==========Partie 3==========

Module URL Node.js

1) Le module d'URL intégré

Le module URL divise une adresse Web en parties lisibles.

Pour inclure le module URL, utilisez la **`require()`** méthode :

```
var url = require('url');
```

Analysez une adresse avec la `url.parse()` méthode et elle renverra un objet URL avec chaque partie de l'adresse en tant que propriétés.

Le code **`url.parse(adr, true)`** est utilisé pour analyser une URL et en extraire les différentes parties qui la composent, telles que le protocole, l'hôte, le chemin, les paramètres de requête, etc.

Voici une explication détaillée de chaque paramètre de la méthode **`url.parse()`** :

- **`adr`** : Il s'agit de l'URL que nous souhaitons analyser.
- **`true`** : Ce deuxième paramètre est un booléen qui indique si les paramètres de la requête doivent être analysés en tant qu'objet (**`true`**) ou en tant que chaîne de requête (**`false`**).

La méthode **`url.parse()`** renvoie un objet qui contient les différentes parties de l'URL analysée. Voici les principales propriétés de l'objet renvoyé :

- **`protocol`** : Le protocole utilisé (http, https, ftp, etc.)
- **`host`** : L'hôte (nom de domaine ou adresse IP) de l'URL.
- **`pathname`** : Le chemin de l'URL.
- **`query`** : Les paramètres de la requête, soit sous forme d'une chaîne de requête (si **`true`** est **`false`**), soit sous forme d'un objet (si **`true`** est **`true`**).
- **`hash`** : le fragment d'URL, également appelé ancre ou hash, fait référence à la partie optionnelle d'une URL qui suit le symbole "#" et qui peut être utilisée pour identifier une section spécifique d'une ressource Web.

En résumé, le code **`url.parse(adr, true)`** analyse l'URL **`adr`** en tant qu'objet et renvoie un objet contenant les différentes parties de l'URL.

Exemple

Divisez une adresse Web en parties lisibles :

```
var url = require('url');
var adr = 'http://localhost:8084/default.htm?year=2017&month=february#section2';
var q = url.parse(adr, true);

console.log(q.protocol); //returns 'http:'
console.log(q.host); //returns 'localhost:8084'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
console.log(qdata.year); //returns '2017'

console.log(q.hash); //returns '#section2:'
```

2) Serveur de fichiers Node.js

Nous savons maintenant comment analyser la chaîne de requête et, dans le chapitre précédent, nous avons appris à faire en sorte que Node.js se comporte comme un serveur de fichiers. Combinons les deux, et servons le dossier demandé par le client.

Créez deux fichiers html et enregistrez-les dans le même dossier que vos fichiers node.js.

été.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Summer</h1>
<p>I love the sun!</p>
</body>
</html>
```

hiver.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Winter</h1>
<p>I love the snow!</p>
</body>
</html>
```

Créez un fichier **t3.js** Node.js qui ouvre le fichier demandé et renvoie le contenu au client. Si quelque chose ne va pas, lancez une erreur 404 :

t3.js :

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  //Le point (".") représente le répertoire courant,
  // donc en ajoutant le point devant le chemin relatif (pathname)
  //extrait de l'URL, cela crée un chemin de fichier relatif
  //qui commence à partir du répertoire courant.
```

```
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });

}).listen(8084);
console.log("Le serveur est en cours d'execution");
console.log("l'adresse : http://localhost:8084/summer.html ");
console.log("l'adresse : http://localhost:8084/winter.html ");
```

Pensez à initier le fichier :

Lancez t2.js :

