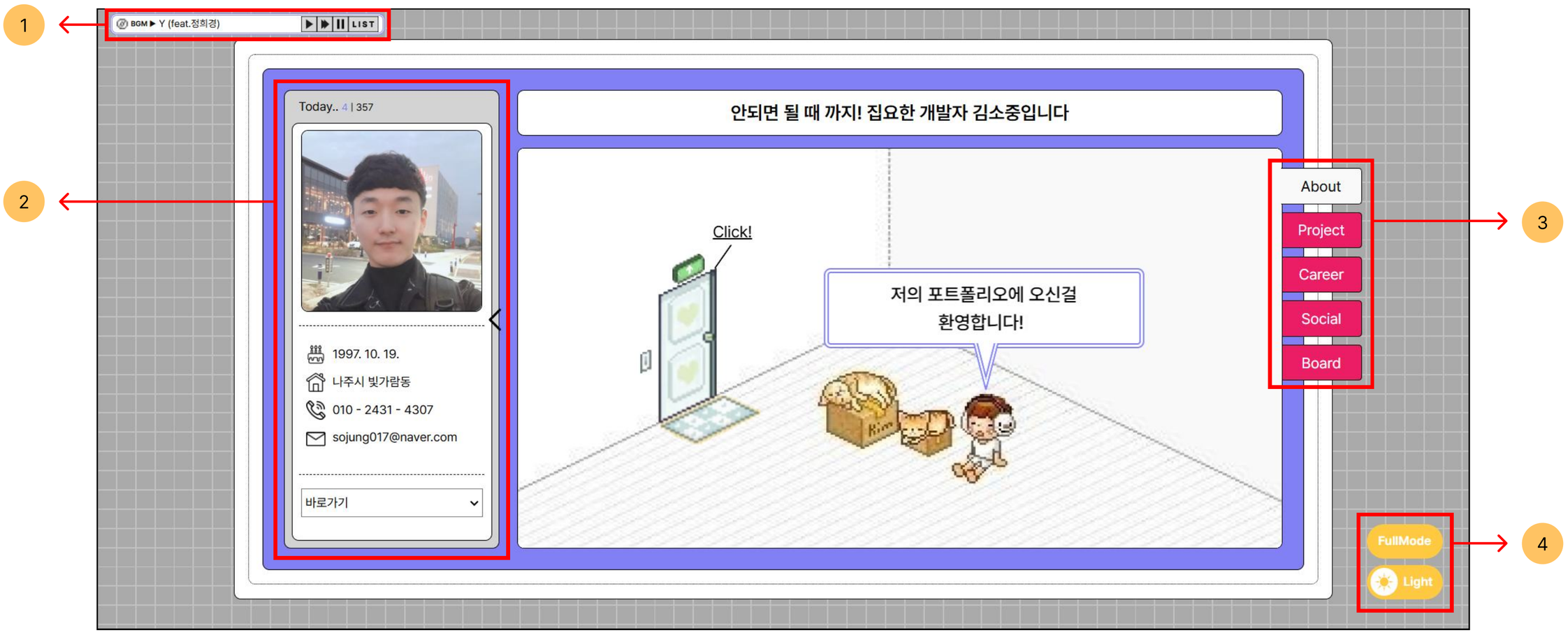
Portfolio: SoJung Kim

1. 전체 페이지 소개

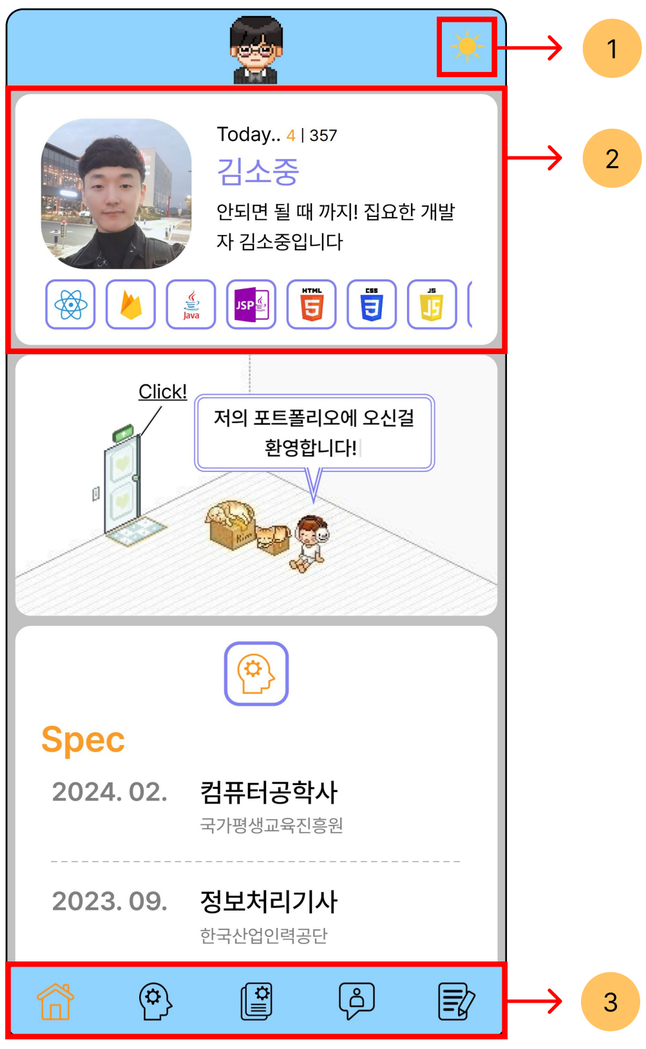


1. 배경음악 재생을 위한 BGM바 입니다.
   react-youtube 라이브러리를 이용하였습니다.

