 위치 조회 서비스



허창범: huhbe@naver.com

황준하: h\_leopold@naver.com

손수영: darae268@naver.com

임다영: gkfka8520@naver.com

박시은: qkrtldms813@gmail.com

김지은: rkam55@naver.com

---

▶ 시연 영상: <https://www.youtube.com/watch?v=q7TwANfWA9U>

🔄 GitHub: <https://github.com/GSITM-Team3/react-traffic-safety/blob/main/README.md>

# 목차

1. 프로젝트 배경 및 개발 환경
2. 핵심 기능 및 역할분담
3. 진행과정
4. 기획서 및 UML
5. 구현결과
6. 트러블 슈팅
7. 프로젝트 회고



# 1-1. 프로젝트 배경



## # 배경

자동차 보유가 보편화되어 현대 사회에서 편리한 교통수단으로 자리 잡고 있습니다.

자동차의 증가와 함께 교통사고 역시 발생 빈도가 높아지고 있습니다.


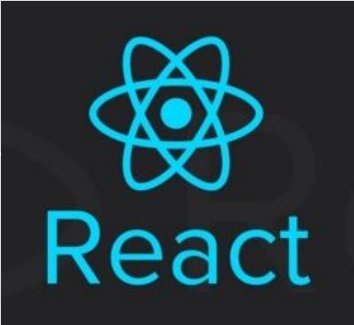




특히 특정 지역에서는 반복적으로 교통사고가 발생하는 **사고다발지역**이 존재합니다.

## # 목표

이 프로젝트는 사고다발지역의 **정보를 제공**함으로써 국민들이 사고다발지역에 대한 경각심을 갖고,

사고 예방과 안전 운전에 대한 의식을 높이는데 중점을 두어 **안전한 도로 환경을 구축**하는데 기여하고자 합니다.

# 1-2. 개발환경

분류	Front-end	
개발도구	HTML5 CSS JavaScript React	 
라이브러리	axios react react-dom react-icons react-router-dom react-virtualized	 <b>React Router</b> 
API	경기데이터드림 API 카카오 맵 API	 

## 2-1. 핵심기능

핵심기능	상세설명
사고유형/지역/사고연도에 따른 사고다발지 조회	<ul style="list-style-type: none"><li>• 사고 데이터 <b>검색 및 조회</b> 기능</li><li>• 사고 유형에 따른 지역/사고 연도를 조회할 수 있는 목록 제공</li></ul>
카카오 맵 연동	<ul style="list-style-type: none"><li>• 카카오 맵 API를 사용하여 <b>지도</b> 제공</li><li>• 원하는 조건에 해당하는 사고 데이터 클릭 시, 해당 위치에 대한 마커가 표시됨</li></ul>
렌더링 최적화	<ul style="list-style-type: none"><li>• react-virtualized 라이브러리의 List 컴포넌트 사용</li><li>• 스크롤 시에 <b>필요한 항목만 동적으로 렌더링</b></li><li>• API로 받은 대량의 데이터를 효율적으로 처리함</li></ul>
교통안전기관연락망	<ul style="list-style-type: none"><li>• 경기도 시/군별 교통안전기관 연락처 및 웹사이트 주소 제공</li><li>• 웹 관리자가 <b>CRUD</b>할 수 있으며, 이용자는 Read only</li></ul>

## 2-2. 역할분담

이름	역할
허창범(팀장)	<ul style="list-style-type: none"><li>프로젝트 구성 및 리딩</li><li>팀 Space 관리 (GitHub, Notion)</li><li><b>Select Box 값과 선택된 카테고리 데이터를 초기화</b>하는 기능 구현</li><li>교통안전기관연락망 CSS</li></ul>
황준하(부팀장)	<ul style="list-style-type: none"><li>프로젝트 구성 및 리딩</li><li>선택된 지역/사고연도에 따른 사고 데이터를 <b>필터링하는 검색 기능</b> 구현</li><li>교통안전기관연락망 CRUD 기능 구현</li><li>최종 Route 작업</li><li>시연 영상 작업</li></ul>
손수영	<ul style="list-style-type: none"><li>사고 유형 카테고리화 및 필터링 구현</li><li>검색 필수 값 미 입력 시 <b>Alert 기능</b> 구현</li><li>회의록 작성 및 PPT 제작</li></ul>