```
C:\Users\Your Name>node t2.js
```

Si vous avez suivi les mêmes étapes sur votre ordinateur, vous devriez voir deux résultats différents lors de l'ouverture de ces deux adresses :

http://localhost:8084/summer.html

Produira ce résultat:

Summer

I love the sun!

http://localhost:8084/winter.html

Produira ce résultat:

Winter

I love the snow!

SERIE EXERCICES

I) SUR le système de fichiers Node.js :

1) Créez un fichier texte "message.txt" et écrivez-y "Bonjour Node.js !" en utilisant la méthode fs.writeFile().
2) Lisez le contenu du fichier "message.txt" à l'aide de la méthode fs.readFile() et affichez-le dans la console.
3) Créez un dossier "documents" avec la méthode fs.mkdir().
4) Copiez le fichier "message.txt" dans le dossier "documents" avec la méthode fs.copyFile().
5) Supprimez le fichier "message.txt" avec la méthode fs.unlink().
6) Renommez le fichier "documents/message.txt" en "documents/nouveau-message.txt" avec la méthode fs.rename().
7) Affichez la liste des fichiers et dossiers présents dans le dossier "documents" avec la méthode fs.readdir().
8) Créez un fichier "data.json" contenant un objet avec des propriétés et des valeurs de votre choix avec la méthode fs.writeFile().
9) Lisez le contenu du fichier "data.json" à l'aide de la méthode fs.readFile() et affichez-le dans la console.
10) Modifiez une propriété de l'objet contenu dans le fichier "data.json" avec la méthode fs.readFile(), puis écrivez les modifications dans le fichier avec la méthode fs.writeFile().

II) SUR 'URL Node.js :

1) Utiliser le module 'url' pour extraire les paramètres de requête d'une URL donnée .
2) Utiliser le module 'url' pour extraire le chemin d'une URL donnée.
3) Utiliser le module 'url' pour résoudre une URL relative en une URL absolue.
4) Utiliser le module 'url' pour parser une URL donnée en ses différentes parties (protocole, hôte, chemin, etc.).
5) Utiliser le module 'url' pour générer une URL à partir de ses différentes parties (protocole, hôte, chemin, etc.).
6) Utiliser le module 'querystring' pour créer une chaîne de requête à partir d'un objet JavaScript.
7) Utiliser le module 'querystring' pour parser une chaîne de requête en un objet JavaScript.
8) Utiliser le module 'url' pour ajouter des paramètres de requête à une URL existante.
9) Utiliser le module 'url' pour extraire le nom de domaine d'une URL donnée.
10) Utiliser le module 'url' pour vérifier si une URL est absolue ou relative.

==========Partie 4==========

NPM Node.js

1) Qu'est-ce que le NPM ?

NPM est un gestionnaire de packages pour les packages Node.js ou les modules si vous le souhaitez.

www.npmjs.com héberge des milliers de packages gratuits à télécharger et à utiliser.

Le programme NPM est installé sur votre ordinateur lorsque vous installez Node.js

NPM est déjà prêt à fonctionner sur votre ordinateur !

2) Qu'est-ce qu'un forfait ?

Un package dans Node.js contient tous les fichiers dont vous avez besoin pour un module.

Les modules sont des bibliothèques JavaScript que vous pouvez inclure dans votre projet.

3) Télécharger un package

Le téléchargement d'un package est très simple.
Ouvrez l'interface de ligne de commande et dites à NPM de télécharger le package souhaité.
Je souhaite télécharger un package appelé "majuscule":

Télécharger "majuscule":

```
C:\Users\Your Name>npm install upper-case
```

Vous avez maintenant téléchargé et installé votre premier package !

NPM crée un dossier nommé **"node_modules"**, où le package sera placé. Tous les packages que vous installerez à l'avenir seront placés dans ce dossier.

Mon projet a maintenant une structure de dossiers comme celle-ci :

```
C:\Users\My Name\node_modules\upper-case
```

3) Utilisation d'un package

Une fois le package installé, il est prêt à être utilisé.

Incluez le package "majuscules" de la même manière que vous incluez n'importe quel autre module :

```
var uc = require('upper-case');
```

Créez un fichier Node.js qui convertira la sortie "Hello World!" en majuscules :

Exemple

```
var http = require('http');
var uc = require('upper-case');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc.upperCase("hello horld! Node js et javascript"));
  res.end();
}).listen(8084);
```

Enregistrez le code ci-dessus dans un fichier appelé "t1.js", et lancez le fichier :

Lancez t1:

```
C:\Users\Your Name>node t1
```

Si vous avez suivi les mêmes étapes sur votre ordinateur, vous verrez le même résultat que l'exemple : http://localhost:8084

==========Partie 5==========

Événements Node.js

Node.js est parfait pour les applications événementielles.

1) Événements dans Node.js

Chaque action sur un ordinateur est un événement. Comme lorsqu'une connexion est établie ou qu'un fichier est ouvert.

1) L'objet "**readStream**" : Cet objet est utilisé pour **lire** des fichiers dans Node.js. Lorsqu'un fichier est lu à l'aide de cet objet, plusieurs événements sont déclenchés, notamment "**open**", "**data**", "end" et "**error**". L'événement "**open**" est déclenché lorsque le fichier est ouvert et prêt à être lu. L'événement "**data**" est déclenché chaque fois qu'une partie du fichier est lue. L'événement "**end**" est déclenché lorsque la fin du fichier est atteinte. Enfin, l'événement "**error**" est déclenché si une erreur se produit pendant la lecture du fichier.

Exemple 1

```
const fs = require('fs');
const readStream = fs.createReadStream('monfichier.txt');

readStream.on('open', function () {
  console.log('File is open');
});

readStream.on('data', (chunk) => {
  console.log(`Chunk reçu: ${chunk}`);
});

readStream.on('end', () => {
  console.log('Lecture du fichier terminée.');
});

readStream.on('error', (err) => {
  console.error(`Erreur de lecture du fichier: ${err}`);
});
```

Exemple 2

