Un serveur Web d’inscription à un colloque, version Express

Reposant sur le module *Connect*, le cadriciel *Express.js* facilite reprend ses méthodes tout en facilitant de nombreuses opérations. Ce TD vous amènera à concevoir une application Web selon les usages en vigueur.

Présentation du code de départ

Dans le dossier *debut*, vous trouverez un sous-dossier *views* contenant l’ensemble des vues de l’application, au format *Pug*. Dans l’autre sous-dossier, simplement le fichier CSV des inscriptions au colloque.

Conception de l’application

Étape 1 : démarrage de l’application

Créez un répertoire de travail avec le nom de votre application et démarrez un nouveau projet *Express* avec `express-generator` :

```
$ mkdir colloque
$ cd colloque
$ npx express-generator
```

Installez le module *Pug* ainsi que les dépendances :

```
$ npm install
$ npm install --save pug
```

Configurez ensuite le script de démarrage dans le fichier *package.json* pour que la commande `npm start` lance l’application *app.js* avec la commande `nodemon` :

```
"scripts": {
    "start": "nodemon ./bin/www"
}
```

Définissez à présent le module *Pug* comme moteur de rendu de vos templates, en modifiant votre application *app.js* :

```
app.set('view engine', 'pug');
```

Vous pouvez maintenant remplacer les vues du répertoire *views* par celles présentes dans le répertoire de départ et ajouter le répertoire *private* à la racine de votre application.

Si vous lancez l’application, le formulaire d’inscription devrait s’afficher :

```
$ npm start
```

Étape 2 : routage des requêtes

Les missions de votre serveur sont toujours les mêmes : il s'agit de récupérer les informations postées dans le formulaire, de les traiter pour les enregistrer dans un fichier CSV et de rediriger le client HTTP vers la page de confirmation. Pour toutes les autres routes, la page 404 devra s'afficher.

Routage de l'inscription

Dans un premier temps, renommez la route *users.js* en *inscription.js*, puis, dans votre application, paramétrez le routeur pour le chemin `/inscription` :

```
var inscriptionRouter = require('./routes/inscription');
app.use('/inscription', inscriptionRouter);
```

Intervenez ensuite sur le fichier de votre routeur pour renvoyer la vue *confirmation.pug*. N'oubliez pas que la méthode utilisée par le navigateur est de type `POST` .

```
router.post('/', function(req, res, next) {
    res.render('confirmation', { title: 'Confirmation de votre inscription' });
});
```

Remarquez la méthode `render()` qui accepte deux arguments :

1. le nom de la vue à afficher ;
2. un objet optionnel dont les clés seront utilisées par la vue pour afficher des messages variables.

Si le paramétrage est conforme, la page de confirmation s'affiche désormais lors de tout nouvel envoi de données par le formulaire.

Routage vers la page 404

Pour résumer, deux routes sont pour l'instant définies :

1. le chemin `/` qui affiche la page d'accueil ;
2. et le chemin `/inscription` qui affiche la page de confirmation.

Si vous demandez maintenant à votre client HTTP d'accéder à une URL fantaisiste, une page d'erreur avec le message **NOT FOUND 404** devrait apparaître. Votre application est par défaut paramétrée pour rediriger toutes les URLs non capturées par les routeurs vers cette page-là. C'est le sens du *middleware* suivant dans le fichier de configuration de votre application :

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
    next(createError(404));
});
```

Toujours grâce à la méthode `render()`, demandez plutôt à votre application de servir la vue 404 :

```
app.use(function(req, res, next) {
    res.render('404');
});
```

Vous pouvez également peaufiner le tout en retournant au serveur un statut 404 :

```
app.use(function(req, res, next) {
    res.status(404);
    res.render('404');
});
```

Étape 3 : enregistrer les informations envoyées

La dernière étape de votre application se chargera de récupérer les données par le formulaire et de les consigner dans le fichier *inscriptions.csv*.

Là encore, *Express.js* vous facilite la tâche. L'objet `req` du *handler* d'une route dispose d'une clé `body` pour représenter les données qui transitent. En l'occurence, il s'agit d'un objet sous la forme :

```
body {
    nom: "Roulois",
    prenom: "Alexandre",
    email: "alexandre.roulois@u-paris.fr",
    statut: "i",
    affiliation: "LLF"
}
```

Comme les clés sont présentées dans le même ordre que sur la ligne d'en-tête du fichier CSV, le traitement de l'enregistrement des données s'en voit facilité.

Dans votre routeur *inscription.csv*, ajoutez déjà un *middleware* pour récupérer la liste des valeurs présentes dans `req.body` :

```
router.post('/', (req, res, next) => {
    const values = Object.values(req.body);
});
```

Convertissez cette liste en une chaîne de caractères correctement formatée pour le format CSV tabulé :

```
router.post('/', (req, res, next) => {
    const values = Object.values(req.body);
    const data = `${ values.join('\t') }\n`;
});
```

Et ajoutez cette chaîne au fichier CSV en utilisant la méthode `fs.appendFile()`, sans oublier de transmettre une erreur éventuelle :

```
const fs = require('fs');

router.post('/', (req, res, next) => {
    const values = Object.values(req.body);
    const data = `${ values.join('\t') }\n`;
    fs.appendFile('./private/files/inscription.csv', data, (error) => {
        if (error) next(error);
        else next();
    });
});
```

Vous trouverez le code final de ce premier TD sur *Express.js* dans le dossier *fin*.