2. Firebase를 이용한 방문자 수가 구현된 프로필 카드입니다.
   버튼을 통해 On/Off 하거나 화면 비율에 따라 자동 숨김됩니다.

3. 메뉴 이동 버튼을 통해 단일 페이지 내에서 메인 영역의 정보가 변경됩니다.

4. ThemeProvider을 사용한 다크모드 버튼과 메인 영역의 확대를 위한 FullMode 버튼입니다.



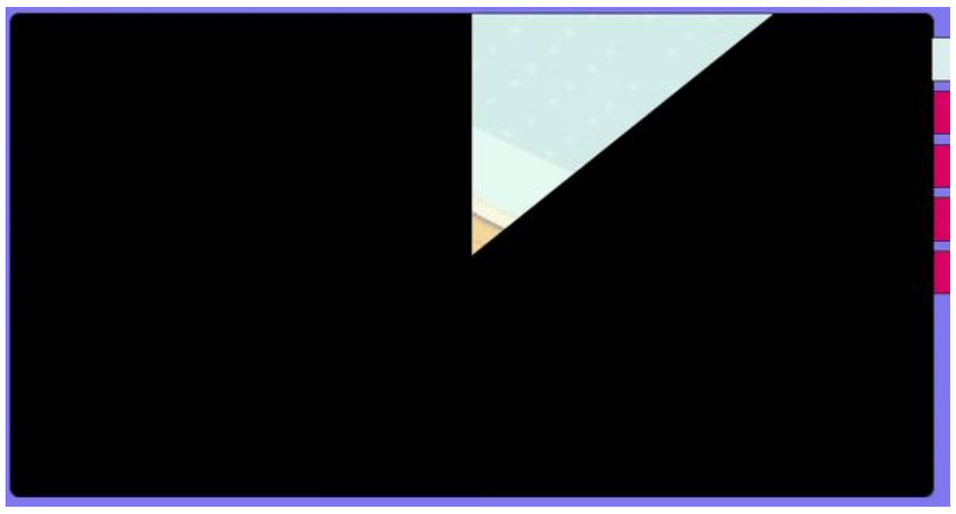
1. ThemeProvider을 사용한 다크모드 버튼입니다.

2. Firebase를 이용한 방문자 수가 구현된 프로필 카드입니다.

3. 메뉴 이동 버튼을 통해 메인 영역이 스크롤됩니다.

2. 각 메뉴 소개

- **About**
  : 미니홈피라는 주제에 걸맞은 자기소개 메뉴입니다.



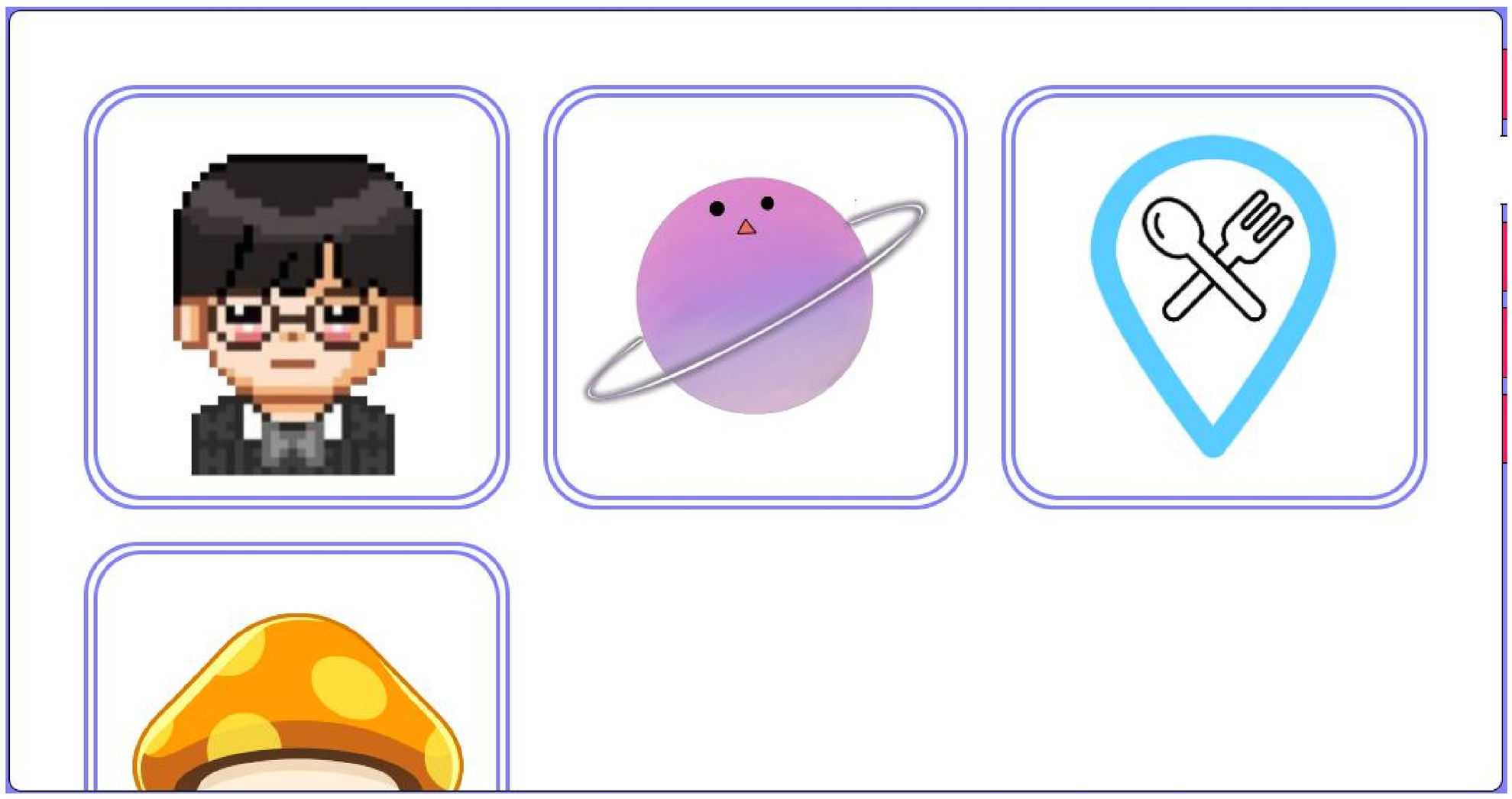
```
const aboutChange = keyframes`
    0% {
       clip-path: polygon(50% 50%, 50% 0, 50% 0, 50% 0, 50% 0, 50% 0, 50% 0, 50% 0, 50% 0, 50% 0);
    } 5% {
       clip-path: polygon(50% 50%, 50% 0, 100% 0, 100% 0,100% 0, 100% 0, 100% 0, 100% 0, 100% 0, 100% 0);
    } 10% {
       clip-path: polygon(50% 50%, 50% 0, 100% 0, 100% 50%, 100% 50%,100% 50%, 100% 50%, 100% 50%, 100% 50%, 100% 50%);
    } 15% {
       clip-path: polygon(50% 50%, 50% 0, 100% 0, 100% 50%,100% 100%, 100% 100%, 100% 100%, 100% 100%, 100% 100%, 100% 100%);
    } 20%{
```

