



# [스파르타코딩클럽] 리액트 심화반 3주차

## [수업 목표]

1. 파일 객체를 다뤄보고 이미지를 처리해본다.
2. mock API를 만들고 사용해본다.
3. form을 다뤄본다.
4. 자바스크립트 함수에 대해 조금 더 깊이 알아본다.

## [목차]

- 01. 오늘 배울 것
  - 02. Mock API 만들기
  - 03. 네트워크 요청 1 - XMLHttpRequest
  - 04. 네트워크 요청 2 - Fetch API와 Axios
  - 05. Quiz\_ 핑하면 풍하기!
  - 06. Polling과 long polling
  - 07. 토큰 기반 인증
  - 08. 웹 저장소
  - 09. 인증처리 - 1. Firebase Authentication 설정
  - 10. 인증처리 - 2. 회원가입 구현하기 (feat. firestore)
  - 11. 인증처리 - 3. 로그인 구현하기
  - 12. Image Upload
  - 13. 끝 & 숙제 설명
- 

## 01. 오늘 배울 것

- ▼ 1) 네트워크 요청, http 프로토콜, 웹 저장소, 인증

## 02. Mock API 만들기



이번 시간은 Mock API를 만들고 테스트까지 해볼건데요!

테스트하기 위한 준비물이 있습니다. 바로바로 POST MAN이라는 프로그램이에요.

포스트맨 사이트에서 포스트맨을 설치하신 후 강의를 진행해주세요! ([설치하러 가기](#) →)

## ▼ 2) Rest API란?



**mock API를 배우기 전에 잠깐!**

REST API가 뭔지 살짝만 읽어보고 가요. 😊

[REST API]

REST API는 REpresentational State Transfer API의 약어예요. 소프트웨어 개발 아키텍처의 한 형태로, 클라이언트와 서버 사이 통신 방식 중 하나입니다.

RESTful은 REST API를 제공하는 웹서비스를 말합니다.

### ▼ REST 구성

- 자원: URI
- 행위: Http method (GET, POST, PUT, DELETE)
- 표현: JSON, XML, ... (대부분의 경우 JSON을 주고 받습니다.)

### ▼ REST 특징

- 클라이언트 - 서버 구조
- 무상태성
- 캐시 처리 기능
- 자체 표현 구조
- 계층화
- 인터페이스 일관성

## ▼ 3) mock API란?



**Mocking API**

mocking api는 실제로 REST API가 구축되어 있지 않더라도, 있는 것처럼 테스트 환경을 만들어주는 거예요. 😊

실제로 뭔가 저장하진 않고 흉내만 내는 거니까 당연히 프로덕션 레벨에서 사용할 수 없어요. API 준비가 덜 된 상태로 개발 진행을 한다거나 지금처럼 공부할 때 쓰기 좋습니다.

## ▼ 4) json-server 설치



json-server는 json 파일로 REST API mock server 구축을 해주는 패키지입니다.

- 폴더 생성

```
mkdir my_mock_api
```

- 프로젝트 초기화

```
yarn init -y
```

- 설치

```
yarn add json-server
```

#### ▼ 5) json-server로 일주일 수면 시간을 알려주는 API 만들기

##### ▼ db.json 만들기

👉 db.json은 디비 역할을 해줄 파일입니다.

```
{
  "sleep_times": [
    {
      "id": 0,
      "day": "월",
      "sleep_time": "10:00"
    },
    {
      "id": 1,
      "day": "화",
      "sleep_time": "10:00"
    },
    {
      "id": 2,
      "day": "수",
      "sleep_time": "10:00"
    },
  ]
}
```

##### ▼ json-server 실행하기

👉 DB를 만들었으니 json-server가 db를 보고 있도록 해줘야 합니다.  
—watch로 db.json을 보도록 해줄 수 있고, 실행할 포트는 —port를 사용해서 지정할 수 있어요. (지정하지 않을 경우, 기본값은 3000번 포트예요!)

```
yarn json-server --watch db.json --port 5001
```

##### ▼ json-server 실행 커맨드 추가하기



매번 실행 명령어를 입력하긴 귀찮죠! 실행 명령어를 추가해봅시다.

```
// package.json
...
"scripts": {
  ...
  "server-start": "json-server --watch db.json --port 5001"
},
...
```

#### ▼ 6) API 테스트하기



포스트맨을 열어봅시다!

GET과 POST를 사용해서 데이터를 가져오고, 넣어줘봅시다.

##### ▼ GET 요청 보내기

- 메시드에 GET을 넣고,
- 주소를 넣어줍니다. endpoint는 sleep\_times예요.

##### ▼ POST 요청 보내기

- 메시드에 POST를 넣고,
- 주소를 넣어줍니다. endpoint는 sleep\_times예요.
- Body에 추가할 데이터를 넣어봅시다.

## 03. 네트워크 요청 1 - XMLHttpRequest

#### ▼ 7) 서버와 클라이언트 동작방식

- 서버와 클라이언트



클라이언트는 요청을 하고, 서버는 응답을 한다!

네! 또 찾아왔습니다. 잊을만 하면 찾아오는 서버와 클라이언트 동작방식 알아보기 시간입니다.



아마 리액트 심화반을 다 들으실 때 즈음엔 제가 서..까지만 말하면 여러분이 버와 클라이언트!라고 말씀하실지도 모르겠네요. 😊

- 서버와 클라이언트가 뭘 통해서 소통할까? → REST API!
- 클라이언트는 REST API를 호출하고, 서버는 응답을 보내준다!

#### ▼ 8) XMLHttpRequest



XMLHttpRequest 객체는 서버와 통신하기 위해 사용하는 객체예요.