## 2-2. 역할분담

이름	역할
임다영	<ul style="list-style-type: none"><li>프로젝트 구성 및 리딩</li><li><b>카카오맵 API를 활용</b>한 Map구현</li><li>사고 유형 카테고리화 및 필터링 구현</li><li>사고 데이터 렌더링 최적화</li><li>교통안전기관연락망 CSS 및 최종 CSS 수정</li><li>시연 영상 작업 및 GitHub 작성</li></ul>
박시은	<ul style="list-style-type: none"><li>컴포넌트 취합(Daily)</li><li>사고 데이터 <b>렌더링 최적화</b></li><li>교통안전기관연락망 CRUD 기능 구현</li><li>교통안전기관연락망 CSS 및 최종 CSS 수정</li></ul>
김지은	<ul style="list-style-type: none"><li><b>CSS/템플릿 제작</b> 및 수정(Daily)</li><li>검색 필수 값 미 입력 시 Alert 기능 구현</li><li>교통안전기관연락망 CSS 및 최종 CSS 수정</li><li>PPT 제작</li></ul>



### 3. 진행 과정



# 회의록

구분	기간	활동
기획/설계	24/05/28(화)	<ul style="list-style-type: none"><li>주제 선정 및 역할 분담</li><li>자료 조사 (API, 통계 등)</li><li>공통 환경 구축 및 배포</li><li>컴포넌트 및 템플릿 설계</li><li>계획 수립 및 구현 기능 결정</li></ul>
구현	24/05/29(수) ~ 24/05/30(목)	<ul style="list-style-type: none"><li>CSS 작업 및 외부 API 연동</li><li>CRUD 기능 구현</li><li>1차 오류 수정</li></ul>
	24/05/31(금)	<ul style="list-style-type: none"><li>1차 테스트</li><li>Route 작업</li></ul>
최종 점검	24/06/03(월) ~ 24/06/04(화)	<ul style="list-style-type: none"><li>2차 오류 수정</li><li>최종 테스트</li><li>프로젝트 문서화</li></ul>
총 개발	24/05/28(화) ~ 24/06/04(화) 총 1주 소요	