1. 싸이월드 감성을 살려주는 자기소개 메뉴입니다.

2. 타이핑 애니메이션으로 텍스트를 출력하여 캐릭터가 말하는 듯한 연출을 하였습니다.

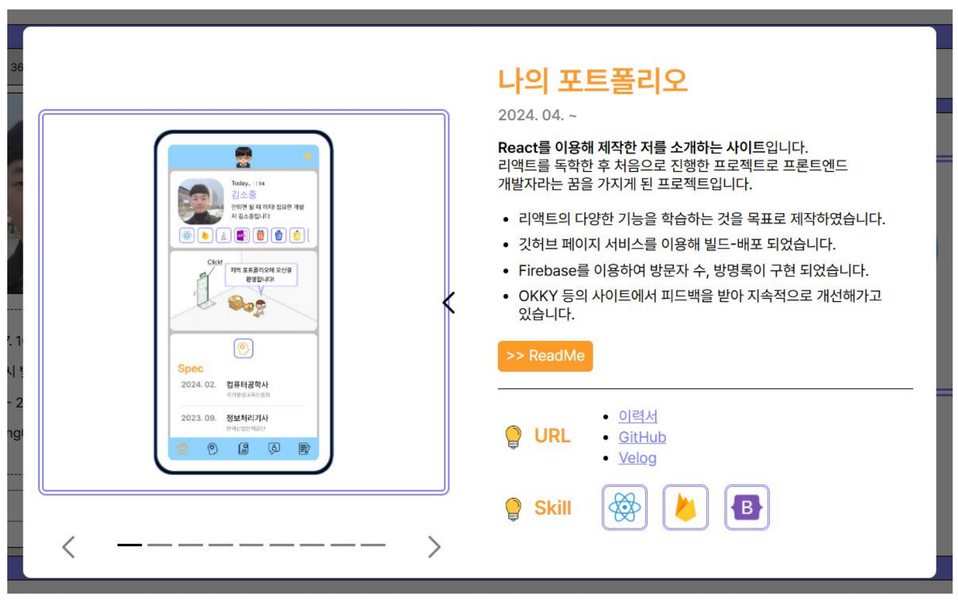
3. 문을 클릭하여 다음 화면으로 전환이 가능하며 CSS로 화면 전환 애니메이션을 만들기 위해 clip-path: polygon()을 이용한 연출을 추가하였습니다.

