FORMATION

JavaScript, perfectionnement

orsys.fr

Glodie Tshimini

Consultant formateur et développeur depuis 2017

Certifié 2AICONCEPT Expert Trainer

Certifié 2AICONCEPT Expert Trainer At POE

Email : contact@tshimini.fr

2AICONCEPT

Certification de formateurs

Présentations

Merci de bien vouloir vous présenter individuellement.

Pour suivre cette formation dans de bonnes conditions, les prérequis sont les suivants :

- Disposer des droits administrateurs sur vos postes
- Avoir des bases de HTML, CSS et JavaScript

Merci de bien vouloir m'indiquer quelles sont vos attentes suite à cette formation.

Horaires et pauses

Les horaires

- 9h - 12h30
- 13h30 – 17h30

Les pauses

- Matin : 15 minutes
- Déjeuner : 1 heure
- Après-midi : 15 minutes

Évaluation formateur et émargement

Chaque matin et chaque après-midi, vous devrez signer les feuilles d'émargement. Les "X" et initiales ne sont pas autorisés, merci de bien vouloir émarger avec une signature.

Le dernier jour de la formation, avant 15h00, vous aurez à saisir une évaluation formateur. Ce document est obligatoire, il doit être rempli consciencieusement et il sera remis à votre entreprise.

Déroulement de la formation

- 40 % théorie 60% pratique
- Daily meeting
- Exercices immédiats d’application du cours
- Code refactoring et code review
- Travaux pratiques en groupe
- Les énoncés des exercices, TP et corrections seront accessibles depuis un dépôt public GitHub

GitHub de la formation

https://github.com/glo10/jpg-28112023

Un formateur à votre écoute

Google Forms

https://docs.google.com/forms/d/e/1FAIpQLSfkchq-Ti_-U2tOVUectnHQZpYHMSIaoJWbc6Dr4fnv3SBFLg/viewform

Plan du cours

Plan

1. POO et DOM
2. Modules
3. Débogage
4. Asynchrone
5. Stockage local
6. Expressions régulières (REGEX)
7. JQuery
8. Refactoring & Code Review
9. NODE JS
10. TypeScript
11. Optimisations
12. Design Pattern

Chapitre 1 : POO et DOM

Rappels programmation orientée objet

- La POO est une méthode de programmation informatique née dans les années 1960.
- Le programme est une collaboration entre des objets.
- Un objet est une entité qui possède des caractéristiques et des comportements.
  - Par exemple, une voiture est un objet qui a les caractéristiques : nombre de porte, couleur et comportements accélérer, freiner.

- Les caractéristiques ou propriétés constituent un ensemble d'attributs définissant l'apparence de l'objet.

- Les comportements ou méthodes constituent un ensemble d'actions réalisables par l'objet.

En JavaScript, tout est objet

- DOM : objets spécifiques à la manipulation du document HTML.

- ***BOM*** : objets spécifiques à la manipulation du navigateur.

- La notation pointée (avec le point) permet d'accéder aux objets, propriétés, méthodes de même hiérarchie ou inférieure.
  - Par exemple ***window.location.href*** pour récupérer l'URL de la page courante.

Source image medium.com/@fknussel

Focus nouveautés ES6 sur les classes et objets

- Import : import d'un module avec le mot-clé ***import*** au lieu de *require()*.
- Export : export d'un module avec le mot-clé ***export*** au lieu de *modules.export=a*.
- Class : structure similaire aux autres langages objets.
- Objet littéral : création des objets avec des accolades *{}*
- Notation fléchée des fonctions =>

```
hello() => { console.log('hello world') }
```

- Déstructuration des tableaux et objets : créer des variables à partir des propriétés d'un objet ou des éléments d'un tableau

Exemples nouveautés ES6

Class

```
// BEFORE ES6
var Car = function (color) {
  this.color = color
}
Car.prototype.accelerate = function() {
  // Do something here
}
Car.prototype.brake = function() {
  // Do something here
}
modules.export = Car
/*----------------------------------*/
// WITH ES6
class Car {
  constructor(color) {
    this.color = color
  }
  accelerate () {
  // Do something here
  }
  brake() {
  // Do something here
  }
}
export default Car
```

Déstructuration

```
const cars = ['twingo', 'clio', 'megane', 'zoe']
const [twingo, clio, ...others] = cars

console.log(twingo) // twingo

console.log(clio) // clio

console.log(others) // ['megane', 'zoe']

const zoe = {
  color: 'black',
  engine: 'electric',
  odomter: 45000
}

const {color} = zoe
console.log('zoe color', color) // black
```

Objet littéral

```
const Car = {
  color: undefined,
  accelerate: () => {
    // Do something here
  },
  brake: () => {
    // Do something here
  },
}
```

DOM : Document Object Model

- Le DOM est une API permettant de représenter et manipuler les éléments constituant une page Web.

- A l'aide des méthodes offertes par l'objet document de JavaScript, une page web peut être modifiée.

Source de l'image centralsupelec

Accéder aux éléments du DOM

- Plusieurs méthodes permettent d'accéder à un ou plusieurs éléments du DOM à partir de leur ID, nom de la balise, nom de la classe etc.

- Nous retiendrons dans le cadre de ce cours uniquement les méthodes ***querySelector*** et ***querySelectorAll*** qui permettent de tout sélectionner à partir des sélecteurs CSS.

  - ***document.querySelector(selector)*** : retourne **le premier élément** du document html qui correspond au sélecteur *selector*

  - ***document.querySelectorAll(selector)*** : retourne **tous les éléments** du document HTML qui correspond au sélecteur *selector* sous forme de **tableau**.

Les autres méthodes et attributs du DOM

- *el.createElement(tag)* : crée l'élément à partir du tag donné.
- *el. insertAdjacentHTML(position, el)* : insère un nouveau nœud HTML dans l'élément *el* par rapport à la position spécifiée.
- Position prend les valeurs suivantes :
  - *Beforebegin* : avant l'élément lui-même ;
  - *Afterbegin* : juste à l'intérieur de l'élément, avant son premier enfant ;
  - *Beforeend* : juste à l'intérieur de l'élément, après son dernier enfant ;
  - *Afterend* : après l'élément lui-même.

- *el.parentElement* : renvoie le parent du nœud (textuel ou html) ou null.
- *el.replaceWith(nodeEl): remplace l'élément courant el par le nœud nodeEl.*
- *el.firstElementChild()* : renvoie le premier nœud enfant de type *element* ou *null*.

Source MDN

Les autres méthodes et attributs du DOM

- *el.attributes.class.value* : valeur de l'attribut class.
- *el.classList.add('alert-danger')* : ajoute la valeur *alert-danger* dans la liste des valeurs de l'attribut class de l'élément.
- *el.classList.remove('alert-danger)* : supprime la class *alert-danger* dans la liste des valeurs de l'attribut class de l'élément.
- *el.textContent* : contenu textuel d'un nœud et de ses descendants.
- *el.outerHTML* : contenu textuel d'un élément et de ses descendants incluant la balise HTML de l'élément.
- *el.remove()* : retire (supprime) l'élément du DOM.
- *elements.length* : nombre des éléments.

Source MDN

Composants Web : CustomElement

- Élément HTML **personnalisé** et **réutilisable**.
- *CustomElement* fonctionne comme les autres éléments natifs du langage HTML.
- Encapsule son propre code JavaScript pour définir son comportement.
- Pour créer un *CustomElement*, il faut créer une classe qui étend la classe ***HTMLElement*** ou ses sous-classes *HTMLInputElement*, *HTMLDivElement* etc.
- Implémenter les méthodes
  - ***.construct()*** : appeler la méthode parent à l'aide de la méthode ***super()***
  - ***.connectedCallback()*** : exécutée au moment où l'élément sera **ajouté** au **DOM**.
  - ***.disconnectedCallback()*** : exécutée au moment où l'élément sera **supprimé** du **DOM**.
- Enregistrer l'élément personnalisé sur la page :
  - Méthode statique *customElements.define('nom-avec-au-moins-un-tiret', Class)*

Implémentation CustomElement

```
class MyCustomElement extends HTMLElement {
  constructor() {
    super() // Création de l'élément
  }

  connectedCallback() {
    // Appelée au moment de l'ajout de l'élément dans le DOM
  }

  disconnectedCallback() {
    // Appelée au moment de la suppression de l'élément du DOM
  }

  /**
  * Il existe d'autres méthodes de la classe mère que vous pouvez implémenter
  * Vous pouvez également ajouter vos propres méthodes pour définir le comportement de l'élément
  */
}

customElements.define("my-tag", MyCustomElement);

/**
 * Implémentation dans le HTML
 * <body>
 * <my-tag></my-tag>
 * </body>
 *
 */
```

Les événements

- Rendre les pages web plus interactives.
  1. **En un**, **sélectionnez** un élément HTML.
  2. **En deux**, ajoutez un écouteur **événement** de type : souris (***click***, *dbclick*, *mouseover*, *mouseout* etc.), formulaire (***focus***, ***blur***, ***change***, ***submit***), clavier (*keydown*, ***keyup***, ***keypress*** etc).
  3. **En trois**, ajoutez une **fonction callback** qui sera exécutée au moment où l'événement aura lieu.

```
const btn = document.querySelector('button')
btn.addEventListener('click', (e) => {
  console.log('Hi, i am the callback function')
  console.log('e for event, object that contains the details of an event ', e)
})
```

Propagation des événements

- Le **troisième argument de la méthode *addEventListener*** (*event*, *callback*, ***boolean***) détermine le flux d'événement.
  (***true*** => ***capture***, ***false*** => ***bubble***).

- Phase de **bouillonnement** (**Bubble**) : propagation de l'événement en remontant la hiérarchie du DOM.
  Comportement par **défaut**.

- Phase de **capture** : propagation de l'événement en **descendant** la hiérarchie du *DOM*.

Propagation des événements

- ***e.preventDefault()*** : empêche le comportement par défaut d'un élément de s'exécuter.

- ***e.stopPropagation()*** : stoppe la propagation de l'événement

Source de l'image laptrinhx

Objet event

- À chaque définition d'un écouteur d'événement à l'aide de la méthode *.addEventListener()*, la fonction callback peut prendre un argument en paramètre qu'on nomme communément *e*.
  Cet argument est un objet qui contient des propriétés et méthodes correspondant à l'événement qui a été déclenché.

- Les événements de type *click*, *mouse* ou *keyup* n'auront pas les mêmes attributs.

- Nous utiliserons dans la plupart du temps l'attribut ***e.target.value*** pour obtenir la valeur de l'élément HTML sur lequel un événement a été greffé.

Objet event

- Vous pouvez utiliser la fonction *console.dir(e)* pour afficher dans la console toutes les informations disponibles d'un événement e.

```
▼ click { target: h1 , buttons: 0, clientX: 34, clientY: 51, layerX: 34, layerY: 51 }
    altKey: false
    bubbles: true
    button: 0
    buttons: 0
    cancelBubble: false
    cancelable: true
    clientX: 34
    clientY: 51
    composed: true
    ctrlKey: false
    currentTarget: null
    defaultPrevented: false
    detail: 1
    eventPhase: 0
  ▶ explicitOriginalTarget: <h1>
    isTrusted: true
    layerX: 34
    layerY: 51
    metaKey: false
    movementX: 0
    movementY: 0
    mozInputSource: 1
    mozPressure: 0
    offsetX: 0
    offsetY: 0
  ▶ originalTarget: <h1>
    pageX: 34
    pageY: 51
    rangeOffset: 0
    rangeParent: null
    relatedTarget: null
    returnValue: true
    screenX: 34
    screenY: 193
    shiftKey: false
  ▶ srcElement: <h1>
  ▶ target: <h1>
    timeStamp: 4792
    type: "click"
  ▶ view: Window http://127.0.0.1:5500/demo/index.html
    which: 1
    x: 34
    y: 51
```

CustomEvent

- Événement personnalisé que l’on peut créer soi-même.
- Cet événement personnalisé fonctionne quasiment de la même manière que les événements natifs, sauf que son exécution est synchrone.