```
const fs = require('fs');
const readStream = fs.createReadStream('monfichier.bin');

readStream.on('data', (chunk) => {
  console.log(`Chunk reçu: ${chunk.length} octets`);
});

readStream.on('end', () => {
  console.log('Lecture du fichier terminée.');
});

readStream.on('error', (err) => {
  console.error(`Erreur de lecture du fichier: ${err}`);
});
```

1) L'objet "**http**" : Cet objet est utilisé pour créer un serveur HTTP dans Node.js. Lorsqu'un client envoie une demande à ce serveur, plusieurs événements sont déclenchés, notamment "**request**", "**connection**", "**close**" et "**error**". L'événement "**request**" est déclenché chaque fois qu'une demande est reçue. L'événement "**connection**" est déclenché chaque fois qu'un client se connecte au serveur. L'événement "**close**" est déclenché lorsque la connexion est fermée. Enfin, l'événement "**error**" est déclenché si une erreur se produit pendant le traitement de la demande.

Exemple 1

```
const http = require('http');

const server = http.createServer(function (req, res) {
  console.log('Request received');
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.write('Hello World!');
  res.end();
});

server.on('connection', function () {
  console.log('Client connected');
});

server.on('close', function () {
  console.log('Server closed');
});

server.on('error', function (err) {
  console.log('Error occurred: ' + err.message);
});

server.listen(8084, function () {
  console.log('Server started on port 8084');
});
```

```
console.log('Adresse: http://localhost:8084/');


setTimeout(function() {
    server.close();
  }, 10000);
```

1) Le code commence par importer le module **http**.
2) Ensuite, un serveur est créé en appelant la méthode **createServer()** du module **http**. Cette méthode prend en paramètre une fonction de rappel qui sera appelée chaque fois qu'une requête HTTP est **reçue**. Cette fonction de rappel prend deux arguments : l'objet **req** (la demande HTTP) et l'objet **res** (la réponse HTTP).

3) Dans cette fonction de rappel, on affiche un message "Request received" sur la console. Ensuite, on configure la réponse HTTP en utilisant la méthode writeHead() pour définir le code d'état de la réponse à 200 (OK) et le type de contenu de la réponse à "text/plain". Ensuite, on écrit le message "Hello World!" dans la réponse en utilisant la méthode write(). Enfin, on termine la réponse en appelant la méthode end().

4) Le serveur écoute également les événements connection, close et error. L'événement connection est déclenché chaque fois qu'un client se connecte au serveur, l'événement close est déclenché lorsque le serveur est fermé, et l'événement error est déclenché lorsque le serveur rencontre une erreur.

5) Enfin, le serveur est démarré en appelant la méthode listen() du serveur en spécifiant le port sur lequel il doit écouter les demandes entrantes. Une fois que le serveur est en cours d'exécution, un message "Server started on port 8084" est affiché sur la console.

Lorsque vous exécutez ce code, un serveur HTTP est créé et écoutera les requêtes entrantes sur le port 8084. Lorsqu'une demande est reçue, le serveur renverra une réponse avec le message "Hello World!". Vous pouvez accéder au serveur en ouvrant un navigateur et en accédant à l'adresse "http://localhost:8084/". Le message "Adresse: http://localhost:8084/" est également affiché sur la console.

Pour voir l'exécution de l'événement close, vous pouvez arrêter le serveur en appelant la méthode close() du serveur. Cela déclenchera l'événement close. Par exemple, vous pouvez ajouter le code suivant à la fin du script pour fermer le serveur après 10 secondes

Lorsque le serveur est fermé, le message "Server closed" sera affiché sur la console.

2) Module Événements

Node.js est une plate-forme de développement JavaScript qui utilise le modèle de programmation asynchrone, ce qui signifie que de nombreux événements peuvent se produire en même temps. La gestion des événements dans Node.js est basée sur la bibliothèque **EventEmitter**, qui permet de déclencher et de gérer des événements.

Voici les étapes de base pour travailler avec des événements dans Node.js :

1) Importez le module **events** : Pour commencer à travailler avec des événements, vous devez importer le module **events** dans votre code.

```
const EventEmitter = require('events');
```

2) Créez un objet **EventEmitter** : Créez un objet **EventEmitter** à partir de la classe **EventEmitter**

```
const myEmitter = new EventEmitter();
```

3) Définissez des événements : Vous pouvez définir des événements en utilisant la méthode **on** de l'objet **EventEmitter**. Vous pouvez également utiliser la méthode **once** pour définir un événement qui ne sera déclenché qu'une seule fois.

```
myEmitter.on('event', () => {
    console.log('Un événement a été déclenché !');
  });

myEmitter.once('eventOnce', () => {
    console.log('Cet événement ne sera déclenché qu\'une seule fois.');
  });
```

4) Déclenchez des événements : Vous pouvez déclencher des événements en utilisant la méthode **emit** de l'objet **EventEmitter**. Vous pouvez également passer des arguments à la méthode **emit** qui seront transmis aux écouteurs d'événements.