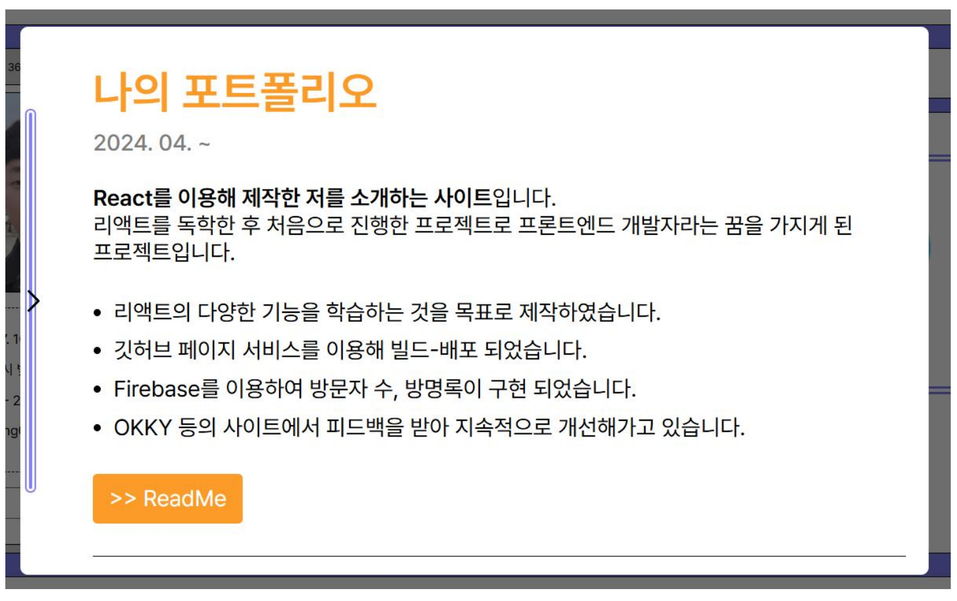
- **Project**
  : react-bootstrap의 Carousel과 react-markdown을 이용한
  프로젝트 메뉴입니다.



1. 프로젝트를 아이콘으로 표현하여 영역의 비율에 따른 display:grid를 사용하여 배치하였습니다.

2. 아이콘을 클릭하여 팝업창으로 상세 화면이 오픈됩니다.



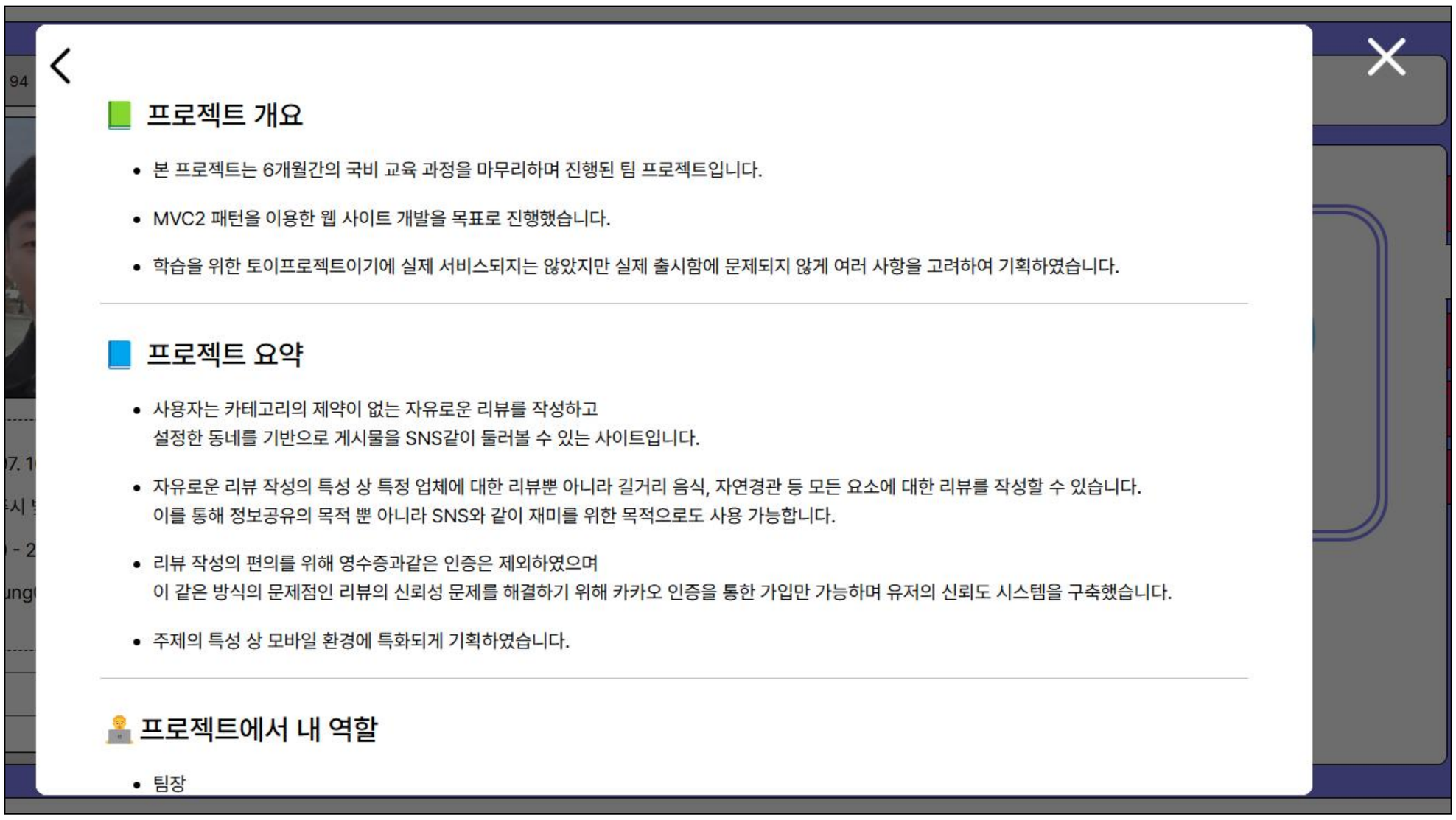


1. 좌측에는 react-bootstrap의 Carousel을 이용하여 프로젝트의 이미지를 확인할 수 있습니다.

