

8. 데이터 포함시키기 (1)

Prof. Seunghyun Park (sp@hansung.ac.kr)

Division of Computer Engineering

학습 목표: 8장. 데이터 포함시키기

- 데이터 요청하기
 - `fetch()`: promise, async/await
 - `XMLHttpRequest()`, `axios.get()`
- React App에서의 데이터 요청 – `useState()`, `useEffect()` 활용
- 웹 스토리지와 활용 예
- Promise 상태 처리
- 렌더 프롭
- 가상화 리스트

```
/* ch08-01-1.html */
```

```
fetch(`https://api.github.com/users/sp-hsu`)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

```
{ login: 'sp-hsu', id: 90921202, node_id: 'MDQ6VXNlcjw...',
  avatar_url: 'https://avatars.githubusercontent.com/u/90921202?v=4', ... }
```

```
▼ Object 
  avatar_url: "https://avatars.githubusercontent.com/u/90921202?v=4"
  bio: null
  blog: ""
  company: null
  created_at: "2021-09-17T15:23:54Z"
  email: null
  events_url: "https://api.github.com/users/sp-hsu/events{/privacy}"
  site_admin: false
  starred_url: "https://api.github.com/users/sp-hsu/starred{/owner}/{/repo}"
  subscriptions_url: "https://api.github.com/users/sp-hsu/subscriptions"
  twitter_username: null
  type: "User"
  updated_at: "2022-10-06T06:45:38Z"
  url: "https://api.github.com/users/sp-hsu"
  ► [[Prototype]]: Object
```

- `fetch()`: url에 대한 비동기 요청을 보내고, promise 반환

※ 네트워크 리소스를 비동기적으로 가져오기 위해 사용

```
const promise = fetch(url, [options]);
```

```
/* ch08-01-6.html */
```

```
const getDataFromUrl = async login => {
  try {
    const response = await fetch(
      `https://api.github.com/users/${login}`);
    const data = await response.json();
    console.log(data);
  }
  catch (err) {
    console.error(err);
  }
}

getDataFromUrl("sp-hsu");
```

```
/* ch08-02-1.html */
const url = "https://api.github.com/users";

const GithubUser = ({ login }) => {
  1 const [data, setData] = React.useState();

  2 React.useEffect(() => {
    if (!login) return;
    fetch(`${url}/${login}`)
      .then(response => response.json())
      .then(setData)
      .catch(console.error)
  }, [login]);

  3 if (data)
    return <pre>{JSON.stringify(data, null, 2)}</pre>;