- La classe ***CustomEvent*** prend en paramètre le nom de l'événement et une fonction callback.

- La **fonction callback** est optionnelle. C’est un **objet** ayant les clés :
  - ***Detail*** : informations (données) sur l’événement ;
  - ***Cancelable*** : annulation de l’événement (possible d’utiliser .preventDefault()) ;
  - ***Bubbles*** : propagation.
- La fonction ***dispatchEvent*** appliquée sur un élément *el*, ***el.dispatchEvent()*** permet de déclencher l’événement.

Implémentation CustomEvent (source image MDN)

```
const form = document.querySelector("form");
const textarea = document.querySelector("textarea");

// Create a new event, allow bubbling, and provide any data you want to pass to the
"detail" property
const eventAwesome = new CustomEvent("awesome", {
  bubbles: true,
  detail: { text: () => textarea.value },
});

// The form element listens for the custom "awesome" event and then consoles the output of
the passed text() method
form.addEventListener("awesome", (e) => console.log(e.detail.text()));

// As the user types, the textarea inside the form dispatches/triggers the event to fire,
and uses itself as the starting point
textarea.addEventListener("input", (e) => e.target.dispatchEvent(eventAwesome));
```

Offline

- Perte de connexion au réseau internet.

- L'événement ***offline*** permet de réagir et effectuer des traitements spécifique lorsque l'utilisateur n'est plus connecté à internet.

- Gestion :
  - Stockez les données en local à l'aide du ***localStorage*** ou cache pour les restituer sans avoir besoin d'internet.

Implémentation offline (source de l'image MDN)

```
//version addEventListener
window.addEventListener("offline", (event) => {
  console.log("La Connexion au réseau est perdu.");
});

// onoffline version
window.onoffline = (event) => {
  console.log("La Connexion au réseau a été perdue.");
};
```

Ce qu’il faut retenir

- Le DOM (Document Object Model) permet de manipuler du HTML et CSS à travers JavaScript (ajout d’éléments, d’événements, modifications des propriétés CSS etc.).
- La programmation orientée objet (POO) permet de solutionner un problème informatique à l’aide des objets ayant des propriétés (caractéristiques) et des méthodes (comportements) qui collaborent.
- POO facilite le partage, la maintenance et l’évolution du code.
- La programmation événementielle repose sur la manipulation des éléments du DOM.

Conseils du formateur

- Utilisez les ***CustomElement*** dès que vous avez des éléments interactifs pouvant disparaitre lors d'une modification du *DOM*.
  Attention, ce n'est pas supporté par tous les navigateurs cf. can i use.

- Utilisez les normes *ECMASCRIPT 6 (ES6 – ES2015)* et +

Chapitre 2 : Modules

Qu’est ce qu’un module ? (source image tech mozfr)

- Un module est un fichier JavaScript contenant du code à partager ou réutiliser dans d’autres scripts JS.

- Plusieurs systèmes d’utilisation existent en JavaScript : ***CommonJS*** et ***ESM*** sont les deux standards les plus utilisés

Différences entre CommonJS et ESM

Les *Bundlers*, que nous verrons plus tard, permettent de résoudre les problèmes d'incompatibilité entre modules.

	CommonJS	ESM
Standard		ES6
Import	*require('path/file.js')*	*import { add } from 'path/file.js'*
Export	*modules.export = add*	*export add*
Extension	*.cjs ou .js*	*.mjs ou .js*
Mode de chargement	synchrone	asynchrone et synchrone
Chargement en HTML côté navigateur	*<script type="script" src=" …"></script>* et utiliser un bundler pour convertir le code	*<script type="module" src=" …"></script>*

Différences entre CommonJS et ESM

(source image lenguajejs)

CIS CommonJS

```
const elem = require('module');

module.exports = {
};
```

node supported | sync | bare imports | browser compatible | from url/cdn

dynamic require | html supported | transpile needed | tree shaking | deno supported

ESM ES Modules

```
import { elem } from './module.js';

export const elem = {
};
```

node supported | async | bare imports | browser compatible | from url/cdn

dynamic imports | html supported | transpile no needed | tree shaking | deno supported

LenguajeJS.com

Le gestionnaire de packages NPM

- ***Node Package Manager*** : gestionnaire de paquets de Node.

- Permet de **télécharger** et de **déployer** les **modules** développés par la communauté des développeurs JavaScript.

- Initialisation d'un nouveau projet : *npm init*
  - Répondre aux questions posées depuis le terminal.

  - A la fin du processus, les fichiers ***package.json*** et ***package-lock.json*** contenant les informations requises et les dépendances sont générés.

- Installation de toutes les dépendances d'un projet contenu dans le fichier *package.json* : *npm install*

Source de l'image npmjs

Le gestionnaire de packages NPM

- Installation d'un module
  - De manière globale : *npm install –g node*
  - De manière locale : *npm install --save typescript*
  - Une dépendance de développement en locale : *npm install –D axios*
- Désinstallation d'un module : *npm uninstall node*
- Mise à jour d'un module : *npm update [--save|--save-dev|-g] typescript*
- Déploiement d'un module sur npmjs.com

Avantages et inconvénients

Avantages

- Logique divisée en plusieurs fichiers
- Moins de responsabilité (responsabilité unique)
- Traitement propre à un besoin
- Plus facile à partager et réutiliser
- Plus facile à tester
- Code plus court et plus lisible

Inconvénient

- Non compatibilité avec tous les navigateurs (principalement les anciennes versions)
- Nécessite un compilateur pour transformer du code JavaScript avec les nouveaux standards en code JavaScript compatibles avec tous les navigateurs
- Débogage un peu plus difficile à partir du code compilé

Le gestionnaire de packages YARN

- Autre gestionnaire de packages de Node.
- Développé par Facebook.
- Il est plus rapide que *NPM*.
- Initialisation d'un projet : *yarn init*
  - Répondre aux questions posées dans le terminal
  - A la fin du processus, les fichiers ***package.json*** et ***yarn.lock*** contenant les informations requises et les dépendances sont générés.
- Installation d'un module
  - De manière globale : *yarn global add jquery*
  - De manière locale en production : *yarn add sqlite3*
  - De manière locale en développement : *yarn add nodemon --dev*
- Désinstallation d'un module : *yarn remove jquery*
- Mise à jour d'un module : *yarn upgrade node*
- Déploiement : yarn publish

Source de l'image meta

Modules bundlers

- Compiler le code JavaScript vers une autre version du langage.
  Par exemple, transformer du code écrit en *ES6* vers du *ES5*, pour qu'ils soient compatibles avec la majorité des navigateurs (+ leurs versions) qui ne supportent pas toutes les nouveautés du langage.

- ***Webpack*** est le plus connu et le plus abouti, cependant, il demande une configuration qui peut être plus ou moins longue et pénible.

- ***Esbuild*** et ***Parcel*** ont l'avantage de pouvoir s'utiliser **sans configuration**.
  Vous pouvez ainsi vous concentrez uniquement sur votre code. Vous devez juste ajouter votre script avec les options souhaitées dans le fichier ***package.json*** pour compiler votre code vers la ou les cibles souhaitées.
- **Vite** plus complet et monte en puissance

Stats

	Stars	Issues	Version	Updated	Created	Size
esbuild	33 929	344	0.15.14	5 days ago	5 years ago	
parcel	41 614	777	2.8.0	11 days ago	10 years ago	
poi	–	–	12.10.3	2 years ago	8 years ago	
webpack	62 088	343	5.75.0	10 days ago	11 years ago	minzipped size 739.1 KB

Source de l'image npmtrends

ESbuild

- Installation

```
npm i -D esbuild
```

- Zéro configuration
- Le plus rapide au niveau des performances du *build et cache* par rapport à ses concurrents
- Support le *TypeScript*
- Script de *build*
- Plugins
- Lien vers documentation complète

Parcel

- Installation
  ```
  npm i -D parcel
  ```
- Zéro configuration
- Optimisations des images
- Minification des fichiers JS, CSS et HTML
- Support le TypeScript
- Script de build
- Plugins
- Lien vers documentation complète

Quelques autres modules utiles

- ***ESLINT*** : analyse statique du code JS avec un fichier de configuration.
- ***Standardjs*** : vérification syntaxe JS sans fichier de configuration (alternatif à *ESLINT*).
- ***Icones.js.org*** : hub de plusieurs types d'icônes, possible de télécharger plusieurs formats.
- ***Vitest*** ou ***Jest*** : pour les tests unitaires.
- ***Cypress*** : pour les tests end to end.
- ***TypeScript*** : superset (surcouche) de JavaScript qui apporte des fonctionnalités supplémentaires tels que le typage, les classes abstraites, etc.
- ***Concurrently*** : lancer plusieurs commandes simultanément.

Installations

Guide d’installation depuis GitHub

Suivez les instructions du fichier d’installation de notre environnement de travail depuis le lien GitHub ou depuis le dépôt local du cours *README.md*.

Les dépendances à télécharger se trouvent depuis le lien GitHub ou depuis le dépôt local du cours *packages.md*.

Ce qu’il faut retenir

- Les modules facilitent le partage et la réutilisation d’une partie de son code.
- ***NPM*** et ***YARN*** sont des gestionnaires de paquets de JavaScript permettant d’ajouter des modules externes dans son projet.

Conseils du formateur

- Codez de façon modulaire.
- Utilisez les modules de la communauté pour répondre à certains besoins.
- Créez vos propres modules pour les partager au sein de la communauté ou en intra.

Chapitre 3 : Débogage

Console

- Parmi les outils de développement (*devTools*), la console permet d'exécuter du code JavaScript en direct et de voir les messages d'information, de warning ou d'erreur sur le CSS et le JavaScript.

- Principalement utilisé pour déboguer le DOM et analyser les erreurs JS.

- Tous les outils de la *devTools* sont accessibles à partir de :

  - Windows/Linux avec la touche F12 ou CTRL + Shift + C ;

  - MacOs avec CTRL + CMD + C .

Débogueur

- Outil de la *devTools* permettant de suivre l'exécution de son code ligne par ligne.
- Avec le suivi pas à pas, vous pouvez voir en temps réel les valeurs de vos variables.
- Vous pouvez également arrêter votre script à des endroits précis.