조금 달리 말해보면 브라우저는 XMLHttpRequest를 이용해서 Ajax요청을 생성하고 전송한다고 할 수 있습니다.

이때, 서버가 브라우저의 요청에 대한 어떤 응답을 돌려주면 같은 XMLHttpRequest 객체가 그 결과를 처리해요!

XML뿐만 아니라 JSON, text 등등 모든 종류의 데이터를 받아오는데 사용할 수 있어요.

HTTP이외의 프로토콜도 지원합니다. (file, ftp도 지원해요!)

([docs 보러가기](#) →).

#### ▼ XMLHttpRequest 객체 생성하기

```
let xhr = new XMLHttpRequest();
```

#### ▼ 요청 만들기

```
xhr.open('GET', 'http://localhost:5001/sleep_times');
```

#### ▼ 요청 보내기

```
xhr.send();
```

#### ▼ 응답 받기

```
// 두가지 방법으로 응답을 받을 수 있어요! 하나는 onreadystatechange를 쓰는 방법,  
// 다른 하나는 onload를 쓰는 방법입니다.  
  
// 1.  
// XMLHttpRequest.readyState 프로퍼티가 변경(이벤트 발생)될 때마다  
// 콜백함수(이벤트 핸들러)를 호출해요!  
xhr.onreadystatechange = function (e) {  
    // 이 함수는 Response가 클라이언트에 도달하면 호출됩니다!  
  
    // readyStates는 XMLHttpRequest의 상태(state)를 반환해요  
    // readyState :::  
    // UNSENT = 0; // XMLHttpRequest.open() 메소드 호출 이전  
    // OPENED = 1; // XMLHttpRequest.open() 메소드 호출 완료  
    // HEADERS_RECEIVED = 2; // XMLHttpRequest.send() 메소드 호출 완료  
    // LOADING = 3; // 응답 기다리는 중(서버 응답 중(XMLHttpRequest.responseText 미완성 상태))  
    // DONE = 4; // 서버 응답 완료  
    if (xhr.readyState !== XMLHttpRequest.DONE) return;  
  
    // status는 response 상태 코드를 반환 : 200 => 정상 응답  
    if(xhr.status === 200) {  
        console.log(xhr.responseText);  
    } else {  
        console.log('Error!');  
    }  
}  
};  
  
//2.  
// load 이벤트는 서버 응답이 완료된 경우에 발생해요!
```

```
xhr.onload = function (e) {
  // status는 response 상태 코드를 반환 : 200 => 정상 응답
  if(xhr.status === 200) {
    console.log(xhr.responseText);
  } else {
    console.log('Error!');
  }
};
```

## 04. 네트워크 요청 2 - Fetch API와 Axios

### ▼ 9) Fetch API



#### Fetch API

Fetch API도 서버와 통신하기 위해 사용하는, XMLHttpRequest의 대체제예요.  
XMLHttpRequest보다 훨씬 사용법이 간단하고 서비스 워커 등 다른 기술을 쓸 때 손쉽게 엮어 사용할 수 있어요.  
([사용법 docs 보러가기](#) →).



fetch()는 구식 브라우저에서는 돌아가지 않아요!

### ▼ .fetch() 생김새 보기



#### fetch()를 호출하면?

브라우저는 네트워크 요청을 보내고 프라미스를 반환됩니다.  
이 프라미스를 사용해서 fetch()를 호출해요!

```
// url - 접근하고자 하는 URL
// options - 선택 매개변수, method나 header 등이 여기에 들어가요!
let promise = fetch(url, [options])
```

### ▼ .fetch() 써보기

#### • get

```
const callSomething = async () => {
  let response = await fetch('http://localhost:5001/sleep_times');

  console.log(response);
}
```

#### • post

```
const callSomething = async () => {
  let data = {
    "id": 6,
    "day": "일",
    "sleep_time": "10:00"
  },
  let response = await fetch('http://localhost:5001/sleep_times', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json;charset=utf-8'
    },
    body: JSON.stringify(data)
  });
  console.log(response);
}
```

## ▼ 10) Axios



XMLHttpRequest를 배이스로한 프라미스 기반 HTTP 클라이언트 라이브러리입니다.  
 악시오스는 node 런타임과 브라우저 양측에서 모두 돌아가요.  
 악시오스는 오늘 알아본 통신 방법 중 가장, 가장 기능이 많은 친구입니다.  
 그 기능을 잘 소개해주는 굉장히 대단한 공식문서가 준비되어 있으니 꼭꼭 읽어보시길 바랍니다!  
**([axios 공식문서 보러가기 →](#))**

### ▼ (1) 설치하기

```
yarn add axios
```

### ▼ (2) 요청하고 응답 받기

- config로 GET 요청해보기

```
axios({
  method: 'get',
  url: 'http://localhost:5001/sleep_times',
})
.then((response) => {
  console.log(response);
});
```

- 요청 메서드로 GET 요청해보기

```
axios.get("http://localhost:5001/sleep_times");
```

- 요청 메서드로 POST 요청해보기

```
let data = {
  id: 6,
  day: "일",
```

```

    sleep_time: "10:00"
  };

  axios.post("http://localhost:5001/sleep_times", data);

```

## 05. Quiz\_핑하면 풍하기!

▼ 11) 🛠️ 지금까지 배운 내용을 바탕으로 핑을 입력하면 풍하고 alert을 띄워주세요!