# 4-1. 기획서

## # 조회 화면

경기도 사고 유형별 위치 조회

header> main으로 이동

사고 유형별 카테고리 클릭 > 클릭한 유형을 List에서 조회

무단횡단사고다발지

보행어린이사고다발지

자전거사고다발지

교통안전기관연락망

교통기관 연락망 클릭 > 검색 및 지도 화면에서 게시판 화면으로 변경

시도군

시도군

사고년도

사고년도

검색

초기화

검색 클릭 > 사고 리스트를 관할 지역과 년도로 조회를 할 수 있음

초기화 클릭 > 기존에 선택했던 검색 조건을 초기화 하고 지도의 위치도 초기화

사고리스트 1

사고리스트 2

사고리스트 3

사고리스트 4

리스트 클릭 시 > 해당 사고의 위치로 변경

기존 default 경기도청

경기도 사고 유형별 위치 조회

시도와 사고년도를 모두 선택해주세요

확인

무단횡단사고다발지

교통안전기관연락망

시도군

시도군

사고년도

사고년도

검색

초기화

시도와 사고 년도를 모두 선택하지 않고 검색을 클릭 > alert 창을 보임으로 사용자에게 필수조건을 알림

사고리스트 1

사고리스트 2

사고리스트 3

사고리스트 4

## # CRUD 화면

경기도 사고 유형별 위치 조회

무단횡단사고다발지

보행어린이사고다발지

자전거사고다발지

교통안전기관연락망

시군명

시군명을 입력하세요

링크

링크를 입력하세요

전화번호

전화번호를 입력하세요

입력버튼 클릭 > 좌측의 입력폼의 내용을 하단의 리스트에 추가

시군명

링크

전화번호

수정

삭제

도시 1

도시 1의 경찰청 링크

도시 1의 안전관리부서 전화번호

도시 2

도시 2의 경찰청 링크

도시 2의 안전관리부서 전화번호

도시 3

도시 3의 경찰청 링크

도시 3의 안전관리부서 전화번호

삭제버튼 클릭 > 해당 내용 삭제

수정버튼 클릭 > 상단의 입력폼으로 정보 전달 > 내용 수정 후 입력버튼 클릭 > 변경된 정보로 업데이트

1

2

3

4

경기도 사고 유형별 위치 조회

시군명, 링크, 전화번호를 모두 입력해주세요

확인

무단횡단사고다발지

교통안전기관연락망

시군명

시군명을 입력하세요

링크

링크를 입력하세요

전화번호

전화번호를 입력하세요

시군명

링크

전화번호

수정

삭제

도시 1

도시 1의 경찰청 링크

도시 1의 안전관리부서 전화번호

도시 2

도시 2의 경찰청 링크

도시 2의 안전관리부서 전화번호

도시 3

도시 3의 경찰청 링크

도시 3의 안전관리부서 전화번호

시군, 링크, 전화번호를 입력하지 않고 확인버튼을 클릭 > alert 창을 보임으로 사용자에게 필수조건을 알림

1

2

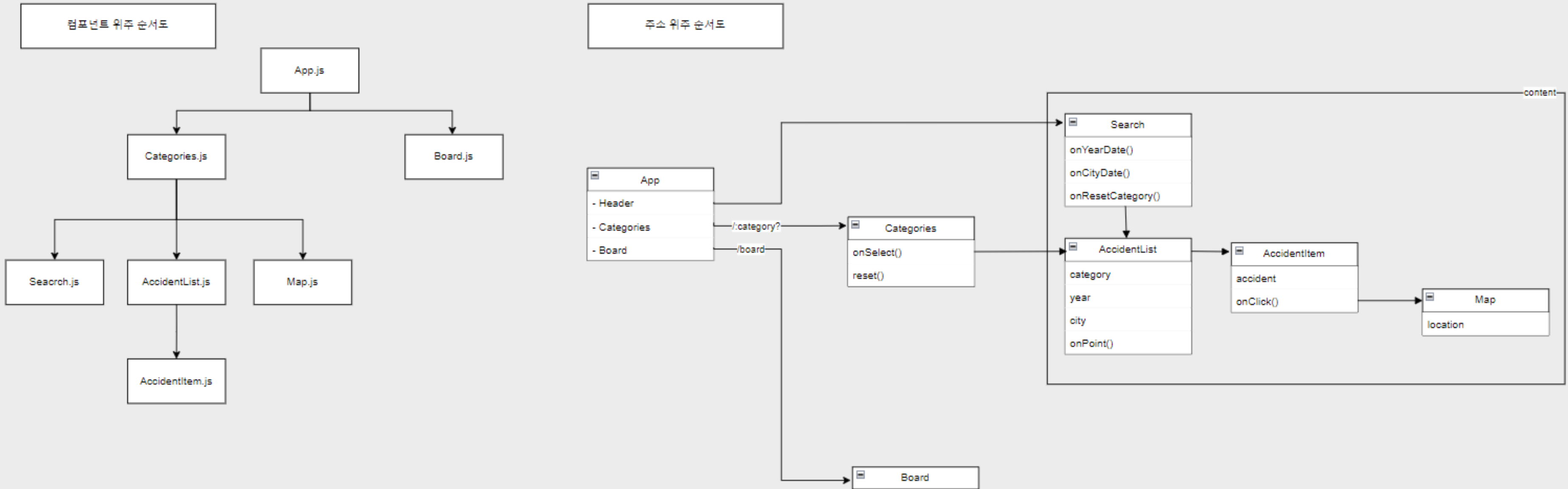
3

4

## 4-2. UML

### # UML

- UML 다이어그램을 활용한 컴포넌트 관계 및 순서도.



## 5. 구현결과

### # 필터링

- 선택된 사고 유형/시도/연도에 해당하는 **기존 데이터와 외부 API의 데이터를 비교**하여 일치하는 데이터를 **filter** 처리
- 조건을 통해 카테고리가 all인지, 시도/사고 연도가 선택됐는지 비교하여 해당되는 값을 반환

경기도 사고 유형별 위치 조회

① 무단횡단사고다발지

② 시도: 시도군  
사고연도: 사고년도  
검색  
초기화

④ 사고유형: 무단횡단사고다발지  
행정자치별: 고양시  
년도: 2019

사고유형: 무단횡단사고다발지  
행정자치별: 고양시  
년도: 2019

사고유형: 무단횡단사고다발지  
행정자치별: 광주시  
년도: 2019

사고유형: 무단횡단사고다발지  
행정자치별: 광주시  
년도: 2019

③ 시도: 시도군  
사고연도: 가평군  
고양시  
과천시  
광명시  
광주시  
구리시  
군포시  
김포시  
남양주시  
동두천시  
부천시  
성남시  
수원시  
시흥시  
안산시  
양주시  
양평군  
여주시  
연천군