Déboguer depuis l'éditeur du code

- Extension *VSCode debugger*.
- Configuration des navigateurs dans le fichier ***.vscode/launch.json***
- Depuis votre code, vous pouvez ajouter le mot-clé ***debugger*** qui permet de stopper l'exécution du code à l'endroit à un endroit précis.

launch.json M

.vscode > launch.json > Launch Targets > {} Chrome Debug

```
1  {
2    // Use IntelliSense to learn about possible attributes.
3    // Hover to view descriptions of existing attributes.
4    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5    "version": "0.2.0",
6    "configurations": [
7      {
8        "type": "firefox",
9        "request": "launch",
10       "reAttach": true,
11       "name": "Firefox Debug",
12       "url": "http://localhost:5500/3-exercices/0-front/templates/html/",
13       "webRoot": "${workspaceFolder}"
14     },
15     {
16       "type": "chrome",
17       "request": "launch",
18       "name": "Chrome Debug",
19       "url": "http://localhost:5500/3-exercices/0-front/templates/html/",
20       "webRoot": "${workspaceFolder}"
21     }
22   ]
23 }
```

index.js M

3-exercices > 0-front > templates > js > exercice3 > index.js > ...

```
8        const input = document.querySelector('[
9        if (input) {
10         geo.disabledField(document.querySelector('[name=city]'))
11         geo.displayCountries(countries)
12         input.addEventListener('change', async (e) => {
13           const cities = await geo.getCities(e.target.value)
14           debugger
15           geo.displayCities(cities)
16         })
17       }
18     }
19   }
20
```

Réseau

- Le réseau est l'outil de la *devTools* consacré à l'analyse des requêtes ***HTTP*** effectuée par une page web, pour charger les différentes ressources telles que les fichiers CSS, JS, images etc.

- Grâce à cet onglet, vous pouvez également voir le statut (code HTTP de réponse) de retour, le type de fichier, la taille et la durée de chaque élément ainsi que la durée totale.

Stockage

- Onglet de la *devTools* permettant d'explorer les types de stockage supportés par le navigateur, ainsi que les données enregistrées pour chaque type de stockage.

- Nous aurons l'occasion d'explorer 3 solutions de stockage local dans la suite du cours.

Méthodes de débogage

Source de l'image codetravelled

Source de l'image medium.com/@gonzaloohk

Exercices

Enoncé de l’exercice

- 0-exercices/1-ex.md

Ce qu’il faut retenir

- La console sert à explorer des données d’information, de sensibilisation ou d’erreur liées à vos fichiers multimédia, CSS, JavaScript.
- Le réseau permet de vérifier que tout se charge correctement.
- Le déboggeur permet de suivre l’exécution de son code pas à pas.
- Le stockage permet de voir tous les types de stockage supportés par le navigateur et les données stockées.
- L’inspecteur permet d’explorer son code HTML.

Conseils du formateur

- Utilisez les extensions disponibles sur les éditeurs pour un débogage plus efficace et plus rapide.

Chapitre 4 : Asynchrone

Définition

- JavaScript est **synchrone**, c'est-à-dire que l'exécution de la ligne suivante du script s'effectue uniquement après la fin de l'exécution de la ligne précédente. On dit que le script est **bloquant**.

- L'**asynchrone** détermine un ensemble de technologies permettant d'effectuer des opérations **non-bloquantes**, c'est-à-dire que les autres instructions (tâches) sont exécutées malgré le fait que la tâche en cours ne soit pas terminée.

- Pour désigner l'asynchrone en JavaScript, on emploie communément le terme ***AJAX*** (***Asynchronous JavaScript And XML***), apparu en **2005**,qui regroupe l'ensemble des solutions offertes par le langage JavaScript.

Synchrone VS Asynchrone

Source de l'image JavaScript.plainenglish.io

Source de l'image medium.com/@vivianyim

Intérêts de l’asynchrone

- Recharger partiellement des pages web
- Exécuter des tâches lourdes sans ralentir l’exécution du reste du Script
- Communiquer avec un serveur externe sans interruption ou blocage de la pile d’exécution principale
- Meilleure expérience utilisateur (Navigation plus flexible)
- Temps de chargement du site Web plus court

Initialement les données étaient envoyées/reçues sous le format XML. Aujourd’hui le format JSON a pris le dessus grâce à ses caractéristiques que nous verrons juste après.

Format textuel XML

- **eXtensible Markup Language**. Comme le HTML, c'est un langage de balisages. Les balises **XML** ne sont pas prédéfinies, l'auteur du fichier doit définir ses propres balises en respectant les normes du langage.

- L'extension du fichier est ***.xml***.

- ***XML*** est principalement utilisé pour stocker des données structurées ou les échanger entre des applications.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <users>
    <item>
      <name>tshimini</name>
      <age>31</age>
      <email>contact@tshimini.fr</email>
      <adresses>
        <city>Paris</city>
        <country>France</country>
      </adresses>
      <adresses>
        <city>Nice</city>
        <country>France</country>
      </adresses>
    </item>
    <item>
      <name>john</name>
      <age>19</age>
      <email>john@doe.com</email>
      <adresses>
        <city>london</city>
        <country>UK</country>
      </adresses>
      <adresses>
        <city>Nice</city>
        <country>France</country>
      </adresses>
    </item>
  </users>
</root>
```

Format textuel JSON

- **JSON** pour ***JavaScript Object Notation***.
- Stocker les informations sous forme de couple de ***clé : valeur*** ; les valeurs pouvant elles-mêmes être des clés contenant un autre sous-ensemble de clés et valeurs.
- L'extension du fichier est ***.json***.

- ***Plus léger, plus lisible pour l'homme, plus rapide à traiter et proche de la notation objet de JS.***
- C'est le **format de prédilection** des échanges des données entre les applications.

```json
{
 "users": [
  {
    "name": "tshimini",
    "age" : 31,
    "email": "contact@tshimini.fr",
    "adresses": [
      {
        "city": "Paris",
        "country": "France"
      },
      {
        "city": "Nice",
        "country": "France"
      }
    ]
  },
  {
    "name": "john",
    "age" : 19,
    "email": "john@doe.com",
    "adresses": [
      {
        "city": "london",
        "country": "UK"
      },
      {
        "city": "Nice",
        "country": "France"
      }
    ]
  }
 ]
}
```

Promise

- **Objet permettant d'effectuer une requête asynchrone depuis ES6.**

- La réponse (une promesse) est disponible ou non dans le futur.

- L'objet ***Promesse*** prend en paramètre deux fonctions ***resolve()*** et ***reject().***
  - *Resolve()* exécutée en cas de succès (réponse positive de la promesse).
  - *Reject()* exécutée en cas d'échec.
- En cas de succès, la réponse peut être récupérée à l'aide de la méthode ***.then()*** de l'objet *Promesse*.
- En cas d'échec, l'erreur peut être capturée par la méthode ***.catch()*** de l'objet *Promesse*.

Implémentation Promise

Implémentation en JS

```
const nb = 11
const myPromise = new Promise((resolve, reject) => {
  if (nb % 2 === 0) {
    resolve('Nombre pair')
  } else {
    reject('Nombre impair')
  }
})

myPromise
.then((pair) => {
  console.log(pair) // Nombre pair
})
.catch(
  error => console.log(error) // Nombre impair
)
```

Les états et enchainements de la promesse

Source de l'image scoutapm

Promise all et Promise allSetteled

- Renvoie une promesse lorsque l'ensemble des promesses données en arguments sous forme de tableau ont été toutes résolues avec succès ou échec.
- Si toutes les promesses sont résolues avec succès, la promesse retournée est un succès, par contre, si une seule échoue, la promesse finale échoue entièrement

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, "foo");
});

Promise.all([p1, p2, p3]).then((values) => {
  console.log(values); // [3, 1337, "foo"]
});
```

- *Promise.allSettled* est similaire à Promise.all, sauf qu'elle renvoie un objet contenant
  - En cas de succès, une propriété ***status*** dont la valeur est « ***fulfilled*** » et une propriété ***value***
  - En cas d'échec, ***status*** vaut « ***rejected*** » et on a une propriété ***reason*** à la place de *value*

Fetch

- **Méthode de prédilection permettant d'effectuer des traitements asynchrones vers un serveur local ou distant.**

- La méthode ***fetch* renvoie une promesse**. En cas de succès, la méthode ***.then()*** de l'API ***fetch*** permet de manipuler le résultat. En cas d'échec, un traitement spécifique peut être effectué dans la méthode *.catch()* .

- La méthode ***fetch*** prend :
  - En **premier** paramètre, un **URL** (**obligatoire**) ;
  - En **second** paramètre, un objet contenant des **options**.
    Dans les options, on peut indiquer la **méthode HTTP**, les informations d'**en-têtes (headers)**, le **cache** etc.

Implémentation de l'API fetch

```
fetch(
  'https://jsonplaceholder.typicode.com/photos',
  {
    method: 'GET',
    headers: new Headers({'Content-Type': 'text/json'}),
    mode: 'cors',
    cache: 'default'
})
  .then((res) => res.json())
  .then((photos) => {
    console.log('photos : ', photos)
  })
  .catch((err) => console.log('error : ', err ))
```

Async / Await

Les mots-clés *async* et *await* fonctionnent de pair. Ces mots-clés représentent une fonctionnalité permettant d'attendre la résolution d'une promesse. L'exécution de la suite du script est bloquée en attendant d'obtenir le résultat.

Sans async/await

```
console.log('Début du script')

const getPhotos =  () => {
  fetch('https://jsonplaceholder.typicode.com/photos')
    .then((res) => res.json())
    .then((photos) => console.log('photos', photos))
    .catch((err) => console.log(err))
}

getPhotos()
console.log('Je serai exécuté en dernier, après le résultat de la fonction getPhotos ')
```

Inspecteur | Console | Débogueur | Réseau | Éditeur de style

Filtrer | Erreurs | Avertisse

```
Début du script
Je serai exécuté en dernier, après le résultat de la fonction getPhotos
photos ▸ Array(5000) [ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, … ]
```

Avec *async/await*

```
(async () => {
  console.log('Début du script')
  const getPhotos =  async () => {
    await fetch('https://jsonplaceholder.typicode.com/photos')
    .then((res) => res.json())
    .then((photos) => console.log('photos', photos))
    .catch((err) => console.log(err))
  }
  await getPhotos()
  console.log('Je serai exécuté en dernier, après le résultat de la fonction getPhotos')
})()
```

Inspecteur | Console | Débogueur | Réseau | Éditeur de style

Filtrer | Erreurs | Avertisse

```
Début du script
photos ▸ Array(5000) [ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, … ]
Je serai exécuté en dernier, après le résultat de la fonction getPhotos
```

XMLHttpRequest

- Objet développé initialement par **Microsoft** pour effectuer des traitements asynchrones. De nos jours, cet objet est **largement moins utilisé**, mais est toujours supporté par les navigateurs.

- Les méthodes :

  - **.abort()** : annule la requête en cours ;

  - ***.getResponseHeaders()*** : headers (en-têtes) ;

  - ***.open(method, url, boolean)*** : démarre la requête avec une méthode *HTTP*, l'*URL* du serveur et le mode synchrone (false) ou asynchrone (true) ;

  - ***.send(data)*** : envoie des données au serveur.

Les propriétés de l'objet XMLHttpRequest

- Propriété .readyState
  - ***0*** : non initialisé, la méthode *.open()* n' a pas encore été appelée.
  - ***1*** : initialisé, la méthode *.send()* n'a pas encore été appelée.
  - ***2*** : *headers* (en-têtes de réponse) disponibles.
  - ***3*** : *body* (données reçues) partiellement disponible.
  - ***4*** : *body* entièrement disponible.

- Propriété .status (Code HTTP du retour du serveur sur la requête effectuée)
  - 20x: OK
  - 40x : erreur côté client (navigateur)
  - 50x : erreur côté serveur

  avec x, un chiffre.

- Propriété .responseText : obtenir les données renvoyées par le serveur au format texte brut.
- Propriété ***.responseXML*** : obtenir les données renvoyées par le serveur au format XML.

