TP2 – Plateforme P2P Web en Node.js

Abdessabour Ahzab,Dadda mohamed laghdaf

20 novembre 2025

Table des matières

Introduction générale

Ce TP a pour objectif de construire pas à pas une application Web en Node.js suivant une architecture modulaire :

— un serveur HTTP minimal,
— un module de routage,
— un ensemble de gestionnaires de requêtes (`requestHandlers.js`),
— la gestion de sessions simples,
— des services de base : `/login`, `/logout`, `/upload`, `/show`, `/find`.

L'application est testée localement sur `http://localhost:8888`.

1 Prérequis et environnement

Les éléments suivants ont été utilisés :
— Node.js (version ≥ 18),
— Initialisation du projet : `npm init -y`,
— Modules internes Node.js : `http`, `url`, `fs`, `child_process`,
— Module d'upload : `formidable`.

2 Construction progressive de l'application

2.1 Étape 1 – Serveur minimal

Un premier serveur HTTP affiche simplement « Hello World » sur le port 8888.

```
const http = require("http");
http.createServer((req, res) => {
  res.write("Hello World");
  res.end();
}).listen(8888);
```

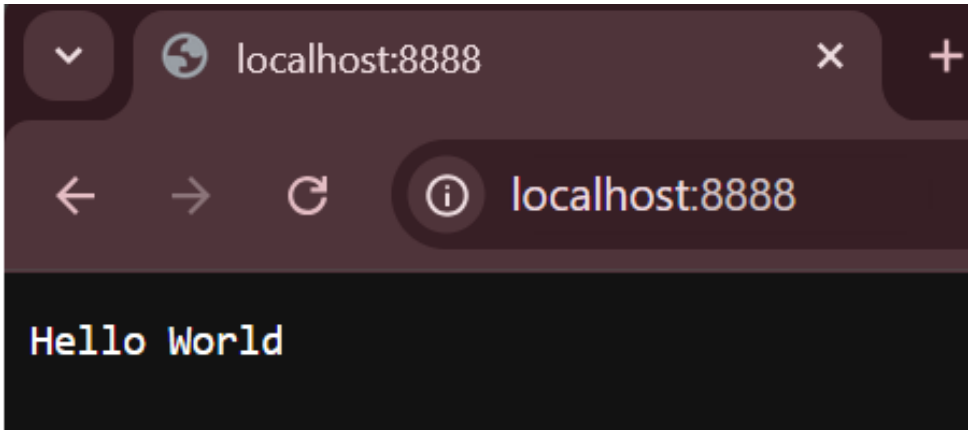

FIGURE 1 – Test du serveur minimal Hello World

2.2 Étape 2 – Modularisation (index/server)

L'application est ensuite découpée en deux modules :
— `server.js` : création du serveur HTTP et gestion de la requête,
— `index.js` : point d'entrée qui appelle `server.start(route, handle)`.

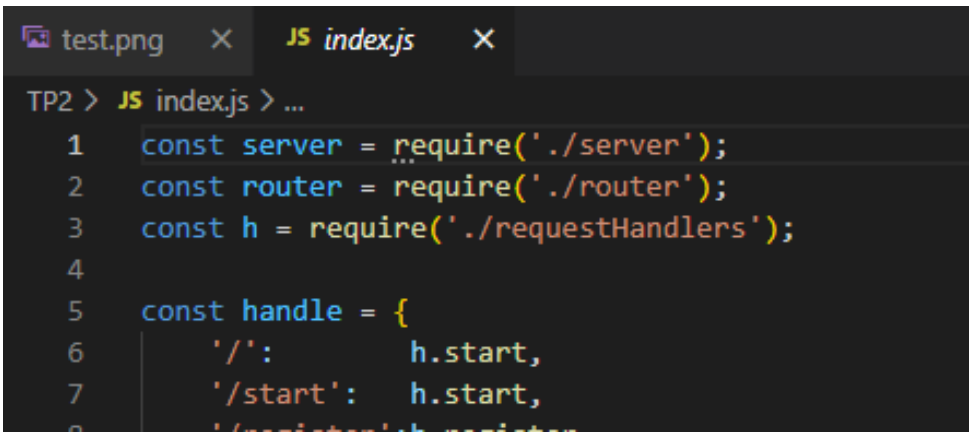

FIGURE 2 – Exécution du serveur modulaire `index.js` + `server.js`

2.3 Étape 3 – Router

Un module `router.js` est ajouté pour rediriger les requêtes vers les bons gestionnaires, en fonction du `pathname` :

```
function route(handle, pathname, response) {
  if (typeof handle[pathname] === 'function') {
      handle[pathname](response);
  } else {
      response.write("404 Not Found");
      response.end();
  }
}
```

