

Récupérez des données sur une API externe

9–11 minutes

Quand on est développeur ou développeuse front-end, on travaille quotidiennement avec des API. On s'en sert pour récupérer des données, par exemple, des données météo ou des séances de cinéma, mais aussi pour communiquer avec des fonctionnalités du navigateur, pour se géolocaliser par exemple.

Dans cette partie, nous allons voir ensemble comment **déboguer une API externe**, autrement dit quand on fait une requête à un service externe, mais aussi comment **déboguer une API interne**, c'est-à-dire une API intégrée au navigateur.

Récupérez des données météo

Notre projet Façadia comporte une page dédiée à chaque capteur. Sur cette page, nous faisons semblant de récupérer les données d'une API interne via le fichier `data/facade-detail-data.json`. En plus de cet appel, nous faisons aussi un deuxième appel API pour récupérer les données météo.

Nous faisons l'appel en direct ou nous passons par un fichier là-aussi ?

Bonne question !

En fait, nous faisons les deux :


- j'ai créé un fichier `weather-api-mocked-data.json` qui correspond aux données renvoyées par l'API. Cela m'évite de dépasser le quota et surtout de donner ma clé d'API en public.

- un fichier existe et nous permet d'appeler l'API directement. Nous verrons ensemble comment le faire fonctionner prochainement.



Donc, pour revenir à Façadia, voici la page dédiée aux capteurs :

Détails du capteur #7



Données du capteur

#	Type de données	Valeur des données
1	ID	232124
2	Marque du capteur	facadiaRT
3	Status	Actif
4	Latitude	12.23
5	Longitude	-18.23
6	Température	17
7	Degré d'humidité	30 %
8	Date de dernière visite	23/02/2021
9	ID du technicien	123456

Bulletin météo

#	Type de données	Valeur des données
1	ID	232124
2	Marque du capteur	facadiaRT
3	Status	Actif
4	Latitude	12.23
5	Longitude	-18.23
6	Température	17
7	Degré d'humidité	30 %
8	Date de dernière visite	23/02/2021
9	ID du technicien	123456

Le premier tableau contient les données du capteur, le deuxième les données du bulletin Météo

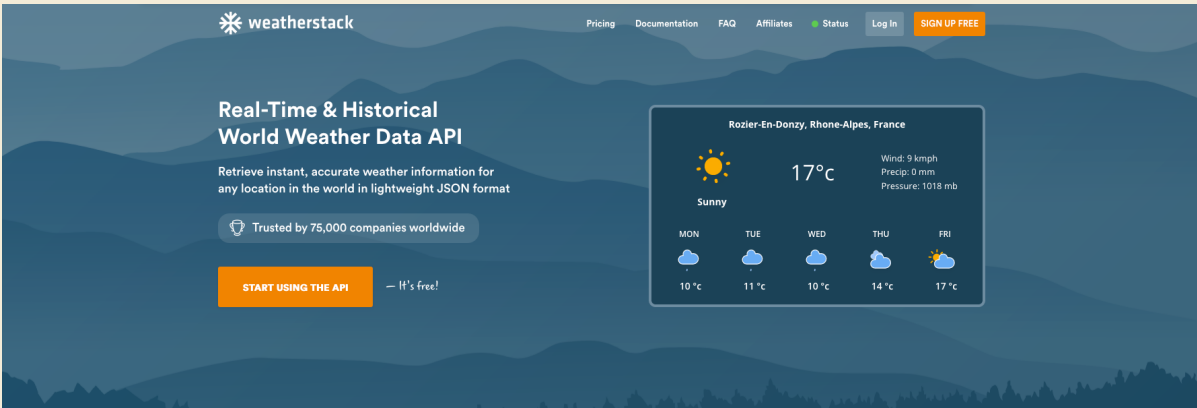
Pour récupérer les données météo, je passe par une API :

[WeatherStack](#).

Comment fonctionne cette dernière ?

Les prochains paragraphes vont être dédiés à la prise en main de l'API et à son fonctionnement.

