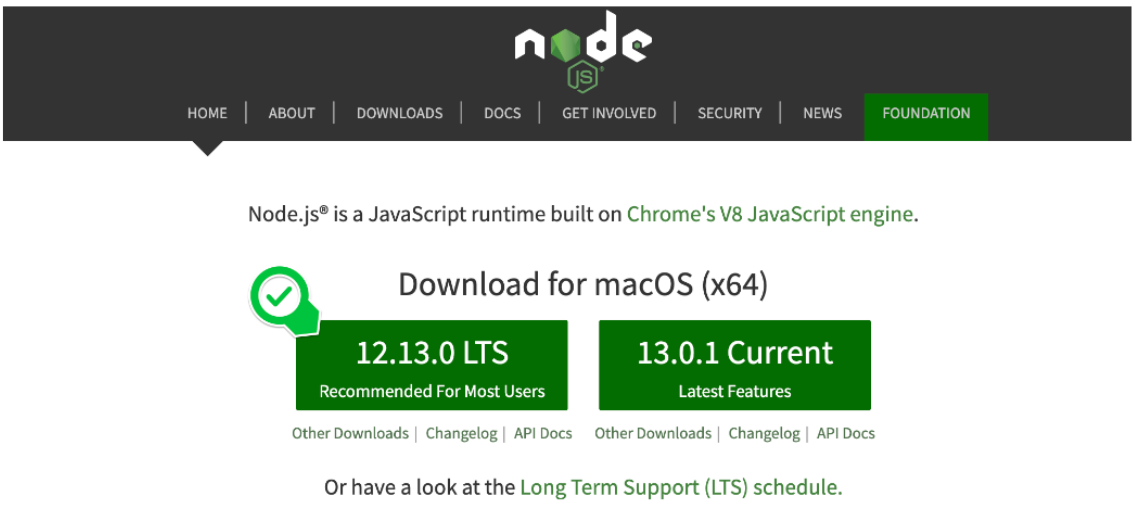
NodeJS est un environnement d'exécution de code JavaScript. Il a été construit sur le moteur JavaScript V8 de Chrome. Il a pour avantage d'être très rapide.

Installation

Rendez-vous sur **Nodes.js® web site**.

Choisissez la version LTS.



Ouvrez le terminal puis exécutez la commande ci-dessous afin d'avoir la version d'installée :

```
node --version
```

Command Line

pwd : indique le répertoire en cours.
ls : liste le contenu.
mkdir nom-dossier : permet de créer un dossier.
cd nom-dossier : permet de se déplacer dans un dossier.
touch nom-fichier : permet de créer un fichier.
node nom-fichier : permet d'exécuter le fichier.
clear : efface le terminal.

Node REPL (Read Evaluation Print Loops)

Il vous permet essentiellement d'exécuter du code ligne par ligne à partir de votre terminal.

.exit : permet d'interrompre le programme, tout comme ctrl + C (deux fois).

Native Node Modules

Lors de l'installation de NodeJS, de nombreux modules ont été installés également. Ce sont essentiellement des bibliothèques qui ont pour objectif de vous faciliter les tâches. Grâce à NodeJS, JavaScript n'est plus prisonnier du navigateur. Cela vous permettra d'interagir directement avec l'ordinateur de l'utilisateur. Par exemple, il nous sera possible d'accéder aux fichiers locaux de son ordinateur.

https://nodejs.org/dist/latest-v18.x/docs/api/

Exemple 1 – Copier un fichier

```
const fs = require('fs')

fs.copyFileSync('file1.txt', 'file2.txt')
```

Exemple 2 – Créer un dossier

```
const fs = require('fs')
const dir = './tmp'

if (!fs.existsSync(dir)){
    fs.mkdirSync(dir, { recursive: true })
}
```

Exemple 3 – Supprimer un fichier ou un dossier

```
const fs = require('fs')

try {
  fs.unlinkSync('file2.txt')

  console.log("Delete File successfully.")
} catch (error) {
  console.log(error)
}
```

Charger un fichier comme un module

Dans votre programme principal, vous allez appeler le fichier externe grâce à la méthode require :

Fichier addition.js

```
const addNumbers = (a,b) => a + b

module.exports = addNumbers
```

Fichier index.js

```
const bar = require('./addition')
// ou import request from 'request'

console.log(bar(5,6))
```

Node Package Manager

npm (node package manager) est le gestionnaire de packages officiel pour Nodejs et il s'agit actuellement du plus grand registre de logiciels au monde. npm fournit un outil de ligne de

commande pour gérer les dépendances des applications, installer, désinstaller et mettre à jour des packages, partager et distribuer du code.