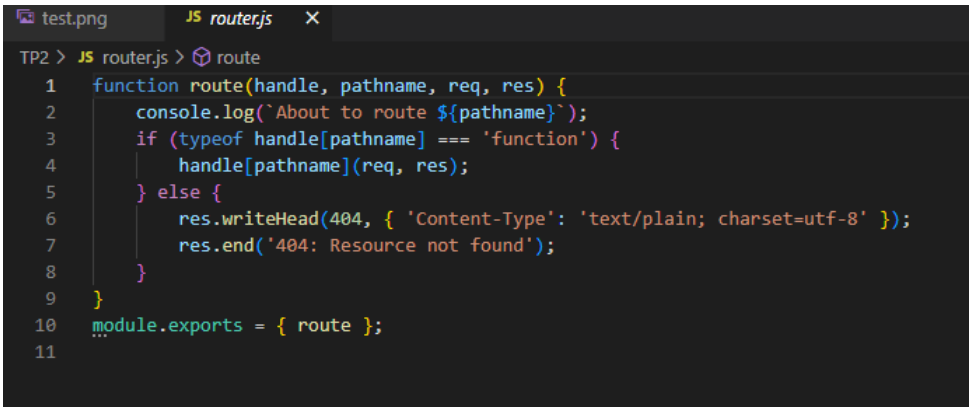

FIGURE 3 – Exécution du serveur modulaire `index.js` + `server.js`

2.4 Étape 4 – Request Handlers

Les principaux services métier sont définis dans `requestHandlers.js` :

- `/login`,
- `/logout`,
- `/upload`,
- `/show`,
- `/find`.

Exemple pour `/login` (version simple) :

```
function login(response) {
  response.writeHead(200, {"Content-Type": "application/json"});
  response.write(JSON.stringify({ ok: true, login: "Abdessabour" }));
  response.end();
}
```

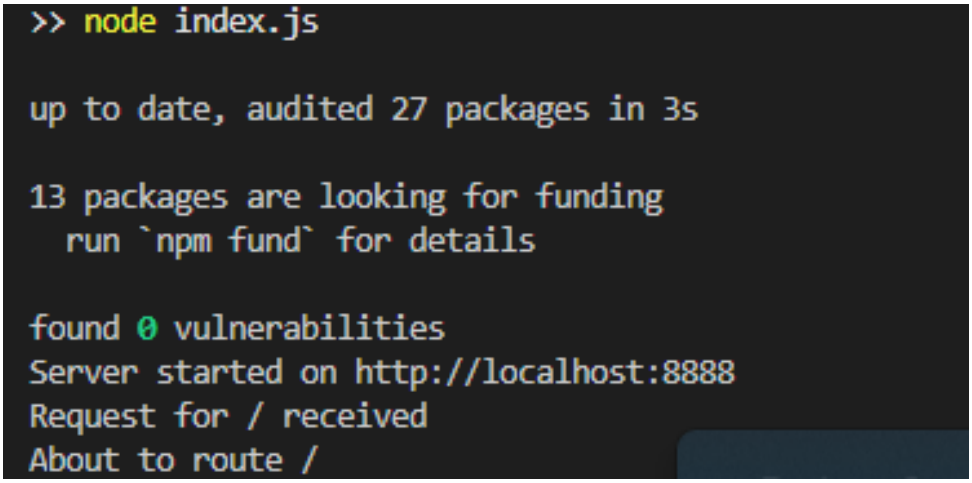

FIGURE 4 – Exécution du serveur Node.js avec les handlers

3 Structure finale du projet

L'arborescence finale du projet est la suivante :

```
.
 index.js
 server.js
 router.js
 requestHandlers.js
 sessions.js
 users.js
 uploads/
 public/
    default.png
 images/
```

Cette structure sépare clairement la logique serveur, le routage, les handlers, les données de session et les ressources statiques.

4 Démonstration des services et captures d'écran

Cette section illustre les différents services exposés par le serveur ainsi que le code associé côté Node.js.