④ 시도: 광명시  
사고연도: 2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022

무단횡단사고다발지

시도: 남양주시  
사고연도: 2015  
검색  
초기화

사고유형: 무단횡단사고다발지  
행정자치별: 남양주시  
년도: 2015

① // 카테고리만 선택했을 때의 필터링

```
const accidentCategoryFilter =
  accidents &&
  accidents.filter((accident) => accident.ACDNT_DIV_NM === category);
```

② // city와 year만 선택했을 때의 필터링

```
const accidentSearchSelectedFilter =
  accidents &&
  accidents.filter(
    (accident) => accident.SIGUN_NM === city && accident.ACDNT_YY === year
  );
```

③ // 세 개의 조건에 대한 필터링

```
const accidentAllSelectedFilter =
  accidents &&
  accidents.filter(
    (accident) =>
      accident.SIGUN_NM === city &&
      accident.ACDNT_YY === year &&
      accident.ACDNT_DIV_NM === category
  );
```

④ // 필터링된 사고 목록

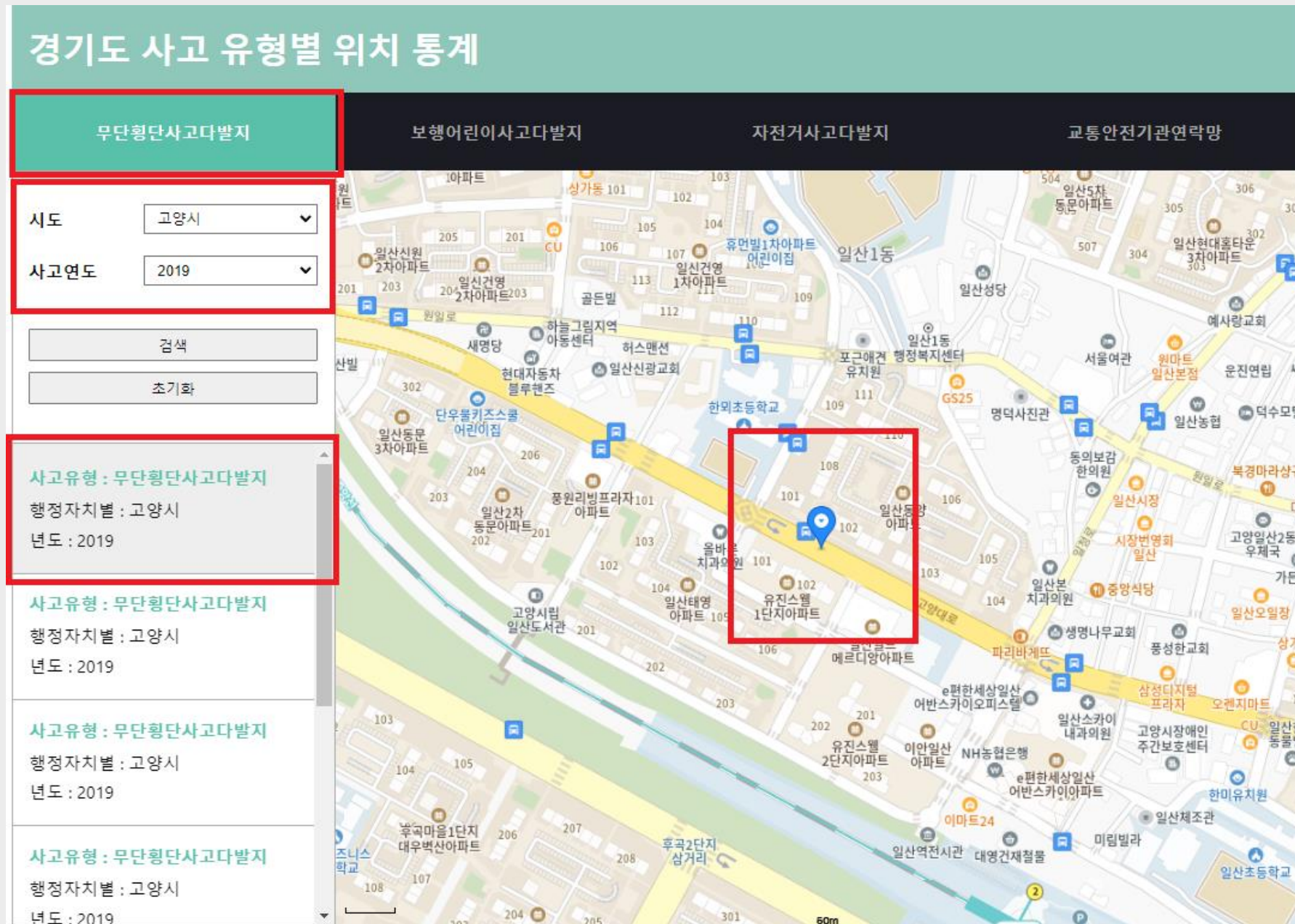
```
const filteredAccidents =
  category === 'all'
    ? !city && !year
      ? allAccidents
      : accidentSearchSelectedFilter
    : city && year
      ? accidentAllSelectedFilter
      : accidentCategoryFilter;
```