Implémentation en JavaScript

```javascript
const getUsers = () => {
  const myPromise = new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest()
    xhr.onreadystatechange = function () {
      if (xhr.readyState === 4 && (xhr.status === 200 || xhr.status === 0)) {
        const res = xhr.responseText
        resolve(res)
      } else if (xhr.status === 404) {
        reject('404 not found')
      }
    };
    xhr.open('GET', 'https://jsonplaceholder.ir/users')
    xhr.send()
  })
  myPromise
  .then((res) => console.log("result", res))
  .catch((e) => console.log("error message", e))
}

getUsers()
```

Exercices

Enoncés des exercices

- 0-exercices/2-ex.md
- 0-exercices/3-ex.md

Ce qu’il faut retenir

- L’asynchrone permet d’effectuer des traitements lourds ou externes sans bloquer le cours d’exécution principal de notre script
- L’asynchrone permet de mettre à jour une partie ou l’intégralité du code HTML sans recharger la page.
- L’asynchrone en VanillaJS (JavaScript natif) s’effectue à travers les objets Promise, XMLHttpRequest, mots-clés async/await, la méthode fetch et les fonctions callback des événements du DOM.

Conseils du formateur

- Pour les traitements asynchrones, utilisez de préférence *fetch* et les *Promises*.
- *XMLHttpRequest* est déprécié depuis ***ES6***, mais il continue d'être supporté par tous les navigateurs.

Chapitre 5 : Stockage local

LocalStorage

- Solution de stockage offerte par les navigateurs. On parle plus communément de Web Storage.

- Le ***localStorage*** permet de stocker des informations sous-forme de clé-valeur avec une grande capacité de stockage allant jusqu'à **5 Mo – 10Mo** selon les navigateurs.

- La particularité du ***localStorage*** par rapport aux autres moyens de stockage du côté client (navigateur) est de pouvoir stocker les informations « sans limite dans le temps ».

```
// Enregistrer des données dans le localStorage
localStorage.setItem('firstname', 'glodie')
localStorage.setItem('age', 31)
localStorage.setItem('email', 'contact@tshimini.fr')

// Récupérer les données depuis le localStorage
localStorage.getItem('firstname') // glodie
localStorage.getItem('age') // 31

// Supprimer les données depuis le localStorage
localStorage.removeItem('age')

// Supprimer toutes les données du localStorage
localStorage.clear()
```

SessionStorage

- Similaire au *localStorage* avec une capacité de stockage moins importante.

- Les informations stockées ne sont disponibles que durant la session en cours.
- ***Un onglet = une session***.
- Les informations ne sont pas partagées entre les différentes sessions de l'utilisateur.

- Avec le *localStorage*, les informations sont partagées avec les différents onglets appartenant au même domaine.

```
// Enregistrer des données dans le sessionStorage
sessionStorage.setItem('firstname', 'glodie')
sessionStorage.setItem('age', 31)
sessionStorage.setItem('email', 'contact@tshimini.fr')

// Récupérer les données depuis le sessionStorage
sessionStorage.getItem('firstname') // glodie
sessionStorage.getItem('age') // 31

// Supprimer les données depuis le sessionStorage
sessionStorage.removeItem('age')

// Supprimer toutes les données du sessionStorage
sessionStorage.clear() // ou fermez l'onglet du site
```

Cookies

- Petits fichiers stockés sur le disque dur d'un visiteur d'une taille inférieure à 4 Ko.

- Des informations supplémentaires sont ajoutées pour permettre au navigateur de sécuriser le cookie.

- **Seul le nom (clé) et la valeur du cookie sont obligatoires**.

- Autres informations du cookie :
  - ***Expires*** : date d'expiration du cookie.
  - ***Secure*** : transmission avec le HTTPS
  - ***SameSite*** : prévient des attaques d'usurpation d'identité sur le web (CRSF)

tz : "Europe%2FParis"
Date de création : "Sun, 13 Nov 2022 13:39:01 GMT"
Dernier accès : "Sun, 13 Nov 2022 21:16:16 GMT"
Domain : ".github.com"
Expiration / Durée maximum : "Session"
HostOnly : false
HttpOnly : false
Path : "/"
SameSite : "Lax"
Secure : true
Taille : 16

Créer, lire et supprimer un cookie

La valeur du cookie ne peut être modifiée en dehors du domaine où il a été créée.

```
// CRÉATION
document.cookie = `course=dom; expires=Thu, 01 DEC 2022 00:00:01 GMT; sameSite=Lax; secure;`

// LECTURE
const getCookie =  (name) => {
  return document.cookie.split('; ').find((row) => row.startsWith(`${name}=`))?.split('=')[1];
}
console.log(getCookie('course')) // dom

// MODIFICATION
document.cookie = 'course=storage; sameSite=Lax; secure;'
console.log(getCookie('course')) //storage

// SUPPRESSION
document.cookie = 'course=;' // or document.cookie('course=dom;expires=Thu, 07 09 2022 00:00:01 GMT;')
console.log(getCookie('course')) // unedfined
```

Objet Map

- Objet contenant des paires de clé-valeur.
- Respecte l'ordre d'insertion.
- Les clés et les valeurs peuvent être des primitives ou objet.

```
const map1 = new Map();

map1.set('a', 1);
map1.set('b', 2);
map1.set('c', 3);

console.log(map1.get('a'));
// expected output: 1

map1.set('a', 97);

console.log(map1.get('a'));
// expected output: 97

console.log(map1.size);
// expected output: 3

map1.delete('b');

console.log(map1.size);
// expected output: 2
```

Source MDN

Sérialisation et désérialisation

- On parle de sérialisation lorsqu'il s'agit de transformer un objet en string (chaîne de caractères) pour le stocker, par exemple dans le *localStorage*.
  La méthode ***JSON.stringify()*** transforme l'objet en chaîne de caractères sérialisés.

- A l'inverse, la désérialisation consiste à reconstruire l'objet d'origine à partir d'une chaîne de caractères sérialisés.
  La méthode ***JSON.parse()*** reforme l'objet à partir de la chaîne de caractères sérialisés.

```
// Stockage d'un objet sérialisé

const person = {
  lastname: 'tshimini',
  firstname: 'glodie',
  age: 31,
  city: 'paris'
}
localStorage.setItem('glodie', JSON.stringify(person)) // transforme l'objet en JSON

/*
Récupérer un texte JSON depuis le localstorage
pour reconstruire l'objet d'origine (désérialisation)
*/
const glodie = JSON.parse(localStorage.getItem('glodie'))
```

DevTools et stockage local

Les données stockées en local sont accessibles depuis l'**onglet storage ou stockage** du navigateur.

Chrome

Firefox

Exercices

Enoncé de l’exercice

- 0-exercices/4-ex.md

Ce qu’il faut retenir

- **localStorage** permet de stocker la plus grande quantité d’informations sur le navigateur de l’utilisateur.
- **sessionStorage** est similaire. Cependant, la capacité de stockage est moins importante et les informations sont supprimées à la fin de la session (fermeture onglet).
- **Cookies** permet de stocker une information à la fois sous forme de clé-valeur avec une capacité très limitée de l’ordre de quelques Ko.

Conseils du formateur

- Choisissez la solution de stockage en fonction du volume d'informations à stocker
- Ne stockez jamais les informations sensibles (mot de passe, coordonnées bancaires etc.) en local

Chapitre 6 : REGEX

Apprendre les expressions régulières (REGEX) en autonomie

Synthèse REGEX

Les caractères

\w : alphanumérique et _

\W : qui n'est pas alphanumérique et _

\d ou [0-9] : chiffres

\D : n'est pas un chiffre

\s : un espace

\S : qui n'est pas un espace

. : tout sauf retour chariot

[^a-c] : tout sauf a, b et c

[def] : uniquement d, e ou f

x|z : x ou z

Les ancres

$: se termine

^ : commence

Quantificateurs

***** : 0, 1 ou n fois

? : 0 ou 1 (optionnel)

+ : 1 et n fois

{1,3} : de 1 à 3

{2,} : au moins 2 fois

Synthèse REGEX

Caractères spéciaux (CS)

{ } [ ] / \ + * . $^ | ?

\ : échapper CS

Les groupes

() : groupe, sous-motif

(?=) : regard avant positif

(?!) **:** regard avant négatif

Les drapeaux

g : toutes les correspondances

m : multiligne

i : insensible à la casse (peu importe minuscule ou majuscule)

Méthodes REGEX avec JavaScript

- ***.test()*** *: renvoie une valeur booléenne (boolean) après évaluation du motif avec le texte fournit.*

```
/*
 * Commence par au moins une lettre
 * Sous groupe exclue, n'est pas suive par _
 * espace ou - optionnel (0 ou 1)
 * Lettres ou chiffres ou espace au moins une fois et se termine par ces caractères
 */
const result = /^[a-z]+(?!_)(\s|-)?[a-z0-9|\s]+$/i.test('hello world')
console.log(result) // true
```

- ***.match() : renvoie les occurrences qui matchent avec le motif ou sous-motif.***

```
console.log('#pk 17de 145fe 1254 fed12 126'.match(/1[0-2][0-9]/g)) // ["125", "126"]

console.log('135 145 182eg 521j'.match(/(1[0-3][0-9])+(\w)$/)) // null

console.log('118acf'.match(/^(1[0-2][0-9])(\w+)$/)) // ["118acf", "118", "acf"]
```

Chapitre 7 : JQuery

JQuery

- Développé en 2006 par *John Resig* (Mozilla).

- Bibliothèque JavaScript « **write less, do more** » :
  - Code **moins verbeux (moins de code)** pour effectuer des opérations sur le *DOM*.
  - **JQuery** règle le problème de compatibilité de JavaScript avec les navigateurs, ce qui justifie son succès et sa popularité auprès des développeurs JS.

- Très populaire durant la décennie précédente, aujourd'hui surpassé par les multiples frameworks ou librairies tels que *React*, *Vue JS*, *Angular*, *Astro*, *Anime*, *Three.js*, *Velocity* etc. et l'évolution de JavaScript lui-même.

- Dernière version **3.6.1** de ***2021***

Téléchargement

- Via le site officiel https://jquery.com/download/ en sauvegardant la libraire en local dans un fichier JavaScript.

- Via les Content Delivery Network (CDN) de Google ou CloudFare

  - Pour les deux cas précédents, ajoutez l'attribut src = lien en local ou distant vers la libraire JQuery entre les balises script

```
<script src="jquery.js"></script>
```

- Via *NPM* ou *YARN*

```
$ npm i jquery
```

```
$ yarn add jquery
```

  - Avec *NPM* ou *YARN*, pour utiliser JQuery, importez le module depuis *node_modules/*

```
import * as $ from 'jquery'
```

Fonction $()

La variable globale JQuery et son alias $ sont disponibles de manière globale ou via un import à partir du fichier source de la librairie.

```
JQuery(function(){
  // script here !
  // Déclenche le script avec JQuery une fois que le document HTML a été chargé.
  // Évite les conflits dus à l'utilisation $ dans des nombreuses librairies JavaScript.
})
```

Sélecteurs JQuery

```
$("#id") // par l'attribut ID de l'élément HTML

$(".class") // par l'attribut CLASS

$("p")  // par le nom de la BALISE (TAG)

$("h1, header, h2") // Combinaison de plusieurs éléments

$("a[href=/contact.html") // par l'attribut HREF
```

Tous les sélecteurs CSS peuvent être utilisés en argument de la fonction ***$()*** pour sélectionner des éléments du **DOM.**

La méthode css

- Affecte une valeur à la propriété CSS pour tous les éléments sélectionnés à l'aide de la fonction *$()*.

```
$("h1").css("color:red")
$("h2").css({"color:blue", "font-size": "20px"})
```

- Retourne la valeur de la propriété CSS du premier élément.

```
$("h3").css("display")
```

- *.hasClass(name)* : retourne un booléen après avoir identifié au moins un élément de la liste des éléments possédant la classe *name*.
- *.addClass(name)* : ajoute la classe *name* à tous les éléments.
- *.removeClass()* : supprime la classe *name* à tous les éléments.
- *.toogleClass()* : ajoute/supprime la classe *name* à tous les éléments.

