**자바스크립트 Fetch API 익히기**

**브라우저 로컬 스토리지**

로컬 스토리지는 사용자가 개발자 도구를 통해 자유롭게 접근하여 데이터를 수정할 수 있으므로, 개발자가 의도한 데이터가 저장되어 있지 않을 수 있습니다.

따라서 로컬 스토리지의 데이터를 가져와 사용하기 전에 데이터 검증 과정이 필요

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**Fetch API 기본 문법 작성**

fetch("https://api.example.com/data")

.then((response) => response.json)

.then((data) => console.log("Result:", data))

.catch((error) => console.error("Error:", error));

**HTTP란?**

**HTTP**는 웹 브라우저와 서버 간에 데이터를 주고받기 위한 프로토콜로, 클라이언트 – 서버 모델에서 가장 일반적으로 사용되는 프로토콜

**HTTP 주요 기능**

**상태 없음 (Stateless):** 상태를 저장하지 않는 프로토콜로, 각 요청은 서버에 독립적으로 처리

**상태 코드(Status code):** 200 성공, 300 리다이렉션을 나타내는 코드 400 클라이언트 오류, 500 서버 오류

**헤더(Headers):** 요청 과 응답은 헤더를 사용하여 추가적인 정보를 전송

Ex Content-Type, Authorization, Accept

**본문(Body):** http 메소드는 요청 본문에 데이터를 담아 서버로 전송

Post, Put 메소드에서 사용, 데이터는 일반적으로 JSON 형식

**Ajax란?**

Ajax는 비동기적 웹 애플리케이션 개발을 위한 기법으로, XMLHttpRequest 혹은 Fetch API를 사용하여 클라이언트와 서버간의 데이터를 동적으로 교환하는 기술

**Ajax의 특징**

비동기 통신: 새로고침 없이 데이터를 받아오거나 전송

**Fetch API 사용법**

const dataContainer = document.getElementByld("data-container")

async function fetchDataAndUpdateUIO (){

try{

//fetch를 동해 데이터 조회

const response =await fetch("https://api.example.com/data");

if (!response.ok){

throw new Error("데이터를 가져오는데 문제가 발생했습니다.");

}

//JSON 형식으로 응답 데이터 파싱

const data = await response.json()

//데이터를 dataContainer에 표시되도록 설정

dataContainer.textContent = JSON.stringify(data)

} catch (error){

console.error("데이터를 가져오는 도중 오류가 발생했습니다:", error)}

}

fetch("https://api.example.com/posts", {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

title: "foo", userld: 1,

}),

})

.then((response) => response.json())

.then((data) => {

console.log(data);

})

.catch((error) => {

console.error("Fetch error:", error);

});

**Content-Type 종류**

**Content-Type**은 HTTP 헤더의 일종으로 요청이나 응답 메시지의 본문의 미디어 타입 지정

**Application/json:** json 형식의 데이터를 전송

**Multipart/form-data:** 파일 업로드와 같이 여려 종류의 데이터를 한번에 전송

**Application/x-www-form-urlencoded:** HTML 폼 데이터를 인코딩하여 전송할 때 사용 key=value 쌍으로 데이터 전송

**Async. Await Fetch API 사용하기**

async function fetchData(){

try {

const response = await fetch("https://api.example.com/posts",{

method: "POST", headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

title: "foo", userid: 1,

}),

});

if (response.ok) {

throw new Error("Network response was not ok");

}

const data = await response.json();

console.log(data);

} catch (error) {

console.error("Fetch error:", error);

}

}

**HTTP 메소드의 개념**

**Get:** 데이터 조회

**Post:** 새로운 데이터 추가

**Put:** 기존 데이터 업데이트

**Delete:** 데이터 삭제

**Patch:** 리소스의 일부를 수정

**Options:** 서버가 지원하는 HTTP 메소드를 확인

**Head:** GET 요청과 유사하지만, 응답 본문을 반환하지 않고 헤더만 반환

**Promise에서 에러 처리**

FetchAPI는 Promsie를 반환 하므로, .then() 과 .catch()를 사용하여 비동기 요청 처리

요청이 성공하면 .then() 블록에서 데이터 처리

실패하면 .catch()블록에서 에러 표시

에러 처리를 위한 세 가지 방법

비 추천

**콘솔에 로깅**

개발자 도구의 콘솔을 사용하여 발생한 에러를 기록

개발 과정에서 쓰지만 비추천이다

운영환경에서는 다른 방법으로 하는게 낫다 에러 페이지 또는 **팝업 (alert)사용**

**오류 페이지 보여주기**

심각한 오류가 발생했을 경우, 브라우저는 사용자에게 오류 페이지를 보여줄 수 있다

Loation.href=’/error’;