# 5. 구현결과

## # 초기화

- 초기화 버튼 클릭 시, 선택된 검색 박스 및 지도를 초기화하는 기능



```
const handleReset = () => {
  setSelectedCity('');
  setSelectedYear('');
  // 기본값으로 설정한 빈 문자열('')을 상위 컴포넌트로 전달하여 초기화합니다.
  onCityData('');
  onYearData('');
  onResetCategory(); // 카테고리 초기화
  navigate('/'); // URL 초기화
  onPoint({
    LAT: '37.2893525',
    LOGT: '127.0535312',
  });
};
```

handleReset 함수는 Select 옵션 값과 그에 해당하는 데이터를 초기 값으로 설정

특정 경로를 지정하는 **useNavigate**를 사용하여 경로를 초기화함

초기값으로 지정한 위도/경도로 위치 설정

```
// 카테고리 초기화 상태 관리
const [resetCategory, setResetCategory] = useState(false);
const onResetCategory = useCallback(() => {
  setCategory('all');
  setResetCategory(true);
  setTimeout(() => setResetCategory(false), 0);
}, []);
```

useCallback 함수를 사용하여 특정 상태를 재설정 함

category는 'all'로 리셋

**setTimeout**으로 resetCategory를 빠르게 false로 변경하여 일시적인 상태 변화 발생

```
<div className="categoryBox">
  <ul>
    {categories.map((c) => (
      <NavLink to={`/${c.name}`} key={c.name}>
        <li
          className={`category ${
            selectedCategory === c.name ? 'active' : ''
          }`}
          key={c.name}
          onClick={() => {
            setSelectedCategory(c.name);
            onSelect(c.name);
          }}
        >
          {c.name}
        </li>
      </NavLink>
    ))}
  </ul>
</div>
```

selectedCategory가 지정한 이름값을 가질 경우,

active로 설정한 CSS값이 적용되므로 초기화 버튼을 통해 적용된 CSS값이 해제됨



# 5. 구현결과

## # 카카오맵 연동

- 원하는 조건에 해당하는 사고 데이터 클릭 시, 해당 위치로 좌표 이동



### 스크립트 파일 읽어오기 함수 정의

```
const script = document.createElement('script');
script.src = src;
script.addEventListener('load', () => {
  resolve();
});
script.addEventListener('error', (e) => {
  reject(e);
});
document.head.appendChild(script);
};
```

### 카카오 맵 읽어오기

```
useEffect(() => {
  const lat = parseFloat(location.LAT);
  const logt = parseFloat(location.LOGT);
  const myScript = new Script(
    'https://dapi.kakao.com/v2/maps/sdk.js?autoload=false&appkey=
  );
```

### Default 좌표 설정(경기도청)

```
myScript.then(() => {
  const kakao = window['kakao'];
  kakao.maps.load(() => {
    const mapContainer = document.getElementById('map');
    const options = {
      center: new kakao.maps.LatLng(lat, logt),
      level: 3,
    };
    const map = new kakao.maps.Map(mapContainer, options);
```

### 위치 업데이트

```
if (map) {
  const moveLatLng = new kakao.maps.LatLng(lat, logt);
  map.setCenter(moveLatLng);
} else {
  map = new kakao.maps.Map(mapContainer, options);
}
```

### 마커 설정

```
const markerPosition = new kakao.maps.LatLng(lat, logt);
const marker = new kakao.maps.Marker({
  position: markerPosition,
});
marker.setMap(map);
});
```