Le DOM

- *.html()* : récupère le contenu HTML du premier élément de la liste.
- *.html(code)* : remplace le contenu des éléments de la liste par le ***code*** HTML.

- *.value* : récupère la valeur du premier élément.
- *.value(val)* : affecte la valeur ***val*** à tous les éléments.

- *.text()* : récupère les nœuds textuels de tous les éléments descendants.
- *.text(content)* : affecte le contenu textuel ***content*** à tous les éléments.

- *.append(code)* : ajoute le ***code*** HTML à la fin de chaque élément de la liste.
- *.prepend(code)* : ajoute le ***code*** HTML au début de chaque élément de la liste.

- *.attr(name, value)* : affecte la valeur ***value*** à l'attribut HTML ***name*** de tous les éléments.

Inspiré du mémento JQuery 4ème édition
Éric Sarrion

Les événements

- *.on(event, callback)* : identique à *document.addEventListener()*.
- .off() : supprime les gestionnaires d'événements de tous les éléments.
- *.trigger(event) :* déclenche l'événement *event* sur tous les éléments.
- *.bind(event, callback) : identique à on(event, callback).*
- *.unbind(event) : supprime les gestionnaires d'événement event de tous les éléments.*
- *.unbind() : supprime tous les gestionnaires d'événements.*

Les effets et animations

- *.show(duration?)* : apparition progressive de l'élément durant la durée duration.
- *.hide(duration?)* : disparation progressive de l'élément durant la durée duration.
- *.toggle(duration?)* : apparition/disparition alternée selon la valeur de la propriété CSS display.
- *.fadeIn(duration?)* : similaire à show mais avec l'évolution de la propriété CSS opacity de 0 à 1. Effet de fondu.
- *.fadeOut(duration?)* : opacité de 1 à 0.
- *.fadeToggle(duration?)* : alterne selon la valeur de la propriété CSS opacity.

PS : ***duration?*** **s**ignifie que le paramètre est **optionnel**. Par défaut il vaut 0ms, ce qui provoque un effet instantané.

Les effets et animations

- *.animate(cssNewProperties, options)* :

  - cssNeWProperties un objet contenant les nouvelles propriétés CSS à remplacer progressivement par les anciennes valeurs en tenant compte des options.

  - options (objet) ayant les propriétés :
    - Durée
    - Elasticité de l'animation
    - etc.

```
$('div').animate({height: '500px', opacity: '0.7'}, "slow")
```

L'AJAX

- *$.ajax()* : globale pour toute requête asynchrone
- *$.get()* : spécifique pour une requête via la méthode HTTP GET
- *$.post()* : spécifique pour une requête via HTTP POST
- Voir la documentation globale sur la méthode ajax()
- Toutes les méthodes précédentes, retourne une promesse.
  - La méthode *.done(callback) permet d'*obtenir le résultat en cas de succès.
  - La méthode *.fail(callback) permet de capturer une erreur survenue au cours du* traitement de la requête.
  - La méthode *always(callback)* s'exécute dans les cas

```
var menuId = $( "ul.nav" ).first().attr( "id" );
var request = $.ajax({
  url: "script.php",
  method: "POST",
  data: { id : menuId },
  dataType: "html"
});

request.done(function( msg ) {
  $( "#log" ).html( msg );
});

request.fail(function( jqXHR, textStatus ) {
  alert( "Request failed: " + textStatus );
});
```

JQuery User Interface

- *Extension de la librairie JQuery dédiée aux :*
- Interactions
  - Drag and drop
  - Redimensionnement des éléments de l'IU
  - Trie des listes
  - Etc.
- Widgets : éléments d'interface prêt à l'emploi
  - Accordéon
  - DatePicker
  - Tooltip
  - Etc.
- Animations
- Thèmes :
  - Plusieurs thèmes à disposition pour les éléments constituant cette bibliothèque
- Démos
- Téléchargement de la librairie

Exercices

Enoncés des exercices

- 0-exercices/5-ex.md
- 0-exercices/6-ex.md

Ce qu’il faut retenir

- ***JQuery*** est une libraire de JavaScript compatible avec tous les navigateurs et très populaire lors de la décennie précédente, aujourd'hui on peut s’en passer.
- Les expressions régulières ou ***REGEX*** permettent de définir des motifs pour contrôler, remplacer des données saisies par l’utilisateur.

Conseils du formateur

- Utilisez ***VanillaJS*** (JavaScript natif) à la place de *JQuery* pour les nouveaux projets.
- Youmightnotneedjquery est une bonne alternative pour transformer du code JQuery en Vanilla JS
- Manipulez les *REGEX* avec prudence, attention à ne pas être trop restrictif.
- Utilisez les modules existants dans la communauté pour contrôler les entrées des données.

Chapitre 8 : Refactoring & Code Review

Refactoring

- Pratique qui consiste à **optimiser son code** soit tout au long du développement soit à la fin du développement.

- Le but est d'avoir un code de **meilleure qualité**, avec des noms de variables pertinents, **optimisation** de ses **algorithmes**, **suppression du code non-utile**, réduction de la complexité, etc.

- Très efficace lorsqu'il est combiné avec le ***Test Driven Development*** (***TDD***), c'est-à-dire, écrire ses tests automatisés avant d 'écrire son code.

Source de l'image infoQ

Code Review

- Un principe de la gestion de projet agile qui consiste à faire ***une revue de code par l'équipe***.
- En général, lorsqu'il y a une politique de versioning du code, l'auteur du code demande une intégration de son travail dans la version du code en production en assignant cette tâche à un collègue de son choix et en assignant également le responsable (Tech Lead) pour la mise en production.

Source de l'image commadot

Comment réaliser un Code Review

Source de l'image evoketechnologies

Ce qu’il faut retenir

- ***Refactoring*** est très efficace lorsqu’il est utilisé avec un développement qui s’appuie sur les tests.
- Le ***Code Review*** permet aux autres membres d’équipe de valider votre code et d’être au courant des évolutions apportées. Cela augmente la qualité du code, réduit le temps de débogage, permet le partage des connaissances.

Conseils du formateur

- Combinez le ***Refactoring***, le ***Code Review*** ainsi que le ***pair programming*** avec l'implémentation des **tests automatisés** (unitaires, d'intégrations, fonctionnelles etc.).

- Explorez le ***clean Code*** en JavaScript.

Exercices

Enoncé de l’exercice

- 0-exercices/7-ex.md

Chapitre 9 : Introduction à NODE JS

Écosystème

Dépendances

- Node est un environnement (technologie) crée en **2009** par **Ryan Dahl**.
- Composé de :

  - Moteur **V8** (C++) de chrome qui **interprète** et **exécute** le code **JavaScript**
  - ***NPM*** (Node Package Module)
  - *OpenSSL*
  - *Zlib* (compression/décompression fichiers)
  - Modules permettant de gérer des protocoles *TCP*, *UDP*, ***HTTP***, *QUIC*

Liste exhaustive de toutes les dépendances

Caractéristiques

- Programmation avec le langage JavaScript
- Multi-plateforme (Linux, Windows, Mac OS).
- Facile à prendre en main.
- Robuste : gestion d'un grand nombre de connexions.
- Rapide grâce à la gestion asynchrone des requêtes.
- Modulaire : développement des paquets par la communauté très actif.
- Faire du développement de plus bas niveau orienté serveur.
- Attention, certains modules côté front (navigateur) ne sont pas disponibles ou existent sous d'autres noms côté serveur.

Principales utilisations

- Serveur Web
- API REST
- Streaming
- Chat
- SPA (Single Page Application)
- IoT (internet des objets)

Installation

- Lien d'installation de Node.js selon son système (OS)
- Version ***LTS*** : Long Time Support, version stable pour la production.
  Actuellement 20.10.0 LTS.
- Version ***Current*** : version en cours de développement avec les dernières nouveautés.
  Actuellement 21.2.0 Current.

Event Driven Model

- *NODE.JS BINDINGS* : librairies de communication entre les systèmes.
- *LIBUV* : gestion des entrées/sorties de manière asynchrone.
  - *Event Queue* : tâches à effectuer.
  - *Event Loop* : système de gestion des tâches.
  - *Worker Threads* : pile d'exécution des tâches (traitements).

Source de l'image medium. om@witkesam

Demo Event Loop

Simulateur loupe

Notre premier script

1. Créez un fichier index.js
2. Ecrivez dans ce fichier

```
console.log(process.versions)
```

3. Exécutez ce fichier depuis un terminal avec la commande *node index.js*

```
console.log(process.versions)
```

MMENTS

TERMINAL

```
Glodie@Glodie MINGW64 /e/formations
$ node index.js
{
  node: '19.0.1',
  v8: '10.7.193.13-node.16',
  uv: '1.43.0',
  zlib: '1.2.11',
  brotli: '1.0.9',
  ares: '1.18.1',
  modules: '111',
  nghttp2: '1.47.0',
  napi: '8',
  llhttp: '8.1.0',
  openssl: '3.0.7+quic',
  cldr: '41.0',
  icu: '71.1',
  tz: '2022b',
  unicode: '14.0',
  ngtcp2: '0.8.1',
  nghttp3: '0.7.0'
}
```

Objet process

- **Gestion** et **contrôle** du **processus** d'exécution du programme en cours.
- Objet ***process*** disponible de manière **globale** dans l'environnement Node.
- Gestion des **interactions** avec **3 propriétés** :
  - ***stdin*** : entrées des données depuis le **flux standard des entrées** (input) ;
  - ***stdout*** : sortie des données depuis le **flux standard des sorties** (output) ;
  - ***stderr*** : sortie des erreurs depuis le **flux standard des erreurs.**

Objet process

- ***process.env***
  - Contient toutes les **variables d'environnement** de votre système où est installé Node.

- ***process.argv***
  - Arguments passés en ligne commande pour lancer le processus courant.

```
index.js demo\node\index.js
console.log(process.argv)

MMENTS
TERMINAL

Glodie@Glodie MINGW64 /e/formations/coderbase/orsys/js-perf-28112022/demo/node
$ node index.js argument1 argument2
[
  'C:\\Program Files\\nodejs\\node.exe',
  'E:\\formations\\coderbase\\orsys\\js-perf-28112022\\demo\\node\\index.js',
  'argument1',
  'argument2'
]
```

Interaction depuis le terminal : package readline

- ***Objet Readline :*** interaction avec un utilisateur à partir du terminal.

  - ***.createInterface()*** crée une interface en ligne de commande en prenant en paramètre les propriétés ***stdin*** et ***stdout*** de l'objet ***process.***
  - ***.close()*** met fin au dialogue.
  - ***.question()*** permet à l'utilisateur de saisir des données depuis le flux standard d'entrée et qui peuvent être récupérées dans une variable.

```
import { createInterface }  from "node:readline/promises"
const app = createInterface({input: process.stdin,output: process.stdout})
app.question('Quel est votre langage de programmation préféré ? ')
.then(res => {
  console.log(`langage préféré : ${res}`)
  app.close()
})
```

```
MINAL   PROBLEMS   OUTPUT   COMMENTS
TERMINAL

Glodie@Glodie MINGW64 /e/formations/coderbase/orsys/js-perf-28112022 (feature/back/typesc
$ node demo/node/index.mjs
Quel est votre langage de programmation préféré ? JavaScript
langage préféré : JavaScript
```

Exercices

Enoncé de l’exercice

- 0-exercices/8-ex.md

Gestion des événements

Classe Events : package events

- Responsable de la gestion des événements de tout type, entrée, sortie, serveurs, données, fin de processus, etc.
- Un bon nombre des classes de Node dérive de cette classe.
- Lorsqu'un événement a lieu, un écouteur d'événement exécute du code défini depuis sa fonction de callback (retour asynchrone).

