Implémentation d'un server

Master 1 – Coding for designer
Jan 24
M.ROCH

Objectif

Tu vas écrire ton propre serveur HTTP avec Node.js, ligne par ligne. À chaque étape, tu vas tester pour t'assurer que tout fonctionne comme prévu.

Un serveur est un programme informatique qui écoute des requêtes provenant de clients (comme un navigateur web) et répond avec des informations ou des fichiers. Il agit comme un intermédiaire entre un utilisateur et les ressources (fichiers, bases de données, etc.) qu'il souhaite consulter ou modifier.

Notre serveur Node.js nous permet:

D'afficher une application web interactive comme avec Live Server.
De conserver des données en local grâce à un fichier JSON.

Étape 1 : Préparer les fichiers du projet

Créez une structure de fichiers comme suit:

```
mon-projet/
├── v1
│   └── index.html
│   └── sketch.js
│   └── server.js
├── toto.json
├── p5.min.js
├── addons/
│   └── p5.sound.js
```

Étape 2 : Démarrer un serveur basique

Ouvre server.js dans ton éditeur de texte et commence avec ceci :

```
// Import du module HTTP de Node.js
const http = require("http");

// Définir le port et l'adresse du serveur
const port = 8000;
const serverUrl = "127.0.0.1";

// Créer le serveur
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello World!");
});

// Lancer le serveur
server.listen(port, serverUrl, () => {
  console.log(`Serveur démarré à l'adresse http://${serverUrl}:${port}`);
});
```

Étape 3 : Tester notre serveur basique

Teste ton serveur

Dans ton terminal, tape :

```
node server.js
```

Si tout va bien, tu devrais voir :

```
"Serveur démarré à l'adresse http://127.0.0.1:8000"
```

Ouvre ton navigateur et va sur http://127.0.0.1:8000. Tu devrais voir

```
Hello World!
```

affiché.

Étape 4 : Ajoute un fichier HTML

Crée un fichier index.html dans le même dossier avec ce contenu:

```
<!DOCTYPE html>
<html>
<head>
  <title>Mon serveur</title>
</head>
<body>
  <h1>Bienvenue sur mon serveur Node.js !</h1>
</body>
</html>
```

Étape 4 : Modifie le serveur pour lire des fichiers

Modifie le serveur pour lire des fichiers

```
const http = require("http");
const fs = require("fs"); // Module pour lire les fichiers
const path = require("path");

const port = 8000;
const serverUrl = "127.0.0.1";

const server = http.createServer((req, res) => {
  // Définir le fichier à servir
  const filePath = path.join(__dirname, "index.html");

  // Lire le fichier
  fs.readFile(filePath, (err, content) => {
    if (err) {
      res.writeHead(500);
      res.end("Erreur serveur");
      return;
    }

    // Réponse avec le contenu du fichier
    res.writeHead(200, { "Content-Type": "text/html" });
    res.end(content);
  });
});

server.listen(port, serverUrl, () => {
  console.log(`Serveur démarré à l'adresse http://${serverUrl}:${port}`);
});
```

Étape 5 : Modifie le serveur pour lire des fichiers

Teste ton serveur
Relance le serveur avec :

```
Ctrl + C                (arrête le server)
node server.js
```

Recharge http://127.0.0.1:8000 dans ton navigateur.

Tu devrais voir la page avec le titre "Bienvenue sur mon serveur Node.js !".

Étape 6 : Gérer des URLs dynamiques

Modifie server.js >

Étape 7 : Écrire dans un fichier

Crée un fichier JSON
Crée un fichier data.json vide :

```
[]
```

Modifie le serveur pour gérer une

Modifie server.js >
Ajoute un système simple pour détecter l'URL demandée en remplaçant le code suivant :

```
const server = http.createServer((req, res) => {
   // Définir le fichier à servir
   const filePath = path.join(__dirname, "index.html");

   // Lire le fichier
   fs.readFile(filePath, (err, content) => {
      if (err) {
         res.writeHead(500);
         res.end("Erreur serveur");
         return;
      }

      // Réponse avec le contenu du fichier
      res.writeHead(200, { "Content-Type": "text/html" });
      res.end(content);
   });
});
```

Par celui ci :

```
const server = http.createServer((req, res) => {
   // Définir le fichier à servir
   const filePath = path.join(__dirname, "index.html");

   // Lire le fichier
   fs.readFile(filePath, (err, content) => {
      if (err) {
         res.writeHead(500);
         res.end("Erreur serveur");
         return;
      }

      // Réponse avec le contenu du fichier
      res.writeHead(200, { "Content-Type": "text/html" });
      res.end(content);
   });
});
```

Test les différentes URLS :

http://127.0.0.1:8000 → Charge la page index.html.
http://127.0.0.1:8000/hello → Affiche "Hello!".
http://127.0.0.1:8000/unknown → Affiche "404 - Page non trouvée".