# 5. 구현결과

## # 렌더링 최적화



```
const rowRenderer = ({ index, key, style }) => (

AccidentItem



className="item"



width={300}



height={117}



accident={filteredAccidents[index]}



onClick={handleItemClick}


```

```
<List

width={width}



height={height}



rowCount={filteredAccidents.length}



rowHeight={117}



rowRenderer={rowRenderer}


```

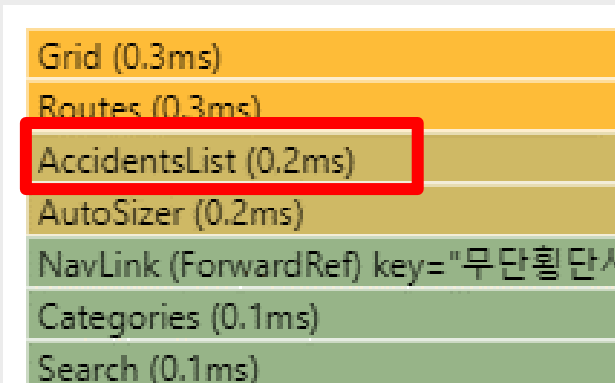
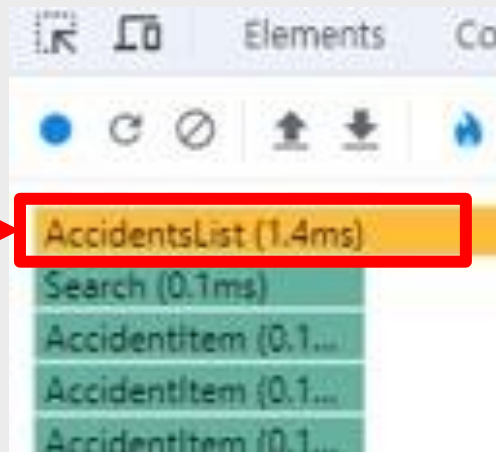
### react-virtualized (List)

맨 처음 렌더링 될 때 AccidentsList의 항목이 모두 렌더링 되면서 AccidentsList를 스크롤 하기 전에 보이지 않는 항목까지 렌더링 되는 것은 비효율적인 방식.

React-virtualized 라이브러리의 **List** 컴포넌트를 사용.

해당 List의 전체 크기와 각 항목의 높이, 각 항목을 렌더링 할 때 사용해야 하는 함수와 배열을 props에 넣어 List 컴포넌트에게 전달하여 최적화함.

이를 통해 스크롤 시, 필요한 항목만 동적으로 렌더링되어 **1.4ms에서 0.2ms**로 낭비되는 자원 감소. API로 받은 대량의 데이터를 효율적으로 처리할 수 있게 됨.



결과

최적화	무단횡단	보행어린이	보행노인
Before	0.5ms of 10.5ms	0.3ms of 2.3ms	0.6ms of 10.9ms
After	0.2ms of 2.1ms	0.1ms of 0.7ms	0.3ms of 3.3ms



## # Router 경로 지정

Header 클릭 시,  
초기 화면을 보여줌

## Categories의 name 값을 받아 주소 경로를 지정

```
<div className="categoryBox">
  <ul>
    {categories.map((c) => (
      <NavLink to={`/${c.name}`} key={c.name}>
        <li>
          className={`category ${
            selectedCategory === c.name ? 'active' : ''
          }`}
          key={c.name}
          onClick={() => {
            setSelectedCategory(c.name);
            onSelect(c.name);
          }}
        >
          {c.name}
        </li>
      </NavLink>
    ))}
    <NavLink to="board">
      <li>
        className={`category ${
          selectedCategory === '교통안전기관연락망' ? 'active' : ''
        }`}
        onClick={() => {
          setSelectedCategory('교통안전기관연락망');
          onSelect('교통안전기관연락망');
        }}
      >
        교통안전기관연락망
      </li>
    </NavLink>
  </ul>
</div>
```