### [해야할 것 순서 정리]

1. mock api 만들기
2. 버튼 만들기
3. fetch나 axios로 "http://localhost:5001/ping"으로 get 요청하기
4. 버튼 눌러서 확인하기

### ▼ [코드스니펫] - App.js

```

import React from "react";
import "./App.css";
import axios from "axios";
function App() {
  // fetch로 핑!
  const fetchPing = async () => {
    const res = await fetch("http://localhost:5001/ping", {
      headers: { "Content-Type": "application/json" },
    });

    const data = await res.json();

    return data;
  };

  const axiosPing = async () => {
    const res = await axios.get("http://localhost:5001/ping", {
      headers: { "Content-Type": "application/json" },
    });

    return res;
  };
  return (
    <div className="App">
      <button onClick={fetchPing}>fetch로 요청하기</button>
      <button onClick={axiosPing}>axios로 요청하기</button>
    </div>
  );
}

export default App;

```

### ▼ [코드스니펫] - db.json

```

{"ping": {"answer": "pong"}}

```



## 06. Polling과 long polling

### ▼ 11) http요청은 유지가 될까?



#### HTTP 프로토콜

클라이언트는 요청을 하고, 서버는 응답을 한다! 그럼 이 요청과 응답은 영원히 이어질까요?  
정답은 **XXX**! 한 번 요청을 보내고 응답을 받으면 그 요청은 거기서 끝이 납니다.

- HTTP의 특징인 비연결성 때문입니다.
- 요청을 주면, 응답을 준다!
- 클라이언트는 요청만, 서버는 응답만 가능!

### ▼ 12) 바뀐 데이터를 받아오기 위한 기법, 폴링



#### 폴링

브라우저가 일정 주기를 두고 요청을 보내는 것

- 주기적으로 요청을 보내고 그에 따른 응답을 받아온다.
- 실시간으로 바뀐 데이터를 받아올 수 없다.

### ▼ 13) 짧은 폴링이 부하를 일으킨다면, 롱 폴링



#### 롱 폴링

요청을 보내고 응답을 지연시키는 것

- 요청을 보내고 서버가 그 응답을 바로 보내지 않는 것
- 데이터의 변동이 있다면 바로 응답을 보낸다.
- 데이터 변동이 없다면 서버는 응답을 일정 시간동안 지연한다.

## 07. 토큰 기반 인증

### ▼ 19) 토큰 기반 인증이란?



### [옛날 이야기 - 세션 기반 인증]

예전에는 **사용자의 로그인 상태를 서버가 전부 가지고** 있었어요. **서버의 세션에 사용자 정보를 넣고 이 사람이 로그인을 했다 안했**다를 전부 기록하고 기억했습니다.

이 세션은 서버의 메모리나 데이터베이스 등에 저장해두는데, 로그인한 사용자가 많아지면 서버에 부하가 많이 오겠죠?

그렇다고 서버를 여러개 놓자니 관리가 까다로워지고요.

→ 그래서 최근에는 오늘 배울 **토큰 기반 인증 방법을 많이 사용**해요!



### [토큰 기반 인증]

토큰 기반 인증은 사용자가 **맞아!**라는 정보를 가진 토큰으로 서로 사용자임을 확인하자는 프로토콜이에요.

공연의 입장권 같은 거라고 생각하시면 좋습니다.

## ▼ 20) OAuth2.0



### OAuth2.0란?

외부서비스의 인증 및 권한부여를 관리하는 프레임워크입니다!

→ Open Authentication, Open Authorization라고 해요. (인증과 허가를 포함해요!)

#### • OAuth 동작 방식 (간단 ver.)

1. **클라이언트** 와 **서버** 사이에 인증(로그인)을 하면 서버가 **access\_token** 을 줍니다.
2. **클라이언트** 는 **access\_token** 을 이용해서 API 요청을 할 수 있어요.
3. **서버** 는 API 요청을 받고, **access\_token** 을 가지고 권한이 있나 없나 확인해서 결과를 **클라이언트** 에 보내줍니다.

#### • OAuth 동작 방식 (외부 서비스 엮음 ver.)



유저가 구글 로그인을 하는 상황이라고 가정하고 생각해봅시다!

공식적으로 쓰는 용어와 함께 가정해볼게요. :)

여기서 **구글**은 **유저의 정보**도 가지고 있을거고, **로그인을 검증**도 해줄거예요. 그러니까 구글은 Resource Server(자원 서버) + Authorization server(권한 서버)겠죠!

유저는 그 정보를 가진 사람이니 Resource Owner(자원 소유자)라고 부릅니다.

(구글에서 주는 유저 정보는 Resource(자원)이라고 부릅니다.)

1. 유저가 **구글** 로그인을 합니다.
  - 자원 소유자가 자원서버에 권한 요청을 한거죠!
2. **구글** 은 로그인할 때 유저가 입력한 정보(아이디, 비밀번호 등)을 보고 **클라이언트(우리 웹사이트!)** 에 접근 권한을 줍니다.

3. 클라이언트는 이 권한을 가지고 `Authorization server(권한 서버)`에 `access_token`을 요청합니다.
4. 클라이언트는 이 `access_token`을 가지고 구글에서 `유저 정보`를 가져올 수 있어요.
5. 구글은 클라이언트가 보낸 `access_token`을 가지고 권한이 있나 없나 확인해서 `결과`를 클라이언트에 보내줍니다.

## ▼ 21) JWT(Json Web Token)



### JWT란?

토큰의 한 형식입니다! 데이터가 JSON 형태로 이루어져 있는 토큰이에요.