2. 우측에는 프로젝트의 설명과 링크, 사용한 기술을 나타내었습니다.

3. 우측의 텍스트의 크기가 작다는 평가를 받아 Carousel의 숨김 기능을 추가하였습니다.

4. ReadMe 버튼을 클릭하여 해당 프로젝트의 리드미 내용을 확인할 수 있습니다.



1. 깃허브에 작성된 ReadMe 파일을 다운받아 프로젝트에 추가하고
   react-markdown을 이용하여 마크다운 문법에 맞게 html에 렌더링하여 나타냅니다.

- **Career & Social**

: 간단한 경력 사항과 깃허브, Velog의 링크 카드가 있는 메뉴입니다.

- **Board**
  : Firebase를 이용하여 구현한 방명록 메뉴입니다.

1. 사용자는 아이디와 비밀번호를 입력하고 프로필 사진을 등록하여 방명록을 작성할 수 있습니다.

2. 정적 웹사이트 호스팅 서비스인 깃허브 페이지를 이용하여 프로젝트를 빌드, 배포하였습니다.
   이 경우 데이터를 저장, 조회하는 등의 백엔드 로직은 따로 구현하여 연결해야 합니다.

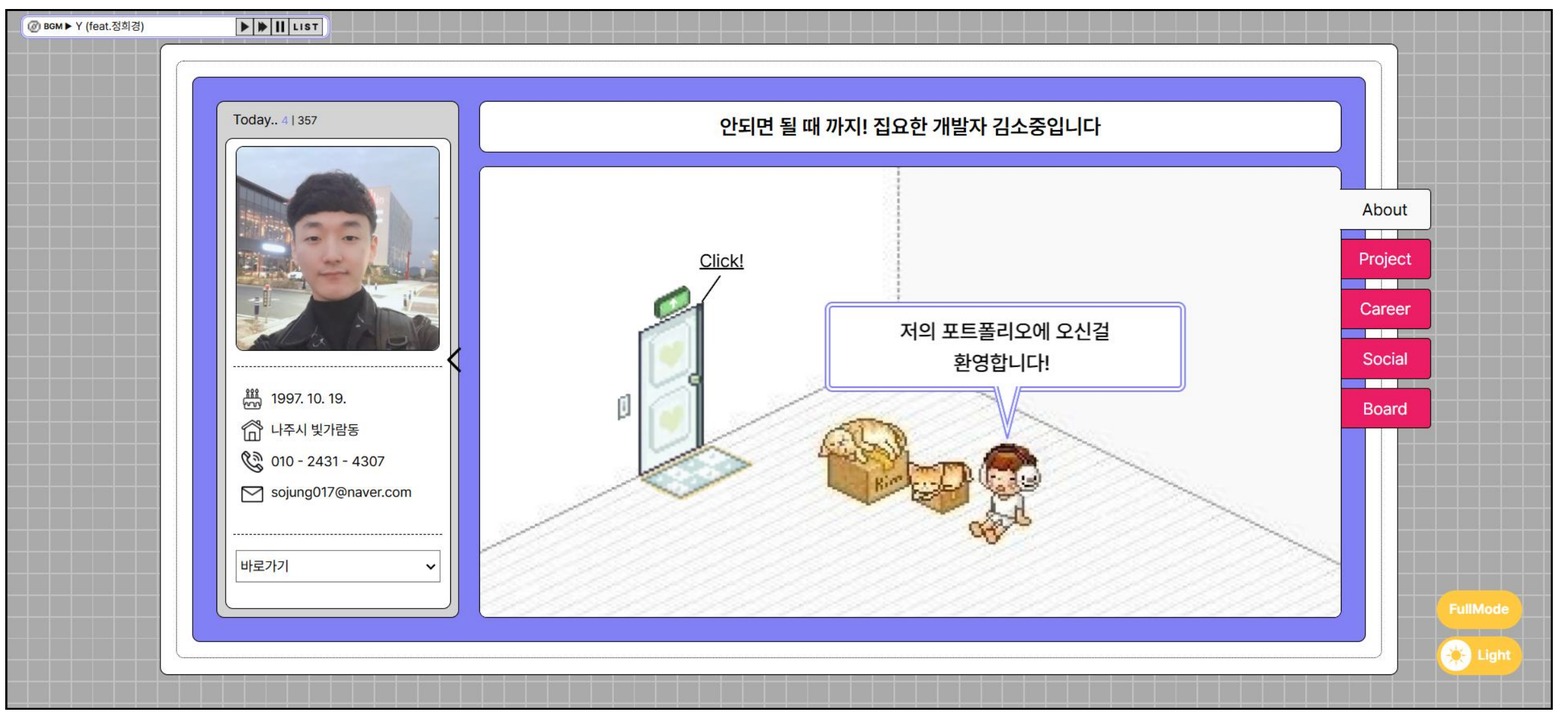
3. 이를 위해 BaaS중 하나인 Firebase를 이용하여 데이터의 저장, 조회, 삭제를 구현하였습니다.