Découvrez l'API WeatherStack



The screenshot shows the WeatherStack website interface. At the top, there's a navigation bar with links: Pricing, Documentation, FAQ, Affiliates, Status, Log In, and a SIGN UP FREE button. The main content area features the text "Real-Time & Historical World Weather Data API" and "Retrieve instant, accurate weather information for any location in the world in lightweight JSON format". Below this, it says "Trusted by 75,000 companies worldwide" and has a "START USING THE API" button with the note "It's free!". On the right, there's a weather widget for "Rozier-En-Donzy, Rhone-Alpes, France" showing a sunny icon, a temperature of 17°C, and a 5-day forecast: MON (10°C), TUE (11°C), WED (10°C), THU (14°C), and FRI (17°C).

WeatherStack est une API météo complète et gratuite. Plutôt cool, non ?

Avant de se lancer dans le code pour voir comment cette API est

intégrée, une bonne pratique de développement est de [découvrir l'API avec Postman et de parcourir la documentation](#).

Pourquoi est-ce que nous avons d'abord utilisé l'API via Postman avant de l'utiliser directement via le JavaScript ?

N'est-ce pas plus rapide de le faire directement via le JavaScript ?

C'est une bonne question.



Découvrir une API via Postman vous permet le plus souvent de bien comprendre comment elle fonctionne avant de vous lancer dans le code. Par exemple, si l'API que vous utilisez ne fonctionne pas, préférez-vous perdre quelques minutes sur Postman ou plusieurs heures sur le JavaScript ? Ça arrive beaucoup plus souvent qu'on peut le croire.



De plus, cela va souvent vous permettre de tester les requêtes et de découvrir le format de réponse. Il arrive parfois que les réponses ne correspondent pas totalement aux attentes.

Dans la vidéo ci-dessous, nous allons voir ensemble comment prendre en main l'API et notamment :

- créer un compte ;
- parcourir la documentation ;
- faire une requête via Postman et découvrir le format de réponse.

Avant de passer à la suite, je vous invite à continuer à tester l'API avec Postman. Parcourez la documentation et tentez des choses. Cela vous permettra de bien comprendre comment l'API fonctionne.



Découvrez l'intégration de l'API WeatherStack sur notre projet fil rouge

Maintenant que vous avez vu comment requêter l'API WeatherStack et le format de ses réponses, nous allons pouvoir

nous intéresser à son intégration côté JavaScript.

Comme je vous le précisais précédemment, j'ai deux manières de récupérer les données météo :

- soit via le fichier de données "mockées" que j'utilise. Il m'évite de faire trop de requêtes quand je fais mes tests.
- soit via une fonction dédiée qui va requêter l'API en fonction des données géographiques du capteur.

```
data > {} weather-api-mocked-data.json > ...
1  {
2    "request": {
3      "type": "LatLon",
4      "query": "Lat 48.88 and Lon 2.38",
5      "language": "en",
6      "unit": "m"
7    },
8    "location": {
9      "name": "Villette",
10     "country": "France",
11     "region": "Ile-de-France",
12     "lat": "48.883",
13     "lon": "2.367",
14     "timezone_id": "Europe/Paris",
15     "localtime": "2021-05-02 18:33",
16     "localtime_epoch": 1619980380,
17     "utc_offset": "2.0"
18   },
19   "current": {
20     "observation_time": "04:33 PM",
21     "temperature": 10,
22     "weather_code": 389,
23     "weather_icons": [
24       "https://assets.weatherstack.com/images/wsymbols01_png_64/wsymbol_0024_thunderstorms.png"
25     ],
26     "weather_descriptions": [
27       "Rain With Thunderstorm"
28     ],
29     "wind_speed": 15,
30     "wind_degree": 360,
31     "wind_dir": "N",
32     "pressure": 1022,
33     "precip": 0.5,
34     "humidity": 66,
35     "cloudcover": 75,
36     "feelslike": 9,
37     "uv_index": 3,
38     "visibility": 10,
39     "is_day": "yes"
40   }
41 }
```

Le fichier `data/weahter-api-mocked-data.json`

Dans la vidéo ci-dessous, vous allez découvrir où sont réalisées ces deux requêtes et mieux comprendre leur fonctionnement.