- 생김새 : **[header].[payload(내용)].[signature(서명)]**
  - header: 토큰 타입과 암호화 방식 정보가 들어갑니다.
  - payload: 토큰에 담은 정보가 name: value 쌍으로 들어갑니다.
  - signature: 서명 정보입니다. secret key를 포함해서 header와 payload 정보가 암호화 되어 들어갑니다.
- 동작 방식 : 토큰 기반 동작 방식으로 움직여요!
  - 유저가 로그인을 시도하면,
  - 서버가 요청을 확인하고 secret key를 가지고 access\_token을 발급합니다.
  - 클라이언트에 JWT를 전달하고
  - 클라이언트는 API 요청을 할 때 Authorization header에 JWT를 담아서 보냅니다.
  - 서버는 JWT의 서명을 확인하고 payload에서 정보를 확인해서 API 응답을 보냅니다.



### JWT vs OAuth?

JWT와 OAuth는 로그인에 많이 쓰이는 두 인증 방식입니다.  
 뭘 택할지 자주 고민하게 되지만 사실 비교하긴 조금 애매해요.  
 JWT는 토큰의 한 형식이고 OAuth는 프레임워크거든요.  
 (OAuth에서 토큰으로 JWT를 사용할 수도 있고요. 😊)



### Authorization 헤더

HTTP Authorization 헤더에 대해 더 알고 싶다면 [이 문서\(링크\)](#)를 참고해주세요!

## 08. 웹 저장소



Http는 1번 요청을 하고 응답을 받으면 연결이 해제됩니다!

즉, 우리가 access\_token을 클라이언트 어딘가에 저장해두어야만 하는 이야기예요!

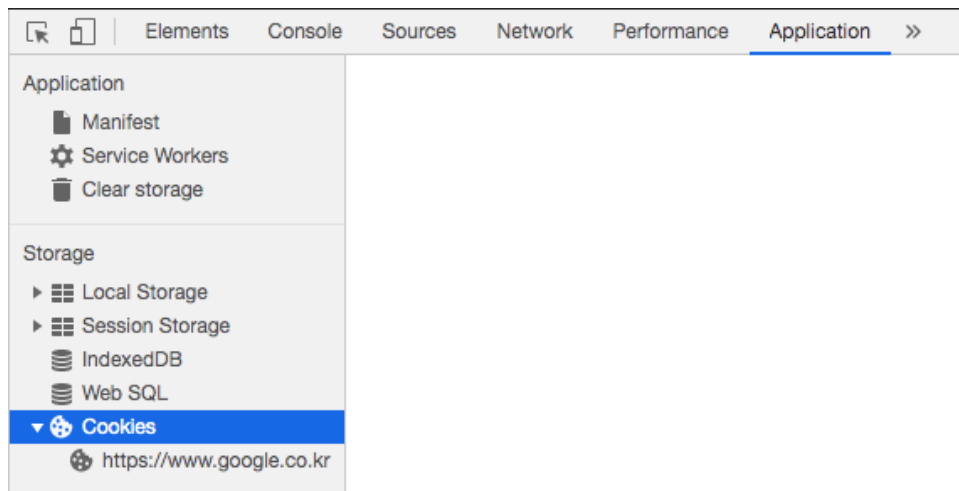
그럼 이 토큰을 어디에 저장하면 좋을지, 클라이언트에서 쓸 수 있는 저장소를 알아보시다. 😊

## ▼ 9) 클라이언트 저장소



클라이언트 저장소 확인하기

개발자도구 → Application 탭 → 좌측 Storage 확인!



## ▼ 10) 쿠키



클라이언트 로컬에 저장되는 **key:value** 형태의 저장소입니다!

약 4KB 정도 저장할 수 있어요.

### • 쿠키 만들기

브라우저에서 개발자 도구를 열고 해봅시다!

이 때 꼭, 어떤 사이트로 들어가신 후에 여셔야 쿠키를 사용할 수 있습니다.

```
// key는 MY_COOKIE, value는 here,
document.cookie = "MY_COOKIE=here;"
```

### • 쿠키 가져오기


```
console.log(document.cookie);
```

### • 쿠키 삭제

쿠키를 삭제할 때는 만료일을 이전으로 돌려서 지우는 방법을 많이 씁니다.

```
document.cookie = "MY_COOKIE=here; expires=new Date('2020-12-12').toUTCString()";
```

## ▼ 11) 세션 스토리지

 HTML5에서 추가된 저장소입니다! 쿠키와 마찬가지로 **key:value** 형태의 저장소예요.

- 세션 스토리지에 저장된 데이터는 브라우저를 닫으면 제거돼요!  
→ 자동 로그인이나, 장바구니같은 **다음에 브라우저를 열었을 때도 유지해야하는 데이터**는 넣기 어렵겠죠!
- 추가하기

```
// key는 MY_SESSION, value는 here인 세션을 만들어요.  
sessionStorage.setItem("MY_SESSION", "here");
```


- 가져오기

```
// key값으로 쉽게 가져올 수 있어요 :)  
sessionStorage.getItem("MY_SESSION");
```

- 삭제하기

```
// 하나만 삭제하고 싶다면, 이렇게 키를 통해 삭제합니다.  
sessionStorage.removeItem("MY_SESSION");  
  
// 몽땅 지우고 싶을 땐 clear()를 쓰면 됩니다. :)  
sessionStorage.clear();
```

## ▼ 12) 로컬 스토리지

 HTML5에서 추가된 저장소입니다! 쿠키와 마찬가지로 **key:value** 형태의 저장소예요.

- 로컬 스토리지는 따로 지워주지 않으면 계속 브라우저에 남아 있어요.  
→ 유저의 아이디, 비밀번호같은 중요한 정보를 넣어두면 아주 위험해요!
- 추가하기

```
// key는 MY_LOCAL, value는 here인 데이터를 저장해요.  
localStorage.setItem("MY_LOCAL", "here");
```

- 가져오기

```
// key값으로 쉽게 가져올 수 있어요 :)
localStorage.getItem("MY_LOCAL");
```

- 삭제하기

```
// 하나만 삭제하고 싶다면, 이렇게 키를 통해 삭제합니다.
localStorage.removeItem("MY_LOCAL");

// 몽땅 지우고 싶을 땐 clear()를 쓰면 됩니다. :)
localStorage.clear();
```

### ▼ 13) 토큰은 어디에 저장해야할까?



예전에는 토큰을 저장할만한 곳이 쿠키 밖에 없었어요. 😞  
HTML5가 나온 후부터는 LocalStorage 에도 토큰을 저장하는 일이 많아졌습니다.

