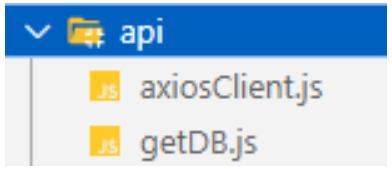
公式HP
https://axios-http.com/ja/docs/intro

インストール；

```
npm install axios
```

詳細コードは上記のリポジトリー確認

1．API設計をするフォルダを作成



2．axiosClient.jsで接続先のURLなどを登録

```
import axios from "axios";

const axiosClient = axios.create({
    baseURL: "http://localhost:5000",
});

export default axiosClient;
```

※server側に本Reactファイルを保存する場合は、外部から通信しないため、
PORT番号以降のエンドポイントを指定したらよい
下記、NotionAppをRenderにデプロイした例

```
import axios from "axios";

// const BASE_URL = "http://localhost:5000/api/v1";
const BASE_URL = "/api/v1";

const getToken = () => localStorage.getItem("token"

const axiosClient = axios.create({
    baseURL: BASE_URL,

});
```

3．getDB.js 内でhttps通信用のObj作成。使用ファイルでdbApiを呼び出す。

```
import axiosClient from "./axiosClient";

const dbApi = {
    getAll: () => axiosClient.get("users"),
    createNewDB: (params) => axiosClient.post("users", params),
    deleteDB: (id) => axiosClient.delete(`users/${id}`),
    updateName: (id, params) => axiosClient.put(`users/${id}`, params),
}

export default dbApi;
```

4．GetAll.jsx 内で３で設定したAPIを叩く例。

```
import React, { useEffect, useState } from 'react'
import dbApi from '../api/getDB';

const createNewDB = async(e) => {
    e.preventDefault();
    try {
        const result = await dbApi.createNewDB({name:newName,email:newEmail,age:newAge})
        console.log(result.data);
        setNewName("");
        setNewEmail("");
        setNewAge("");
    } catch (err) {
        alert(err);
    }
}
```

・裏でDB接続する場合は、async awaitの非同期処理を実施
・try {} catch {} 文でうまくいかなかった場合の対応も書く
・サーバー側の処理としてreq.body内に含めたいオブジェクトはそのまま
　キーも設定してparamsとして渡しにいく。

Node.js Express で出てくる req, res, next について

https://qiita.com/syumiwohossu/items/f9ee317f31adc3ad387b

```
app.get('/hoge', preProcess1, preProcess2, mainProcess);
function preProcess1(req, res, next) {
    console.log('１個目の前処理したよ！');
}
function preProcess2(req, res, next) {
    console.log('2個目の前処理したよ！');
}
function mainProcess(req, res) {
    res.send('メインの処理をしたよ!');
}
```

実はこのように `/hoge` に対するアクションを何個も登録することができます。
まずはこれにリクエストを送ってみましょう。

```
curl -X GET localhost:8000/hoge

## server 側
1個目の処理をしたよ!
```

```
app.get('/hoge', preProcess1, preProcess2, mainProcess);
function preProcess1(req, res, next) {
    console.log('１個目の前処理したよ！');
    next();
}
function preProcess2(req, res, next) {
    console.log('2個目の前処理したよ！');
    next();
}
function mainProcess(req, res) {
    res.send('メインの処理をしたよ!');
}
```

この場合、

```
curl -X GET localhost:8000/hoge
メインの処理をしたよ！
## server 側
1個目の処理をしたよ!
2個目の処理をしたよ！
```

と出力され、無事にHTTPレスポンスを返して処理を終えることができました。
next()関数によって、次の処理へ制御を渡すことができます。

複数の処理を登録でき、ミドルウェアを設定できる。次の処理に進むかどうかはnext()が制御する