Comprenez les messages d'erreur d'une API externe

Vous savez désormais comment vous servir de l'API de WeatherCast. Il est maintenant temps de se pencher sur les messages d'erreur. En effet, pour bien déboguer une API interne, il

est important de comprendre :

- les codes HTTP renvoyés par l'API ;
- les messages renvoyés par l'API en cas d'erreurs.

Découvrez les messages d'erreur provenant d'une API

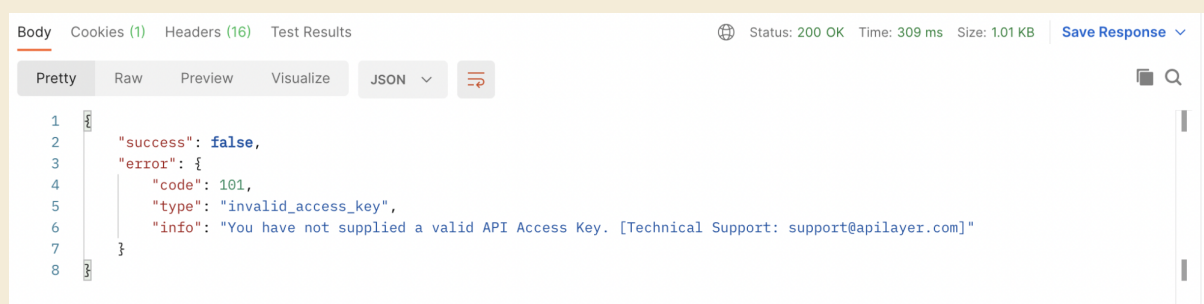
Quand on développe une API et qu'on souhaite la rendre publique, autrement dit, la mettre à disposition d'autres développeurs externes à notre structure, il y a souvent des règles à respecter :

- Il est important d'utiliser des noms au lieu des verbes pour chaque endpoint (cela correspond à l'URL appelé). Ce sont les méthodes HTTP (GET, POST, PUT, etc.) qui vont préciser l'action à réaliser.
- les status code HTTP doivent vous permettre de traiter les erreurs de façon propre. Par exemple, un status code 400 correspond à des données côté client non valides, 403 à une erreur d'authentification, etc.
- l'API doit contenir sa version dans son URL. Cela permet d'aider les développeurs utilisant cette API à réaliser des montées de versions.

J'ai cru voir un statut code de 200 tout à l'heure sur l'API alors qu'il y avait une erreur. C'est normal ?

Bonne remarque !

En effet, bien que les développeurs derrière les API essayent de vous faciliter la vie en affichant des messages d'erreur explicites. Les bonnes pratiques ne sont pas toujours respectées et il faut parfois faire avec.

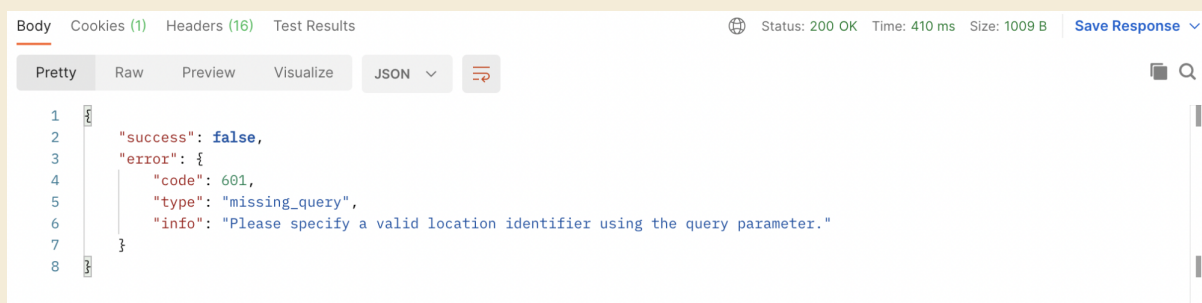


Exemple de message d'erreur de l'API WeatherStack

Dans la capture d'écran ci-dessus, j'ai volontairement modifié mon `access_key` pour faire en sorte d'avoir une erreur. Vous pouvez noter que :

- Le corps (body) du message d'erreur est assez explicite. La clé d'API est invalide.
- Le statut de la requête est 200 (alors qu'il devrait plutôt être 401). Comme je vous le disais, il n'y a pas précisément de règles.