- 왜 쿠키보다 로컬 스토리지에 저장했을까?
  - 쿠키보다 더 많은 정보를 저장할 수 있다.  
(쿠키 4KB, 로컬 스토리지 5MB)
  - 쿠키처럼 모든 http 통신에 딸려들어가지 않는다.
- 그럼 무조건 로컬 스토리지에 토큰을 넣을까?
  - 아니요! 로컬 스토리지는 말그대로 로컬에 데이터가 다 남아있으니 보안상 취약해지기 쉬워요.  
**프로젝트 성향에 맞춰 저장 장소는 그때그때 달라져야 합니다.**

## 09. 인증처리 - 1. Firebase Authentication 설정

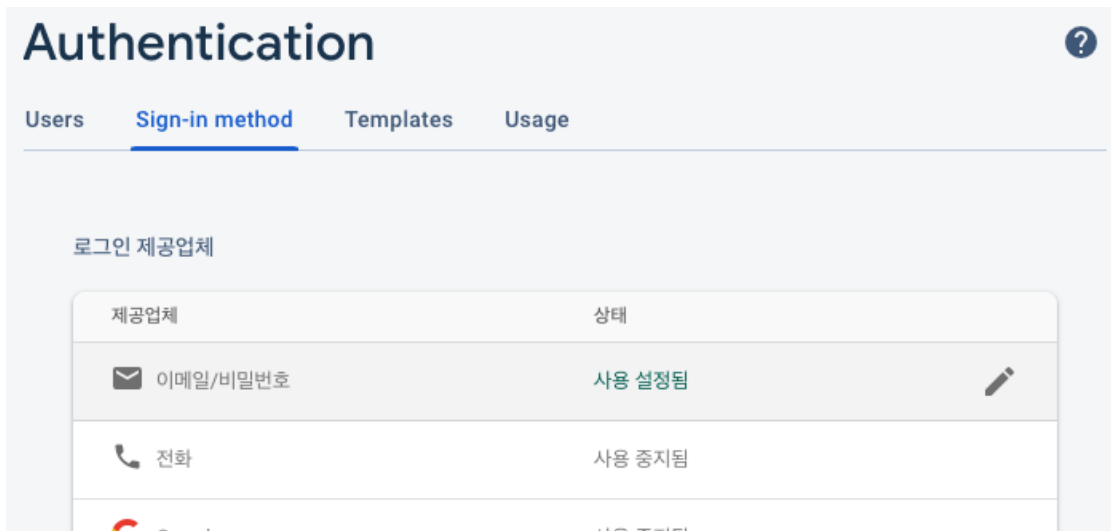
### ▼ 22) Firebase Authentication 둘러보기



파이어베이스에서 제공하는 인증 기능이에요.  
구글 로그인, 페이스북 로그인 같은 소셜 로그인 기능, 파이어베이스에 직접 사용자를 추가(회원가입), 관리하는 기능을 제공합니다. 😊  
([공식 문서 바로가기](#) →)

### ▼ 23) Authentication 설정하기

- 파이어베이스 대시보드 → Authentication에서 이메일/비밀번호 항목을 사용설정 해줍니다.



- 파이어베이스 설치하기

```
# 패키지 설치
yarn add firebase
```

- shared/firebase.js에서 auth 설정을 해줍니다!

```
import firebase from "firebase/app";
import "firebase/auth";

const firebaseConfig = {
  // 인증정보!
};

firebase.initializeApp(firebaseConfig);
const auth = firebase.auth();

export { auth };
```

## 10. 인증처리 - 2. 회원가입 구현하기 (feat. firestore)

### ▼ 24) Firestore 설정하기

👉 firebase authentication은 우리가 원하는 모든 회원 정보를 담을 수 없어요. 😞  
우리는 firestore를 써서 집주소를 저장해볼거예요!

- 파이어베이스 대시보드 → Firestore로 이동합니다!
- shared/firebase.js에서 firestore 설정을 해줍니다!

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";

const firebaseConfig = {
  // 인증 정보!
};

const app = initializeApp(firebaseConfig);

export const auth = getAuth();
export const db = getFirestore(app);
export default app;
```

## ▼ 25) 회원가입



### 중요한 건 순서!

저희는 백엔드 api가 없이도 할 수 있도록 firebase authentication을 사용하지만, 백엔드 api를 사용한다고 해서 순서가 달라지진 않아요. (구현 방식은 쯤~금 다를 수 있지만요!)

중요한 건,

**1. 회원 정보를 받고, 2. 회원가입을 위한 서버 요청을 하고, 3. 응답(토큰 등!)을 받아온다!**  
이 순서입니다.

그 외의 처리는 그때 그때 사용하는 것에 따라, 프로젝트 요구사항에 따라 달라질 수 있어요.

## ▼ firebase Authentication과 firestore를 사용한 회원 가입하기

### 1. firebase.js에 만들어둔 auth, firestore 가져오기

```
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";

// TODO: Replace the following with your app's Firebase project configuration
const firebaseConfig = {
  // 인증정보!
};

const app = initializeApp(firebaseConfig);

export const auth = getAuth();
export const db = getFirestore(app);
export default app;
```

### 2. auth.createUserWithEmailAndPassword()로 가입 시키기