Pour installer un package localement à l'aide de npm, dans la console, exécutez :

```
npm install <package-name>
```

Lors de l'installation globale d'un package (ce qui le rend accessible de n'importe où sur l'ordinateur), vous utiliserez l'indicateur global :

```
npm install <package-name> --global
```

Pour initialiser npm, exécutez dans la console :

```
npm init
```

Cela aura pour effet de créer le fichier package.json à la source de votre application.
Le package.json est un fichier très important dans le développement de Node.js. Il contient des informations qui décrivent votre application ; vous pouvez le considérer comme le fichier manifeste de votre application.
Le fichier package.json est également utile pour suivre les dépendances et les dépendances de développement de votre application (expliquées brièvement).

Comme son nom l'indique, il s'agit d'un fichier JSON (objet JavaScript contenant des paires clé-valeur) qui fournit des métadonnées sur votre application (package).

Crypto application

Dans votre terminal, exécutez :

```
mkdir crypto-app && cd crypto-app
```

Cette commande va créer d'abord le répertoire crypto-app qui contiendra notre application, puis change le répertoire actuel en répertoire crypto-app afin que nous soyons à l'intérieur.

Puis exécutez :

```
touch app.js
```

Et pour finir :

```
npm init –yes
```

Place au code

Dans le fichier app.js inscrivez :

```
console.log('Welcome to the cryptoApp!!!')
```

Pour démarrer notre application, exécutez le code suivant dans la console (assurez-vous que votre répertoire de travail actuel est le répertoire crypto-app).

```
node app
```

Dans app.js, nous chargerions le module https dans notre application que nous utiliserions pour faire des requêtes à l'API cryptoCompare.
Le module https est un module intégré, nous n'aurions donc pas besoin de l'installer :

```
const https = require('https')
```

Nous allons définir nos options dans le fichier app.js et nous utiliserons un objet. Mais d'abord, nous allons assigner à une variable les symboles de crypto-monnaie par défaut que nous allons convertir :

```
const cryptoSymbols = 'BTC,ETH,XRP,LTC'
```

Nous mettrons également les devises par défaut dans lesquelles nous voulons nos conversions dans une variable comme celle-ci :

```
const currencies = 'USD,EUR,GBP,NGN'
```

Maintenant, notre api :

```
const options = {
  hostname: 'min-api.cryptocompare.com',
  path: `/data/pricemulti?fsyms=${cryptoSymbols}&tsyms=
${currencies}`
}
```

https://min-api.cryptocompare.com/data/pricemulti?fsyms=BTC,ETH,XRP,LTC&tsyms=USD,EUR,GBP,NGN

Remarque : Le nom d'hôte n'inclut pas le protocole (c'est-à-dire http ou https). Cependant, si vous transmettez une chaîne au lieu d'un objet aux options, vous devrez alors inclure le protocole comme suit :

const options = ‘https://min-api.cryptocompare.com/data/pricemulti?fsyms=${ cryptoSymbols }&tsyms=${ currencies }‘

La méthode https.get prend en compte deux paramètres :

- Options — un object, un string ou une URL.
- Callback function(optional) — écoute et gère l'événement de réponse.

Ensuite, toujours dans app.js, nous allons écrire la fonction pour la requête GET. Dans votre code, ajoutez ce qui suit :

```
https.get(options, (res) => {
   let body = '';
   res.setEncoding('utf8');
   res.on('data', (data) => {
      body += data;
   });
})
```

Dans la première ligne, nous appelons la méthode https.get et transmettons l'objet options que nous avons créé précédemment ainsi qu'une fonction de rappel en tant que paramètres. Notre rappel prend en compte le ou les objets de réponse qui sont créés une fois que les en-têtes de réponse ont été reçus.

Nous créons une variable (body) qui est initialement vide dans le but de stocker le flux de données entrant du serveur.
Nous avons également défini le codage de l'objet de réponse sur utf-8 afin qu'il n'y ait aucun problème lors du décodage des données du flux.

Ensuite, nous ajoutons un écouteur à l'objet de réponse pour l'événement "data" (l'événement "data" se produit chaque fois que nous recevons un bloc de données du flux de données).
Cet écouteur gérera les données de l'objet de réponse en ajoutant les données entrantes à ce qui est déjà disponible dans la variable de corps.

