



# Modules/Loaders/Bundlers JavaScript



# Modules JavaScript



## ▸ JavaScript à sa conception

- Objectif : créer des interactions côté client, après chargement de la page
- Exemples de l'époque :
  - Menu en rollover (image ou couleur de fond qui change au survol)
  - Validation de formulaire

## ▸ JavaScript aujourd'hui

- Applications front-end, back-end, en ligne de commande, de bureau, mobiles...
- Applications pouvant contenir plusieurs centaines de milliers de lignes de codes (Front-end de Facebook > 1 000 000 LOC)
- Il faut faciliter le travail collaboratif, en plusieurs fichiers et en limitant les risques de conflit

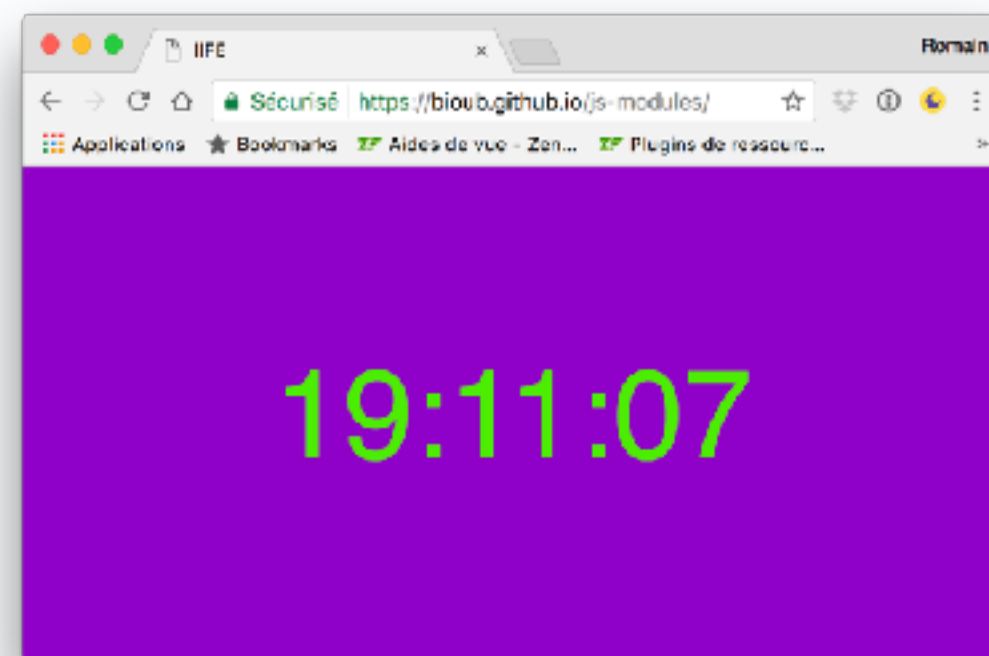
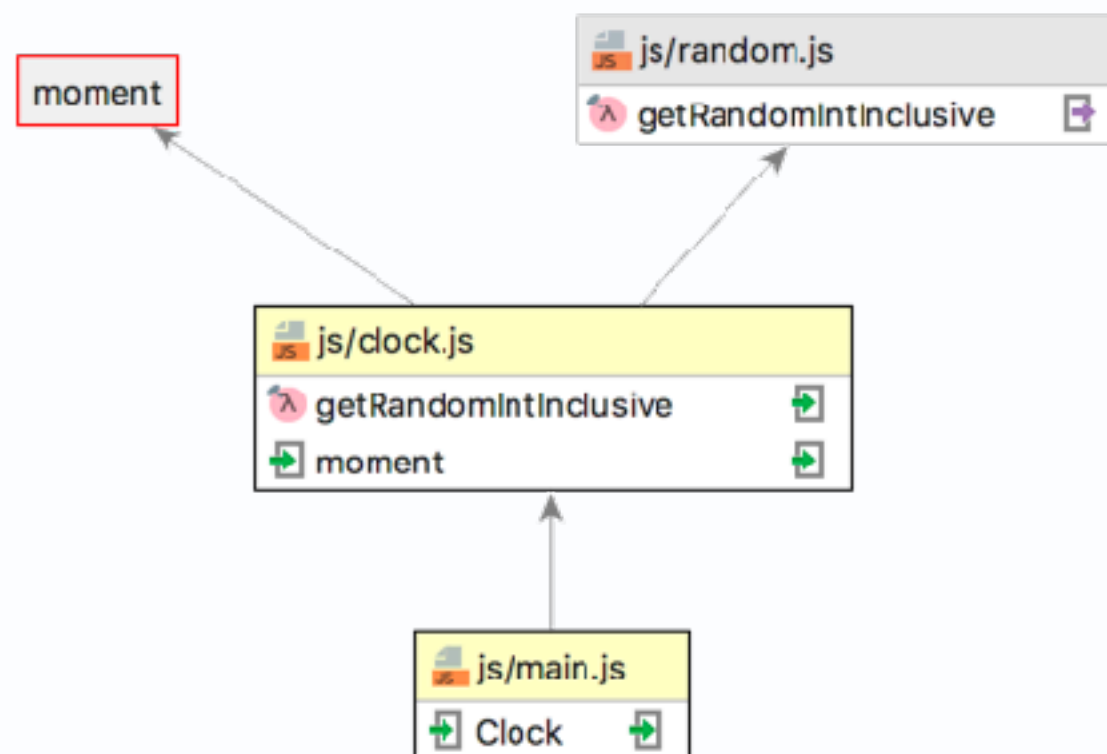