```
import { createUserWithEmailAndPassword } from 'firebase/auth';
import './App.css';
import { auth, db } from './shared/firebase';
```



```
function App() {
  const signup = async () => {

    const user = await createUserWithEmailAndPassword(auth, "devdev@aaa.com", "devdev123!");
    console.log(user);
  }

  return (
    <div className="App">

      <button onClick={signup}>회원가입</button>
    </div>
  );
}

export default App;
```

### 3. 가입에 성공했다면? → firestore에 유저 정보를 바로 업데이트하기

```
import { createUserWithEmailAndPassword } from 'firebase/auth';
import './App.css';
import { auth, db } from './shared/firebase';
import { collection, addDoc } from "firebase/firestore";

function App() {
  const signup = async () => {

    const user = await createUserWithEmailAndPassword(auth, "devdev@aaa.com", "devdev123!");
    console.log(user);

    const user_data = await addDoc(collection(db, "users"), {user_id: "1111", name: "perl"});
    console.log(user_data.id);
  }

  return (
    <div className="App">

      <button onClick={signup}>회원가입</button>
    </div>
  );
}

export default App;
```

#### ▼ 회원 가입 화면 만들고 회원가입하기

- 뷰 만들기

```
//Signup.js
import React from "react";

const Signup = () => {
  const id_ref = React.useRef();
  const name_ref = React.useRef();
  const pw_ref = React.useRef();

  return (
    <div>
      아이디 : <input ref={id_ref} /> <br />
      이름 : <input ref={name_ref} /> <br />
      비밀번호 : <input ref={pw_ref} />
    </div>
  );
}
```

```

        <button>회원가입</button>
      </div>
    );
  };

export default Signup;

```

- 회원 가입 유틸리티

- App.js

```

import './App.css';
import { auth, db } from './shared/firebase';

import React from 'react';
import Signup from './Signup';

function App() {

  return (
    <div className="App">
      <Signup />
    </div>
  );
}

export default App;

```

- Signup.js

```

import React from 'react';
import {
  createUserWithEmailAndPassword,
  onAuthStateChanged,
} from 'firebase/auth';
import './App.css';
import { auth, db } from './shared/firebase';
import { collection, addDoc } from 'firebase/firestore';

const Signup = () => {
  const id_ref = React.useRef();
  const name_ref = React.useRef();
  const pw_ref = React.useRef();

  const signupFB = async () => {
    const user = await createUserWithEmailAndPassword(
      auth,
      id_ref.current.value,
      pw_ref.current.value
    );
    console.log(user);

    const user_data = await addDoc(collection(db, "users"), {
      user_id: id_ref.current.value,
      name: name_ref.current.value,
    });
    console.log(user_data.id);
  };

  return (
    <div>
      아이디 : <input ref={id_ref} /> <br />
      이름 : <input ref={name_ref} /> <br />
      비밀번호 : <input ref={pw_ref} />
    </div>
  );
};

```

```

        <button onClick={signupFB}>회원가입</button>
      </div>
    );
  };

  export default Signup;

```

## 11. 인증처리 - 3. 로그인 구현하기

### ▼ 26) 로그인하기

👉 로그인 화면을 만들고 id, pw를 입력한 다음 로그인하도록 만들어봅시다!  
그리고 로그인 여부에 따라 로그아웃 버튼을 볼 수 있게끔도 해봐요. 😊

- 뷰 만들기

#### ▼ Login.js

```

import React from "react";

const Login = () => {
  const id_ref = React.useRef();
  const pw_ref = React.useRef();

  return (
    <div>
      아이디 : <input ref={id_ref} /> <br />
      비밀번호 : <input ref={pw_ref} />
      <br />
      <button>로그인</button>
    </div>
  );
};

export default Login;

```

- 로그인

#### ▼ Login.js

```

import React from "react";
import { signInWithEmailAndPassword } from "firebase/auth";
import { auth } from "../shared/firebase";

const Login = () => {
  const id_ref = React.useRef();
  const pw_ref = React.useRef();

  const loginFB = async () => {
    const user = await signInWithEmailAndPassword(
      auth,

```

```

        id_ref.current.value,
        pw_ref.current.value
    );
};

return (
    <div>
        아이디 : <input ref={id_ref} /> <br />
        비밀번호 : <input ref={pw_ref} />
        <br />
        <button onClick={loginFB}>로그인</button>
    </div>
);
};

export default Login;

```

### ▼ App.js

```

import './App.css';
import { auth, db } from './shared/firebase';
import { collection, addDoc } from 'firebase/firestore';

import React from 'react';
import Signup from './Signup';
import Login from './Login';

function App() {

    return (
        <div className="App">
            <Signup />
            <Login />
        </div>
    );
}

export default App;

```

## • 로그인 체크하기

### ▼ App.js

```

import { onAuthStateChanged } from 'firebase/auth';
import './App.css';
import { auth } from './shared/firebase';

import React from 'react';
import Signup from './Signup';
import Login from './Login';

function App() {
    const [is_login, setIsLogin] = React.useState(false);

    console.log(auth.currentUser);

    const loginCheck = async (user) => {
        if (user) {
            setIsLogin(true);
        } else {
            setIsLogin(false);
        }
    };
};

```

