Lab 2 : Module Http

Exercice 1 : lire des données à partir d'un serveur distant

1. Dans votres espace de travail, créer un répertoire /http
2. Dans le nouveau répertoire, ajouter un fichier nommé request.js
3. Copier le code suivant dans le nouveau fichier

```
var https = require("https");
var fs = require("fs");

var options = {
        hostname: "en.wikipedia.org",
        port: 443,
        path: "/wiki/George_Washington",
        method: "GET"
};

var req = https.request(options, function(res) {

        var responseBody = "";

        console.log("Response from server started.");
        console.log(`Server Status: ${res.statusCode} `);
        console.log("Response Headers: %j", res.headers);

        res.setEncoding("UTF-8");

        res.once("data", function(chunk) {
                console.log(chunk);
        });

        res.on("data", function(chunk) {
                console.log(`--chunk-- ${chunk.length}`);
                responseBody += chunk;
        });

        res.on("end", function() {
                fs.writeFile("george-washington.html", responseBody, function(err)
{
                        if (err) {
                                throw err;
                        }
                        console.log("File Downloaded");
                });
        });

});

req.on("error", function(err) {
        console.log(`problem with request: ${err.message}`);
});

req.end();
```

4. Lancer le programme et remarquer la création d'un nouveau fichier contant les données récupérées à partir de la page wikipedia. Remarquer également la différence entre res.once et res.on

Exercice 2 : Créer un serveur et servir une page HTML

1. Dans le répertoire /http, ajouter un fichier nommé serveur.js
2. Copier le code suivant dans le nouveau fichier

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

var server = http.createServer(function(req, res) {

        res.writeHead(200, {"Content-Type": "text/html"});

        res.end(`
                <!DOCTYPE html>
                <html>
                        <head>
                                <title>HTML Response</title>
                        </head>
                        <body>
                                <h1>Serving HTML Text</h1>
                                <p>${req.url}</p>
                                <p>${req.method}</p>
                        </body>
                </html>
        `);

});

server.listen(3000);

console.log("Server listening on port 3000");
```

3. Lancer le programme et remarquer l'affichage de la page web avec l'URL et la méthode http utilisées.

Exercice 3 : Servir des fichiers statiques

1. Copier le répertoire /public fourni avec ce lab dans le répertoire /http
2. Sous /http, créer un fichier nommé fileserver.js
3. Copier le code suivant dans le nouveau fichier

```
var http = require("http");
var fs = require("fs");
var path = require("path");

http.createServer(function(req, res) {

        console.log(`${req.method} request for ${req.url}`);

        if (req.url === "/") {
                fs.readFile("./public/index.html", "UTF-8", function(err, html) {
                        res.writeHead(200, {"Content-Type": "text/html"});
                        res.end(html);
                });

        } else if (req.url.match(/.css$/)) {

                var cssPath = path.join(__dirname, 'public', req.url);
                var fileStream = fs.createReadStream(cssPath, "UTF-8");

                res.writeHead(200, {"Content-Type": "text/css"});
```

```
                fileStream.pipe(res);
        } else if (req.url.match(/.jpg$/)) {

                var imgPath = path.join(__dirname, 'public', req.url);
                var imgStream = fs.createReadStream(imgPath);

                res.writeHead(200, {"Content-Type": "image/jpeg"});

                imgStream.pipe(res);

        } else {
                res.writeHead(404, {"Content-Type": "text/plain"});
                res.end("404 File Not Found");
        }

}).listen(3000);


console.log("File server running on port 3000");
```

4. Lancer le programme et remarquer l'affichage de la page web définie dans /public/index.html.

Exercice 4 : Renvoyer des données au format json

1. Copier le répertoire /data fourni avec ce lab dans le répertoire /http
2. Sous /http, créer un fichier nommé api.js
3. Copier le code suivant dans le nouveau fichier

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

var data = require("./data/inventory");

http.createServer(function(req, res) {

        if (req.url === "/") {
                res.writeHead(200, {"Content-Type": "text/json"});
            res.end(JSON.stringify(data));
        } else if (req.url === "/instock") {
                listInStock(res);
        } else if (req.url === "/onorder") {
                listOnBackOrder(res);
        } else {
                res.writeHead(404, {"Content-Type": "text/plain"});
                res.end("Whoops... Data not found");
        }

}).listen(3000);

console.log("Server listening on port 3000");


function listInStock(res) {

        var inStock = data.filter(function(item) {
                return item.avail === "In stock";
        });

        res.end(JSON.stringify(inStock));
```

```
}

function listOnBackOrder(res) {

        var onOrder = data.filter(function(item) {
                return item.avail === "On back order";
        });

        res.end(JSON.stringify(onOrder));

}
```

4. Lancer le programme, ouvrir le navigateur à l'URL (http://localhost:3000) et remarquer le résultat renvoyé selon l'URL passée.
5. Dans votre navigateur, changer l'URL vers http://localhost:3000/instock puis vers http://localhost:3000/onorder
Que remarquez-vous ?
Essayer d'identifier les lignes du code source précédent qui permettent de traiter les différentes URL.

Exercice 5 : Récupèrer des paramètres passés avec POST

1. Dans le répertoire /http/public, vérifier la présence du fichier form.html
2. Sous /http, ajouter le fichier formserver.js
3. Copier le code suivant dans le nouveau fichier

```
var http = require("http");
var fs = require("fs");

http.createServer(function(req, res) {

        if (req.method === "GET") {
                res.writeHead(200, {"Content-Type": "text/html"});
            fs.createReadStream("./public/form.html", "UTF-8").pipe(res);
        } else if (req.method === "POST") {

                var body = "";

                req.on("data", function(chunk) {
                        body += chunk;
                });

                req.on("end", function() {

                        res.writeHead(200, {"Content-Type": "text/html"});

                        res.end(`

                                <!DOCTYPE html>
                                <html>
                                        <head>
                                                <title>Form Results</title>
                                        </head>
                                        <body>
                                                <h1>Your Form Results</h1>
                                                <p>${body}</p>
                                        </body>
                                </html>
                        `);
                });
        }
```

```
}).listen(3000);

console.log("Form server listening on port 3000");
```

4. Lancer le programme et ouvrir le navigateur à l'adresse (http://localhost:3000). Remarquer l'affichage du formulaire puisque la méthode http utilisée est GET.
5. Remplir le formulaire et valider. Remarquer que cette fois c'est le résult de la méthode POST qui est affiché.
N.B : Pour exploiter les données récupérées, nous avons besoin de les parser. Nous n'allons pas réaliser cette tâche avec node directement, nous verrons que les choses sont plus simples avec le framework Express.