**디버깅**

**API 플랫폼을 통해 디버깅하기**

간편한 API 요청 생성: 직관적인 UI를 통해 다양한 HTTP 요청을 생성하고 관리

**실시간 요청 모니터링**: 요청과 응답을 실시간으로 모니터링하여 성능 및 상태를 확인

**풍부한 테스트 환경:** Pre-request, Scripts 와 Test 기능을 활용하여 자동화된 테스트를 작성할 수 있습니다.

PostMan

**RestAPI 와 GraphQL**

**Rest API**

자원을 이름을 구분하여 해당 자원의 상태를 주고받는 모든 것을 의미

자원 중심

행위 표현: HTTP 메소드를 통해 자원에 대한 행위를 정의

**Get:** 데이터 조회

**Post:** 새로운 데이터 추가

**Put:** 기존 데이터 업데이트

**Delete:** 데이터 삭제

**Patch:** 리소스의 일부를 수정

**Options:** 서버가 지원하는 HTTP 메소드를 확인

**Head:** GET 요청과 유사하지만, 응답 본문을 반환하지 않고 헤더만 반환

HTTP에서 동작하기 때문에, HTTP의 특징을 모두 가지고 있습니다.

유니폼 인터페이스

상태를 관리하지 않음

캐싱 가능

계층화 시스템

**단점**

**표준의 한계:** 일반적인 웹 기술에 의존, 특정 구현에서는 표준이나 가이드라인이 부족 일관성이 떨어짐

**확장성의 한계:** 자원과 행위 별로 URI가 분리되기 때문에, 자원의 수가 증가할 수록 관리해야 하는 URI의 수도 증가하게 되어 유지보수가 어려워짐

**유연성 부족:** HTTP 메소드 종류가 제한적이라 모든 애플리케이션의 요구 충족시키기 어려움

**GraphQL**

**GraphQL**은 페이스북에서 개발한 쿼리 언어로, 클라이언트가 원하는 데이터 요청할 수 있는 강력한 API 질의 언어

**유연한 데이터 요청:** GraphQL은 클라이언트가 필요한 데이터를 직접 지정할 수 있게 해줍니다. 하나의 요청에서 필요한 필드와 관계를 지정하여 서버로부터 원하는 데이터만을 효율적으로 가져옴

**단일 엔드포인트:** REST API와 달리 GraphQl은 단일 엔드포인트를 사용 클라이언트는 하나의 엔드포인트 에서 모든 데이터 요청

실시간으로 하기 때문에 실시간으로 데이터 수정 및 가능

**단점**

**데이터 노출 위험**: 클라이언트가 필요한 데이터를 직접 지정할 수 있는 특성 때문에 잘못된 설정으로 인해 민감한 데이터가 노출

**쿼리 폭탄:** 복잡하고 대량의 데이터를 한번에 요청할 수 있는 GraphQL 쿼리를 이용하여 서버에 과도한 부하를 줄 수 있음

**보안 취약점**

**공개 API 활용**

**API 검색 및 선택**

원하는 기능 또는 데이터를 제공하는 공개 API를 찾습니다.

이를 위해 검색 엔진이나 API 집약 사이트 활용할 수 있습니다.

예를 들어, 지도 데이터 날씨 정보, 금융 데이터, 소셜미디어 데이터등 다양한 분야의 API 존재

**API 문서 확인**

API 제공자의 공식 문서를 찾아서 API의 사용방법, 엔드포인트 URL , 요청 및 응답 형식, 인증 방법 등을 확인

문서에서는 API의 기능 설명, 예제 코드, 제한 사항 서비스 정책 등을 자세히 설명

인증 획득

인증 정보를 발급

**API 테스트**

API를 호출하여 실제 데이털르 가져오는 테스트 진행

응답하는 데이터 형식과 내용을 확인하고, 필요에 따라 데이터를 가공할 방법을 계획

**데이터 처리 및 통합**

받아온 데이터를 애플리케이션에 필요한 형태로 처리하고, UI에 표시하거나 다른 시스템과 통합합니다.

데이터를 실시간으로 업데이트하거나 일정 주기로 호출하여 최신 정보를 유지하는 방법을 구현