```

    React.useEffect(() => {
      onAuthStateChanged(auth, loginCheck);
    }, []);

    return (
      <div className="App">
        <Signup />

        {is_login ? (
          <div>
            <p>환영합니다!</p>
            <button>로그아웃</button>
          </div>
        ) : (
          <Login />
        )}
      </div>
    );
  }
}

export default App;

```

- 로그아웃

- ▼ App.js

```

import { onAuthStateChanged, signOut } from "firebase/auth";
import "../App.css";
import { auth } from "../shared/firebase";

import React from "react";
import Signup from "../Signup";
import Login from "../Login";

function App() {
  const [is_login, setIsLogin] = React.useState(false);
  console.log(auth.currentUser);

  const loginCheck = async (user) => {
    if (user) {
      setIsLogin(true);
    } else {
      setIsLogin(false);
    }
  };

  const logout = () => {
    signOut(auth).then(() => {
      setIsLogin(false);
    });
  };

  React.useEffect(() => {
    onAuthStateChanged(auth, loginCheck);
  }, []);

  return (
    <div className="App">
      <Signup />

      {is_login ? (
        <div>
          <p>환영합니다!</p>
          <button onClick={logout}>로그아웃</button>
        </div>
      ) : (
        <Login />
      )}
    </div>
  );
}

export default App;

```

```

    ) : (
      <Login />
    )}
  </div>
);
}

export default App;

```

## 12. Image Upload

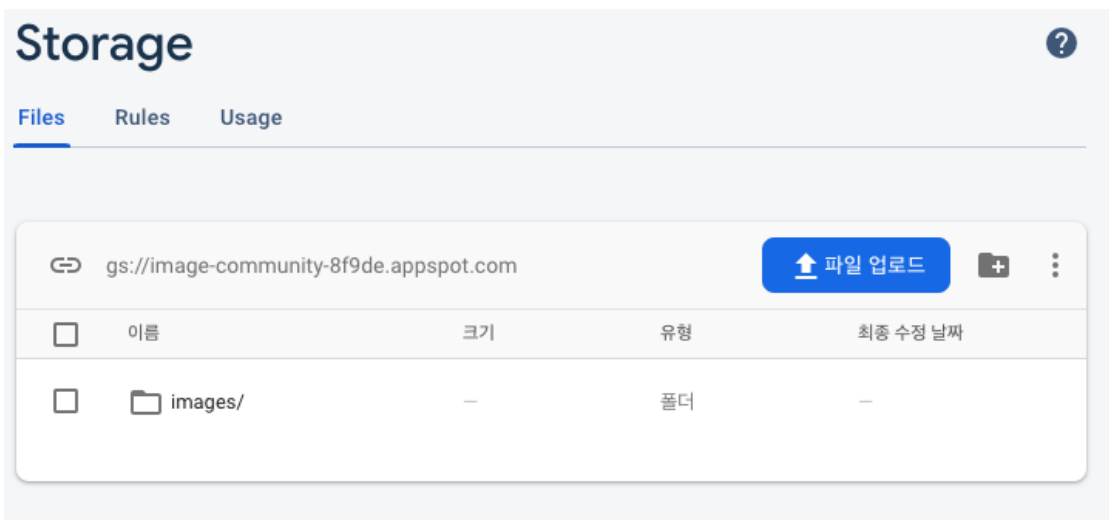
### ▼ 27) 파이어베이스 Storage 만들기

#### ▼ Storage 시작하기



파이어베이스의 대시보드 좌측의 Storage를 클릭  
그리고 시작하기 버튼을 누르면 아래처럼 스토리지를 사용할 수 있어요.

스토리지 사용 설정 하신 후, images 폴더를 하나 만들고 시작하세요!



#### ▼ Storage 설정하기

##### • storage 연결

```

import firebase from "firebase/app";
import "firebase/auth";
import "firebase/firestore";
import "firebase/storage";

const firebaseConfig = {
  apiKey: "여러분의 api key를 여기에!",
  ... // 그 외 설정들이 여기에 들어가요! apiKey와 함께 넣어주세요!
};

```

```

firebase.initializeApp(firebaseConfig);

const apiKey = firebaseConfig.apiKey;
const auth = firebase.auth();
const firestore = firebase.firestore();
const storage = firebase.storage();

export{auth, apiKey, firestore, storage};

```

- rules 세팅하기

```

// 파이어베이스 콘솔 -> Storage에서 규칙(rules) 탭으로 이동!아래처럼 바꿔주기!
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=*}* {
      allow read, write: if request.auth != null;
    }
  }
}

```

## ▼ 28) 이미지 업로드 해보기



src/shared/Upload.js에서 시작! 😎

`<input type="file"/>` 은 파일을 선택하면 onChange 이벤트에 선택한 파일이 객체로 넘어와요.

지금부터 onChange에서 파일 객체를 가져와서 state에 저장해줬다가, 저장 버튼을 누르면 firebase의 Storage에 저장하게 할거예요.

자, 그럼 우리 업로드할 이미지 하나 얼른 준비해서 해볼까요! 😊

## ▼ input에 접근하기 위해 ref를 사용합니다!

```

import React from "react";

const Upload = (props) => {

  const fileInput = React.useRef();

  const selectFile = (e) => {
    // e.target은 input이죠!
    // input이 가진 files 객체를 살펴봅시다.
    console.log(e.target.files);
    // 선택한 파일이 어떻게 저장되어 있나 봅시다.
    console.log(e.target.files[0]);

    // ref로도 확인해봅시다. :)
    console.log(fileInput.current.files[0]);
  };

  return (
    <React.Fragment>
      <input type="file" ref={fileInput} onChange={selectFile} />
    </React.Fragment>
  );
};

```

```
};

export default Upload;
```

## ▼ Storage에 업로드해보기

👉 버튼을 하나 만들고, 버튼을 누르면 스토리지에 업로드 해보기!

👉 **?.** 는 옵셔널 체이닝이라고 해요.