4.1 Page d'accueil (`http://localhost:8888`)

L'application démarre sur une page HTML statique située dans `public/index.html` et servie par le handler `start`.

FIGURE 5 – Page d'accueil du TP2

Handler correspondant :

```
// requestHandlers.js
function start(response) {
  fs.readFile("public/index.html", function(err, data) {
    response.writeHead(200, {"Content-Type": "text/html"});
    response.write(data);
    response.end();
  });
}
```

4.2 Formulaire d'inscription → /login

La page d'accueil contient un formulaire avec les champs `nom`, `prenom`, `login`, `password`, envoyé vers le service `/login` en méthode GET.

FIGURE 6 – Appel de `/login` depuis le navigateur

Extrait HTML :

```
<form action="/login" method="GET">
  <input type="text"     name="nom">
  <input type="text"     name="prenom">
  <input type="text"     name="login">
  <input type="password" name="password">
  <input type="submit"   value="S'inscrire">
</form>
```

4.3 Service `/login` (retour JSON)

Lorsqu'un utilisateur soumet le formulaire, le serveur renvoie une réponse JSON confirmant le login reçu.

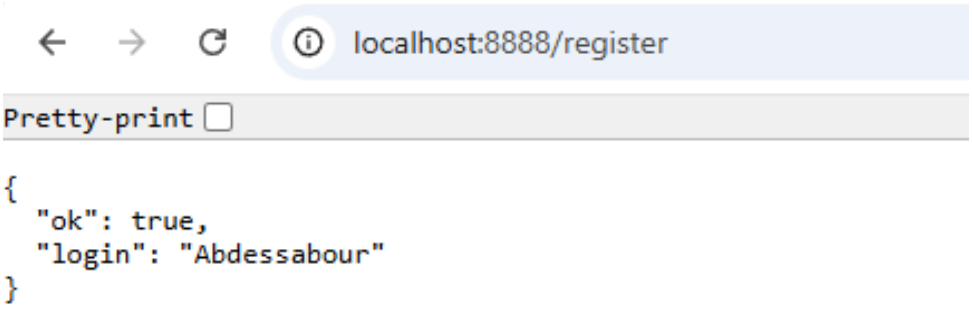

FIGURE 7 – Réponse JSON du service `/login`

Handler utilisé :

```
function login(response, query) {
  const result = { ok: true, login: query.login };
  response.writeHead(200, {"Content-Type":"application/json"});
  response.write(JSON.stringify(result));
  response.end();
}
```

4.4 Upload d'image via `/upload`

L'utilisateur peut téléverser une image qui sera sauvegardée dans le répertoire `uploads/` grâce au module `formidable`.

FIGURE 8 – Formulaire d'upload d'image

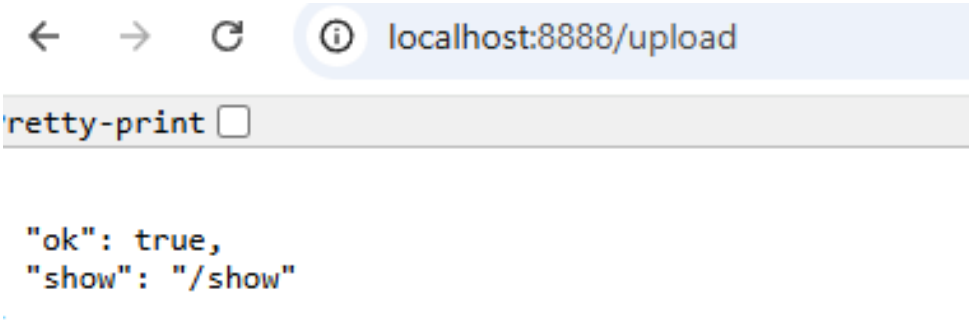

FIGURE 9 – Résultat après upload de l'image

Handler :

```
function upload(response, request) {
  const form = new formidable.IncomingForm();
  form.uploadDir = "uploads";

  form.parse(request, function(err, fields, files) {
    fs.rename(files.file.path, "uploads/image.png", () => {
      response.writeHead(200, {"Content-Type":"text/plain"});
      response.write("Image uploadée avec succès.");
      response.end();
    });
  });
}
```

4.5 Service `/show` (affichage de l'image)

Le service `/show` renvoie la dernière image uploadée ou une image par défaut.

