REST API와 비동기 통신

API는 **Application Programming Interface**의 약자입니다.

말 그대로 애플리케이션(프로그램)이 다른 프로그램과 소통할 수 있도록 도와주는 인터페이스입니다.

쉽게 말해,

> "API는 프로그램끼리 데이터를 주고받는 통로" 라고 이해하면 됩니다.

1. REST API란?

- REST(Representational State Transfer)는 **서버와 클라이언트 간 데이터를 주고받는 아키텍처 스타일**
- HTTP 프로토콜의 메서드(GET, POST, PUT, DELETE 등)를 활용해 **자원(Resource)을 URL로 표현**
- 서버는 JSON, XML 등 형식으로 데이터를 제공
- 주요 특징:
  - **무상태성(Stateless)**: 요청 간 상태를 서버가 저장하지 않음
  - 자원을 URI로 식별
  - HTTP 메서드에 따라 작업 구분

예시 URL

`https://api.example.com/books/123` → 123번 책 리소스 접근

2. REST API 호출 방법(클라이언트 측)

호출 방식	설명	코드 예시	특징
AJAX (jQuery.ajax)	jQuery 기반 XMLHttpRequest 비동기 호출	`$.ajax({ url: "...", method: "GET" })`	콜백 기반, 최신 버전은 Promise 스타일 메서드 지원
Fetch API	최신 브라우저 표준, Promise 기반	`fetch("url").then(res => res.json())`	간결하고 직관적, 기본 Promise 지원

Axios	HTTP 클라이언트 라이브러리, Promise 지원	`axios.get("url").then(res => ...)`	편리한 기능 많고 Promise 기본 지원
XMLHttpRequest	브라우저 내장 API, AJAX의 기초	`var xhr = new XMLHttpRequest(); ...`	콜백 기반, 직접 Promise 래핑 필요

3. 로컬 JSON 파일과 REST API 차이

구분	로컬 JSON 파일	REST API
데이터 위치	내 컴퓨터 또는 서버 내 저장된 파일	외부 서버의 동적 데이터
접근 방식	파일 경로나 상대 경로로 직접 접근	URL + HTTP 메서드로 요청
데이터 갱신	수동 업데이트 필요	실시간 최신 데이터 제공 가능
기능	단순 파일 읽기	다양한 필터링, 인증, 조작 등 동적 기능 지원

로컬 JSON 파일 활용

1. 로컬 JSON 파일이란?

- **로컬 JSON 파일**은 프로젝트 내에 저장된 `.json` 형식의 데이터 파일을 의미합니다.
- 서버가 아닌 내 컴퓨터 혹은 웹 서버에 직접 저장되어 있어 URL 경로나 상대 경로로 접근합니다.
- 예를 들어, `data/books.json` 같은 경로로 접근할 수 있습니다.

2. 로컬 JSON 파일 사용 목적

- 서버 없이 정적인 데이터를 불러올 때
- 개발 초기, API가 없을 때 테스트용으로 사용
- 간단한 데이터 바인딩 및 화면 표시

3. 로컬 JSON 파일 불러오는 방법

3-1. jQuery AJAX 사용 ⇒ 예제

```
$.ajax({
  url: 'data/books.json',
  dataType: 'json',
```

```
  success: function(data) {
    console.log(data);
  },
  error: function(err) {
    console.error('로컬 JSON 불러오기 실패', err);
  }
});
```

3-2. Fetch API 사용

```
fetch('data/books.json')
  .then(response => response.json())  // JSON 파싱
  .then(data => {
    console.log(data);  // JSON 데이터 사용
  })
  .catch(error => {
    console.error('로컬 JSON 불러오기 실패:', error);
  });
```

3-3. XMLHttpRequest 사용

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'data/books.json', true);
xhr.onload = function() {
  if (xhr.status === 200) {
    const data = JSON.parse(xhr.responseText);
    console.log(data);
  } else {
    console.error('로컬 JSON 불러오기 실패', xhr.status);
  }
};
xhr.send();
```

---

4. 비동기 작업과 Promise

- **비동기(Asynchronous) 작업**: 요청 후 응답 대기 중에도 다른 작업을 수행 가능
- **동기와 비동기의 비교**

4-1. 동기 예제 ⇒ 앞 작업이 끝나야 다음 코드 실행