Méthodes

- *obj.addListerner(eventName, listernerCallback)* : ajout d'un gestionnaire d'événement à l'objet *obj*.
  Paramètres :
  - *eventName* : nom de l'événement
  - *listernerCallback* : fonction de rappel exécuté lorsque l'événement se produit.
- *obj.on(eventName, listenerCallback)* : alias de la fonction *.addListener()*.
- *obj.emit(eventName)* : émission de l'événement *eventName* sur l'objet *obj*.

index.mjs demo\node\index.mjs\app

```
import { EventEmitter } from 'events'
const app = new EventEmitter()
app.on('app_start', () => {console.log('app started on http://localhost:3000')})
// Something else here
app.emit('app_start')
```

MMENTS

TERMINAL

```
Glodie@Glodie MINGW64 /e/formations/coderbase/orsys/js-perf-28112022/demo/node (feature/ba
$ node index.mjs
app started on http://localhost:3000
```

Exercices

Enoncé de l’exercice

- 0-exercices/9-ex.md

Serveur Web

Serveur Web : package http

- Gestion des connexions via le protocole HTTP
- Méthodes
  - ***createServer([requestListener])*** :

    Crée un serveur et retourne un objet ***http.server*** qui hérite de la classe ***events.EventEmitter***

  - ***.listen(port, [callback])*** :

    Définit le **port d'écoute** sur lequel un client doit se connecter pour communiquer avec le serveur.
    La callback est facultative.

index.mjs demo\node\index.mjs\webServer\createServer() callba

```
import { createServer } from 'http'
console.info('App on http://localhost:8080')
const webServer = createServer(() => {
  console.log('Welcome!')
  process.exit(1)
})
webServer.listen(8080)
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-2
App on http://localhost:8080
Welcome!

[Done] exited with code=1 in 3.217 seconds
```

Objet Request

- Objet de la classe ***http.incomingMessage*** qui dérive de la classe ***stream.readable***.

- Contient les informations transmises par un client ***HTTP*** au serveur ***HTTP***, on peut y trouver les propriétés suivantes :

  - ***request.url*** : *URL* de la requête sans le nom de domaine et le port ;

  - ***request.method*** : méthode HTTP associée à la requête ;

  - ***request.headers*** : en-têtes de la requête.

Objet Response

- Objet de la classe ***http.ServerResponse*** qui dérive de la classe ***stream.writable***.

- Possède les méthodes suivantes :

  - ***response.write(chunk, [encoding], [callback])*** : **écrit** la donnée ***chunk*** sur le **flux d'écriture** du corps du message ;

  - ***response.end([chunk], [encoding], [callback])*** : **ferme** le flux d'écriture ;

  - ***response.writeHead(statusCode, [headers])*** : indique le statut HTTP de retour et ajoute les informations d'en-têtes.

Implémentation d'un serveur Web

```
import { createServer } from 'http'
console.info('App on http://localhost:3000')
createServer((req, res) => {
  const headers = { 'Content-Type': 'application/html; charset=utf-8' }
  if (/\/$/.test(req.url)) {
    res.writeHead(200, headers)
    res.write('<h1>Welcome</h1>')
  } else {
    res.writeHead(404, headers)
    res.write('not found')
  }
  res.end()
}).listen(3000)
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\demo\node\index.mjs"
App on http://localhost:3000
```

GET http://localhost:3000

http://localhost:3000

GET http://localhost:3000

Params Authorization Headers (1

Query Params

KEY

x

Key

Body Cookies Headers (5) Test Result

Pretty Raw Preview Visualiz

Welcome

Exercices

Enoncé de l’exercice

- 0-exercices/10-ex.md

Gestion des fichiers

Gestion des fichiers : package fs

- Lire/écrire :
  - ***fs.readFile('file.md', {encoding: 'utf-8'}, (err, data) => ({...}) ;***
  - ***fs.writeFile('file.md', 'my content', (err) => ({...}) .***
- Créer/supprimer fichiers et dossiers :
  - *fs.unlink(path, callback), fs.mkdir(path, callback) et fs.rmdir(path, callback) .*
- Informations sur les fichiers et dossiers, ouvrir/fermer, lister, etc.
- La gestion des fichiers peut se faire de manière :
  - Asynchrone ;
  - Synchrone ;

    Les méthodes ont le suffixe Sync ex: ***readFileSync() .***
  - Avec des promesses depuis les packages avec un suffixe */promises*

    Exemple ***import { readFile } from 'node:readFlile/promises'***

Implémentation écriture et lecture d'un fichier

```
37  import {readFile, writeFile} from 'node:fs'
38
39  writeFile('hello.md', 'hello world', (err) => {
40    if(!err) {
41      readFile('hello.md', {encoding: 'utf-8'}, (er, data) => {
42        if(!er) console.log('content : ', data)
43      })
44    }
45  })
46
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\demo\nc
content :  hello world

[Done] exited with code=0 in 0.203 seconds
```

Les streams : package fs

- Utile pour la gestion des fichiers très volumineux.
- L'écriture ou la lecture s'effectue par flux de données (bout d'information découpé de plus petite taille).
- La méthode ***createReadStream()*** permet de gérer un flux de lecture.
- La méthode ***createWriteStream()*** permet de gérer un flux d'écriture.
- La méthode ***pipe()*** appliqué à un ***stream*** permet de synchroniser 2 flux.
  Par exemple un flux de lecture avec un flux d'écriture pour copier un fichier d'un *disque A SSD* vers *un disque B SATA* (ils n'ont pas la même vitesse de transfert des données).

Implémentation streams lecture et écriture

```
import { createWriteStream } from 'node:fs'
import { request } from 'node:https'
const hostname = 'raw.githubusercontent.com'
const path = '/dr5hn/countries-states-cities-database/master/countries%2Bstates%2Bcities.json'
const countries = createWriteStream('country-states.json')
request({hostname, path, method: 'GET'}, (res) => {
  res.pipe(countries)
  res.on('end', () => { console.log('fin de lecture de la page web')})
  countries.on('close', (err) => {
    if(err) console.log('erreur', err)
    console.log('fin de la copie !')
  })
}).end()
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\demo\node\index.mjs"
fin de lecture de la page web
fin de la copie !

[Done] exited with code=0 in 34.148 seconds
```

Package url

- Gestion des différents composants d'un URL.

- Propriétés
  - *href* : URL
  - *protocol*
  - *hostname*
  - *hash* : l'ancre (après #)
  - *pathname* : partie après le port

Source de l'image usbforwindows

- Méthodes
  - *.parse(url, [bool])* : retourne un objet contenant les différents composants d'un URL. Le paramètre *bool* est vrai le *queryString* (partie après ? Dans l'URL) sera également parsé.
  - *resolve(from, to)* : construit un URL relative ou absolue en combinant les informations du *from* et *to*.

Exercices

Enoncés des exercices

- 0-exercices/11-ex.md
- 0-exercices/12-ex.md

Base de données SQLite 3

Installation et création d'une base de données

- Installation

```
npm i sqlite3
```

- Mode de lecture
  - *OPEN_READONLY*
  - *OPEN_READWRITE*
  - *OPEN_CREATE*

```
import sqlite3 from 'sqlite3'
const db = new sqlite3.Database('./database.sqlite', sqlite3.OPEN_READWRITE, (err) => {
  if (err) {
    console.error(err.message)
  }
  console.log('Connected to the database !')
})
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\3-exercices\1-back\demo.mjs"
SQLITE_CANTOPEN: unable to open database file
Connected to the database !

[Done] exited with code=0 in 0.264 seconds
```

- Création de la base de données sur :
  - Mémoire (***:memory:***) ;
  - Disque avec un chemin vers le fichier de la base de données comme illustré sur l'image ci-dessus.

Les Méthodes pour effectuer les requêtes SQL

- *Méthodes*
  - ***.run(sql, [params], [callback(err, [row])])*** : pour les requêtes de création, insertion, suppression et modification.
  - ***.get(sql, [params], callback(err, row)])*** : pour les requêtes de sélection avec la récupération du premier résultat.
  - ***.all(sql, [params], callback(err, row)])*** : pour les requêtes de sélection d'une table entière.
  - ***.close([callback(err)])*** : ferme la connexion à la base de données

- Propriétés
  - ***lastID*** : ID du dernier élément inséré dans la table lors d'une requête d'insertion
  - ***changes*** : informations sur les lignes affectés par une requête.

Mode d’exécution

- ***.serialize()***
  - Exécution des requêtes de manière séquentielle, une requête après l’autre selon l’ordre d’écriture dans le code.

- ***.parallelize()***
  - Exécution des requêtes en parallèle.

Implémentation base de données SQLite3

```
import sqlite3 from 'sqlite3'
const db = new sqlite3.Database(':memory:', sqlite3.OPEN_READWRITE, (err) => {
  if (err) {console.error(err.message)}
  console.log('Connected to the database !')
})
db.serialize(() => {
  db
    .run('CREATE TABLE product(id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(50))', (err) => {
      if(!err) console.log('Table created !')
    })
    .run('INSERT INTO product(name) VALUES (?),(?), (?)', ['tomato', 'apple', 'orange'], function (err) {
      if(!err) console.log(`${this.changes} affected and lastID ${this.lastID}`)
    })
})
db.parallelize(() => {
  db
    .all('SELECT * FROM product', function (err, rows) {
      if(!err) console.log('results before update', rows)
    })
    .run('UPDATE product SET name=? WHERE id=?', ['salad', 3], function (err){
      if(!err) console.log(`updated, ${this.changes} row affected`)
    })
    .all('SELECT * FROM product', function(err, rows) {
      if(!err) console.log('results after update', rows)
    })
})
db.close(() => {console.info('database closed !')})
```

```
[Running] node "e:\formations
Connected to the database !
Table created !
3 affected and lastID 3
results before update [
  { id: 1, name: 'tomato' },
  { id: 2, name: 'apple' },
  { id: 3, name: 'orange' }
]
updated, 1 row affected
results after update [
  { id: 1, name: 'tomato' },
  { id: 2, name: 'apple' },
  { id: 3, name: 'salad' }
]
database closed !
```

Exercices

Enoncé de l’exercice

- 0-exercices/13-ex.md

JSON WEB TOKEN

Json Web Token

- Installation : *npm install jsonwebtoken*
- *Documentation*
- Pour gérer l'authentification d'un utilisateur sur un serveur.
- Données chiffrées (*token*) permettant d'identifier un utilisateur.
  Le *token* n'est pas stockée sur le serveur comme c'est le cas pour un identifiant unique de session, mais il est envoyé par le client (navigateur) à chaque requête vers le serveur.
- Le serveur peut vérifier les informations stockées dans le *token* à partir de la signature du *token* (clé secrète) qui est partagé entre le serveur et le client.
- La méthode *.sign(data, secret, options, callback)* : génère un *token*.
- La méthode *.verify(token, secret, [options, callback])* : vérifie la validité du *token*.

Décomposition du Json Web Token

- ***Header***
  - En-tête qui contient sous la forme d'un *JSON* des informations sur l'algorithme et le type de *token* utilisé, le tout encodé en *base64*.