사고유형 : 보행어린이사고다발지  
행정자치별 : 오산시  
년도 : 2021

사고유형 : 보행어린이사고다발지  
행정자치별 : 의정부시  
년도 : 2021

**사고유형 : 보행어린이사고다발지**  
**행정자치별 : 평택시**  
**년도 : 2021**

사고유형 : 보행어린이사고다발지  
행정자치별 : 화성시  
년도 : 2021

15



# 5. 구현결과

## # 교통안전기관연락망

- 관리자는 CRUD 가능, 이용자는 Read only

### Read (조회)

경기도 사고 유형별 위치 조회

무단횡단사고다발지	보행안전사고다발지	자전거사고다발지	교통안전기관연락망
시군명	시군명을 입력하세요		
링크	링크를 입력하세요		
전화번호	전화번호를 입력하세요		
시군명	링크	전화번호	수정 / 삭제
가평군	<a href="https://ggbpolice.go.kr/gp/mainPage.do">https://ggbpolice.go.kr/gp/mainPage.do</a>	031-580-1228	✎ ✕
고양시	<a href="https://ggbpolice.go.kr/gy/mainPage.do">https://ggbpolice.go.kr/gy/mainPage.do</a>	031-830-5868	✎ ✕
과천시	<a href="https://www.ggpolicy.go.kr/gc/">https://www.ggpolicy.go.kr/gc/</a>	02-2149-4353	✎ ✕
광명시	<a href="https://www.ggpolicy.go.kr/gm/">https://www.ggpolicy.go.kr/gm/</a>	02-2093-0351	✎ ✕
광주시	<a href="https://www.ggpolicy.go.kr/gj/">https://www.ggpolicy.go.kr/gj/</a>	031-790-7352	✎ ✕
구리시	<a href="https://ggbpolice.go.kr/guri/mainPage.do">https://ggbpolice.go.kr/guri/mainPage.do</a>	031-560-9352	✎ ✕
군포시	<a href="https://www.ggpolicy.go.kr/gunpo/">https://www.ggpolicy.go.kr/gunpo/</a>	031-390-9352	✎ ✕
김포시	<a href="https://www.ggpolicy.go.kr/gimpo/">https://www.ggpolicy.go.kr/gimpo/</a>	031-950-2352	✎ ✕
남양주시	<a href="https://ggbpolice.go.kr/ny/mainPage.do">https://ggbpolice.go.kr/ny/mainPage.do</a>	031-579-8871	✎ ✕

### Pagination

pageNumbers[ ]  
페이지 넘버링 생성

경기도 사고 유형별 위치 통계

무단횡단사고다발지	보행안전사고다발지	자전거사고다발지	교통안전기관연락망
시군명	시군명을 입력하세요		
링크	링크를 입력하세요		
전화번호	전화번호를 입력하세요		
시군명	링크	전화번호	수정 / 삭제
하남시	<a href="https://www.ggpolicy.go.kr/hn/">https://www.ggpolicy.go.kr/hn/</a>	031-790-0352	✎ ✕
화성시	<a href="https://www.ggpolicy.go.kr/hssb/">https://www.ggpolicy.go.kr/hssb/</a>	031-379-9352	✎ ✕

### Create (등록)

#### onSubmit( )

시군명, 링크, 전화번호 입력란 구현  
사용자의 입력 값을 저장

테스트1

테스트1

테스트1

### Create 결과

#### handleSearch( )

입력란에 모두 기입했는지  
확인하기 위한 alert 함수 사용

localhost:3000 내용:  
시도와 링크, 전화번호를 모두 입력해주세요.

확인

시군명

링크

전화번호

### Update (수정)

#### onEdit( )

수정 버튼을 통해 내용을 수정

테스트1

테스트2

테스트1

### Update 결과

시군명	링크	전화번호	수정 / 삭제
테스트2	<a href="#">테스트2</a>	테스트2	✎ ✕
가평군	<a href="https://ggbpolice.go.kr/gp/mainPage.do">https://ggbpolice.go.kr/gp/mainPage.do</a>	031-580-1228	✎ ✕
고양시	<a href="https://ggbpolice.go.kr/gy/mainPage.do">https://ggbpolice.go.kr/gy/mainPage.do</a>	031-830-5868	✎ ✕
과천시	<a href="https://www.ggpolicy.go.kr/gc/">https://www.ggpolicy.go.kr/gc/</a>	02-2149-4353	✎ ✕

### Delete (삭제)

#### onDelete( )

데이터 삭제

시군명	링크	전화번호	수정 / 삭제
가평군	<a href="https://ggbpolice.go.kr/gp/mainPage.do">https://ggbpolice.go.kr/gp/mainPage.do</a>	031-580-1228	✎ ✕
고양시	<a href="https://ggbpolice.go.kr/gy/mainPage.do">https://ggbpolice.go.kr/gy/mainPage.do</a>	031-830-5868	✎ ✕
과천시	<a href="https://www.ggpolicy.go.kr/gc/">https://www.ggpolicy.go.kr/gc/</a>	02-2149-4353	✎ ✕
광명시	<a href="https://www.ggpolicy.go.kr/gm/">https://www.ggpolicy.go.kr/gm/</a>	02-2093-0351	✎ ✕