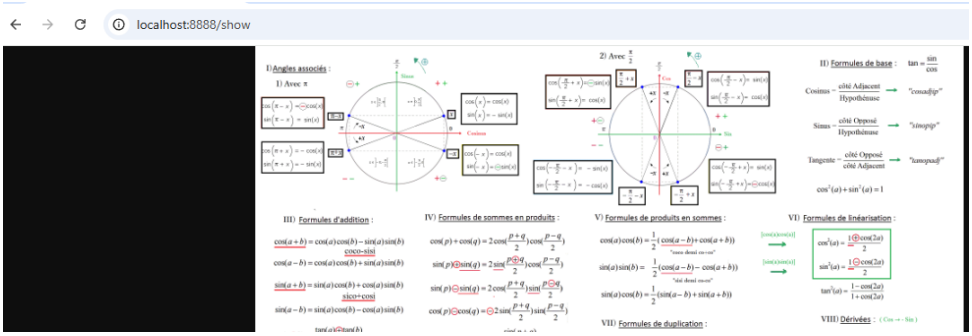

FIGURE 10 – Affichage de l'image servie par `/show`

Handler :

```
function show(response) {
  fs.readFile("uploads/image.png", function(err, data) {
    response.writeHead(200, {"Content-Type":"image/png"});
    response.write(data);
    response.end();
  });
}
```

4.6 Service `/find` (listing de fichiers)

Le service `/find` exécute la commande système `find /` et renvoie la liste des fichiers. L'appel est asynchrone via `child_process.exec`.

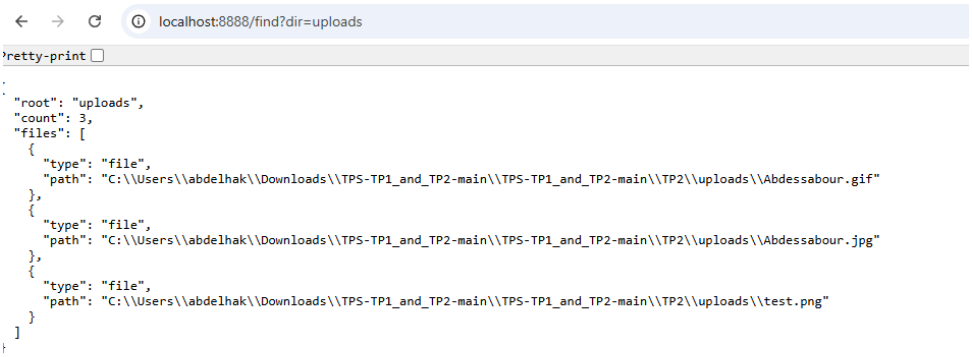

FIGURE 11 – Résultat du service `/find`

Code :

```
function find(response) {
  exec("find /", function(err, stdout, stderr) {
    response.writeHead(200, {"Content-Type":"text/plain"});
    response.write(stdout);
    response.end();
  });
}
```

4.7 Service `/logout`

Le service `/logout` clôt la session courante en renvoyant une réponse JSON.

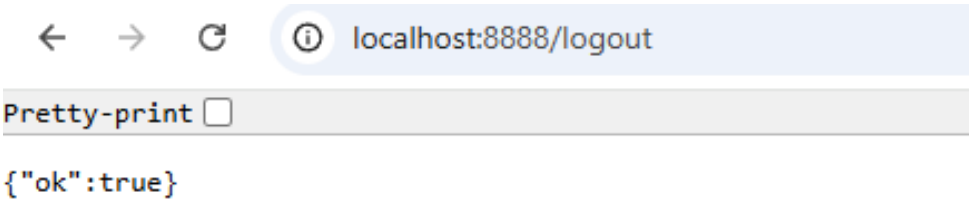

FIGURE 12 – Appel du service `/logout`

Handler :

```
function logout(response) {
  response.writeHead(200, {"Content-Type":"application/json"});
  response.write(JSON.stringify({ ok: true, message: "Logout OK" }));
  response.end();
}
```

Conclusion

Ce TP a permis de mettre en place une application Node.js structurée en modules (server, router, handlers) et d'implémenter plusieurs services typiques : authentification simple, upload et affichage de fichiers, exploration du système de fichiers et gestion basique de sessions.