```
myEmitter.emit('event');
myEmitter.emit('eventOnce');

// Vous pouvez également transmettre des arguments
// aux écouteurs d'événements.
myEmitter.emit('eventWithArgs', 'arg1', 'arg2');
```

5) Supprimez des écouteurs d'événements : Vous pouvez supprimer des écouteurs d'événements en utilisant la méthode **removeListener** ou la méthode **removeAllListeners** de l'objet EventEmitter.

```
a) myEmitter.removeListener('event', listener);
b) myEmitter.removeAllListeners('event');
```

a) Méthode myEmitter.removeListener('event', listener):

La méthode **removeListener** est utilisée pour supprimer un **écouteur d'événement** spécifique attaché à un **émetteur d'événement**.

Voici un exemple où un **écouteur** d'événement nommé "**messageHandler**" est attaché à un **émetteur** nommé "**myEmitter**", puis supprimé à l'aide de la méthode **removeListener** :

```
const EventEmitter = require('events');

// Créer un nouvel émetteur d'événements
const myEmitter = new EventEmitter();

// Définir un écouteur d'événement nommé "messageHandler"
const messageHandler = () => {
  console.log('Un message a été reçu');
};
myEmitter.on('message', messageHandler);

// Supprimer l'écouteur d'événement "messageHandler"
myEmitter.removeListener('message', messageHandler);
```

Dans cet exemple, la méthode **removeListener** est utilisée pour supprimer l'écouteur d'événement "**messageHandler**" de l'événement "message" attaché à l'émetteur "**myEmitter**".

b) Méthode myEmitter.removeAllListeners('event'):

La méthode **removeAllListeners** est utilisée pour supprimer tous les **écouteurs** d'un événement spécifique ou de tous les événements attachés à un émetteur d'événement.

Voici un exemple où plusieurs **écouteurs** d'événements sont attachés à un émetteur nommé "**myEmitter**", puis tous supprimés à l'aide de la méthode **removeAllListeners** :

```
const EventEmitter = require('events');

// Créer un nouvel émetteur d'événements
const myEmitter = new EventEmitter();

// Définir deux écouteurs d'événement nommés
//"messageHandler" et "errorHandler"

const messageHandler = () => {
  console.log('Un message a été reçu');
};
myEmitter.on('message', messageHandler);
```

```
const errorHandler = () => {
  console.log('Une erreur s\'est produite');
};
myEmitter.on('error', errorHandler);

// Supprimer tous les écouteurs d'événement
// attachés à l'émetteur "myEmitter"
myEmitter.removeAllListeners();
```

La gestion des événements est une technique importante en Node.js, car elle permet de créer des applications performantes et évolutives. Elle est particulièrement utile pour les applications qui doivent gérer de nombreux événements en même temps, comme les serveurs Web ou les applications en temps réel.

---

3) EXEMPLES

Exemple 1 :

```
var events = require('events');
var eventEmitter = new events.EventEmitter(); // syntaxe CommonJS

//Créez un gestionnaire d'événements :
var myEventHandler = function () {
  console.log('I hear a scream!');
}

//Affectez le gestionnaire d'événements à un événement :
eventEmitter.on('scream', myEventHandler);

//Lancer l'événement 'scream' :
eventEmitter.emit('scream');
```

Dans l'exemple ci-dessous, nous avons créé une fonction qui sera exécutée lorsqu'un événement "cri" est déclenché.
Pour déclencher un événement, utilisez la `emit()` méthode.

Exemple 2 :

```
var events = require("events");
var eventEmitter = new events.EventEmitter();


//Créez un gestionnaire d'événements :
var myEventHandler = function (nb) {
  console.log("I hear a scream!", nb, "fois");
};

//Affectez le gestionnaire d'événements à un événement :
eventEmitter.on("scream", myEventHandler);

//Lancer l'événement 'scream' :
eventEmitter.emit("scream", 5);
```

Exemple 3 :

```
const EventEmitter = require('events');
const myEmitter = new EventEmitter(); // syntaxe ES6

// Enregistrement d'un gestionnaire d'événements pour l'événement 'myEvent'
myEmitter.on('myEvent', (data) => {
  console.log('L\'événement "myEvent" a été émis avec les données suivantes :',
data);
});

// Émission de l'événement 'myEvent'
myEmitter.emit('myEvent', {message: 'Ceci est un message'});
```

Lorsque cet exemple est exécuté, la fonction de gestion d'événements enregistrée est appelée et affiche le message "L'événement "myEvent" a été émis avec les données suivantes : { message: 'Ceci est un message' }" dans la console.

Exemple 4 :