1 2 3 4

## 6. 트러블 슈팅

### # Search

#### [ 문제배경 ]

- API 데이터를 AccidentsList 컴포넌트에서 호출하여 Search 컴포넌트로 전달 후, Search 컴포넌트 필터링 하는 구조.
- 두 컴포넌트가 많은 양의 데이터를 주고 받는 과정에서 **렌더링 및 성능 저하** 문제 발생.

#### [ 해결방법 ]

- AccidentsList 컴포넌트에서만 API를 호출하는 것으로 변경.
- Search 컴포넌트에서 사용자가 선택한 값을 AccidentsList 컴포넌트의 **props로 전달하는 방식**으로 리팩토링.
- 이를 통해 Search 컴포넌트에서 API 데이터를 불러오지 않고 필터링을 할 수 있게 되어 성능 저하의 문제 해결.

### # Map

#### [ 문제배경 ]

- API를 로딩하는 스크립트 태그의 경우, HTML파일 안에서의 위치는 상관없으나 실행 코드보다 먼저 선언되어야 한다는 원칙이 위배되어 오류 발생.

#### [ 해결방법 ]

- 스크립트가 성공적으로 로드된 후 카카오 맵이 **차례대로 호출 및 실행**되도록 코드 개선.
- React는 기본적으로 SPA가 기반인 프로그램이므로, 카카오 객체는 스크립트가 온전히 로드된 이후에 인식되기 때문에 스크립트 요소에 **로드(Load) 이벤트**가 필요함.

### # Router

#### [ 문제배경 ]

- 하나의 컴포넌트에서 2개의 path 값을 사용하고자 할 때, 각각의 경로에 대해 **중복되는 코드**가 작성되는 문제.

#### [ 해결방법 ]

- Path 값에 category 파라미터를 변수처리하여 **'/:category?'**로 작성.
- category 파라미터를 선택적으로 사용할 수 있게 되었으며 코드 중복 또한 개선.

## 7. 프로젝트 회고

### # Liked

- 리액트와 관련된 다양한 라이브러리 및 도구들을 활용하고, 리액트의 대표적인 특성 중 하나인 **컴포넌트에 대한 이해도**를 향상시킬 수 있어서 좋았습니다.
- 맡은 업무에만 집중하는 것이 아니라, 팀으로서 서로의 부족한 부분을 보완하기 위해 **지속적인 소통과 협력**을 통해 팀원들과의 협업이 원활했고 완성을 하는 데 있어서 유익한 경험을 할 수 있어서 좋았습니다.

### # Learned

- 리액트 프로젝트를 진행함으로써 라우터 구현을 위해 효율적으로 컴포넌트를 구성하고, 상태 관리, 스크롤 최적화 등 리액트만의 장점을 가진 기능을 수행해 봄으로써 리액트에 대한 경험을 쌓고, 컴포넌트 관리와 **요구사항에 대한 유연한 대처**에 대해 배울 수 있었습니다.
- 팀원들 간의 코드 작성 스타일의 차이로 충돌 문제가 발생할 수 있음을 깨달았고, 이를 통해 협업 과정에서 **팀원들과의 소통과 규칙**의 중요하다는 것을 알게 되었습니다.
- **코드의 가독성과 유지 보수**를 위해 코드를 간결하게 작성하는 것의 중요성을 깨달았습니다.

### # Longed For

- 사용자의 입장에서 처음 보는 화면을 완벽히 이해하지 못할 수 있기 때문에 한눈에 기능을 파악할 수 있도록 더 **세세한 구조와 흐름을 기획하는 능력**을 더 길러야 한다고 생각합니다. 이러한 능력을 통해 사용자가 편리하게 사용할 수 있는 기능과 화면을 구현하도록 노력할 것입니다.
- 팀원들의 가능한 자원을 최대한 활용하여 요구사항을 잘 이해하고, **효율적인 코드 리뷰 및 테스트 전략**을 도입하여 코드 품질을 높이고, 안정적인 서비스를 제공할 수 있도록 개선하고 싶습니다.

# 감사합니다

---

경기도 사고 유형별 위치 조회 서비스