URL /ecriture
Ajoute ce bloc dans la partie if...else du serveur :

```
} else if (req.url === "/ecriture") {
   // Nouvelle donnée
   const newData = { message: "Salut", timestamp: new Date().toISOString() };

   // Lire et mettre à jour le fichier JSON
   fs.readFile("./data.json", "utf8", (err, data) => {
      if (err) {
         res.writeHead(500);
         res.end("Erreur lors de la lecture du fichier JSON");
         return;
      }

      const json = JSON.parse(data || "[]");
      json.push(newData);

      fs.writeFile("./data.json", JSON.stringify(json, null, 2), (err) => {
         if (err) {
            res.writeHead(500);
            res.end("Erreur lors de l'écriture du fichier JSON");
            return;
         }

         res.writeHead(200, { "Content-Type": "text/plain" });
         res.end("Données écrites dans le fichier JSON !");
      });
   });
}
```

Test les différentes URLS :
Lance http://127.0.0.1:8000/ecriture dans ton navigateur.
Regarde le fichier data.json. Tu devrais voir une nouvelle entrée avec message et timestamp.

Tu as maintenant un serveur fonctionnel :

Il sert des fichiers statiques.
Il gère plusieurs routes (/, /hello, /ecriture).
Il peut lire et écrire dans un fichier JSON.

Retirer :

Étape 8 : Modifier l'interface HTML

Ajoutons un formulaire HTML dans index.html pour permettre à l'utilisateur d'entrer son nom et son email.

```
<form action="/ecriture" method="POST" class="form-example">
  <div>
    <label for="name">Nom :</label>
    <input type="text" name="name" id="name" required />
  </div>
  <div>
    <label for="email">Email :</label>
    <input type="email" name="email" id="email" required />
  </div>
  <button type="submit">Envoyer</button>
</form>
```

Étape 9 : Configurer le serveur pour gérer le formulaire

Pour gérer les données envoyées par le formulaire, nous devons :

Lire les données reçues via l'envoi du formulaire.
Les stocker dans le fichier data.json.
Voici le code pour mettre à jour server.js :

Étape 10 : Formulaire HTML à utiliser

```
if (req.url === "/ecriture") {
  // Nouvelle donnée
  const newData = { message: "Salut", timestamp: new Date().toISOString() };

  // Lire et mettre à jour le fichier JSON
  fs.readFile("./data.json", "utf8", (err, data) => {
    if (err) {
      res.writeHead(500);
      res.end("Erreur lors de la lecture du fichier JSON");
      return;
    }

    const json = JSON.parse(data || "[]");
    json.push(newData);

    fs.writeFile("./data.json", JSON.stringify(json, null, 2), (err) => {
      if (err) {
        res.writeHead(500);
        res.end("Erreur lors de l'écriture du fichier JSON");
        return;
      }

      res.writeHead(200, { "Content-Type": "text/plain" });
      res.end("Données écrites dans le fichier JSON !");
    });
  });
}
```

Remplacer par :

```
if (req.url.startsWith("/ecriture")) {
  // Récupérer les données de la requête GET
  const url = new URL(req.url, `http://${req.headers.host}`);
  const name = url.searchParams.get("name");
  const email = url.searchParams.get("email");

  if (!name || !email) {
    res.writeHead(400, { "Content-Type": "text/plain" });
    res.end("Erreur : Paramètres 'name' et 'email' requis.");
    return;
  }

  // Créer une nouvelle entrée
  const newEntry = {
    name,
    email,
    timestamp: new Date().toISOString(),
  };

  // Lire le fichier JSON existant
  fs.readFile("data.json", "utf8", (err, data) => {
    if (err) {
      res.writeHead(500);
      res.end("Erreur lors de la lecture du fichier JSON");
      return;
    }

    const jsonData = JSON.parse(data || "[]");
    jsonData.push(newEntry);

    // Écrire les nouvelles données dans le fichier JSON
    fs.writeFile("data.json", JSON.stringify(jsonData, null, 2), (err) => {
      if (err) {
        res.writeHead(500);
        res.end("Erreur lors de l'écriture du fichier JSON");
        return;
      }

      // Répondre à l'utilisateur avec une confirmation
      res.writeHead(200, { "Content-Type": "text/plain" });
      res.end("Données enregistrées avec succès !");
    });
  });
}
```

à utiliser

```
<form action="/ecriture" method="GET">
  <div>
    <label for="name">Nom :</label>
    <input type="text" name="name" id="name" required />
  </div>
  <div>
    <label for="email">Email :</label>
    <input type="email" name="email" id="email" required />
  </div>
  <button type="submit">Envoyer</button>
</form>
```

Pour modifier la structure de donnée il faut d'un coté ajouter des "entrée" (<input> en html) dans le fichier html et il faut ajouter ces même entrées au server (const email = url.searchParams.get("email"); pour le server.js)

Ensuite on ajoute ces entrées (qui sont maintenant stockée dans des variables) dans l'objet qui sera ajouté à notre json !