Nous aurons besoin d'ajouter un autre écouteur, cette fois, pour l'événement "end".
Cet écouteur d'événement sera déclenché lorsque nous aurons reçu tous les en-têtes de réponse et les blocs de données.
La fonction de rappel correspondante convertirait nos données d'une chaîne (stockée dans la variable body) en JSON, puis afficherait les données reçues.

Ajoutez ce code ci-dessous juste en dessous de l'écouteur "data" :

```
res.on('end', () => {
   try {
      const output = JSON.parse(body);
      console.log('Current cryptocurrency exchange rates');
      console.log(output);
   }
   catch(err) {
      console.log(`Error parsing JSON from
server:${err.message}`);
   }
});
```

Notez que nous avons utilisé un bloc try/catch pour nous permettre d'intercepter et de gérer les erreurs d'analyse des données JSON.
En cas de succès, le JSON est stocké dans la variable de sortie, sinon, s'il y a une erreur lors de l'analyse du JSON, nous enregistrons simplement le message d'erreur.

Enfin, nous ajouterons un écouteur pour l'événement 'error' à la méthode https.get. Cet écouteur gérera les erreurs pouvant résulter de la résolution DNS, de l'analyse HTTP ou des échecs de connexion.

Formater l'application

Notre application n'a pas l'air si mal, mais nous pouvons faire mieux. Formatons notre application de manière à avoir chaque crypto-monnaie sur sa propre ligne et les valeurs d'échange correspondantes en tant qu'objet en dessous.
Pour y parvenir, nous aurions besoin d'itérer sur l'objet de sortie, mais nous le ferons indirectement.
Tout d'abord, nous aurons besoin de convertir la chaîne cryptoSymbols (qui contient toutes nos crypto-monnaies) en un tableau afin que nous puissons parcourir celle-ci, puis l'utiliser pour parcourir l'objet de sortie.

Dans app.js, localisez le bloc try et remplacez :

```
console.log(output);
```

par :

```
const symbols = cryptoSymbols.split(',');
symbols.map((symbol) => {
   console.log(`${symbol}`);
   console.log(output[`${symbol}`]);
   console.log('**********************************************
   ******');
});
```

Tout d'abord, nous convertissons la variable cryptoSymbols d'une chaîne en un tableau à l'aide de la méthode split.
Nous utilisons ensuite la méthode map pour boucler sur chaque crypto-monnaie dans le tableau que nous avons créé, en imprimant le nom de chacun, ses taux de change correspondants à partir de la sortie, puis une ligne fantaisie.

```
Current cryptocurrency exchange rates
BTC
{ USD: 21196.13, EUR: 21345.9, GBP: 18700.59, NGN: 17999555 }
**********************************************************
ETH
{ USD: 1609.82, EUR: 1621.88, GBP: 1419.48, NGN: 1367066.2 }
**********************************************************
XRP
{ USD: 0.4841, EUR: 0.4874, GBP: 0.4271, NGN: 411.11 }
**********************************************************
LTC
{ USD: 70.02, EUR: 70.5, GBP: 61.79, NGN: 59452.53 }
**********************************************************
```

Accepter l'entrée de l'utilisateur (Yargs)

Pour rendre notre application plus flexible, nous laisserons les utilisateurs choisir les crypto-monnaies et les devises qu'ils souhaitent.
L'utilisateur pourra les saisir sur la ligne de commande grâce au package appelé yargs que nous installerions dans notre application.

Pour installer yargs, dans la console, exécutez :

```
npm install --save yargs
```

Nous utilisons l'indicateur de sauvegarde afin que le package soit stocké comme l'une des dépendances de l'application dans notre package.json.
Si notre application est téléchargée par un utilisateur, il devra exécuter npm install et toutes les dépendances du package.json seront automatiquement installées sur sa machine.

Au début du fichier app.js, inscrivez :

```
const argv = require('yargs').argv
```

L'utilisateur pourra transmettre deux arguments qui sont des cryptoSymbols et des devises dns la ligne de commande.
Le module yargs analysera ces arguments et les rendra disponibles dans notre application.
Nous allons maintenant modifier nos variables cryptoSymbols et devises dans app.js pour qu'elles ressemblent à ceci :

```
const cryptoSymbols = argv.cryptoSymbols || 'BTC,ETH,XRP,LTC';
const currencies = argv.currencies || 'USD,EUR,GBP,NGN';
```

argv.cryptoSymbols et argv.currencies font référence aux arguments saisis par l'utilisateur sur la ligne de commande.
Nous vérifions simplement si l'utilisateur a passé des arguments via la ligne de commande. Si les arguments existent, les variables reçoivent les valeurs, sinon, les valeurs par défaut sont utilisées.