- Objectifs d'un module JavaScript
  - Créer une portée au niveau du fichier
  - Permettre l'export et l'import d'identifiants (variables, fonctions...) entre ces fichiers qui auront désormais leur propre portée
- Principaux systèmes existants
  - IIFE / Function Wrapper
  - CommonJS
  - AMD
  - UMD
  - SystemJS
  - ES6 (statiques mots clés import / export)
  - ESNext : import() (fonction asynchrone)

# Modules JavaScript - Introduction



- ▶ Exemple utilisé pour la suite



- ▶ Le point d'entrée de l'application est le fichier `main.js`, qui dépend de `Clock` défini dans le fichier `clock.js`, qui dépend lui-même de `getRandomIntInclusive` du fichier `random.js` et `moment` défini dans le projet Open Source `moment.js`
- ▶ Exemples : <https://github.com/bioub/js-modules>
- ▶ Démo : <https://bioub.github.io/js-modules/>



## ► Immediately-invoked function expression (IIFE)

```
// random.js
(function (global) {
  'use strict';

  var getRandom = function() {
    return Math.random();
  };

  var getRandomIntInclusive = function(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(getRandom() * (max - min + 1)) + min;
  };

  global.getRandomIntInclusive = getRandomIntInclusive;
})(this);
```

- Une function expression appelée immédiatement
  - Limite la portée des identifiants (getRandom, getRandomIntInclusive)
  - Permet de renommer localement des dépendances (this → global)



## ► Import / Export en IIFE

```
// clock.js
(function(global, moment, random) {
  'use strict';

  var Clock = function(parentElt) {
    this.parentElt = parentElt;
  };

  Clock.prototype.update = function() {
    var n = random(0, 255);
    document.body.style.backgroundColor = 'rgb('+n+', '+n+', '+n+')';
    this.parentElt.innerHTML = moment().format('HH:mm:ss');
  };

  Clock.prototype.start = function() {
    this.update();
    setInterval(this.update.bind(this), 1000);
  };

  global.Clock = Clock;
})(this, moment, getRandomIntInclusive));
```

- Pour importer on utilise des variables globales, éventuellement en paramètres d'entrée de la fonction pour pouvoir les renommer localement (getRandomIntInclusion → random)
- Pour exporter on crée des variables globales en étendant l'objet global



- ▶ Limiter les risques de conflits en IIFE

Exemple : API Maps de Google

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<div id="map"></div>
<script>
  function initMap() {
    var uluru = {lat: -25.363, lng: 131.044};
    var map = new google.maps.Map(document.getElementById('map'), {
      zoom: 4,
      center: uluru
    });
    var marker = new google.maps.Marker({
      position: uluru,
      map: map
    });
  }
</script>
<script async defer src="https://maps.googleapis.com/maps/api/js?callback=initMap"></script>
</body>
</html>
```





## ► Limiter les risques de conflits en IIFE

```
(function(global, google) {  
  'use strict';  
  
  var GMap = function (parentElt, options) {  
    // ...  
  };  
  
  var GMarker = function (parentElt, options) {  
    // ...  
  };  
  
  global.google = google || {};  
  google.maps = google.maps || {};  
  google.maps.Map = GMap;  
  google.maps.Marker = GMarker;  
  