Avant de passer à la suite, faisons une deuxième erreur dans notre requête. Ici, je vais mal orthographier le paramètre `query` en le nommant `qery` .



```
Body Cookies (1) Headers (16) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "success": false,
3   "error": {
4     "code": 601,
5     "type": "missing_query",
6     "info": "Please specify a valid location identifier using the query parameter."
7   }
8 }
```

Message d'erreur sur l'API

Ici, le message d'erreur nous dit qu'il manque le paramètre `query` et on nous demande de spécifier une position valide en utilisant le paramètre "query".

Une fois encore, le message d'erreur est plutôt compréhensible même si le code HTTP reste à 200.

Cela vous donnera une meilleure compréhension de l'API.



À vous de jouer !



Vous venez de voir que les messages d'erreur des API n'étaient pas uniformisés entre les API. Mais alors, comment pouvez-vous

les déboguer ?

Simplifiez vos problèmes en les isolant

Ce conseil ne s'applique d'ailleurs pas que pour le débogage de vos API. De manière générale, il est toujours important d'essayer de **tester de façon isolée** les différentes parties de votre code.

C'est ce que nous venons justement de faire pour notre API externe. Nous avons d'abord testé les requêtes de l'API via Postman avant de les intégrer à notre code.

Utilisez l'onglet réseau du navigateur

Au début du cours, vous avez pu découvrir tous les outils dédiés au débogage. Vous avez pu prendre en main la console, le débogueur, l'inspecteur, il ne manquait plus que la partie réseau.



Cet onglet peut vous donner des informations très intéressantes sur les appels API comme, par exemple, la requête réalisée, la réponse reçue par l'API et les en-têtes HTTP. Cet outil vous permet de rapidement **voir si une de vos requêtes échoue**.

Avant de poursuivre sur la vidéo, je vais vous raconter une petite anecdote. Au moment de réaliser ce cours, j'ai eu un bug avec l'API WeatherCast et j'ai mis un peu de temps à comprendre d'où il venait. J'ai utilisé la console et l'onglet Network pour essayer de voir d'où il venait. Au final, c'était un bug assez bête mais comme quoi, ça arrive à tout le monde et tout le temps.



Attrapez vos erreurs

Dernier point avant de passer au prochain chapitre. Pensez à **traiter vos messages d'erreur**. Quand vous travaillez avec la méthode fetch, vous travaillez avec des promesses. Les promesses ont plusieurs statuts :

- Pending (en cours),
- Resolve (résolue),
- et Reject (rejetée).

La méthode fetch traite les erreurs de la manière suivante :

```
const _retrieveWeatherForecastMockedData = () =>
fetch('/data/weather-api-mocked-data.json')
    .then(res => res.json()) // Ici, le cas de
succès
    .catch(err => console.log("Oh no", err)) //
Ici, le cas d'erreur
```

Dans le snippet de code ci-dessous, le `.catch` correspond au traitement de l'erreur dans le cadre de la méthode fetch.

Dans la vidéo ci-dessous, nous allons voir ensemble comment attraper les erreurs de l'API météo aussi bien avec les données mockées qu'avec les données de l'API.

Vous êtes maintenant paré pour déboguer des API externes !

En résumé

- Avant de vous lancer dans l'intégration d'une API côté code, essayez toujours de passer par Postman pour tester votre API.
- Les API externes respectent le plus souvent des bonnes pratiques dans leur conception : status code, versionnement, etc.

Une connaissance de ces bonnes pratiques peut vous permettre de comprendre et déboguer plus rapidement une API. Cela dit, il arrive parfois que les API ne respectent pas les bonnes pratiques.

- L'onglet Network vous permet d'analyser la requête et la réponse renvoyées par les API.

Vous savez maintenant comment déboguer une API externe. Il est temps de vous intéresser au débogage des API internes !

- <#>

