Avant propos

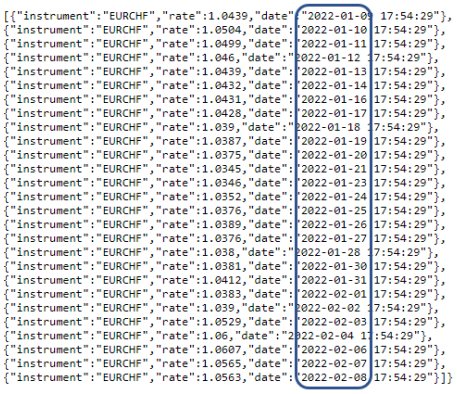
A la lecture du sujet et un peu de réflexion, j'ai identifié certains point clés sur lesquels il fallait que je me renseigne. Au travers du test j'ai aussi rencontré certains obstacles. J'ai donc rédigé quelques points qui ont été importants dans ma résolution de ces problématiques ci-dessous.

Découverte des GET de l'API:

La récupération des données était le premier point sur lequel j'ai pensé me renseigner.

Pour ça il fallait vite se renseigner sur le vocabulaire des pairs de devises, les taux de change, A la lecture de la doc de l'API, j'ai rapidement trouvé les requête Cross, Rate et RateHistory.

Je les ai tout de suite testés manuellement dans la barre de recherche pour savoir quoi chercher et dégrossir les paramètre 'to', 'from' et 'dateFormat' qui été peu documenté.



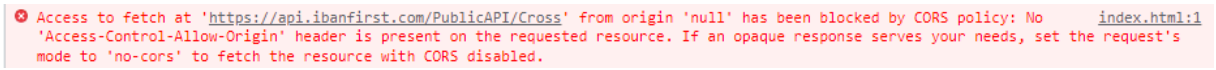
Puis, j'ai fais pas mal de screenshot des structures de données avec le genre de valeurs retournées qui pourraient m'aider pour continuer mes recherches et préparer le terrain.

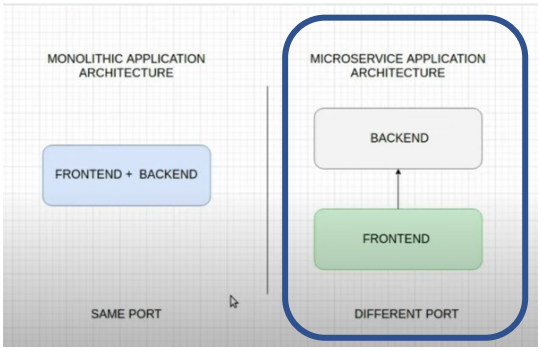
Fetching réel des données

Problèmes d'en-tête avec les CORS Policy

Puis dans un petit script javascript, j'ai voulu récupérer ces données dans des structures de données.

Travaillant sur un serveur local (port 3000), j'ai eu des problèmes liés au CORS Policy qui pour des raison de sécurité ne me bloquait le retour des données. Voici l'erreur





Cela s'explique à cause des CORS Policy qui permettent de protéger de certaines attaques pirates. En effet L'entête **`Access-Control-Allow-Origin`** renvoie une réponse indiquant si les ressources peuvent être partagées avec une origine donnée.

Pour régler cette erreur il y avait plusieurs options plus ou moins chronophages et optimales à mettre en place. Pour des raisons de temps alloué, j'ai opté pour une solution très temporaire. Cette solution consiste à installer l'extension **Moesif** de Chrome et de l'activiter pour faire des requêtes. Et bien sûr d'utliser Chrome comme navigateur.

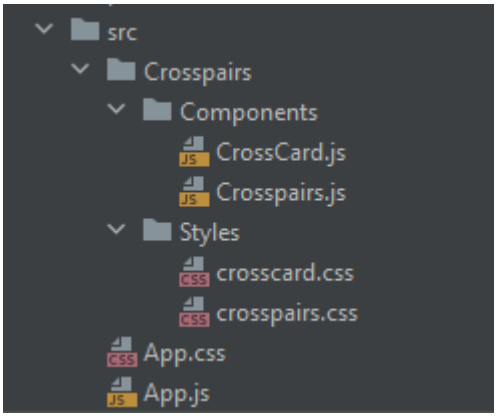
Accès refusé à l'API

Une fois l'erreur de CORS Policy outre-passé, j'ai pu tester les objets retoruné par l'API.