4. 아이디, 비밀번호, 작성일자, 게시글은 Firestore에 저장되며
   프로필 사진 파일은 Storage에 저장됩니다.

1. 해당 기능을 이용하여 방문자 수를 구현할 수 있습니다.

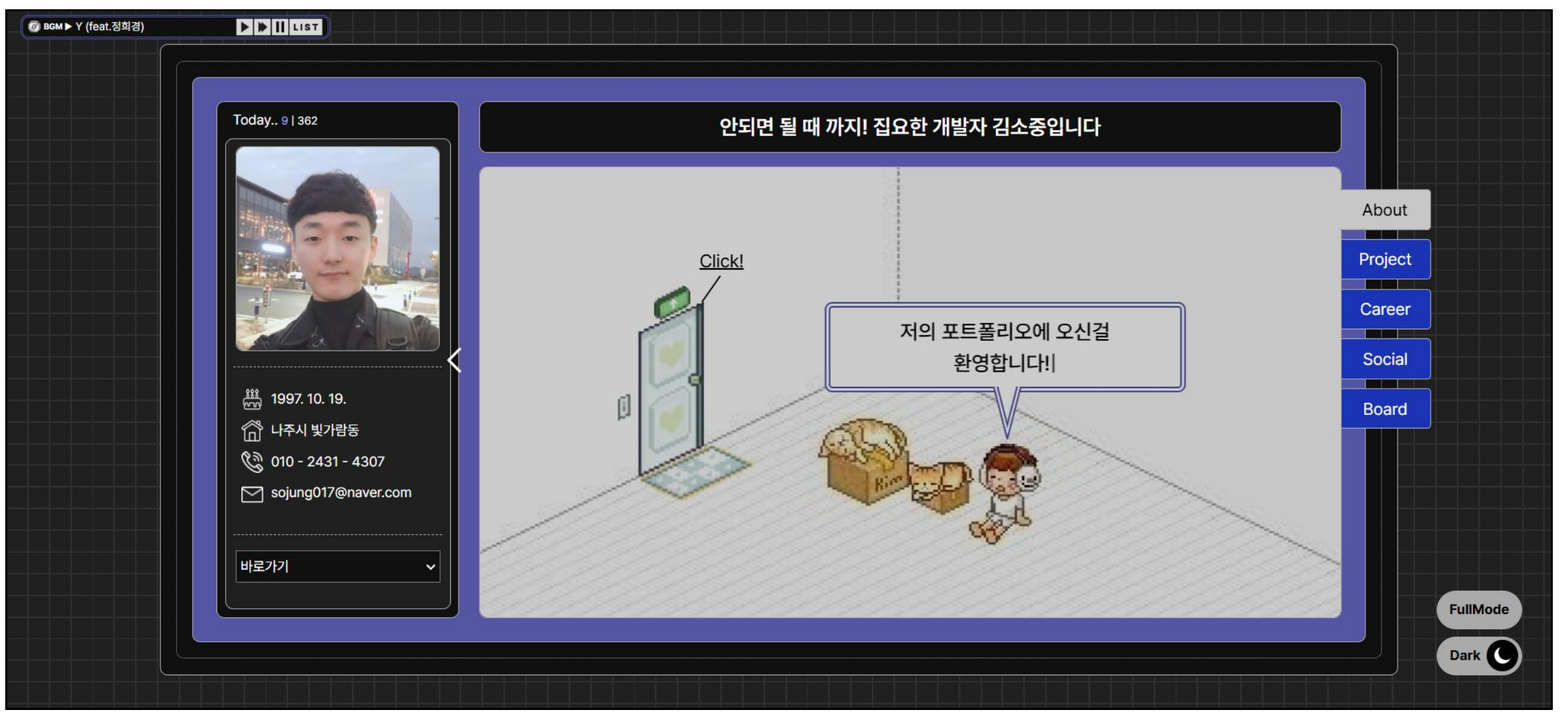
2. 데이터베이스에 오늘 날짜에 해당하는 필드를 추가하고
   localStorage.setItem() 을 이용해 브라우저에 날짜를
   업데이트해서 서로 비교하는 방식을 사용하여
   새로고침마다 Today 값이 오르는 것을 방지할 수 있습니다.

3. 모드 변경



1. 페이지의 비율을 계산하여 데스크탑 / 모바일 모드로 변경됩니다.
   ※ 초기에는 모드에 따라 컴포넌트 내에서 구조만 변경되는 방식으로 제작하였으나 모바일의 작은 화면에서 가독성이 좋지 않아 아얘 다른 컴포넌트가 렌더링되는 방식으로 변경하였습니다.

2. 데스크탑 모드에서는 메인 영역의 컴포넌트가 변경되는 방식으로,
   모바일 모드에서는 화면이 스크롤되는 방식으로 구현하였습니다.



1. 라이트/다크 모드 버튼을 통해 전체적인 색상의 변경이 가능합니다.

