Javascript et le réseau

1. Les rappels
2. Réseau

Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

1. Les modèles OSI et TCP/IP

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

1. Protocole http / https

Deux types de requêtes : classique et ajax

* Classique
* Requete qui recharge la page

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

* Ajax provient du javascript :
* Requete qui ne recharge pas la page.
* Récupère des données sur un serveur
* Envoie des données sur le serveur.
* Composition d’une requete http :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Ou :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Envoyer ou recevoir de data en javascript ne dépend pas de ECMAScript mais de Wep APIs

L’Api propose :

* XmlHttpRequest (old school)
* Fetch (utilisé aujourd’hui)

1. Requete http avec fetch

Utilisation d’un site : <http://jsonplaceholder.typicode.com> un fake Online REST API

Fetch retourne une promesse.

* Utilisation :
* Utilisation de base (utilisation du GET sans option par défaut) :

Une image contenant texte, Police, logiciel, Logiciel multimédia

Description générée automatiquement

La promesse de fetch renvoie un objet Response :

Type : cors indique qu’on fait une requete sur un autre serveur.

url : de la requete

redirect : si c’est redirige

status : le status code http

headers : les headers de la reponse

body : le corps de reponse.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Le body peut etre traiter avec des promesses et des méthodes propose par l’API

L’objet Response propose des méthodes qui renvoient des Promesses aussi.

Ex avec response.json() :

On peut recevoir la reponse sans pour autant avoir le body complet.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

En utilisant les await / async plus conviviable : (Attention on n’oublie pas le try/catch si on a une erreur)

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Il existe d’autres methodes que json() si on veut sur un autre format notre donné : par exemple text() ;

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

Il existe aussi :

* formData()
* blob() qui permet de recuperer au format binaire avec un type.
* blobarrayBuffer() qui permet de lire la réponse et de la parser au format ArrayBuffer.

1. Fetch et une requete POST

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Dans cet exemple on veut envoyer un objet en post à la ressource.

* Il faut indiquer des options en deuxieme argument de fetch (un objet)
  + Method : la methode http utilisée
  + Body : le corps, attention de bien stringifier avant (autrement objet.toString() ce qui pose probleme)
  + Headers :
    - Il faut definir les headers pour indiquer ce qu’on envoit
    - Content-type : pour indiquer le type de contenu.

On lance et récupère notre promesse :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. CORS

Cross-Origin Requests

Par défaut, si on n’avait pas de solution de sécurité on pourrait faire des récupérations sur n’importe quel serveur, ce qui poserait un problème évident.

Une image contenant texte, capture d’écran, diagramme, Rectangle

Description générée automatiquement

Ce type de requete s’appelle des CORS.

* Sur les requetes simples (GET, HEAD, POST)

Notre requete va rajouter en en-tete ORIGINE : domaine.fr pour indiquer d’où provient la requete. Cet element n’est pas modifiable.

Le serveur va nous renvoyer : Access-Control-Allow-Origin : domaine.fr pour indiquer si il donne l’autorisation au domaine.

Par exemple gmail ne donne pas l’autorisation.

* Sur les autres requetes (PUT, PATCH, DELETE, etc…)
* Le navigateur envoie une requete en amont de type « preflight »

Avec ceci pour demander l’autorisation :

Une image contenant texte, Police

Description générée automatiquement

Si le serveur accepte il renvoie dans sa réponse en header :

Une image contenant texte, Police

Description générée automatiquement

Et aussi :

Access-Control-Allow-Method : GET POST …. Pour indiquer les méthodes http autorisées.

Si il accepte :

Le navigateur renvoie la bonne requete et le serveur la traite en renvoyant en reponse le Access-Contrôle-Allow-Origin : domaine.fr.

1. Options avancées des requêtes fetch

Dans la methode fetch on a vu 4 choses :

* url
* dans la partie option :
  + method
  + body
  + header

dans la partie option on peut rajouter :

* cache : ici indique sur la requete si l’on souhaite recuperer la version en cache. (default, force-cache, no-cache, no-store, only-if-cached, reload)
* credentials : indique si on envoie un cookie en cas d’authentification (include, omit, same-origin)
* integrity : ‘sha256-hashhh’, peu utilisé, pour la vérification d’intégrité. Il permer de passer un hash d’integrité.
* Keepalive : permet de finir une requete http quoi qu’il arrive (meme si on quitte l’onglet)
* mode :permet de preciser le mode core de la requete (navigate, pour utiliser les requetes simples – get – post – head - no-cors, same-origin et cors)
* redirect : si on autorise les redirection
  + fallow : suit les redirections du serveur
  + error : erreur en cas de redirection
  + manuel : ne pas suivre les redirections. Mais la propriété url de la reponse du serveur contiendra l’URL de la redirection et redirected sera à true. Le JS pourra ainsi manuellement faire la redirection si besoin.
* Signal : est une option qu’on va utiliser pour arreter une requete avec un AbortController().