A force de laisser touné en boucle un SetInterval qui rafraichissait les données, l'API m'a interdit temporairement l'accès aux données.

Préparation pendant le ban de l'API

Ne pouvant plus rien fetché en attendant que le serveur me redonne l'accès, je me suis dis que j'allais préparé le balisage HTML, le style CSS qui allait accueillir ces données.



En me basant sur tout les screens shots que j'avais fait des requêtes manuelles. J'ai determiné cette structure :

`Crosspairs : Un composant se peuplant d'une liste de 'Crosscard'`

Ps : il faudrait changer le nom en 'CrossList '

`CrossCard : Un composant correspondant à une paire de devises qui permet de (current Rate, RateHistory...)`

Le but est simple il faillait que l'Application principale App.js retourne une balise <Crosspairs />, puis , à partir de la liste de paires de devises que Crosspairs doit récupèrer, il fallait itérer dynamiquement des <CrossCard /> qui détaillent les informations de la paire de devise qui lui est confiée via ses props.

Population des Crosscard par Crosspair

Itération de composant et 'populating'

Je voyais comment structurer mes composants, mais je ne savais pas comment générer itérativement à partir d'une liste pour constituer des 'Card'. Alors j'ai fais mes recherche et j'ai opter pour l'utilisation de 'map'.

Itérer des cross sans pouvoir fetcher l'array qui nous interresse ?

Il me fallait un faux semblant, alors j'ai mis en place un solution temporaire utilisant un array similaire à ce que renvoyait l'API.

```
const manualCrossListPopulation = [
    { instrument: 'EURCHF', type: "Major" },
    { instrument: 'EURGBP', type: "Major" },
    { instrument: 'EURUSD', type: "Major" },
    { instrument: 'USDCHF', type: "Major" },
    { instrument: 'USDEUR', type: "Major" },
    { instrument: 'USDGBP', type: "Major" },
];
```

UseState, UseEffect & Binding

Lors de mes différents exercices pour découvrir React, j'ai découvert les bindings utilisant les useState et les useEffect. Je me suis donc beaucoup inspiré de ce que j'avais déjà fais en l'adaptant au test. Et j'ai poursuivi mes recherches pour mieux comprendre les fameux hooks d'état. Ce sont des éléments indispensables pour rendre dynamique une application, et j'ai hâte de profiter de tout leur potentiel.

L'historique de variation des mois passés

Je me rends compte que ce n'est pas tout à fait ce qui était demandé.

Toutefois, après mes observations des retours du GET RateHistory/{instrument}, j'ai voulu concevoir une fonction relativement générique pour pouvoir calculer le rapport entre le taux de change Actuel et le taux d'il y a 1, 2, 3 ou 4 mois auparavant. Ce qui permet d'avoir un pourcentage d'évolution du taux de change depuis les derniers mois.

Screenshot de *de CrossCard.js*

```
////Getting API content
const urlRequest = rateHistoryURLRequest(month)
const response   = await fetch(urlRequest);
const data       = await response.json();

////Compute and update the overtime rate ratio
let indexWanted = data.rateHistory.length - 1;
let newestRate = data.rateHistory[indexWanted].rate; // the rate of the cross pair at 'now' date
let oldestRate = data.rateHistory[0].rate;           // the rate of the cross pair at 'from' date
let overTimeRate = (newestRate * 100 / oldestRate);  // the ratio between now rate and starting date rate
```

Comment le pourcentage d'évolution est calculé

```
const loadCrossInfos = async () => {
    ////getting rates from API
    const currentRate = await getRate().catch((err) => {return "Unknown";});
    const oneMonthHistory = await getRateHistory( month: 1).catch((err) => {return "Unknown";});
    const threeMonthHistory = await getRateHistory( month: 3).catch((err) => {return "Unknown";});

    ////updating states
    setRate(currentRate);
    setOverOneMonth(oneMonthHistory);
    setOverThreeMonth(threeMonthHistory);
}
```

Paramètre 'month' qui permet de sélectionner le nombre de mois en arrière par rapport à aujourd'hui