```
// importe le module 'events' et assigne
// la classe EventEmitter à la constante EventEmitter
const EventEmitter = require('events');


//définit une nouvelle classe appelée MonEvenement qui hérite
// de la classe EventEmitter en utilisant l'héritage
//la classe MonEvenement aura toutes les méthodes et propriétés
// de la classe EventEmitter.
class MonEvenement extends EventEmitter {}

//crée une instance de la classe MonEvenement
// en appelant son constructeur
const monInstance = new MonEvenement();

// Abonnement à l'événement 'maNotification'
monInstance.on('maNotification', () => {
  console.log('La notification a été reçue.');
});

// Émission de l'événement 'maNotification'
monInstance.emit('maNotification');
```

SERIE EXERCICES

1) Créer un programme Node.js qui utilise le module 'events' pour créer et gérer des événements personnalisés.

2) Créer un programme Node.js qui utilise le module 'events' pour créer et gérer des événements personnalisés avec des **arguments**.

3) Créer un programme Node.js qui utilise le module 'events' pour créer et gérer plusieurs événements personnalisés avec des arguments et un événement personnalisé pour signaler la fin de l'exécution.

4) Créer un programme Node.js qui lit le contenu d'un fichier texte, en supprime toutes les voyelles, puis écrit le résultat dans un nouveau fichier texte.

5) Créez un **EventEmitter** qui écoute une variable de comptage et qui émet un événement chaque fois que le compteur atteint un multiple de 10. Ajoutez un écouteur d'événements qui se déclenche chaque fois qu'un multiple de 10 est atteint et qui affiche le nombre de fois que le compteur a été atteint.

6) Créez un **EventEmitter** qui écoute une file d'attente de messages et qui émet un événement chaque fois qu'un nouveau message est ajouté à la file d'attente. Ajoutez un écouteur d'événements qui se déclenche chaque fois qu'un nouveau message est ajouté à la file d'attente et qui affiche le nombre total de messages dans la file d'attente.

CORRECTION SERIE EXERCICES

1) Créer un programme Node.js qui utilise le module 'events' pour créer et gérer des événements personnalisés.

Instructions :

1) Créez un fichier JavaScript et nommez-le 'eventEmitter.js'.
2) Importez le module 'events' en utilisant la syntaxe require.
3) Créez un objet EventEmitter en instanciant la classe EventEmitter.
4) Ajoutez un gestionnaire d'événements pour l'événement 'message' qui imprime le message 'Un message a été reçu'.
5) Déclenchez l'événement 'message' en utilisant la méthode emit().
6) Exécutez le programme en utilisant Node.js.

Solution avec détails:

1) Créez un nouveau fichier JavaScript et nommez-le 'eventEmitter.js'.
2) Importez le module 'events' en utilisant la syntaxe require:

```
const EventEmitter = require('events');
```

3) Créez un objet EventEmitter en instanciant la classe EventEmitter:

```
const myEmitter = new EventEmitter();
```

4) Ajoutez un gestionnaire d'événements pour l'événement 'message' qui imprime le message 'Un message a été reçu':

```
myEmitter.on('message', () => {
    console.log('Un message a été reçu');
  });
```

5) Déclenchez l'événement 'message' en utilisant la méthode emit():

```
  myEmitter.emit('message');
```

6) Exécutez le programme en utilisant Node.js en exécutant la commande suivante dans le terminal:

```
node eventEmitter.js
```

code complet:

```
const EventEmitter = require('events');
const myEmitter = new EventEmitter();

myEmitter.on('message', () => {
  console.log('Un message a été reçu');
});

myEmitter.emit('message');
```

2) Créer un programme Node.js qui utilise le module 'events' pour créer et gérer des événements personnalisés avec des **arguments**.

Instructions :

1) Créez un fichier JavaScript et nommez-le 'eventEmitterArgs.js'.
2) Importez le module 'events' en utilisant la syntaxe require.
3) Créez un objet EventEmitter en instanciant la classe EventEmitter.
4) Ajoutez un gestionnaire d'événements pour l'événement 'message' qui prend un argument et l'imprime.
5) Déclenchez l'événement 'message' en utilisant la méthode emit() avec un argument.
6) Exécutez le programme en utilisant Node.js.

Solution avec détails :

1) Créez un nouveau fichier JavaScript et nommez-le **'eventEmitterArgs.js'**.
2) Importez le module 'events' en utilisant la syntaxe require:

```
const EventEmitter = require('events');
```

3) Créez un objet EventEmitter en instanciant la classe EventEmitter:

```
const myEmitter = new EventEmitter();
```

4) Ajoutez un gestionnaire d'événements pour l'événement 'message' qui prend un argument et l'imprime :

```
myEmitter.on('message', (arg) => {
    console.log('Le message est : ', arg);
  });
```

5) Déclenchez l'événement 'message' en utilisant la méthode emit() avec un argument:

```
 myEmitter.emit('message', 'Bonjour tout le monde !');
```

6) Exécutez le programme en utilisant Node.js en exécutant la commande suivante dans le terminal :

```
node eventEmitterArgs.js
```

code complet:

```
const EventEmitter = require('events');
const myEmitter = new EventEmitter();

myEmitter.on('message', (arg) => {
  console.log('Le message est : ', arg);
});

myEmitter.emit('message', 'Bonjour tout le monde !');
```

3) Créer un programme Node.js qui utilise le module 'events' pour créer et gérer plusieurs événements personnalisés avec des arguments et un événement personnalisé pour signaler la fin de l'exécution.

Instructions :

1) Créez un fichier JavaScript et nommez-le 'eventEmitterMulti.js'.
2) Importez le module 'events' en utilisant la syntaxe require.
3) Créez un objet EventEmitter en instanciant la classe EventEmitter.
4) Ajoutez plusieurs gestionnaires d'événements pour différents événements qui prennent des arguments et les impriment.
5) Ajoutez un événement personnalisé qui signalera la fin de l'exécution.
6) Déclenchez les différents événements avec des arguments.
7) Déclenchez l'événement de fin d'exécution.
8) Exécutez le programme en utilisant Node.js.

Solution avec détails :

1) Créez un nouveau fichier JavaScript et nommez-le 'eventEmitterMulti.js'.
2) Importez le module 'events' en utilisant la syntaxe require :

```
const EventEmitter = require('events');
```

3) Créez un objet EventEmitter en instanciant la classe EventEmitter :

```
const myEmitter = new EventEmitter();
```

4) Ajoutez plusieurs gestionnaires d'événements pour différents événements qui prennent des arguments et les impriment :

```
myEmitter.on('event1', (arg1, arg2) => {
    console.log('Le premier argument est :', arg1, 'Le deuxième argument est :',
arg2);
  });

myEmitter.on('event2', (arg1, arg2) => {
    console.log('Le premier argument est :', arg1, 'Le deuxième argument est :',
arg2);
  });

myEmitter.on('event3', (arg1, arg2) => {
    console.log('Le premier argument est :', arg1, 'Le deuxième argument est :',
arg2);
  });
```

5) Ajoutez un événement personnalisé qui signalera la fin de l'exécution :

```
myEmitter.on('end', () => {
    console.log('Fin d\'exécution');
  });
```

6) Déclenchez les différents événements avec des arguments :

```
myEmitter.emit('event1', 'Hello', 'world');
myEmitter.emit('event2', 'Bonjour', 'tout le monde');
myEmitter.emit('event3', 'Hola', 'mundo');
```

7) Déclenchez l'événement de fin d'exécution :

```
myEmitter.emit('end');
```

8) Exécutez le programme en utilisant Node.js en exécutant la commande suivante dans le terminal :

```
node eventEmitterMulti.js
```

code complet:

```
const EventEmitter = require('events');
const myEmitter = new EventEmitter();


myEmitter.on('event1', (arg1, arg2) => {
    console.log('Le premier argument est :', arg1, 'Le deuxième argument est :',
arg2);
  });


myEmitter.on('event2', (arg1, arg2) => {
    console.log('Le premier argument est :', arg1, 'Le deuxième argument est :',
arg2);
  });

myEmitter.on('event3', (arg1, arg2) => {
    console.log('Le premier argument est :', arg1, 'Le deuxième argument est :',
arg2);
  });

myEmitter.on('end', () => {
    console.log('Fin d\'exécution');
  });

myEmitter.emit('event1', 'Hello', 'world');
myEmitter.emit('event2', 'Bonjour', 'tout le monde');
myEmitter.emit('event3', 'Hola', 'mundo');

myEmitter.emit('end');
```

4) créer un programme Node.js qui lit le contenu d'un fichier texte, en supprime toutes les voyelles, puis écrit le résultat dans un nouveau fichier texte.

Instructions :

1) Créez un fichier JavaScript et nommez-le 'manipulationFichier.js'.
2) Importez le module 'fs' en utilisant la syntaxe require.
3) Utilisez la méthode 'readFile' pour lire le contenu du fichier texte à manipuler.
4) Supprimez toutes les voyelles du contenu du fichier en utilisant une expression régulière.
5) Utilisez la méthode 'writeFile' pour écrire le nouveau contenu dans un nouveau fichier texte.
6) Exécutez le programme en utilisant Node.js.

Solution avec détails :

1) Créez un nouveau fichier JavaScript et nommez-le 'manipulationFichier.js'.
2) Importez le module 'fs' en utilisant la syntaxe require :

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

3) Utilisez la méthode 'readFile' pour lire le contenu du fichier texte à manipuler :