})(global, google);
```

- Pour limiter les conflits de nom on simule des namespaces.
- Il ne faut créer qu'une seule variable globale du nom de la société, nom du projet... (google dans l'exemple)



## ► Utilisation dans le Navigateur

```
<!DOCTYPE html>
<html lang="en">
<head></head>
<body>
  <div class="clock"></div>
  <script src="node_modules/moment/moment.js"></script>
  <script src="js/random.js"></script>
  <script src="js/clock.js"></script>
  <script src="js/main.js"></script>
</body>
</html>
```

- Avec les modules IIFE c'est au développeur de créer les balises script
- Les imports/exports se faisant via des variables globales, il faut maintenir ces balises dans l'ordre des dépendances

## ► Utilisation dans Node.js

- Node.js incluant son propre système de module, il n'est pas recommandé d'utiliser des modules IIFE



## ▸ CommonJS (parfois CJS)

Projet visant à créer des API communs pour du développement JavaScript hors navigateur (console, GUI...)

Exemple : standardiser l'accès aux fichiers

Le projet propose une norme pour le chargement de modules utilisé entre autre par Node.js

<http://wiki.commonjs.org/wiki/Modules/1.1.1>

## ▸ Création d'un module

```
// calcullette.js
exports.ajouter = function(nb1, nb2) {
  return Number(nb1) + Number(nb2);
};
```

## ▸ Utilisation

```
// main.js
var calc = require('./calcullette');

console.log(calc.ajouter(2, 3)); // 5
```



## ▸ Export

```
// (function (exports, require, module, __filename, __dirname) {  
'use strict';  
  
var getRandom = function() {  
    return Math.random();  
};  
  
exports.getRandomIntInclusive = function(min, max) {  
    min = Math.ceil(min);  
    max = Math.floor(max);  
    return Math.floor(getRandom() * (max - min + 1)) + min;  
};  
// });
```

- Le code s'exécute dans une fonction, pas besoin de la créer (ici en commentaire)
- Cette fonction contient un paramètre exports, qui est un objet que l'on pourra récupérer à l'import (il n'y a plus qu'à l'étendre)



## ▸ Import

```
// (function (exports, require, module, __filename, __dirname) {  
'use strict';  
  
var moment = require('moment');  
var getRandomIntInclusive = require('./random').getRandomIntInclusive;  
  
var Clock = function(parentElt) {  
  this.parentElt = parentElt;  
};  
  
// ...  
// });
```

- A l'import on utilise la fonction `require()`
  - Si le fichier est local on commencera toujours par `./` ou `../`
  - Sinon on fait référence à une installation via npm ou bien un fichier se trouvant dans le binaire de Node (`fs`, `http`, `readline...`)
  - La fonction `require` est synchrone et peut s'utiliser dans un `if` ou une boucle



## ‣ Export (autre chose qu'un objet)

```
// (function (exports, require, module, __filename, __dirname) {  
'use strict';  
  
var Clock = function(parentElt) {  
  this.parentElt = parentElt;  
};  
  
// ...  
  
module.exports = Clock;  
  
// });
```

- Lorsque l'on exporte autre chose qu'un objet, on peut écraser exports en écrivant module.exports (ici on exporte une fonction constructeur)
- Une bonne pratique pourrait être d'importer en début de fichier et d'exporter toujours en fin de fichier (en pratique c'est rarement le cas)



- ▶ Utilisation dans le Navigateur
  - Il faut passer par une bibliothèque externe :
    - Soit un bundler comme browserify (historique) ou webpack (moderne)
    - Soit un loader comme SystemJS
  - Les fichiers décrivant eux-même leurs dépendances, il suffit d'indiquer un point d'entrée dans l'application (ou plusieurs)
- ▶ Utilisation dans Node.js
  - Node.js supporte par défaut les modules CommonJS



- ▶ **Asynchronous Module Definition**

CommonJS ne permettant de charger des modules côté client sans transformation préalable. Des développeurs ont imaginé la syntaxe AMD.

- ▶ **Fonctionnement**

L'utilisation de modules AMD se fait via 2 fonctions globales : `require()` et `define()`. `define()` permet de définir un module et ses dépendances, `require` définit un point d'entrée dans l'application.





## ▸ Export