- ***Payload***
  - les données qui pourront être récupérées lors de la vérification. De même encodé en *base64*.

- ***Signature***
  - Concaténation du *header*, *payload* et de la clé secrète partagée entre le client et le serveur.

Illustration Json Web Token

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Source de l'image JWT.io

BCRYPT

Bcrypt : module bcrypt

- Installation : *npm install bcrypt*
- Documentation
- Cryptage irréversible : impossible de revenir vers la donnée initiale

- Cas d'utilisation des mots de passe avec les méthodes suivantes :
  - ***.genSalt([nb, version], callback)***

    Génère un sel, par défaut nb=10 et version=b
  - ***.hash(plain, salt, callback)***

    Génère un mot de passe hashé à partir du mot de passe brut et du sel
  - ***.compare(plain, hashDb, callback)***

    Compare les 2 hash (hash crypté depuis la saisie brut *plain* de l'utilisateur et *hashDb* en base de données.
- Les méthodes *hash*, *compare*, *genSalt* peuvent s'utiliser de manière **asynchrone**, avec des **promesses** ou **synchrone** avec le suffixe Sync, exemple ***.hashSync()***

Implémentation hachage

```
db.serialize(() => {
  db.run('CREATE TABLE user(id INTEGER PRIMARY KEY AUTOINCREMENT, password VARCHAR(255))', (err) => {
    if(!err) console.log('Table user created !')
  })
  const salt = 10
  const PlainPassword = '>j5LqNgHw*%74Y7'
  hash(PlainPassword, salt, (err1, hashPassword) => {
    if (!err1) {
      db
        .run('INSERT INTO user(password) VALUES (?)', hashPassword, function (err2) {
          if(!err2) console.log(`${this.changes} affected and lastID ${this.lastID}`)
        })
        .all('SELECT id, password FROM user', function (err3, row) {
          if(!err3) console.log('user in database ', row)
        }).close(() => {console.info('database closed !')})
    }
  })
})
```

TERMINAL PROBLEMS OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\3-exercices\1-
Connected to the database !
Table user created !
1 affected and lastID 1
user in database  [
  {
    id: 1,
    password: '$2b$10$XGzI3iRaUQA2rAxbVKHUUuTg9xgV7.aT4Rxvrik8tWMP8PjYF4M.K'
  }
]
database closed !
```

Implémentation comparaison

```
const plain = '>j5LqNgHw*%74Y7'
const fromDb = '$2b$10$Xp2HON7h.Qe3fBOJJc9hOef9sm2MknOW2x.kt.6WOGpwrPofltFWC'
compare(plain, fromDb)
.then((ok) => {
  if(ok) console.log('same password')
  else throw new Error('not same password')
}).catch(ko => console.log('error : ', ko))
```

TERMINAL PROBLEMS OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\3-exercices\1-back\demo.mjs"
same password

[Done] exited with code=0 in 0.33 seconds
```

Exercices

Enoncé des exercices

- 0-exercices/14-ex.md
- 0-exercices/15-ex.md

Ce qu'il faut retenir

SAVE

- Node est un ensemble de technologie écrit en C, C++ et JavaScript permettant de réaliser des applications Web côté Serveur, SPA, IoT etc.
- La gestion des fichiers et des *streams* s'effectue à l'aide du paquet ***fs*.**
- Le paquet ***http*** permet de réaliser un serveur Web.
- Sécurisez les mots de passe à l'aide du paquet ***bcrypt*.**
- L'authentification d'un utilisateur lors d'une requête peut être géré à l'aide du paquet ***jsonwebtoken*.**

Conseils du formateur

- ***Bcrypt*** est indispensable pour crypter les mots de passe utilisateur.
- Ne jamais stocker des informations sensibles en utilisant JWT.
- Vous pouvez aller plus loin en utilisant ***Express***, *framework* de *NODE* facile à prendre en main.
- Vous pouvez allez plus loin avec les bases de données en explorant les plugins ***MySQL*** ou ***MongoDB*** pour des échanges avec des bases données distantes.

Chapitre 10 : Introduction

TypeScript

Qu’est-ce que TypeScript ?

- ***Superset*** de JavaScript : une surcouche du langage JavaScript avec des nouvelles fonctionnalités.
- Créer en 2012 par *Microsoft* pour combler les lacunes du langage JavaScript.
- Fonctionnalités phares : les **types** et les **classes.**

Installation & configuration

- Installation

```
npm install --save-dev  typescript
```

- Configuration du fichier ***tsconfig.json***

```
{
  "compilerOptions": {
    "module": "NodeNext",
    "esModuleInterop": true,
    "outDir": "dist",
    "target": "ESNext",
    "strict": true,
    "noEmitOnError": true,
    "noUncheckedIndexedAccess": true,
    "declaration": true,
    "declarationDir": "dist/d",
    "experimentalDecorators": true
  },
  "include": [
    "src"
  ],
  "exclude": [
    "dist"
  ],
}
```

- [Documentation complète options configuration TypeScript]

Utilisation du TypeScript

1. Créer un fichier avec une extenstion *.ts*

2. Typer
   - Variables
   - Propriétés
   - Paramètres des fonctions
   - Fonctions avec le type de retour

3. Compiler pour transformer le code en JavaScript avec la cible souhaitée (option *target* dans la configuration)

```
> tsc
```

Type

- Types primitifs et leurs dérivés
  - boolean
  - string
  - string[]
  - number
  - number[]
  - {}
  - { [key : string] : string }
  - Etc.

- Types des modules développés par la communauté disponibles sur :
  - npmjs.com
  - Dépôt GitHub des types

- Types personnalisés

```
// Types primitifs
const myNb : number = 3
const isFine : boolean = true

const ltName : string = 'tshimini'

const europreSeasons : string[] = [
  'winter',
  'fall',
  'spring',
  'summer'
]

const numbers : number[] = [0, 1, 2, 3, 4,]
const wrongNumbers : number[] = [
  0,
  '1'    Type 'string' is not assignable to type 'number'.
]
type USER = {
  id: number,
  firstname: string,
  lastname?: string
}
// Type personnalisé
const user : USER = {
  id: 1,
  firstname: 'glodie'
}
```

Readonly

- Mot-clé qui spécifie qu'une variable, paramètre, retour d'une fonction sera disponible uniquement en lecture seule. C'est-à-dire qu'il n'est pas possible de modifier sa valeur lors de son utilisation.

```
// Type personnalisé
const user : {
  readonly id: number,
  firstname: string,
  lastname?: string
} = {
  id: 1,
  firstname: 'glodie'
}

user.id = 4    Cannot assign to 'id' because it is a read-only property.
```

- ? Indique que la propriété est facultative.

Type narrowing

- Une variable, propriété, paramètre, retour d'une fonction peut avoir un ou plusieurs types de retour selon le contexte.

- Le ***narrowing*** permet de réduire les types possibles d'une entité à partir de l'évaluation d'une condition.
  Pour cela, vous pouvez utiliser l'une des méthodes ou mots-clés suivants :

  - *Array.isArray()*
  - *Typeof*
  - *instanceof*
  - *In*

```
function size(input: string|number|string[]|number[]|{name: string}|null|undefined) {
  if(Array.isArray(input) || typeof input === 'string') {
    return input.length
  }
  else if(input instanceof Object && 'name' in input) {
    return input.name.length
  }
  return -1
}
```

Types any, unknown

- Le type ***any*** permet de pouvoir affecter n'importe quelle valeur à l'entité.

- Le type ***unknown*** est pratique pour le cas de figure ou le type n'est pas connu à la définition, mais il sera connu et précisé lors de l'affectation d'une valeur.

```
function size(input: unknown) {
  if(Array.isArray(input) || typeof input === 'string') {
    return input.length
  }
  else if(input && 'name' in input) {
    return input['name'].length
  }
  return -1
}
```

enum

- ***enum*** est une liste fermée des valeurs possibles pour une entité, par exemple les mois de l'année, les saisons européennes, etc.

```typescript
enum SEASONS {
  winter,
  fall,
  spring,
  summer
}

const fall : SEASONS = SEASONS.fall
console.log('current season :', fall) // 1
```

TypeScript

```js
var SEASONS;
(function (SEASONS) {
    SEASONS[SEASONS["winter"] = 0] = "winter";
    SEASONS[SEASONS["fall"] = 1] = "fall";
    SEASONS[SEASONS["spring"] = 2] = "spring";
    SEASONS[SEASONS["summer"] = 3] = "summer";
})(SEASONS || (SEASONS = {}));
const fall = SEASONS.fall;
console.log('current season :', fall); // 1
export {};
```

Transformation du code à gauche en JS

Types génériques (source image Bytefer)

- Les types génériques sont des alias qu'on peut utiliser pour éviter la répétition dans son code lorsqu'on utilise plusieurs fois le même type.
- Généralement, pour les alias, on choisit une seule lettre, souvent la lettre T.
- S'il y a plusieurs alias à créer, on utilisera la suite de l'alphabet U, V, etc.

Class

Par rapport à JavaScript, TypeScript ajoute l'encapsulation (visibilité) des propriétés et des méthodes :

- *Public* : accessible à tout le monde ;
- *Protected* : accessible par les classes filles (dérivées, sous-classes) ;
- *Private*: accessible uniquement depuis la classe.

```typescript
class Car {
  public model : string
  protected maxVelocity : number
  private color : string

  public constructor (model : string, maxVelocity: number, color : string) {
    this.model = model
    this.maxVelocity = maxVelocity
    this.color = color
  }
}
```

TypeScript

```javascript
class Car {
    model;
    maxVelocity;
    color;
    constructor(model, maxVelocity, color) {
        this.model = model;
        this.maxVelocity = maxVelocity;
        this.color = color;
    }
}
export {};
```

Transformation en JavaScript

Classe abstraite

- Également la possibilité décrire des classes abstraites qui servent de modèle pour les classes dérivées.

- Les classes dérivées, dites concrètes doivent implémenter les méthodes abstraites de la classe abstraite. Par définition, une classe abstraite ne peut pas être instanciée.

TypeScript

```typescript
abstract class Vehicle {
  protected model : string
  protected maxVelocity : number
  protected color : string
  abstract describe() : string
}
class Trotinette extends Vehicle {
  describe(): string {
   return 'trotinette électrique'
  }
}
```

JavaScript

```javascript
class Vehicle {
}
class Trotinette extends Vehicle {
    describe() {
        return 'trotinette électrique';
    }
}
export {};
```

Interface

- Similaire aux types avec la particularité de pouvoir fusionner les interfaces entre elles.
- On peut les utiliser pour ajouter des nouveaux types aux variables déjà définies comme par exemple les variables *window*, *document*, etc.

```typescript
type adress = { num : number, city: string, street: string}
interface addr { address : adress }
interface addr { postCode : string }
interface addr { country : string }
class Address implements addr {
  address: adress
  postCode: string
  country: string
  constructor(addr: adress, postCode : string , country: string) {
    this.address = addr
    this.postCode = postCode
    this.country = country
  }
}
const paris = new Address(
  {num : 1, city: 'paris', street : 'latin'},
  '75001',
  'france'
)
console.dir(paris)
```

```typescript
class Address {
    constructor(addr, postCode, country) {
        this.address = addr;
        this.postCode = postCode;
        this.country = country;
    }
}
const paris = new Address(
    { num: 1, city: 'paris', street: 'latin' },
    '75001',
    'france'
);
console.dir(paris);
export {};
```

Créer un fichier de déclaration TypeScript

- TypeScript se sert des fichiers de définition dont l'extension est ***.d.ts*** pour connaître les types attendus par les variables, propriétés, etc.

