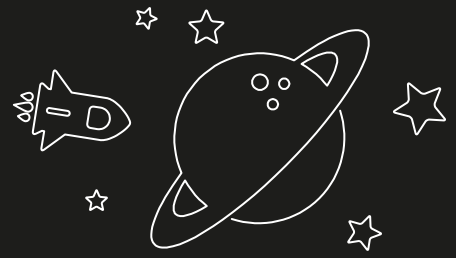

Extras
JS



XMLHttpRequest

2

Signature des méthodes

.open(method, url, async, user, pass)

-> method : “GET” / “POST”
-> url : URL de la requête
-> async : requête asynchrone/synchrone
-> user : username (HTTP BASIC)
-> pass : password (HTTP BASIC)

.send(data)

-> date : données à envoyé au serveur

.setRequestHeader(header, value)

-> header : nom du header à modifier
-> value : value du header

Utilisation du XMLHttpRequest

```
// Initialisation de l'objet
var request = new XMLHttpRequest();

// Spécification de la requête (synchrone)
request.open("GET", "test.php", false);
// Spécification de la requête (asynchrone)
request.open("GET", "test.php", true);

// Envoi GET
request.send();
// Envoi POST
request.send(data);
```

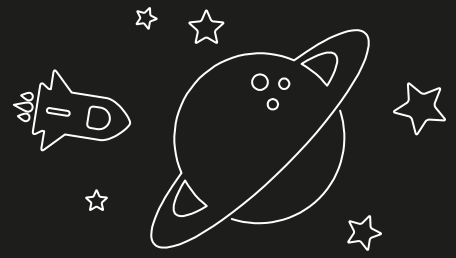
```
// Récupération du résultat (synchrone)
request.send();
request.getResponseText();
request.getStatus();

// Récupération du résultat (asynchrone)
request.onload = function(event) {
  console.log(this.status); // retourne le status code
  console.log(this.responseText); // retourne le
body
}
request.send();
```

Utilisation de FormData

- Objet permettant de gérer facilement les données POST à partir d'un formulaire

```
// Initialisation
var formElement = document
        .getElementById("formRegister");
var data = new FormData(formElement);
// Ajout de valeurs customs au formulaire
data.append("customValue", "myValue");
// Envoie de la requête
request.send(data);
```



Destructuration d’objets

6

Utilisation de la déstructuration/décomposition

// Principe de base

```
var [a, b, c , d, e = 5] = [1, 2, 3, 4];
console.log(a, b, c, d, e); // 1 2 3 4 5
```

// Filtrer les valeurs

```
var [a, b] = [1, 2, 3, 4];
console.log(a, b); // 1 2

var [a, , c] = [1, 2, 3, 4];
console.log(a, c); // 1 3
```

// Utilisation de ... (spread/rest operator)

```
var [a, b, ...c] = [1, 2, 3, 4];
console.log(a, b, c); // 1 2 [3, 4]
```

Le spread operator doit toujours être en dernier

// Changement de nom des variables

```
var {a: aa, b} = {a: 1, b: 2, 3, 4};
console.log(aa, b); // 1 2
```

// Décomposition imbriquée

```
var {a: aa, b: {b1: bb}} = {a: 1, b: {b1: 21, b2:22}};
console.log(aa, bb); // 1 21
```

Seul les valeurs les plus imbriquées sont conservées

// Equivalent JS pure => la décomposition génère de nouveaux objets

```
var right = {a: 1, b: 2, 3, 4};
var {a: aa, b} = right;
==> var aa = Object.assign({}, right.a);
var b = Object.assign({}, right.b);
```