```
// random.js
define(function() {
  'use strict';

  return {
    getRandom: function () {
      return Math.random();
    },
    // ...
    getRandomIntInclusive: function (min, max) {
      min = Math.ceil(min);
      max = Math.floor(max);
      return Math.floor(this.getRandom() * (max - min + 1)) + min;
    }
  };
});
```

## ▸ define

- La définition d'un module se passe dans le callback de la fonction define
- Pour exporter on utilise la valeur de retour
- Si plusieurs valeurs à exporter il suffit de retourner un tableau ou un objet



## ► Import

```
// clock.js
define(['moment', './random'], function(moment, random) {
  'use strict';

  var getRandomIntInclusive = random.getRandomIntInclusive;

  var Clock = function(parentElt) {
    this.parentElt = parentElt;
  };

  // ...

  return Clock;
});
```

- Pour importer on passera un tableau en premier paramètre du define
- Le callback sera appelé avec les retours des modules importés, dans le même ordre



## ▸ Point d'entrée

```
// main.js
require(['./clock'], function(Clock) {
    'use strict';

    var clockElt = document.querySelector('.clock');
    var clock = new Clock(clockElt);
    clock.start();
});
```

## ▸ Require

- La fonction require permet de définir un point d'entrée dans l'application
- C'est elle qui va inclure les balises scripts sur la page
- Idéalement elle ne devrait être utilisée qu'un seul fois par page
- Les scripts chargent de manière asynchrone et s'exécute lorsqu'il n'ont pas de dépendances ou que toutes leurs dépendances ont été résolues



- Utilisation dans le Navigateur
  - Il faut passer par une bibliothèque externe :
    - Soit un loader comme curl, Require.js ou plus récemment SystemJS
    - Soit un bundler comme webpack qui va faire une transformation
- Utilisation dans Node.js
  - Non pertinent



- ▶ Universal Module Definition (UMD)
  - ▶ Un module universel, qui est à la fois AMD, CommonJS et parfois IIFE  
<https://github.com/umdjs/umd>
  - ▶ Peut s'utiliser avec tous les loaders et les bundlers existants (et sans rien si IIFE)
  - ▶ Pertinent pour les projets où l'on ne peut pas prédire la techno qui sera utilisée pour charger le module
  - ▶ Exemple de module UMD :
    - ▶ jQuery
    - ▶ Lodash → « The Lodash library exported as a UMD module. »  
<https://github.com/lodash/lodash>
    - ▶ Angular → @angular/core/bundles/core.umd.js



## ▸ Exemple de module UMD

```
(function (root, factory) {  
  if (typeof exports === 'object') {  
    // CommonJS  
    module.exports = factory(require('moment'), require('./random'));  
  } else if (typeof define === 'function' && define.amd) {  
    // AMD  
    define(['moment', './random'], function (moment, random) {  
      return factory(moment, random);  
    });  
  } else {  
    // IIFE (global var)  
    root.Clock = factory(root.moment, root.random);  
  }  
})(this, function (moment, random) {  
  'use strict';  
  
  var getRandomIntInclusive = random.getRandomIntInclusive;  
  
  var Clock = function(parentElt) {  
    this.parentElt = parentElt;  
  };  
  
  // ...  
  
  return Clock;  
}));
```



- Utilisation dans le Navigateur :
  - Sans rien si UMD inclus IIFE (pas toujours le cas)
  - Avec un bundler : browserify ou webpack
  - Avec un loader : curl, Require.js, SystemJS
- Utilisation dans Node.js
  - Avec la fonction require



- ▶ ECMAScript 2015 / ECMAScript 6

La nouvelle version de JavaScript prévoit une syntaxe pour l'utilisation de module. A l'heure actuelle (juillet 2015), ni les navigateurs ni Node.js ou io.js ne supportent cette syntaxe.

- ▶ Babel / Traceur

Babel et Traceur sont des bibliothèques qui permettent de transpiler du code ES6 en ES5 et ainsi l'utiliser sur les moteurs actuels.

- ▶ Installation :

```
npm install -g babel-cli
```

- ▶ Utilisation

Toutes les fichiers JS du répertoire src vers le répertoire dist :

