Effectuer des appels REST

Rappels sur HTTP

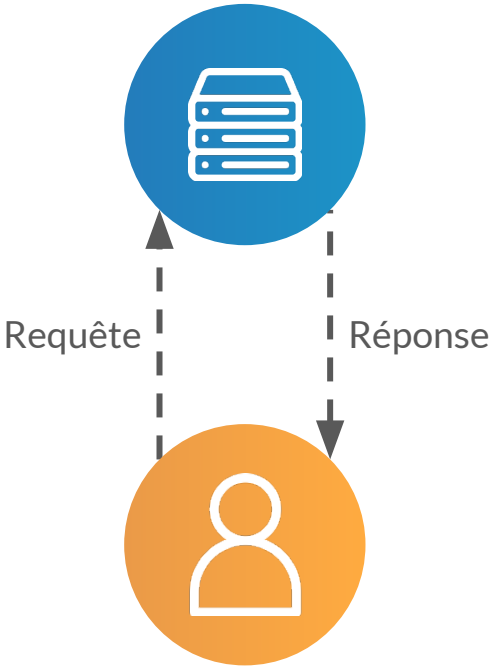
HTTP

Hyper**T**ext **T**ransfert **P**rotocol

Protocol créé pour le web

Suit le modèle de communication client-serveur

Stateless par nature



Requêtes HTTP

Possède plusieurs attributs :

- Une méthode (ou verbe)
- Des headers
- Un corps

Méthodes HTTP

GET

POST

PUT

DELETE

PATCH

HEAD

CONNECT

OPTIONS

TRACE

Réponses HTTP

Possède également plusieurs attributs :

- Un code de retour
- Des headers
- Un corps

Codes de retour

200 Ok

201 Created

202 Accepted

204 No content

400 Bad Request

401 Unauthorized

404 Not Found

418 I’m a Teapot

500 Internal Server Error

501 Not Implemented

REST

Web service

Protocol de communication web

Permet l'échange de données entre un client et un serveur

Fonctionne sur le format requête / réponse

En grande majorité en XML ou JSON

Les types les plus connus : SOAP et REST

REST

REpresentational **S**tate **T**ransfert

Architecture logicielle créée pour les web services

- architecture client-serveur
- stateless
- cacheable
- interface uniforme
- possibilité d'envoyer du code à la demande

REST et HTTP

REST est fortement lié à HTTP

La méthode HTTP va représenter le type de requête pour du CRUD :

- POST → Création de données (create)
- GET → Récupération de données (read)
- PUT → Mise à jour de données (update)
- DELETE → Suppression de données (delete)

Une même URL peut fournir différents services avec des méthodes différentes

Exemple 1/2

```
GET https://thronesapi.com/api/v2/Characters/0

Réponse : 200 OK
Body :
{
  "id": 0,
  "firstName": "Daenerys",
  "lastName": "Targaryen",
  "fullName": "Daenerys Targaryen",
  "title": "Mother of Dragons",
  "family": "House Targaryen",
  "image": "daenerys.jpg",
  "imageUrl": "https://thronesapi.com/assets/images/daenerys.jpg"
}
```

Exemple 2/2

```
POST https://thronesapi.com/api/v2/Characters
Body requête :
{
  "id": 0,
  "firstName": "testFirstName",
  "lastName": "testLastName",
  "fullName": "testFullName",
  "title": "testTitle",
  "family": "testFamily",
  "image": "testImage",
  "imageUrl": "testImageUrl"
}

Réponse : 200 OK
```

Axios

Présentation

React ne s'occupant que de la partie affichage, aucun outil n'est fourni permettant de communiquer avec un serveur REST.

Plusieurs librairies externes existent pour cela, dont **Axios** (https://axios-http.com).

Installation :

```
$ npm install axios
```

Promises

Une **Promise** en JavaScript représente un traitement asynchrone : on sait quand il démarre, mais on ne peut pas savoir quand il termine.

Il est possible cependant de réagir à la complétion d'une Promise en passant une fonction de callback à sa fonction **`then`**.

Si une erreur se produit lors du traitement de la Promise, alors c'est sa fonction **`catch`** qui sera appelée.

```
const p = Promise.resolve('test'); // Crée une Promise à partir d'une valeur

p.then((v) => {
  console.log(v); // Affichera 'test'
}).catch((error) => ...) // Sera appelé si une erreur est levée lors du traitement de la Promise
```

Exemple basique

```
import axios from 'axios';

axios.get('https://www.boredapi.com/api/activity')
  .then((res) => { // Requête réussie
    console.log('data', res.data);
  })
  .catch((e) => { // Echec de la requête
    console.log('error', e);
  });
```

Axios fournit pour chaque méthode HTTP une fonction permettant d'effectuer un appel HTTP avec cette méthode-là.

Si l'appel réussi, le **`then`** de la Promise retournée est appelé.

Sinon, le catch est appelé avec l'erreur représentant la raison de l'échec de la requête.

Les fonctions d'appel

Les fonctions axios.get/post/put/… prennent d'autres valeurs permettant de changer les paramètres de la requête :

- `url` : L'URL du service REST avec lequel communiquer.
- `data` : **Uniquement pour post, put et patch.** Données à envoyer au serveur.
- `config` : Configuration de la requête :
  - `headers` : Headers de la requête
  - `url` : URL du service REST
  - `data` : Données à envoyer

Manipuler la réponse

Axios englobe la réponse HTTP dans un objet pouvant être manipulé, possédant plusieurs propriétés :

`data` :	Données retournées par le service REST
`status` :	Statut de la réponse (200, 204, 401, …)
`statusText` :	Message du statut de la réponse (OK, ACCEPTED, UNAUTHORIZED, …)
`headers` :	Headers de la réponse
`config` :	Configuration de la requête ayant générée cette réponse
`request` :	Requête créée par Axios pour générer cette réponse

Intégration avec React

Utilisation de useEffect et useState 1/2

Il est souvent nécessaire de récupérer des données au chargement de la page.

Pour faire cela, il est recommandé d'utiliser le hook `useEffect` afin de ne faire les appels REST nécessaires qu'une seule fois au chargement de la page.

Il est également tout à fait possible d'effectuer un appel REST en réaction à un clic sur un bouton par exemple, ou à la validation d'un formulaire, dans un callback.

Utilisation de useEffect et useState 2/2

```
const DisplayActivity: React.FC = () => {
  const [activity, setActivity] = useState<string>();
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    axios.get('https://www.boredapi.com/api/activity')
      .then((res) => {
        setActivity(res.data.activity);
        setLoading(false);
      });
  }, []);

  return (
    (loading && !activity) ?
      <em>Loading ...</em> :
      <b>{activity}</b>
  );
};
```

Exercice

Exercice

Utilisez l'API https://thronesapi.com/ pour récupérer la liste des personnages de Game of Thrones plutôt que votre liste statique au chargement de la page.

Documentation :
https://thronesapi.com/swagger/index.html?urls.primaryName=Game%20of%20Thrones%20API%20v2