1. Annuler des requetes en cours

Pour annuler une requete on utilise un Objet natif : AbortController.

On ajoute un event « abort » sur signal qui permet d’indiquer l’action à effectuer lors de l’abort.

La methode .abort() permet de lancer l’action d’« abord » des requetes rattachées.

Pour rattacher un controller abord à un fetch/requete, il suffit de rajouter l’option signal et rattacher signal de l’objet AbordController.

Lors de l’abord la promesse passe au catch et non au then.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Un ControllerAbort peut être rattache à plusieurs fetch.

1. Les objets FormData.

L’objet FormData va permettre de recuperer le contenu d’un form HTML.

Dans cet exemple ici on a preventDefault le submit d’un form et on a recuperer le contenu du form en utilisant un objet FormData(form) qu’on a initialisé.

FormData propose plusieurs méthodes et propriétés.

.append(‘clef’, ‘value’) //pour rajouter des valeurs au form (meme si il existe)

.set(‘clef’, ‘value’) //idem mais avec l’écrasement

.get(‘clef’) // pour recuperer une valeur à une clef particuliere.

.delete(‘clef’) //pour supprimer un clef spécifique

Le formData peut etre parcouru comme un objet simplement.

Chaque elements une association : clef / valeur.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Une fois terminé on envoi le formData dans le body du fetch.

On lance la requete.

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement

1. La Web API URL
2. Introduction

//utilisation assez simple avec l’objet URL

const url = new URL(‘http://….’) ;

Le contenu des propriétés dans URL :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

La partie search permettra de jouer avec les querystring. Et searchParams fournit un ensemble de méthodes permettant de pouvoir manipuler les querystrings.

L’avantage d’utiliser cet objet est de pouvoir respecter la norme quand en encode l’url entre autres.

Dans notre cas si on mettait un espace dans une value ou key d’une querystring cela ne fonctionnerait pas tel quel cependant grâce à cette objet l’url fournit serait correctement encodé en remplaçant les caractères non autorisés.

La propriété href contient l’intégralité de l’url.

1. searchParams

dans l’objet URL on a un objet searchParams qui va permettre de manipuler plus simplement les queryStrings.

Les méthodes utiles :

.append(‘key’, ‘value’)

.get(‘’)

.getAll(‘’)

.delete(‘’)

.set(‘key’, ‘value’)

1. XMLHttpRequest
2. Utilisation

C’est un objet plus ancien qu’on utilisé avant fetch.

Etapes d’utilisation de base pour un GET:

* Creation de l’instance XMLHttpRequest()
* .open(‘methode’, ‘url’) :
* Indiquer le responseType pour le type de reponse
* .send() // pour envoyer
* Ajouter les events :
  + load : se lance quand on a le retour complet
  + error : se lance quand on a une erreur
  + progress : se lance plusieurs fois pour l’avancement de la requete. (renvoie la quantite de data envoye)

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Etapes d’utilisation de base pour un POST :

Les changements par rapport au GET :

* onn doit rajouter un header content-type pour indiquer le format de ce qui est envoyé (ici du JSON)
* dans .send(JSON.stringify(obj)) ont doit renvoyer l’objet js stringifie pour l’envoyer au serveur.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

1. Etats de la requetes dans le cas ou on utilise pas progress.

Voici les états d’une request avec XMLHttpRequest :

UNSENT = 0;

OPENED = 1;

HEADERS\_RECEIVED = 2;

LOADING = 3;

DONE = 4;

On peut s’abonner à onreadystatechange pour ecouter les changements d’état :

requete.onreadystatechange = () => {

if (requete.readyState === 1) {

// Requête configurée

}

if (requete.readyState === 2) {

// Entêtes de la réponse reçues

}

if (requete.readyState === 3) {

// Chargement en cours

}

if (requete.readyState == 4) {

// Terminée

}

};

1. Utilisation avancée de XMLHttpRequest

* Timeout (qui n’est pas disponible sur le fetch) :

A partir de l’objet xhr on peut definir une propriété timeout en ms.

Quand la requete depasse cette duree la requete est arrete et un event ‘timeout’ est leve.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* Methode abord() de XMLHttpRequest()

Cette méthode directement accessible depuis l’objet permet d’arreter la requete ce qui est plus simple qu’avec fetch qui a besoin du ControllerAbord.

xhr.abord() ;

cela produit un event ‘abort’

* Traquer l’état d’upload de fichier (non disponible sous fetch)
* On recupere le formulaire qu’on wrap dans un formData
* L’event de upload ‘loadstart’ se lance quand on a un lancement d’upload
* L’event de upload ‘progress’ se lance à chaque changement de progression
  + L’objet event recu à :
    - Loaded pour la quantité chargée
    - Total pour la donnée totale.
    - Il reste facile de calculer le %.
* L’event de upload ‘loadend’ se lance à la fin de l’upload.
* Et error quand il y a une erreur.
* On envoie le formdata dans la méthode send().

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Description générée automatiquement