Node JS - Activités N02

Activité N01

Application pratique

1. Creation du fichier diagnostic.js

```
const os =require("os"); // importation du module `os` pour interagir avec l'operating system

// partie responsable pour l'affichage des infos comme :
console.log("Plateforme :", os.platform());
// la palteform de l'OS (Kernel)

console.log("Architecture :", os.arch());
// architecture de l'OS - CPU

console.log("CPU :", os.cpus().length,"coeurs");
// nombre des coeurs du processeur

console.log("Memoire Totale :", os.totalmem());
// capacite totale du RAM

console.log("Memoire libre :", os.freemem());
// capacite libre du  RAM

console.log("Uptime (en heure) :", (os.uptime()/3600).toFixed(2));
// nombe des heures du dernier demmarage
```

2. lancement du script `app.js`

```
node app.js
```

Reponses aux questions

1. *Quelle est la différence entre `os.platform()` et `os.arch()` ?*
   - `os.plateform()` : retourne la plateforme de l'operating system
   - `os.arch()` : retourne l'architecture de processeur
2. *À quoi pourrait servir cette information dans une application réelle (ex : tableau de bord système) ?*
   - Par exemple, le monitoring de la consommation des ressource sur notre serveur, gestion et automatisation des builds, interaction avec les operating systems ...

Activité N02

Application pratique

1. Création du fichier `explorateur.js`

```
const fs=require("fs"); // importation du module `fs` (filesystem)
const path=require("path"); // importation du module `path`

fs.readdir(__dirname,(err,files))=>{ //
  if(err) return console.log("Error :", err.message);
  console.log("Contenu du dossier :", files);
  files.foreach(f=>console.log(path.join(__dirname,f)));
}
```

2. Ecriture dans le fichier `log.txt`

```
let fileList = fs.readdirSync(__dirname);
fileList.forEach((file) => {
  const filePath = `${__dirname}/${file}`;
  const stats = fs.statSync(filePath);
  const creationDate = stats.birthtime;
  fs.appendFile(
    "./log.txt",
    `file name : ${file}  Created at ${creationDate}\n`,
    (err) => {
      if (err) {
        return err;
      }
    }
  );
});
```

3. lancement du script `explorateur.js`

```
node explorateur.js
```

Activité N03

Application pratique

```
// importation et creation d'une instance de la calsse `EventEmitter`
const EventEmitter = require("events");
const emitter = new EventEmitter();

// decrlaration un événement `utilisateurConnecté`
emitter.on("utilisateurConnecté", (data) => {
  console.log(`Nouvelle connexion : ${data.nom} (${data.id})`);
});

// declachement de l'evenement
emitter.emit("utilisateurConnecté", { id: 1, nom: "Asma" });
```

Reponses aux questions

1. *Que se passe-t-il si l'écouteur est enregistré après l'émission de l'événement ?*
   - L'evenement ne sera pas déclenché.
2. *Peut-on avoir plusieurs écouteurs pour un même événement ?*
   - On peut avoir plusieurs écouteurs pour un même événement.

Activité N04

Application pratique

1. Création du fichier `logger.js`

```
// importation du module events
const EventEmitter = require("events");

// Création du classe Logger qui herite de la classe EventEmitter
class Logger extends EventEmitter {
// methode pour afficher et émetter un événement
  log(message) {
    console.log("LOG :", message);
    // declachement de l'événement messageLogged avec les données
    this.emit("messageLogged", {message,date: new Date()});
  }
}

module.exports = Logger; // Export du classe Logger
```

2. Création du fichier `app.js`

```
// Importation de classe Logger du module logger.js
const Logger = require('./Logger');

// Création d'une instance de Logger
const logger = new Logger();

// Enregistrement d'un écouteur pour l'événement `messageLogged`
logger.on("messageLogged", (data) => {
    console.log("Événement capturé :", data);
});

// Appel de la méthode log qui déclenchera l'événement
logger.log("Application démarrée !");
```

3. lancement du script `app.js`

```
node app.js
```

Reponses aux questions

1. *Quelle est la différence entre une instance directe d'EventEmitter et une classe qui l'etend ?*
   - **Instance directe** : pour créer et gérer des événements simples.
   - **Classe étendue** : pour créer des évènements d'une manière libre et réutilisables qui se basent sur les instances directes définies par le module `events`.
2. *Pourquoi encapsuler la logique dans une classe ?*
   - pour organiser le code

Activité N05

Application pratique

Création du fichier `server.js`

```
const http = require("http"); // Importation du module HTTP

// Création du serveur
const server = http.createServer((req, res) => {
  if (req.url === "/") { // Route pour la page d'accueil
    res.write("Bienvenue sur notre serveur Node.js !");
    res.end();
  }

  // Route pour l'API des etudiants
  else if (req.url === "/api/etudiants") {
        res.writeHead(200, { "Content-Type": "application/json" });
        res.end(JSON.stringify(["Asma", "Youness", "Oussama"]));
  }

  // Route pour les pages non trouvees
  else {
    res.writeHead(404);
        res.end("Page non trouvée");
  }
});

// Demarrage du serveur sur le port 3000
server.listen(3000, () => console.log("Serveur en écoute sur le port
3000..."));
```

Activité N06

Application pratique

1. Création du fichier `logger.js`

```
const fs = require("fs"); // importation fu filesystem
const EventEmitter = require("events"); // importation du EventEmitter

// Creation du class `Logger` et l'heritage du classe `EventEmitter`
class Logger extends EventEmitter {
  log(message) { // methode pour ecrire et declancher un evenement
    fs.appendFileSync("log.txt", message + "\n");
    this.emit("messageLogged", { message, date: new Date() });
  }
}

module.exports = Logger; // exportation de la classe `Logger`
```

1. Création du fichier `app.js`

```
const http = require("http"); // import du module http
const Logger = require('./Logger'); // import du classe Logger
const logger = new Logger(); // creation d'une instance

logger.on("messageLogged", (data) => {
  console.log("Événement capturé :", data);
}); // creation d'un evenement `messageLogged`

const server = http.createServer((req, res) => {
  logger.log(`Requête reçue : ${req.url}`);
  res.end("Requête enregistrée !");
}); // creation du serveur http

// lancement de l'ecoute sur le port 4000
server.listen(4000, () => console.log("Serveur sur le port 4000..."));
```

Reponses aux questions

1. *Que signifie “non-bloquant” dans le contexte du module `fs` ?*

- C'est a dire que les instructions ne bloquent pas les autres instructions de programme qui suivent ces instructions de s'executer meme si ces instructions ne sont pas encore finis.

2. *Comment les événements permettent-ils de découpler les modules ?*
   - Un module peut émettre un événement, et un autre peut l'écouter sans que le premier ait besoin de le connaître. ce que rend le code plus facile à maintenir.
3. *Pourquoi un serveur HTTP Node peut-il gérer des milliers de connexions avec un seul thread ?*
   - Un serveur HTTP Node peut gérer des milliers de connexions avec un seul thread grâce à son fonctionnement asynchrone.
4. *Quelle serait la prochaine étape (ex: Express, middlewares, JSON parsing) ?*
   - JSON parsing