- Les fichiers de déclaration aident également l'éditeur pour l'auto-complétion.

- Regrouper les fichiers de déclaration dans un dossier à part.

- La plupart des modules ont des paquets contenant la description de chaque paramètre et fonctions de ce dernier. Le [dépôt GitHub DefinitelyTyped]() répertorie plusieurs types.

- Le paquet ***dts-gen*** permet de générer des paquets des modules n'ayant pas de fichiers de déclaration.

```
npx i -D dts-gen && npx dts-gen -m nameOfModuleWithoutDeclarationFile
```

Créer un fichier de déclaration TypeScript

- Dans le cas où vous utiliseriez un module qui n'a pas de fichier de déclaration des types. Vous devez le créer avec au moins les propriétés et méthodes que vous utilisiez dans votre code.

```
export declare function compareSync(data: string | Buffer, encrypted: string): boolean;
export declare function compare(data: string | Buffer, encrypted: string): Promise<boolean>;
export declare function compare(
  data: string | Buffer,
  encrypted: string,
  callback: (err: Error | undefined, same: boolean) => any,
): void;
```

Exercices

Enoncé des exercices

GitHub

- 0-exercices/16-ex.md
- 0-exercices/17-ex.md

Chapitre 11 : Optimisations

Web Workers

- Le cours d'exécution principal **délègue la réalisation** d'une ou plusieurs tâches aux *web workers*.
- Un *web worker* s'exécute sur un **nouveau** ***thread*** pour effectuer les opérations qui lui ont été assignés.
- A la fin des opérations, *le web worker* notifie le *thread* principal.
- Le thread principal exécute la réponse renvoyée par le *web worker*.
- Gains :
  - Les traitements lourds n'impactent pas l'interactivé du site.
  - L'utilisateur peut effectuer d'autres actions en parallèle.
  - Interactions plus fluides.
- Inconvénients
  - Pas accès aux objets du document (*cookie*, *history*, *location*, *DOM*, etc), il faut notifier le fichier principal pour que ce dernier puisse effectuer les opérations précédentes

Implémentation Web Workers

Côté Front

- Voir la documentation

Côté Back

- Voir la documentation

Techniques d’optimisation et écologiques

- **YAGNI** : *You ain’t gonna need it* (**développez que ce qui est nécessaire**).
- **Compresser** les **fichiers** HTML et documents en plus du JS et CSS.
- Préférer le format **SVG** aux autres formats images.
- Modifier le **DOM** et les **propriétés CSS en une seule fois**.
- En fonction de la fréquence des requêtes et d’utilisation des données optez pour une solution de **Cache**.
- Utiliser les animations que lorsque c’est pertinent et utile.
- N’embarquez pas des modules en production dont vous n’exploitez que 10 % des fonctionnalités (ex : intégralité de Bootstrap pour n’utiliser que 3 - 4 classes).
- **Dépendances à jours**.
- Ajouter les **indexes** et **procédures stockées** au niveau des **bases de données SQL** pour les requêtes les plus fréquentes.

Techniques d'optimisation et écologiques

- Utiliser les avantages des requêtes **asynchrones**.
- Utiliser ***promise.all()*** lorsque vous avez plusieurs promesses.
- Avoir **un fichier de sortie JS** qui inclut le CSS.
- Utiliser le protocole ***HTTP 2*** et le certificat ***SSL*** (HTTPS), plus rapide et sécurisé).
- **Valider son code** HTML/CSS avec ***W3C Validator*** et JS à l'aide de **standardJs** ou *ESLint*.
- Créer des processus enfants.
- Gérer les parties susceptibles de déclencher des **erreurs** avec un **try/catch**.
- En **production**, Utiliser **PM2** au lieu de Node pour lancer votre Application. **PM2 redémarre automatiquement l'application en cas de crash**.

Modules performance Node : Autocannon

- Simuler les tests de performance

- Effectuer des requêtes HTTP avec la possibilité d'indiquer le nombre de connexion simultanée, délai d'exécution des tests, etc.

- Avoir des statistiques sur la charge que peut supporter l'application :
  - Latence (temps de réponse) ;
  - Nombre de requêtes simultanées pour mesurer le débit ;
  - Quantité des données lues.

- Usage :
  - *En ligne de commande* , ***autocannon [options] URL***
  - *Script*

Résultat simulation autocannon

autocannon -c 10 –d 10 localhost:3000

```
Running 10s test @ http://localhost:3000
10 connections


┌─────────┬──────┬──────┬───────┬──────┬─────────┬─────────┬──────────┐
│ Stat    │ 2.5% │ 50%  │ 97.5% │ 99%  │ Avg     │ Stdev   │ Max      │
├─────────┼──────┼──────┼───────┼──────┼─────────┼─────────┼──────────┤
│ Latency │ 0 ms │ 0 ms │ 0 ms  │ 1 ms │ 0.02 ms │ 0.16 ms │ 16.45 ms │
└─────────┴──────┴──────┴───────┴──────┴─────────┴─────────┴──────────┘

┌───────────┬─────────┬─────────┬─────────┬─────────┬─────────┬─────────┬─────────┐
│ Stat      │ 1%      │ 2.5%    │ 50%     │ 97.5%   │ Avg     │ Stdev   │ Min     │
├───────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┤
│ Req/Sec   │ 20623   │ 20623   │ 25583   │ 26271   │ 25131.2 │ 1540.94 │ 20615   │
├───────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┼─────────┤
│ Bytes/Sec │ 2.29 MB │ 2.29 MB │ 2.84 MB │ 2.92 MB │ 2.79 MB │ 171 kB  │ 2.29 MB │
└───────────┴─────────┴─────────┴─────────┴─────────┴─────────┴─────────┴─────────┘


Req/Bytes counts sampled once per second.

251k requests in 10.05s, 27.9 MB read
```

Source de l'image autocannon

Module performance Node : Clinic.js

- Identifie et propose des solutions pour régler les problèmes de performance.
  - CPU
  - RAM
  - Event Loop
  - Etc.
- Outils
  - ***Doctor*** : diagnostic les problèmes de performance de l'application ;
  - ***Flame*** : donne le temps d'exécution d'une fonction au niveau du CPU et le temps qu'il passe en haut de la pile ;
  - ***BubbleProf*** : aide à résoudre les problèmes liés au blocage de l'*Event Loop*.
- A **combiner** avec **AutoCannon** (cf Documentation)

Diagrammes clinic.js

clinic doctor --autocannon [-c 100 –d 5] -- node server.js

Source de l'image clinic.js

Exercices

Enoncé de l’exercice

- 0-exercices/18-ex.md

Chapitre 12 : Introduction aux Design Patterns

Qu’est-ce qu’un Design Pattern ?

- Les designs patterns viennent initialement du domaine logiciel.

- Un *Design Pattern* ou ***patron de conception*** est une solution de haut niveau (sans implémentation) apporté et validé par la communauté pour résoudre un problème récurrent dans la conception logiciel. Autrement dit la solution à implémenter varie d’un langage à un autre et d’un développeur à un autre.

- On compte initialement 23 Designs Patterns repartis en 3 groupes par ses créateurs ***Erich Gamma, Ralph Johnson, Richard Helm*** et ***John Vlissides*** en 1994:
  - Patrons de **création** : solutions pour résoudre les problèmes de création des objets.
  - Patrons de **structure** : solution pour rassembler des objets et des classes dans des packages.
  - Patrons de **comportement** : solution pour définir la responsabilité de chaque objet et de les faire communiquer.

Design Pattern de création

Singleton

- Une seule instance d'une classe A (un seul objet de la classe A dans toute l'application).

- Permet de réduire l'utilisation des variables globales qui peuvent être modifié et provoquer des effets de bord sur le reste de l'application. Le singleton une fois implémenté n'est pas modifiable.

- Souvent utilisé dans la mise en place des bases données, en effet, on peut se contenter d'avoir une seule instance pour gérer les traitements au niveau de la base de données. A chaque accès à la base de données pour un traitement, l'instance unique sera utilisée.

- Le singleton n'est pas testable, il est aussi considéré comme un anti-pattern, car il ne respecte pas toutes les bonnes pratiques de la programmation.

Singleton exemples d'implémentation

Avec un constructeur

```
class Pi {
  constructor() {
    if(!Pi.instance) {
      console.log('first time')
      this.value = 3.14
      Pi.instance = this
    }
    return Pi.instance
  }
}
const p1 = new Pi(),
p2 = new Pi(),
p3 = new Pi()

console.log(p1.value) // first time et 3.14
console.log(p2.value) // 3.14
console.log(p3.value) // 3.14
console.log(p1 === p2, p2 === p3, p3 === p1) // true true true
```

Avec une méthode statique

```
class Pi {
  static value

  static getValue() {
    if(!this.value) {
      this.value = 3.14
      console.log('first time')
    }
    return this.value
  }
}

Pi.getValue() // first time et 3.14
Pi.getValue() // 3.14
Pi.getValue() // 3.14
Pi.getValue() // 3.14
Pi.getValue() // 3.14
Pi.getValue() // 3.14
```

Factory

- Construire des objets similaires sans se soucier de leurs particularités au moment de la création.

- Simplifie la construction des objets, mais augmente considérable le nombre des classes.

```
class Car {}
class Ship {}
class Bus {}
class Plane {}
class VehicleFactory {
  constructor(type) {
    switch(type.toLowerCase()) {
      case 'car':
        return new Car()
      case 'ship':
        return new Ship()
      case 'bus':
        return new Bus()
      case 'plane':
        return Plane()
      default:
        throw new Error('unknown vehicle')
    }
  }
}
const car = new VehicleFactory('car')
const bus = new VehicleFactory('bus')
// const truck = new VehicleFactory('truck') // error
console.log(
  car instanceof Car,// true
  bus instanceof Bus // true
)
```

Design Pattern de structure

Adapter

- Permettre à 2 objets incompatibles de pouvoir collaborer.

- On l'utilise principalement lors de la migration vers une nouvelle version d'un objet sans avoir à tout modifier. La classe adaptatrice aura la même signature que l'ancienne version de l'objet, cependant dans ces entrailles, il utilisera la nouvelle version.

Design Pattern de comportement

Observer

- Plusieurs observateurs souscrivent auprès d'un sujet pour une tâche.

- Lorsque la tâche est exécutée, le sujet notifie tous les observateurs souscripteurs.

- C'est un Design Pattern très utilisé, notamment dans la programmation événementielle. Par exemple, lors de l'utilisation de la méthode *.addEventListener()* du DOM.

Exercices

Enoncés des exercices

- 0-exercices/19-ex.md
- 0-exercices/20-ex.md

Mise en production

Mise en production

- Déploiement sur un serveur dédié ou un service cloud.

- Serveur dédiée local ou chez un fournisseur d'hébergement.
  - Les installations à effectuer :
    - Node ;
    - PM2 .
- Cloud
  - Tout est fourni via les plateformes, uniquement une configuration à faire.
  - Déploiement continue avec via git (système de versioning).
  - Fournisseurs
    - Heroku
    - CloudFare pages
    - Vercel
    - Netlify

Travaux pratiques

Enoncés TP

- 1-tp/0-front.md
- 1-tp/1-liaison-front-back.md
- 1-tp/2-design-pattern.md

Émargements

Pensez à vérifier les feuilles d'émargement

Évaluations formateur

N’oubliez pas les évaluations formateur avant de partir.

Fin
Merci d’avoir participer à cette formation

orsys.fr