Étant donné que l'utilisateur peut transmettre ses propres valeurs, une erreur peut survenir si les arguments ne sont pas valides. Le serveur renverrait une erreur si le cryptoSymbols ou la currencies transmise n'est pas valide. Dans ce cas, l'objet de réponse ne renverra aucune valeur d'échange mais ressemblera à ceci :

```
{ Response: 'Error',
Message: 'There is no data for any of the toSymbols USDy .',
Type: 1,
Aggregated: false,
Data: [],Warning: 'There is no data for the toSymbol/s USDy ',
HasWarning: true }
```

Une fois que nous avons notre réponse du serveur, nous vérifions si la réponse contient un "Message" comme ci-dessus. Si c'est le cas, nous savons qu'il y a une erreur et nous afficherons simplement la 'Réponse' et l'erreur 'Message' :

```
console.log('Current cryptocurrency exchange rates');
if (output.Message)
   return console.log(`${output.Response}:${output.Message}`);
```

Exécutons notre application :

```
node app --cryptoSymbols=BTC --currencies=EUR
```

```
BTC
{ EUR: 21387.5 }
************************************************************
```

Ou :

```
node app --currencies=EUR,USD --cryptoSymbols=BTC,LTC
```

```
BTC
{ EUR: 21386.9, USD: 21185.74 }
************************************************************
LTC
{ EUR: 70.46, USD: 69.85 }
************************************************************
```

Ajoutons de la couleur !

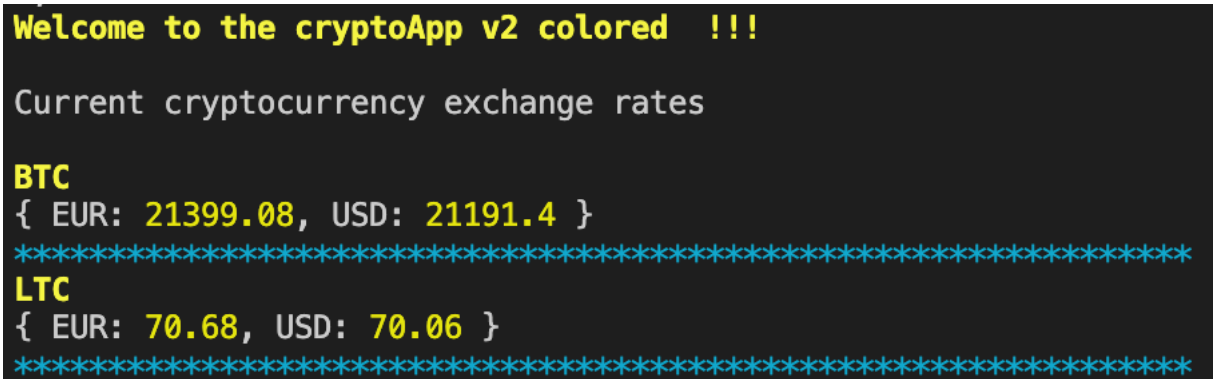
```
npm install colors-cli --save
```

```
const color = require('colors-cli/safe');
```

Ensuite, nous créons des variables correspondant aux couleurs que nous allons utiliser :

```
// color for error messages
const errorColor = color.red.bold;

// color for the welcome message and cryptoSymbols
const appColor = color.yellow.bold;

// color for our fancy line
const lineColor = color.cyan;
```

```
// -- old code
// ++ new code

-- console.log('Welcome to the cryptoApp!!!')
++ console.log(appColor('Welcome to the cryptoApp!!!'));

-- console.log(`${symbol}`);
++ console.log(appColor(`${symbol}`));

-- console.log('**********************************************
***');
++ console.log(lineColor('**********************************
***'));

-- console.log(`Request failed, ${err.message}`);
++ console.log(errorColor(`Request failed, ${err.message}`));

-- console.log(`Error parsing JSON from CoinAPI:${err}`);
++ console.log(errorColor(`Error parsing JSON from
CoinAPI:${err}`));
```


Welcome to the cryptoApp v2 colored !!!
Current cryptocurrency exchange rates
BTC
{ EUR: 21399.08, USD: 21191.4 }
****
LTC
{ EUR: 70.68, USD: 70.06 }
****