2. 버튼을 통해 state를 변경하고 해당 값으로
   ThemeProvider의 value를 변경합니다.
   key-value로 정의하여 묶어둔 value를 따로 파일로
   관리함으로써 유지보수가 간편해집니다.

```
src > style > theme.jsx > dark
1  export const light = {
2      dark: false,
3      bg_pro_hov: '#000000a0',
4      bg_mobile_container: 'white',
5      bg_mobile_item: 'white',
6      bg_mobile_menu: '#91d3ff',
7      bg_mobile_icon: 'rgb(129 129 245)',
8      bg_mobile_tech: 'white'
```

3. ThemeProvider을 사용함으로써 styled-components 내에서 변수를 이용한 스타일링이 가능합니다.

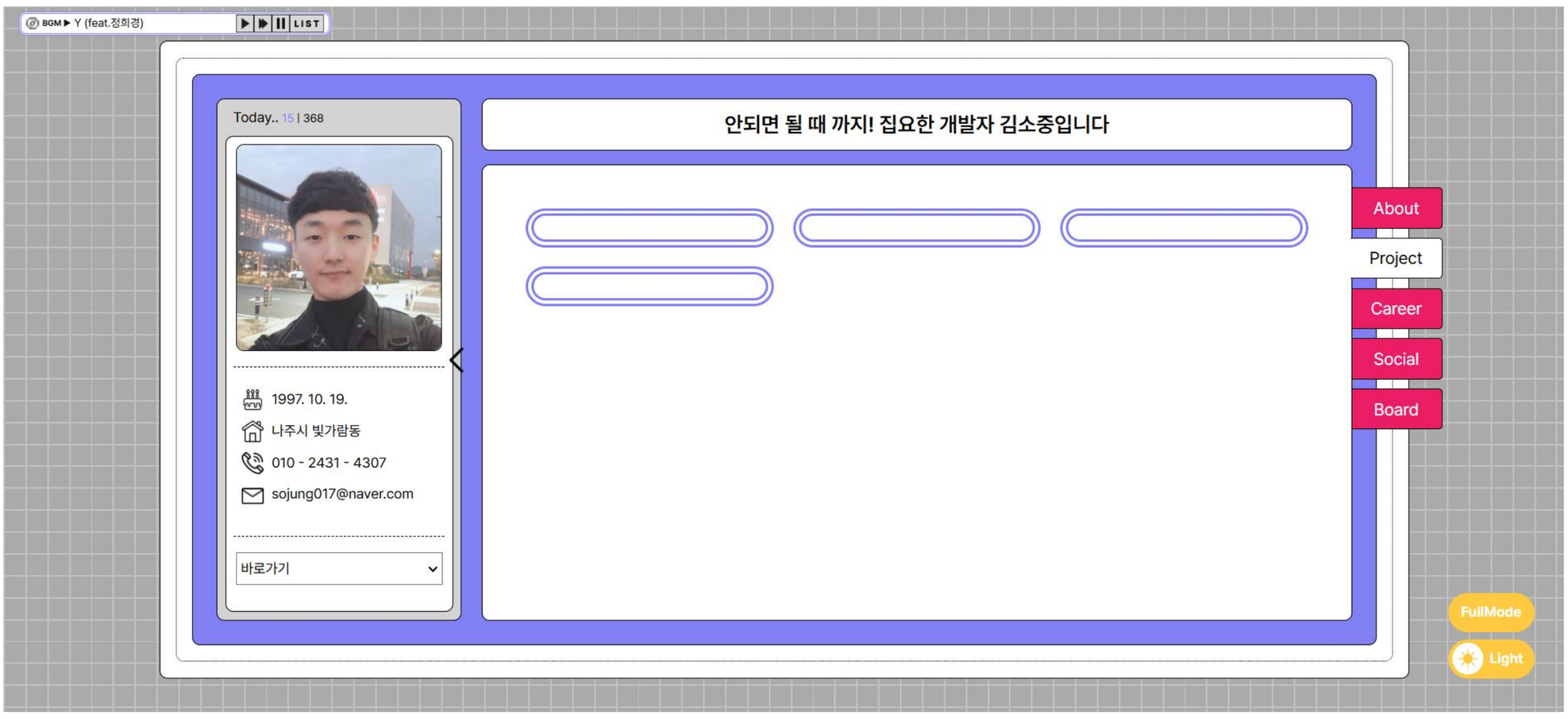
1. 데스크탑 모드에서 메인 영역의 크기가 화면 대비 작은 느낌이 있습니다.
   해당 불편사항을 해소하기 위해 전체화면 모드를 사용할 수 있습니다.

2. width, height를 100vw, 100vh로 확대하면 해당 영역의 비율이 손상되어
   메인 영역의 크기와 브라우저의 크기의 비율로 transform: scale()을 사용해 확대하였습니다.

3. 좌측 메뉴 버튼 영역은 마우스 hover시에만 노출됩니다.

4. 시행착오

- **이미지 프리 로딩**



- 문제
  : 메뉴를 선택하여 컴포넌트가 렌더링되는 방식으로 인해 처음 메뉴를 선택할 때에 이미지 로딩이 지연되는 현상이 있었습니다.

```jsx
import { useEffect } from "react";

function PreLoadingImage(props) {

    const UrlArr = [
        "profile.png",
        "project/starIcon.png",
        "project/myIcon.png"
    ];

    function preloadImage() {
        UrlArr.forEach((url) => {
            const image = new Image();
            image.src = `${process.env.PUBLIC_URL}/images/${url}`;
        })
```

- 해결 방안
  : 이미지를 불러오는 역할을 하는 컴포넌트를 생성하여 최상위 컴포넌트에 넣어줌으로써 사용감을 향상시켰습니다.

이런 방식을 이용하여 프리 로딩이 되는 이유는 브라우저 캐싱 때문입니다.