```
fs.readFile('fichier_a_manipuler.txt', 'utf8', (err, data) => {
    if (err) throw err;
    console.log('Contenu du fichier avant manipulation :', data);
  });
```

4) Supprimez toutes les voyelles du contenu du fichier en utilisant une expression régulière :

```
const contenuSansVoyelles = data.replace(/[aeiou]/gi, '');
```

5)Utilisez la méthode 'writeFile' pour écrire le nouveau contenu dans un nouveau fichier texte :

```
fs.writeFile('fichier_manipule.txt', contenuSansVoyelles, (err) => {
    if (err) throw err;
    console.log('Le fichier a été manipulé et enregistré avec succès !');
  });
```

6) Exécutez le programme en utilisant Node.js en exécutant la commande suivante dans le terminal :

```
node manipulationFichier.js
```

Le programme devrait lire le contenu du fichier texte à manipuler, supprimer toutes les voyelles, puis écrire le nouveau contenu dans un nouveau fichier texte.

code complet:

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

fs.readFile('fichier_a_manipuler.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('Contenu du fichier avant manipulation :', data);

  const contenuSansVoyelles = data.replace(/[aeiou]/gi, '');

  fs.writeFile('fichier_manipule.txt', contenuSansVoyelles, (err) => {
    if (err) throw err;
    console.log('Le fichier a été manipulé et enregistré avec succès !');
  });
});
```

5) Créez un **EventEmitter** qui écoute une variable de comptage et qui émet un événement chaque fois que le compteur atteint un multiple de 10. Ajoutez un écouteur d'événements qui se déclenche chaque fois qu'un multiple de 10 est atteint et qui affiche le nombre de fois que le compteur a été atteint.

```
const EventEmitter = require('events');
class Counter extends EventEmitter {
  constructor() {
    super();
    this.count = 0;
  }
  increment() {
    this.count++;
    if (this.count % 10 === 0) {
      this.emit('multiple_of_10', this.count);
    }
  }
}
const counter = new Counter();

counter.on('multiple_of_10', (count) => {
  console.log(`Le compteur a atteint un multiple de 10 (${count} fois)`);
});

for (let i = 0; i < 100; i++) {
  counter.increment();
}
```

6) Créez un EventEmitter qui écoute une file d'attente de messages et qui émet un événement chaque fois qu'un nouveau message est ajouté à la file d'attente. Ajoutez un écouteur d'événements qui se déclenche chaque fois qu'un nouveau message est ajouté à la file d'attente et qui affiche le nombre total de messages dans la file d'attente.

```
const EventEmitter = require('events');
class MessageQueue extends EventEmitter {
  constructor() {
    super();
    this.messages = [];

  }
  addMessage(message) {
    this.messages.push(message);
    this.emit('message_added', this.messages.length);
  }
}
const messageQueue = new MessageQueue();
messageQueue.on('message_added', (count) => {
  console.log(`La file d'attente contient maintenant ${count} messages`);
});
messageQueue.addMessage('Message 1');
messageQueue.addMessage('Message 2');
messageQueue.addMessage('Message 3');
```

==========Partie 6==========

Télécharger des fichiers Node.js

1) Module Formidable

Il existe un très bon module pour travailler avec les téléchargements de fichiers, appelé "**Formidable**".

Le module Formidable peut être téléchargé et installé à l'aide de NPM :

```
C:\Users\Your Name>npm install formidable
```

Après avoir téléchargé le module Formidable, vous pouvez inclure le module dans n'importe quelle application :

```
var formidable = require('formidable');
```

---

2) Télécharger des fichiers

Vous êtes maintenant prêt à créer une page Web dans Node.js qui permet à l'utilisateur de télécharger des fichiers sur votre ordinateur :

Étape 1 : Créer un formulaire de téléchargement

Créez un fichier Node.js qui écrit un formulaire HTML, avec un champ de téléchargement :

Exemple

Ce code produira un formulaire HTML :

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<form action="fileupload" method="post" enctype="multipart/form-
data">');
  res.write('<input type="file" name="filetoupload"><br>');
  res.write('<input type="submit">');
  res.write('</form>');
  return res.end();
}).listen(8084);
```

Étape 2 : Analyser le fichier téléchargé

Incluez le module Formidable pour pouvoir analyser le fichier téléchargé une fois qu'il atteint le serveur.

Lorsque le fichier est téléchargé et analysé, il est placé dans un dossier temporaire sur votre ordinateur.

Exemple

Le fichier sera téléchargé et placé dans un dossier temporaire :