  4 return <div>No data</div>;
}


const root =
  ReactDOM.createRoot(document.getElementById('root'));
root.render(<GithubUser login="sp-hsu" />);
```

1 1-1) 최초 렌더링 시에는 **data**가 존재하지 않음

4 1-2) **data**가 존재하지 않으면, No data 반환 후 렌더링

2 2-1) 첫 렌더링 이후,
login 값에 따라 **useEffect()** 실행

1 2-2) **data**가 상태 값 변경으로 다시 렌더링

3 2-3) **data**를 JSON 문자열로 변환하여 반환
JSON.stringify() 

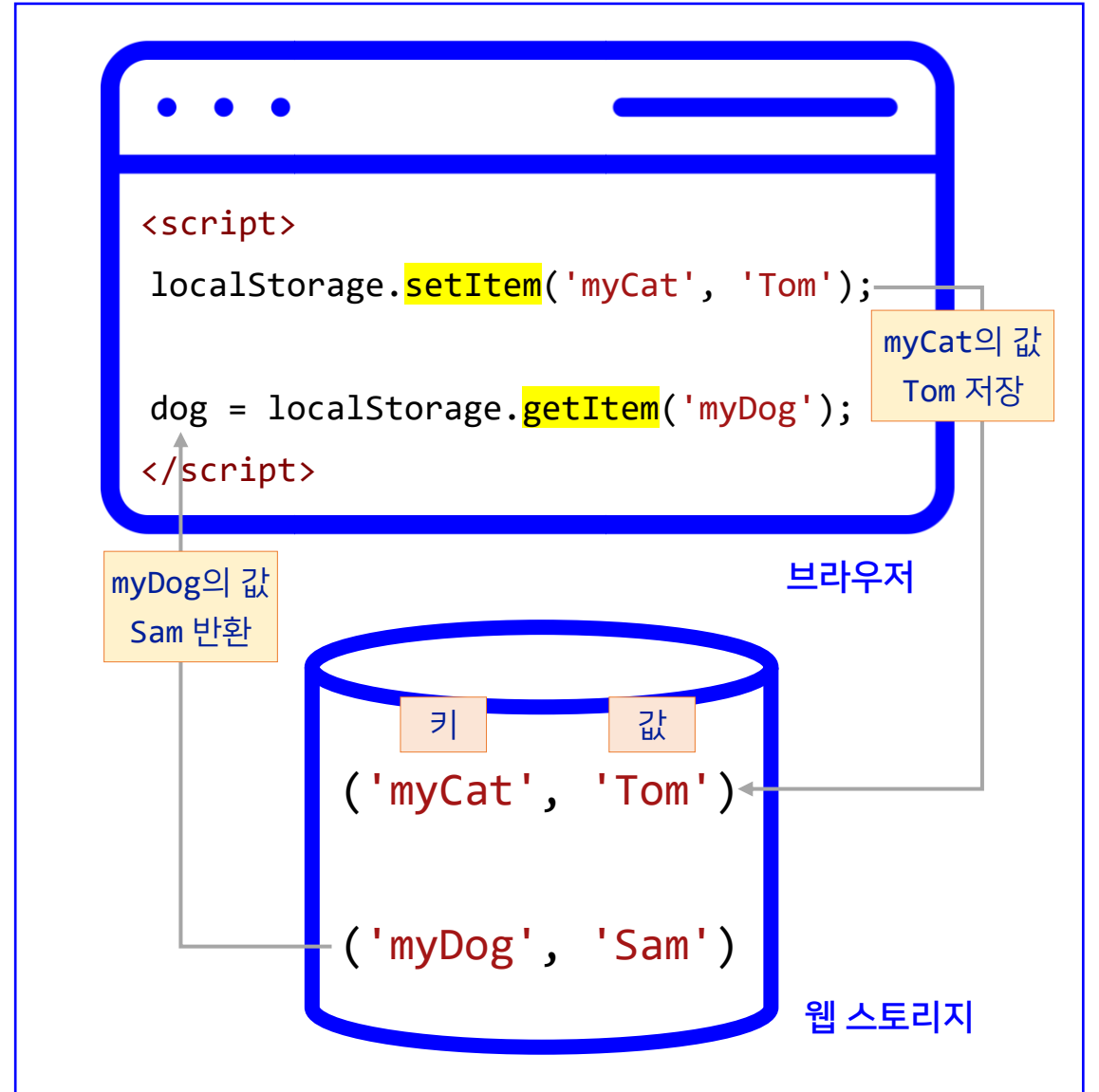
```
{
  "login": "sp-hsu",
  "id": 90921202,
  "node_id": "MDQ6VXNlcjkwOTIxMjAy",
  "avatar_url": "https://avatars.githubusercontent.com/u/90921202?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/sp-hsu",
  "html_url": "https://github.com/sp-hsu",
  "followers_url": "https://api.github.com/users/sp-hsu/followers"
```

웹 스토리지 활용

<https://developer.mozilla.org/ko/docs/Web/API/Window/sessionStorage> 

<https://developer.mozilla.org/ko/docs/Web/API/Window/localStorage> 

- 웹 스토리지의 데이터
 - 키와 값을 문자열 형태로 저장
 - (키, 값)의 쌍으로 구성된 아이템 단위
- 세션 스토리지
 - 데이터를 사용자 세션 단위로 저장
 - 윈도우마다 별도 세션 스토리지 별도 생성, 윈도우 간 공유하지 않음
 - 윈도우 닫힐 때 세션 스토리지 소멸
- 로컬 스토리지
 - 윈도우와 무관하게 웹 서버 (웹 사이트) 당 하나씩 생성
 - 웹사이트의 모든 웹페이지가 로컬 스토리지 공유
 - 브라우저 종료, PC 종료시에도 유지



사용자 로컬 컴퓨터

웹 스토리지 활용 예