이미 불러왔던 이미지는 캐시에 저장되기 때문에 나중에 같은 경로로 이미지를 불러오려고 할 때 서버가 아닌 캐시에서 불러오기 때문에 속도가 향상되는 것입니다.

- **팝업창의 z-index**

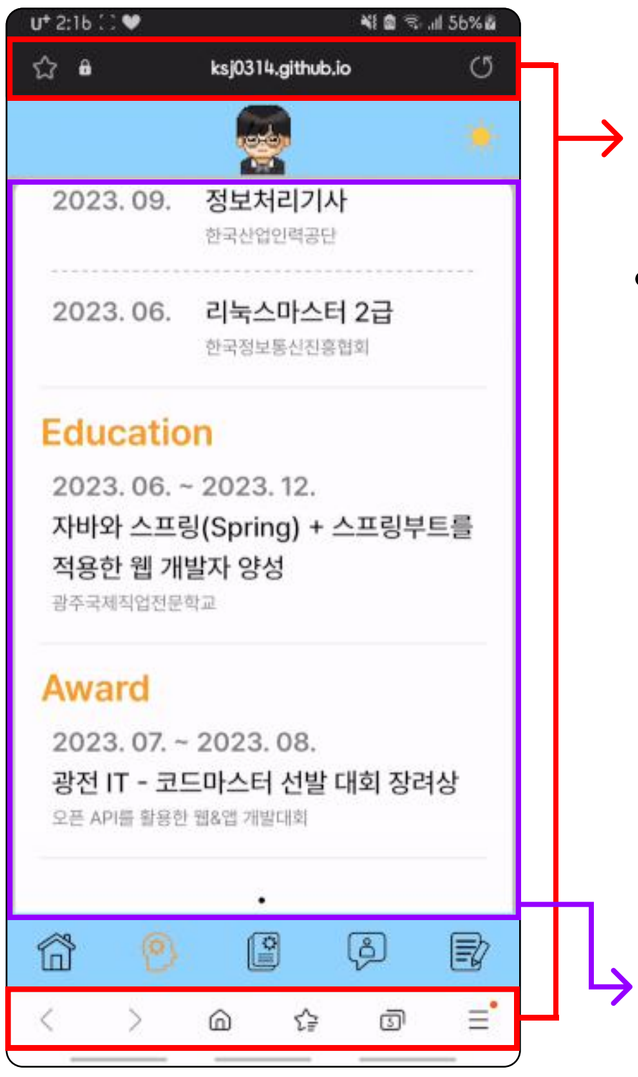
- 문제
  : 컴포넌트의 계층 구조 때문에 팝업 창을 position: fixed 설정하여 z-index를 높여도 상위 계층 컴포넌트간의 index 순위로 인해 원하는 우선 순위가 적용되지 않습니다.
  (위 사진에서 Project 버튼이 팝업창의 반투명 background에 가려지지 않음)

```
import ReactDOM from 'react-dom';

function ChildComponent (props){
  return ReactDOM.createPortal(
    <>…
    </>, document.body  // 자식 요소를 body에 직접 렌더링
  );

export default ChildComponent;
```

- 해결 방안
  : React Portal을 통해 DOM 구조와는 독립적으로 컴포넌트를 렌더링할 수 있습니다.

  해당 컴포넌트를 body에 직접 렌더링하게 되어 원하는 z-index 적용이 가능합니다.

- **모바일 웹에서 주소창, 하단 메뉴 UI 자동 숨기기**



자동 숨김이 안됨

- 문제
  : 모바일 웹에서는 사용자가 스크롤 시 주소창과 하단 메뉴 UI가 자동으로 나타나거나 숨겨집니다.

  이러한 자동 숨김 기능은 루트영역에서 스크롤이 발생할때 이루어집니다.

  해당 프로젝트에서는 좌측의 사진 처럼 중앙의 컴포넌트 내에서 스크롤이 이루어지기에 자동으로 상·하단 UI가 숨겨지는 기능이 작동하지 않는 문제가 있었습니다.

중앙의 메인 컴포넌트 영역

- 해결 방안 1.
  : 루트 영역에서 스크롤이 발생되면 해당 기능은 문제 없이 작동합니다.

  중앙의 영역에 overflow: scroll을 해제하고 페이지 전체 영역에서 스크롤되게 변경하면 간단하게 해결이 가능합니다.

```
const handleTouchStart = (event) => {…
};

const handleTouchMove = (event) => {…
};

const handleTouchEnd = () => {…
};

  window.addEventListener('touchstart', handleTouchStart);
  window.addEventListener('touchmove', handleTouchMove, { passive: false });
```

- 해결 방안 2.
  : 1번 방안을 적용하는 것은 원하는 작동 방식이 아니었습니다.
  메뉴를 선택해 전체 영역이 스크롤되고 선택된 메뉴에 대한 정보를 나타내는 영역에서 스크롤이 작동하기를 원했기 때문입니다.

1. 해당 영역에 overflow: hidden을 설정하여 스크롤 기능을 제거합니다.
2. 루트 영역의 스크롤 이벤트에 해당 영역의 스크롤을 조작하는 코드를 추가합니다.

forwardRef()와 useImperativeHandle()를 이용하여 App.js에 해당 영역의 스크롤 상태를 변경하는 기능을 넘겨주고
window.addEventListener()를 이용하여 touchstart, touchmove, touchend에 적절히 추가합니다.