```
console.log("1. 요청 시작");

function syncTask() {
  for (let i = 0; i < 3_000_000_000; i++) {} // 시간 지연
  console.log("2. 서버 응답 완료");
}

syncTask();

console.log("3. 다음 작업 실행");
```

콘솔 출력 결과

```
1. 요청 시작
2. 서버 응답 완료
3. 다음 작업 실행
```

4-2. 비동기 예제 ⇒ 요청 후 기다리지 않고 다음 코드 실행

```
console.log("1. 요청 시작");

setTimeout(() => {
  console.log("2. 서버 응답 완료");
}, 1000);

console.log("3. 다음 작업 실행");
```

콘솔 출력 결과

```
1. 요청 시작
3. 다음 작업 실행
2. 서버 응답 완료
```

4-3. **Promise**

- **Promise**: 자바스크립트 비동기 처리 객체
  - 상태: Pending(대기), Fulfilled(성공), Rejected(실패)
  - `.then()`, `.catch()`로 결과 처리
- **async/await**: Promise 기반 비동기 코드를 더 간결하게 작성하는 문법

Promise 예제 ⇒ Pending → Fulfilled / Rejected

```
const promise = new Promise((resolve, reject) => {
  console.log("1. Promise 실행 (Pending)");

  setTimeout(() => {
    resolve("2. 작업 성공 (Fulfilled)");
    // reject("2. 작업 실패 (Rejected)");
  }, 1000);
});

promise
  .then(result => {
    console.log(result);
  })
  .catch(error => {
    console.log(error);
  });

console.log("3. Promise 생성 후 바로 실행됨");
```

콘솔 출력

```
1. Promise 실행 (Pending)
3. Promise 생성 후 바로 실행됨
2. 작업 성공 (Fulfilled)
```

async / await로 변환한 예제

```
const promise = new Promise((resolve, reject) => {
  console.log("1. Promise 실행 (Pending)");

  setTimeout(() => {
    resolve("2. 작업 성공 (Fulfilled)");
  }, 1000);
});

async function run() {
  try {
    const result = await promise; // Promise 완료까지 대기
    console.log(result);
  } catch (error) {
    console.log(error);
  }
}

run();

console.log("3. Promise 생성 후 바로 실행됨");
```

콘솔 출력

```
1. Promise 실행 (Pending)
3. Promise 생성 후 바로 실행됨
2. 작업 성공 (Fulfilled)
```

---

5. AJAX와 Promise

- AJAX는 비동기 HTTP 요청 기술의 총칭
- jQuery `$.ajax()`는 기본적으로 콜백 방식이나, 최신 jQuery는 `.done()`, `.fail()` 등 Promise 스타일도 지원
- `fetch()`와 `axios`는 기본적으로 Promise 반환

기본 형식 (Promise 스타일 사용)

```
$.ajax({
  url: '요청 URL',
  method: 'GET',        // 또는 'POST', 'PUT', 'DELETE'
  dataType: 'json',     // 또는 'text', 'html', 'script', 'x
ml'
  data: {               // 선택 사항
    key1: 'value1',
    key2: 'value2'
  }
})
.done(function(response) {
  // 요청이 성공했을 때 실행
  console.log('성공:', response);
})
.fail(function(jqXHR, textStatus, errorThrown) {
  // 요청이 실패했을 때 실행
  console.error('실패:', textStatus, errorThrown);
})
.always(function() {
  // 성공이든 실패든 항상 실행
  console.log('AJAX 요청이 완료되었습니다.');
});
```

메서드 설명

메서드	설명
`.done()`	성공 시 실행되는 콜백 등록 (`then()` 과 유사)
`.fail()`	실패 시 실행되는 콜백 등록 (`catch()` 와 유사)
`.always()`	성공/실패 상관없이 항상 실행 (`finally()` 와 유사)

예제