```
var http = require('http');
var formidable = require('formidable');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
      var form = new formidable.IncomingForm();
      form.parse(req, function (err, fields, files) {
       res.write('File uploaded');
       res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post" enctype="multipart/form-
data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8084);
```

Étape 3 : Enregistrer le fichier

Lorsqu'un fichier est téléchargé avec succès sur le serveur, il est placé dans un dossier temporaire.

Le chemin d'accès à ce répertoire se trouve dans l'objet "files", passé en troisième argument dans la `parse()` fonction de rappel de la méthode.

Pour déplacer le fichier dans le dossier de votre choix, utilisez le module File System, et renommez le fichier :

Exemple

Incluez le module fs et déplacez le fichier dans le dossier actuel :

```
var http = require('http');
var formidable = require('formidable');
var fs = require('fs');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      var oldpath = files.filetoupload.filepath;
      var newpath = 'C:/SS/' + files.filetoupload.originalFilename;
      fs.rename(oldpath, newpath, function (err) {
        if (err) throw err;
        res.write('File uploaded and moved!');
        res.end();
      });
 });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('<form action="fileupload" method="post" enctype="multipart/form-
data">');
    res.write('<input type="file" name="filetoupload"><br>');
    res.write('<input type="submit">');
    res.write('</form>');
    return res.end();
  }
}).listen(8080);
```

Le programme utilise également les modules **formidable** et **fs**. **formidable** est un module qui permet de **traiter les données de formulaire HTML**, y compris les **fichiers téléchargés**. fs est un module qui permet de manipuler les fichiers et les répertoires.

Lorsque l'utilisateur accède à l'URL racine du serveur, le programme envoie un formulaire HTML qui permet à l'utilisateur de télécharger un fichier. Lorsque l'utilisateur **soumet** le formulaire, le programme vérifie si la requête est pour l'URL **/fileupload**. Si c'est le cas, il **utilise formidable pour analyser les données de formulaire et récupérer le fichier téléchargé**. Ensuite, il utilise **fs** pour renommer le fichier téléchargé et le déplacer vers le répertoire spécifié.

Si la requête n'est pas pour **/fileupload**, le programme envoie simplement le formulaire HTML pour télécharger un fichier.

Ce code est un exemple d'utilisation de Node.js pour créer un serveur HTTP qui permet de télécharger un fichier et de le sauvegarder sur un disque dur. Voici une explication plus détaillée de chaque partie du code :

```
var http = require('http');
//La première ligne importe le module http de Node.js qui permet de créer un
//serveur HTTP.

var formidable = require('formidable');
//La deuxième ligne importe le module formidable qui permet de traiter facilement
//les formulaires envoyés en POST.



var fs = require('fs');
//La troisième ligne importe le module fs (file system) qui permet de travailler
//avec le système de fichiers.

http.createServer(function (req, res) {
//La quatrième ligne crée un serveur HTTP en appelant la fonction createServer()
//du module http. Cette fonction prend une fonction de rappel qui est appelée à
//chaque fois qu'une requête HTTP est reçue par le serveur.

    if (req.url == '/fileupload') {
//Cette ligne vérifie si l'URL de la requête est /fileupload.
// Si c'est le cas, cela signifie que l'utilisateur
//a soumis le formulaire pour télécharger un fichier.


        var form = new formidable.IncomingForm();
// Cette ligne crée une nouvelle instance de
// la classe IncomingForm du module formidable.
// Cette classe permet d'analyser les données de formulaire HTML
// et de récupérer les fichiers téléchargés.


        form.parse(req, function (err, fields, files) {
// Cette ligne utilise la méthode parse de l'instance form
// pour analyser les données de formulaire.
// Elle prend en paramètre une fonction de rappel
//qui sera exécutée une fois que l'analyse est terminée.
//La fonction de rappel prend trois paramètres :
// - 'err' pour les erreurs éventuelles,
// - 'fields' pour les champs du formulaire
// - 'files' pour les fichiers téléchargés.


            var oldpath = files.filetoupload.filepath;
// Cette ligne récupère le chemin d'accès au fichier téléchargé
//en accédant à la propriété 'filepath'
// de l'objet 'files.filetoupload'
// '.filetoupload' est le nom du champ de formulaire
// qui contient le fichier.


      var newpath = 'C:/SS/' + files.filetoupload.originalFilename;
```

```
// Cette ligne définit le nouveau chemin d'accès
// pour le fichier téléchargé. Il s'agit du répertoire
// de destination pour le fichier.
// Le nom du fichier est récupéré à partir
// de la propriété originalFilename de l'objet  files.filetoupload.


            fs.rename(oldpath, newpath, function (err) {
//Cette ligne utilise la méthode rename du module fs
// pour renommer et déplacer le fichier téléchargé
// vers le nouveau chemin d'accès.
//Elle prend en paramètre une fonction de rappel
// qui sera exécutée une fois que l'opération est  terminée.

// Cette ligne vérifie si une erreur s'est produite
// lors du déplacement du fichier.
//  Si c'est le cas, une exception est levée.
                if (err) throw err;
```
