

Un serveur Web d'inscription à un colloque

D'autres méthodes de connexion à un serveur existent en dehors de `GET`. Ce TD vous amènera à manipuler une requête de type `POST` en envoyant des données, via un client HTTP, à un serveur HTTP chargé de mettre à jour un fichier CSV.

La première partie du TD se consacre à paramétrer le serveur pour distribuer correctement les tâches aux différentes URLs appelées, quand la seconde partie est centrée sur la configuration d'un client HTTP qui envoie les données.

Présentation du site Web

Le site Web est constitué de trois pages HTML, présentes dans le dossier *debut* :

- *index.html*, un formulaire d'inscription qui envoie à l'adresse `/inscription` ;
- *confirmation.html*, une page de confirmation de la prise en compte de l'inscription au colloque ;
- *404.html*, une page d'erreur au cas où le navigateur chercherait à atteindre une route non attribuée.

En plus de ces trois pages, vous trouverez un fichier au format CSV (*Comma-Separated Values*) qui contient la liste des inscrit.es au colloque avec un enregistrement par ligne au-delà de la ligne d'en-tête. Les informations recueillies sont celles du formulaire, dans l'ordre : nom, prénom, email, statut et affiliation.

Conception du serveur

Une fois configuré et prêt à l'écoute, deux missions importantes seront dévolues à votre serveur :

1. servir les différents fichiers en fonction des routes ;
2. mettre à jour le fichier *inscriptions.csv*.

Étape 1 : configurer un serveur prêt à écouter

Pour cette première étape, configurez un serveur en écoute sur le port 3000 et qui ne s'occupe que d'une seule chose : afficher la page d'accueil.

Tout d'abord, incluez les modules nécessaires dans le fichier *server.js* :

```
// modules requis
const http = require('http');
const fs = require('fs');
const stream = require('stream');
```

Paramétrez ensuite le serveur avec la méthode `http.createServer()` de telle manière à créer un flux en lecture avec la page *index.html*. Vous utiliserez la méthode `fs.createReadStream()` et mettrez en place un pipeline entre ce flux et celui de l'objet `response` :

```
// configuration serveur
const server = http.createServer( (request, response) => {
  // page d'accueil dans un flux en lecture
  let index = fs.createReadStream('./index.html');
  // pipeline entre flux en lecture et flux en écriture
```

```

    stream.pipeline(index, response, (error) => {
      if (error) console.log(error);
    });
  });
// serveur en écoute sur le port 3000
server.listen(3000);

```

Vous pouvez lancer le serveur dans un terminal :

```
$ node server.js
```

Visitez à présent la page <http://localhost:3000> qui devrait vous afficher le formulaire d'inscription.

Étape 2 : servir les différentes routes

Trois cas de figures doivent être envisagés :

- le cas normal où le formulaire est affiché ;
- le cas où le message de confirmation s'affiche après la soumission du formulaire ;
- et le cas où aucune de ces pages n'est appelée.

Pour le premier cas, la route `/` devrait simplement afficher le fichier `index.html`. Pour le second cas, c'est un peu plus complexe. L'attribut `action` du formulaire HTML vous informe que, lors de la soumission, le navigateur renvoie à l'adresse `./inscription`. Il ne vous reste plus qu'à programmer la redirection vers la page `confirmation.html`.

Pour tous les autres cas, vous devriez rediriger vers la page `404.html`.

La première étape consiste à analyser la requête envoyée par le navigateur :

```

// module url
const url = require('url');

// configuration serveur
const server = http.createServer( (request, response) => {
  ...
  // analyse url
  let filename = url.parse(request.url).pathname;
  ...
});

```

Ensuite, en fonction de la route appelée, redirigez vers la bonne page :

```

// routes d'accès
switch (filename) {
  case '/': filename = './index.html'; break;
  case '/inscription': filename = './confirmation.html'; break;
}

```

Enfin, grâce à la méthode `exists()` du module `fs`, paramétrez la redirection vers la page 404 en cas de fausse route :

```
// flux en écriture : fichier HTML envoyé
fs.exists(filename, (exists) => {
  const headers = {'Content-Type': 'text/html'};
  if (!exists) {
    filename = './404.html';
    response.writeHead(404, headers);
  }
  else response.writeHead(200, headers);
  let file = fs.createReadStream(filename);
  stream.pipeline(file, response, (error) => {
    if (error) console.log(error);
  });
});
```

Étape 3 : mettre à jour le fichier des inscriptions

Pour mettre à jour le fichier des inscriptions, rien de bien compliqué : il s'agit simplement de savoir ajouter des données à un fichier existant. L'opération est réalisée avec la méthode `fs.appendFile()` :

```
fs.appendFile('./inscriptions.csv', data, (error) => {
  if (error) console.log(error);
});
```

La vraie question est de savoir comment récupérer le contenu de la constante `data`. Comme un objet `request` est aussi un flux en lecture, il émet un événement `data` dès lors que des données lui parviennent. En l'écouter, vous récupérerez les informations issues de la soumission du formulaire :

```
// compléter inscriptions.csv
request.on('data', (chunk) => {
  // traitement des données transmises
  ...
  // ajout dans le fichier
  fs.appendFile('./inscriptions.csv', data, (error) => {
    if (error) console.log(error);
  });
});
```

À présent, le formatage des données. À moins d'avoir installé le module nécessaire, considérez le fichier CSV comme un fichier de texte simple et formatez la chaîne de caractères à insérer dans le fichier avec des tabulations (`\t`) entre chaque champ et un retour à la ligne (`\n`) en fin d'enregistrement :

```
{nom}\t{prenom}\t{email}\t{statut}\t{affiliation}\n
```

Vous devrez alors découper le `chunk` reçu (de la forme `nom=x&prenom=x`) avec la méthode `split()` pour isoler les composants de la requête.

Remarque : les données sont encodées pour le Web. Il vous sera peut-être nécessaire d'utiliser une fonction de décodage comme `decodeURIComponent()` pour enregistrer les bons caractères dans le fichier.

```
// compléter inscriptions.csv
request.on('data', (chunk) => {
  let data = String();
  let search = String();
  for (let param of chunk.toString().split('&')) {
    const q = param.split('=');
    search += `${decodeURIComponent(q[1]) }\\t`;
  }
  data = `${search}\\n`;
  fs.appendFile('./inscriptions.csv', data, (error) => {
    if (error) console.log(error);
  });
});
```

Vous trouverez le code final des documents HTML et JavaScript dans le dossier *fin* de ce premier TD.