```
const a = {b: 2};
let a_value = a? a.b : 0; → let a_value = a?.b;
```

위와 같이 삼항 연산자 등을 써서 어떤 객체 안에 든 속성이 있나 없나 확인하고 그 속성을 불러와야할 때 유용합니다!

```
import React from "react";
import { Button } from "../elements";
import {storage} from "../firebase";

const Upload = (props) => {
  const fileInput = React.useRef();

  const selectFile = (e) => {
    // e.target은 input이죠!
    // input이 가진 files 객체를 살펴봅시다.
    console.log(e.target.files);
    // 선택한 파일이 어떻게 저장되어 있나 봅시다.
    console.log(e.target.files[0]);

    // ref로도 확인해봅시다. :)
    console.log(fileInput.current.files[0]);
  };

  const uploadFB = () => {
    let image = fileInput.current?.files[0];
    const _upload = storage.ref(`images/${image.name}`).put(image);

    // 업로드!
    _upload.then((snapshot) => {
      console.log(snapshot);
    });
  }

  return (
    <React.Fragment>
      <input type="file" ref={fileInput} onChange={selectFile} />
      <Button _onClick={uploadFB}>업로드하기</Button>
    </React.Fragment>
  );
};

export default Upload;
```



## ▼ 이미지 링크 가져오기

```
const uploadFB = () => {  
  let image = fileInput.current?.files[0];  
  const _upload = storage.ref(`images/${image.name}`).put(image);  
  
  // 업로드!  
  _upload.then((snapshot) => {  
    console.log(snapshot);  
  
    // 업로드한 파일의 다운로드 경로를 가져오자!  
    snapshot.ref.getDownloadURL().then((url) => {  
      console.log(url);  
    });  
  });  
};  
  
}
```



### [알면 재미있는 이야기]

앗! 드디어 <input/>(인풋 태그)의 `type="file"` 을 썼어요!  
유저가 내 컴퓨터에 있는 파일을 가져오게 해줍니다.

유저가 input으로 업로드할 파일을 선택하면, 인풋태그는 `FileList`라는 객체에 파일들을 담아 보관해요.

그리고 이 `FileList`에 있는 파일 하나, 하나가 `File` 객체죠!  
만약 유저가 선택한 파일이 이미지라면, 이 이미지의 이름이 뭔지, 크기는 어떻게 되는 지 같은 정보를 가지고 있어요. (물론 파일 내용도요!)

우리는 이 `File`객체를 사용해서 저장소(우리는 파이어베이스의 `Storage`를 사용하죠!)에 파일을 업로드해주는 거예요.



### [알면 재미있는 이야기]

우리가 사용한 것 중에 조금 생소한 객체가 있었는데, 눈치 채셨나요?  
그렇습니다! `FileReader`라는 객체를 사용했죠!

<input/>(인풋 태그)를 사용해서 유저가 선택한 파일을 `File` 객체로 받아오는 건 이제 알고 있죠!  
`FileReader`는 이 파일의 내용을 읽어 옵니다.  
이게 무슨 뜻이냐면, 파일을 읽어왔으니 미리 보기도 만들 수 있고 다른 이름으로 저장도 할 수 있게 해준다는 거예요. 😎 아주 유용하죠!

더 자세한 내용이 궁금하다면 [링크\(링크\)](#)를 통해 알아볼 수 있어요.

## 13. 끝 & 속제 설명



이번 주차에도 많은 걸 배웠네요! 이번 주차 핵심 내용은 네트워크 요청입니다.

TIL 페이지에서도 네트워크 요청 안해볼 수 없겠죠!

1. TIL을 위한 mock api를 만들어봅시다.
2. axios를 설치하고 mock api로 GET과 POST 요청을 보내봅시다.
3. Main 컴포넌트가 마운트 되는 순간에 GET을 보내기 위해 useEffect를 사용해요!

## ▼ 예시 화면

### • GET

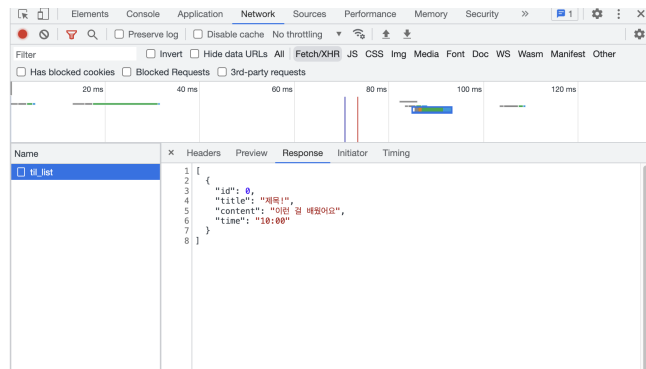
#### TIL

제목!

이런 걸 배웠어요

10:00

추가



### • POST

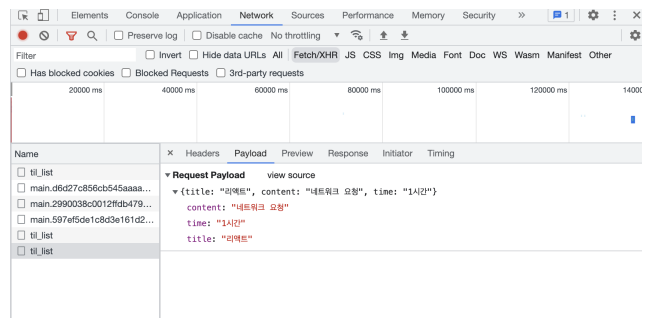
#### TIL

제목!

이런 걸 배웠어요

10:00

추가



[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/fb0954bf-40b7-4882-bd98-ce99f9450ca1/homework\\_w3.zip](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/fb0954bf-40b7-4882-bd98-ce99f9450ca1/homework_w3.zip)