```
/* ch08-03-1.html */
const loadJSON = key => key && JSON.parse(localStorage.getItem(key));
const saveJSON = (key, data) =>
  localStorage.setItem(key, JSON.stringify(data));

const GithubUser = ({ login }) => {
  1 const [data, setData] = React.useState(loadJSON(`user:${login}`));
  2 React.useEffect(() => {
    if (!data || data.login !== login) return;
    const {name, login, avatar_url, location} = data;
    saveJSON(`user:${login}`, {name, login, avatar_url, location});
  }, [data]);

  3 React.useEffect(() => {
    if (!login || (data && data.login === login)) return;
    fetch(`https://api.github.com/users/${login}`)
      .then(response => response.json())
      .then(setData)
      .catch(console.error);
  }, [login]);

  4 if (data) return <pre>{JSON.stringify(data, null, 2)}</pre>;
  5 return <div>No data</div>;
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<GithubUser login="sp-hsu" />);
```

1 첫 렌더링 `useState()`

No data

Paused in debugger

5 . 빈 로컬 스토리지

. data는 null

Storage
▼ Local Storage
file://

렌더링 후

`useEffect()`

2 3 . fetch()로 data 변경

1 4 . 상태 변경으로 재 렌더링

재렌더링 후

`useEffect()`

2 3 . data 일부를 로컬 스토리지 입력

Storage
▼ Local Storage
file://
▼ Session Storage
file://

```
{
  "login": "sp-hsu",
  "id": 90921202,
  "node_id": "MDQ6VXNlcjkwOTIzMjAy",
  "avatar_url": "https://avatars.githubusercontent.com/u/90921202?v=4",
  "following": 0,
  "created_at": "2021-09-17T15:23:54Z",
  "updated_at": "2022-10-06T06:45:38Z"
}
```

Key	Value
user:sp-hsu	{"name":null,"login":"sp-hsu","avatar_url":"https://avatars.githubusercontent.com/u/90921202?v=4","location":null,"login":"sp-hsu","name":null}

1 재 방문 (새로 고침)

: 로컬 스토리지에 데이터 포함

4 . data 읽어와서 렌더링

렌더링 후

`useEffect()`

2 데이터 저장

`useEffect()`

3 data.login === login 종료

Storage
▼ Local Storage
file://

Key	Value
user:sp-hsu	{"name":null,"login":"sp-hsu","avatar_url":"https://avatars.githubusercontent.com/u/90921202?v=4","location":null,"login":"sp-hsu","name":null}

Promise 상태 처리

/* ch08-04-1.html */

```
const url = "https://api.github.com/users";
```

```
const GithubUser = ({ login }) => {
```

- 1

```
const [data, setData] = React.useState();  
const [loading, setLoading] = React.useState(false);  
const [error, setError] = React.useState();
```

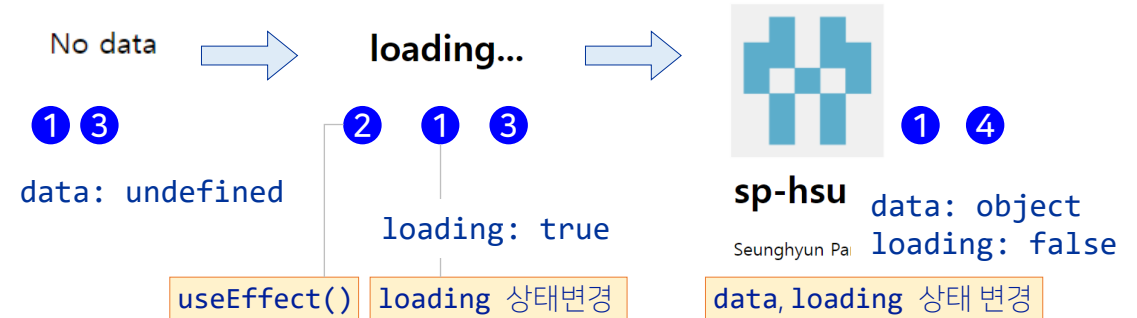
```
React.useEffect( () => {
```

- 2

```
  if (!login) return;  
  setLoading(true);  
  fetch(`${url}/${login}`)  
    .then(response => response.json())  
    .then(setData)  
    .then(() => setLoading(false))  
    .catch(setError)  
}, [login]);
```

- 3