```
$.ajax({
  url: 'https://jsonplaceholder.typicode.com/users/1',
  method: 'GET',
  dataType: 'json'
})
.done(function(data) {
  console.log('사용자 이름:', data.name);
})
.fail(function() {
  console.error('데이터를 불러오는 데 실패했습니다.');
})
.always(function() {
  console.log('요청 처리 완료');
});
```

6. 주요 REST API 활용 예

서비스명	설명	주소
Kakao Developers	책 검색, 지도, 번역, 로그인 등	https://developers.kakao.com
Naver Developers	뉴스, 블로그, 쇼핑, 영화, 이미지 검색	https://developers.naver.com
OpenWeatherMap	날씨 정보 제공	https://openweathermap.org/api
TMDB	영화 정보 제공	https://developer.themoviedb.org/
GitHub API	깃허브 사용자, 리포지토리 등 정보 제공	https://docs.github.com/en/rest
REST Countries	국가별 정보 제공	https://restcountries.com/
Unsplash API	고해상도 이미지 검색	https://unsplash.com/developers
Google Maps API	지도, 장소 검색, 거리 계산	https://developers.google.com/maps

7. Postman 기본 사용 가이드

7-1. Postman이란?

Postman은 **REST API를 테스트하고 검증하기 위한 도구**이다.

프론트엔드 코드나 브라우저 화면 없이도 서버 API 요청을 직접 보내고,

요청(Request)과 응답(Response)을 확인할 수 있다.

Postman 사용 목적

- 프론트엔드 개발 전 API 동작 확인
- 서버 API 디버깅
- REST API 구조 학습
- 협업 시 API 명세 검증

7-2. REST API 요청의 기본 구성 요소

Postman에서 보내는 모든 요청은 아래 요소들로 구성된다.

7.2.1 HTTP Method

- GET: 데이터 조회
- POST: 데이터 생성
- PUT / PATCH: 데이터 수정
- DELETE: 데이터 삭제

7.2.2 URL (Endpoint)

- 요청을 보낼 서버 주소
- API 제공자가 문서로 제공

예:

```
https://dapi.kakao.com/v3/search/book
```

7.2.3 Query Parameters (Params)

- GET 요청에서 URL 뒤에 붙는 옵션 값
- 검색 조건, 페이지 크기 등을 전달

형식:

```
?key=value&key=value
```

7.2.4 Headers

- 인증 정보, 데이터 형식 등을 전달

예:

```
Authorization: KakaoAK {REST_API_KEY}
```

7.2.5 Body

- POST / PUT 요청에서 서버로 전달하는 데이터
- GET 요청에서는 일반적으로 사용하지 않음

7-3. API URL은 어디에서 확인하는가?

API 요청에 사용하는 URL(Endpoint)은 **API 제공자의 공식 문서**에서 확인한다.

7.3.1 카카오 책 검색 API URL 출처

아래 URL은 임의로 만든 것이 아니라,

Kakao Developers 공식 문서에 명시된 REST API 엔드포인트이다.

```
https://dapi.kakao.com/v3/search/book
```

7.3.2 카카오 개발자 문서에서 확인하는 경로

1. Kakao Developers 사이트 접속
2. 상단 메뉴에서 `문서` 선택
3. `REST API` 선택
4. `검색(Search)` 카테고리 선택
5. `책 검색` API 문서 확인

해당 문서에서 다음 정보를 확인할 수 있다.

- Request URL (Endpoint)
- HTTP Method
- Query Parameters (target, query, size 등)
- Header (Authorization)
- Response 예시

7.3.3 URL 구조 해석

```
https://dapi.kakao.com/v3/search/book?target=title
```

- Base URL (Endpoint)
  - `https://dapi.kakao.com/v3/search/book`
- Query Parameter
  - `target=title` : 제목 기준 검색

`target=title` 또한 문서에 정의된 옵션 값이다.

7-4. Postman 기본 화면 구성

상단 영역

- HTTP Method 선택
- Request URL 입력
- Send 버튼

중단 탭 영역

- Params: Query Parameter 설정
- Headers: Header 설정
- Body: 요청 데이터 작성
- Authorization: 인증 설정

하단 영역

- Response Body: 서버 응답 데이터

- Status Code: HTTP 상태 코드
- Response Time: 응답 시간

7-5. Postman으로 GET 요청 보내기 (실습 예제)

7.5.1 예제 API

카카오 책 검색 API (제목 기준 검색)

7.5.2 비교용 AJAX 코드1

```
$.ajax({
  method: "GET",
  url: "https://dapi.kakao.com/v3/search/book?target=titl
e",
  data: {
    query: "자바스크립트",
    size: 10
  },
  headers: {
    Authorization: "KakaoAK {REST_API_KEY}"
  }
});
```

7.5.3 비교용 AJAX 코드2 ⇒ done, fail 포함 구문

```
$.ajax({
  method: "GET",
  url: "https://dapi.kakao.com/v3/search/book?target=titl
e",
  data: { query: "자바스크립트", size: 10 },
  headers: { Authorization: "KakaoAK YOUR_REST_API_KEY" },
})
.done(function(response) {
  console.log(response.documents);
})
.fail(function() {
```