```
babel src --out-dir dist/
```



# Modules JavaScript - ECMAScript 2015 / ES6



- Support dans Node.js

Expérimental depuis Node 8 (--experimental-modules)

Pas avant 2018 : <https://twitter.com/addyosmani/status/859296190323597313>

- Supporté par :

Chrome 60+, Meteor 1.3+

# Modules JavaScript - ECMAScript 2015 / ES6



```
// src/number-converter.js
var exports = {};

exports.convert = function(nb) {
  return Number(nb);
};

export default exports;
```

```
// src/calcullette.js
import numberConverter from './number-converter';

var exports = {};

exports.ajouter = function(nb1, nb2) {
  return numberConverter.convert(nb1) + numberConverter.convert(nb2);
};

export default exports;
```

```
// src/main.js
import calc from './calcullette';

console.log(calc.ajouter(2, 3)); // 5
```



# Loaders JavaScript



- **Require.js**

Bibliothèque permettant le chargement de modules AMD.

- **Plugins**

Quelques plugins existent pour charger par exemple des fichiers CSS, des fichiers de traduction, etc...

- **r.js**

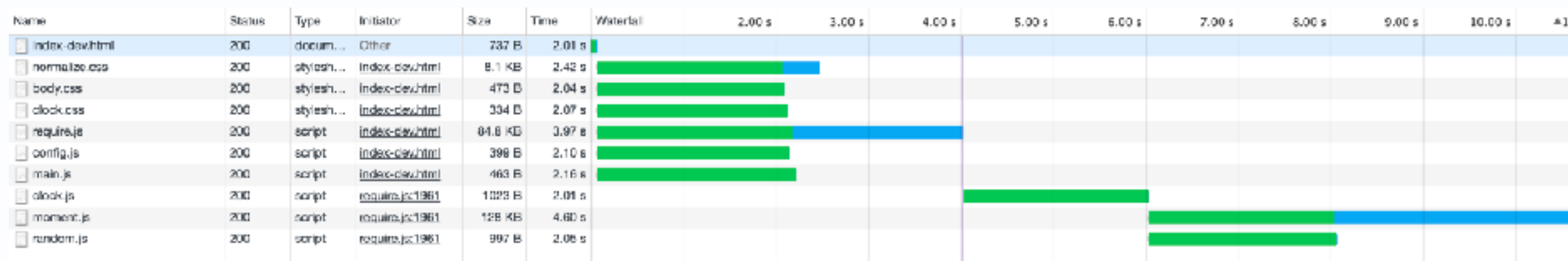
Il est possible d'utiliser un builder pour créer un seul fichier de production, le chargement se ferait sinon en « escalier ».



# Loaders JavaScript - Require.js



## ► Chargement en escalier





- ▶ **SystemJS**

SystemJS est un loader universel qui sait charger des modules CommonJS, AMD, ES6 et IIFE dans les navigateurs et sous node.js

<https://github.com/systemjs/systemjs>

- ▶ **jspm**

Afin de faciliter le chargement de modules installés via des gestionnaires de dépendances, jspm permet le chargement de packages npm ou bien de

- ▶ **SystemJS Builder**

Afin de faciliter le chargement de modules installés via des gestionnaires de dépendance



# Bundlers JavaScript



- ▶ Browserify

Permet de charger des modules CommonJS côté client.

- ▶ Installation :

`npm install -g browserify`

- ▶ Transormation en code client :

`browserify main.js > calculette-browser.js`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <script src="calculette-browser.js"></script>
</body>
</html>
```





- **webpack**

webpack est un bundler universel, il sait charger n'importe quel type de modules, AMD, CommonJS, ES6

- Des loaders supplémentaires permettent de charger des fichiers CSS / LESS / SCSS / i18n / ...
- Depuis sa version 2, webpack supporte nativement les modules ES6, alors qu'il fallait utiliser un transpileur comme Babel dans la version 1.
- Le projet open-source le plus financé avec Vue.js (+ 300k\$ par an), son développeur contribue à webpack à temps plein.

# Bundlers JavaScript - webpack



```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');

module.exports = {
  entry: './src/js/index.js',
  output: {
    path: __dirname + '/dist',
    filename: 'bundle.[hash].js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /(node_modules|bower_components)/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: [["env", {
              "targets": {
                "browsers": ["> 1% in FR"]
              }
            }]]
          }
        }
      }
    ]
  },
  plugins: [
    new CleanWebpackPlugin(['dist']),
    new HtmlWebpackPlugin({
      template: './src/index.html'
    })
  ]
};
```