```
  if (loading) return <h1>loading...</h1>;  
  if (error) return <pre>{JSON.stringify(error, null, 2)}</pre>;  
  if (!data) return <div>No data</div>;
```



- 4

```
  return (  
    <div className="githubUser">  
      <img src={data.avatar_url} alt={data.login}  
        style={{ width:300 }} />  
      <div>  
        <h1>{data.login}</h1>  
        {data.name && <p>{data.name}</p>}  
        {data.location && <p>{data.location}</p>}  
      </div>  
    </div>  
  );  
}
```

```
const root =  
  ReactDOM.createRoot(document.getElementById('root'));  
root.render(<GithubUser login="sp-hsu" />);
```

render prop

- 렌더 프롭

- 렌더링되는 프로퍼티 (프로퍼티로 전달되는 컴포넌트)
- 컴포넌트 사이에 함수 전달을 위해 props를 활용

```
/* ch08-05-1.html */
const csehsu3 = [
  { track: "...", subject: "...", ... },
  ...
];

const App = () => (
  <ul>
    {csehsu3.map( (info, i) => (
      <li key={i}>{info.track} - {info.subject}</li>
    ))}
  </ul>
)

const root =
  ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

track	subject
Mobile-SW	AMP
Web Eng.	WF1
Bigdata	DB Design
D.Contens	VR

```
/* ch08-05-2.html */
const csehsu3 = [...];

const List = ({ data=[], renderEmpty, renderItem }) => {
  return !data.length ? renderEmpty : (
    <ul>
      {data.map( (item, i) => (
        <li key={i}>{renderItem(item)}</li> ))}
    </ul>
  );
}

const root =
  ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <List
    data={csehsu3}
    renderEmpty={<p>This list is empty</p>}
    renderItem={info => <>{info.track} - {info.subject}</>}
  />
);
```


학습 정리: 8장. 데이터 포함시키기 1

- 데이터 요청하기
 - `fetch()`: promise, async/await
 - `XMLHttpRequest()`, `axios.get()`
- React App에서의 데이터 요청 – `useState()`, `useEffect()` 활용
- 웹 스토리지와 활용 예
- Promise 상태 처리
- 렌더 프롭
- 가상화 리스트

데이터 요청하기: XMLHttpRequest()

<https://developer.mozilla.org/ko/docs/Web/API/XMLHttpRequest> 

```
/* ch08-01-2.html */

const xhr = new XMLHttpRequest();
xhr.onreadystatechange = () => {
  if (xhr.readyState === xhr.DONE) {
    (xhr.status === 200 || xhr.status === 201) ?
      console.log(xhr.responseText) :
      console.error(xhr.responseText);
  }
}
xhr.open('GET',
  'https://api.github.com/users/sp-hsu');
xhr.send();
```

```
/* ch08-01-3.html */

const getDataFromUrl = (method, url) =>
  new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest();
    xhr.onreadystatechange = () => {
      if (xhr.readyState === xhr.DONE){
        (xhr.status === 200 || xhr.status === 201) ?
          resolve(xhr.response) : reject(xhr.response);
      }
    };
    xhr.open(method, url);
    xhr.send();
  });

getDataFromUrl('GET',
  'https://api.github.com/users/sp-hsu')
  .then(data => console.log(JSON.parse(data)))
  .catch(err => console.error(err))
```

- Axios: Promise 기반의 브라우저와 node.js를 위한 HTTP 클라이언트 라이브러리
 - 브라우저: XMLHttpRequest() 사용
 - 서버 사이드: node.js의 http 모듈 사용

```
const promise = axios.get(url, [config]);
```

```
/* ch08-01-4.html */
```

```
axios.get('https://api.github.com/users/sp-hsu')  
.then(result => console.log(result))  
.catch(error => console.log(error))
```

https://axios-http.com/kr/docs/req_config 

url, method, baseURL, headers, params, data 등을
객체로 지정하여 전달

```
/* ch08-01-5.html */
```

```
(async () => {  
  try {  
    const result = await axios.get(  
      'https://api.github.com/users/sp-hsu');  
    console.log(result.data);  
  }  
  catch(err){  
    console.error(err);  
  }  
})();
```