```
  alert("API 호출 실패");
});
```

7.5.3 Postman 설정 방법

1) Method 선택

- GET

2) Request URL 입력

```
https://dapi.kakao.com/v3/search/book
```

3) Params 설정

Key	Value
target	title
query	자바스크립트
size	10

자동 생성 URL:

```
https://dapi.kakao.com/v3/search/book?target=title&query=자바스크립트&size=10
```

4) Headers 설정

Key	Value
Authorization	KakaoAK {REST_API_KEY}

주의: KakaoAK 뒤에는 공백이 반드시 필요하다.

5) 요청 전송

- Send 클릭
- Response 영역에서 결과 확인

7-6. 정상 응답 예시

```
{
  "documents": [
    {
      "title": "자바스크립트 완벽 가이드",
      "authors": ["David Flanagan"],
      "price": 45000
    }
  ],
  "meta": {
    "total_count": 1234
  }
}
```

7-7. AJAX와 Postman 설정 대응표

AJAX	Postman
method	Method
url	Request URL
data	Params
headers	Headers
success / error	Response

7-8. 자주 발생하는 오류

8.1 401 Unauthorized

- Authorization 헤더 누락
- REST API 키 오류
- KakaoAK 문자열 오타

8.2 400 Bad Request

- 필수 파라미터(query) 누락
- 파라미터 이름 오타

7-9. 카카오 REST API 간단 사용법

7.9.1 애플리케이션 등록 및 REST API 키 발급

- 카카오 개발자 사이트 접속: https://developers.kakao.com
- 카카오 계정으로 로그인
- 애플리케이션 등록 후 REST API 키 발급

7.9.2 카카오 개발자 사이트 접속 및 로그인

- 접속 주소: https://developers.kakao.com
- 카카오 계정으로 로그인한다.

7.9.3 애플리케이션 등록 절차

1. 상단 메뉴에서 **내 애플리케이션** 클릭
2. **새 애플리케이션 추가하기** 버튼 클릭
3. 애플리케이션 이름 입력 (예: Book Search App)
4. 회사명은 개인일 경우 임의 입력 가능 (예: Personal)
5. 저장 클릭

7.9.4 REST API 키 확인 방법

1. 애플리케이션 등록 후 앱 설정 화면으로 이동
2. 좌측 메뉴에서 **앱 키** 선택
3. 아래 키 목록 확인
    - JavaScript 키
    - REST API 키 (사용 대상)
    - Admin 키 (백엔드 전용)
4. **REST API 키** 복사

7.9.5 API 사용을 위한 플랫폼 설정

1. 좌측 메뉴에서 **플랫폼** 선택
2. 플랫폼 추가

3. 웹 사용 시 **웹 플랫폼** 선택
4. 도메인 입력
    - 개발 환경: http://localhost
    - 운영 환경: 서비스 도메인 (예: https://example.com)

7.9.6 카카오 책 검색 API 사용 설정

- 좌측 메뉴의 **제품 설정 > 카카오 API** 확인
- 책 검색 API는 별도 활성화 없이 즉시 사용 가능

7-10. 정리

Postman은 REST API 요청을 코드 없이 테스트할 수 있는 도구이며,

카카오 REST API 사용 시에는 반드시 **공식 개발자 문서에서 URL과 파라미터를 확인**하고

REST API 키를 발급받아 Authorization 헤더에 포함해야 한다.

Method, URL, Params, Headers를 문서에 맞게 설정하면

프론트엔드 코드와 동일한 API 요청을 보낼 수 있다.

---

8. 비동기 AJAX 호출 예제

XMLHttpRequest + Promise 감싸기

```
function getData(url) {
  return new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url, true);
    xhr.onload = () => {
      if(xhr.status >= 200 && xhr.status < 300) {
        resolve(JSON.parse(xhr.responseText));
      } else {
        reject(new Error(`HTTP error: ${xhr.status}`));
      }
    };
```

```
    xhr.onerror = () => reject(new Error('Network error'));
    xhr.send();
  });
}

getData('https://api.example.com/data')
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Fetch API (Promise 기본 제공)

```
fetch('https://api.example.com/data')
  .then(res => {
    if (!res.ok) throw new Error('Network response error');
    return res.json();
  })
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

async/await 활용

```
async function fetchData() {
  try {
    const res = await fetch('https://api.example.com/dat
a');
    if (!res.ok) throw new Error('Network error');
    const data = await res.json();
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}

fetchData();
```