Un serveur Web de gestion des références bibliographiques

Précédemment, vous avez programmé une petite application qui, après avoir interrogé l'API de recherche HAL, affichait les références bibliographiques d'un·e contributeur·rice en fonction de son identifiant HAL. Au cours de ce TD, vous apprendrez à analyser la syntaxe d'un document JSON avec le cadriciel *Express.js* afin de vous en servir comme d'une base de données compatible avec l'archive ouverte HAL. Vous mettrez notamment en place des routes pour ajouter, modifier et supprimer des références de votre base.

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* ; et, dans le sous-dossier *private*, un exemple de fichier JSON pour servir de base à votre application. Les données ont été récupérées en interrogeant l'API HAL à l'aide de paramètres prédéfinis. Modifiez uniquement le paramètre `authIdHal_s` afin d'obtenir un résultat similaire pour votre identifant HAL.

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 biblio && cd biblio
$ 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, une page intitulée *Références bibliographiques* devrait s'afficher :

```
$ npm start
```

Étape 2 : lister les routes nécéssaires

Votre application ne s'occupera que de quelques tâches de base. Pour chacune d'entre elles, définissez une route :

- (**GET**) `/` : lister les références ;
- (**GET**) `/docs/new` : formulaire pour ajouter une référence ;
- (**GET**) `/docs/update/{id}` : formulaire pour modifier une référence ;
- (**POST**) `/docs/new` : ajouter une référence ;
- (**PUT**) `/docs/update/{id}` : modifier une référence ;
- (**DELETE**) `/docs/delete/{id}` : supprimer une référence.

Étape 3 : routage vers la page 404

Comme lors du précédent TD, configurez le *middleware* chargé de gérer le message d'erreur 404 dans le fichier *app.js* de telle manière qu'il affiche le contenu de la vue *404.pug* :

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

Étape 4 : lister les références bibliographiques

À l'ouverture de la page d'accueil de votre application, vous aimeriez obtenir la liste de vos références bibliographiques, triées une première fois par ordre alphabétique et une deuxième fois par ordre antéchronologique d'année de publication.

Votre intervention à cette étape se situera à deux niveaux :

- au niveau du contrôle des données dans la route *index.js* ;
- et au niveau de leur affichage dans la vue *index.pug*.

Lecture des données de la base

Occupez-vous dans un premier temps de l'importation des données issues de la base. Comme vous communiquez avec un fichier présent en local, vous mobiliserez la méthode `readFile()` du module *File System*.

Importez tout d'abord le module dans le fichier *routes/index.js* :

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

Puis affectez le contenu du fichier *db.json* à une variable `rows` :

```
router.get('/', (req, res, next) => {
    var rows;
    fs.readFile('./private/db.json', 'utf8', function (err, data) {
        if (err) throw err;
        rows = JSON.parse(data);
    });
});
```

Si vous analysez la structure du fichier JSON extrait depuis l'API de recherche HAL, les références bibliographiques sont listées dans un objet `rows.response.docs`. Itérez dessus pour récupérer le contenu de la clé `label_s` et l'enregistrer dans un nouvel objet `documents` :

```
router.get('/', (req, res, next) => {
    var rows;
    var documents = Array();
    fs.readFile('./private/db.json', 'utf8', function (err, data) {
        if (err) throw err;
        rows = JSON.parse(data);
        for (doc of rows.response.docs) {
            documents.push({
                ref: doc.label_s
            });
        }
    });
});
```

Transmettez enfin le nouveau objet `documents` à la vue *index.pug*, en le triant par ordre alphabétique :

```
res.render('index', {
    title: 'Références bibliographiques',
    docs: documents.sort()
});
```

Affichage d'une première liste de références

Éditez à présent la vue *index.pug* afin d'afficher une liste non ordonnée des références bibliographiques :

```
ul.list-group.list-group-flush.mt-3
    each doc in docs
        li.list-group-item= doc.ref
```

Remarque : un problème d'affichage survient. Les signes typographiques chevrons apparaissent en effet sous une forme encodée. Pour les décoder, l'une des méthodes consiste à faire appel à la méthode `decode()` du module *Node.js HTML Entities*.

Installez-le :

```
npm install html-entities --save
```

Importez-le ensuite dans le fichier de votre route :

```
const entities = require('html-entities');
```

Et appliquez la méthode `decode()` à l'objet `doc.label_s` :

```
documents.push({
    ref: entities.decode(doc.label_s)
});
```

Classer les références par année de publication

Pour le moment, vos références bibliographiques s'affichent telles quelles sur la page d'accueil alors que votre projet initial consistait à les trier par année de publication. Pour parvenir à votre but, vous allez d'abord construire une liste des années à partir du champ `publicationDate_tdate` de votre base de données pour, à l'affichage, formuler une condition au moment de l'itération sur l'objet `docs`.

Dans votre route, instanciez un nouvel objet `years` grâce au constructeur `Set()` qui permet de créer un ensemble dont l'avantage est de ne conserver que des données dédoublonnées :

```
var years = new Set();
```

Puis, au moment de parcourir l'objet `rows.response.docs`, isolez l'année de publication du champ `publicationDate_tdate` qui est au format ISO 8601 (p. ex. 2021-01-31T12:52:07Z) avec la méthode `substring()` afin de l'ajouter à l'ensemble `years` :

```
for (doc of rows.response.docs) {
    const year = doc.publicationDate_tdate.substring(0, 4);
    years.add(year);
    …
}
```

Transmettez enfin la liste des années, triée par ordre antéchronologique :

```
res.render('index', {
    title: 'Références bibliographiques',
    docs: documents.sort(),
    years: [...years].sort().reverse()
});
```

Remarque : une variante de la syntaxe de décomposition (`[...years]`) peut être obtenue avec la méthode `Array.from()` :

```
res.render('index', {
    title: 'Références bibliographiques',
    docs: documents.sort(),
    years: Array.from(years).sort().reverse()
});
```

Il ne vous reste plus qu'à modifier la vue pour itérer sur l'objet `years` et exprimer la condition d'affichage de chaque référence :

```
each year in years
    h2= year
    ul.list-group.list-group-flush.mt-3
        each doc in docs
            if year == doc.year
                li.list-group-item= doc.ref
```

Pour aller plus loin : un menu de sélection par type de document

Votre objectif ici est de créer des boutons sur le modèle de celui intitulé *Ajouter une référence* pour chaque type de document présent dans votre base de données.

Pour réaliser votre objectif, vous devrez :

- instancier un nouvel ensemble `docTypes` et le remplir avec le champ `docType_s` ;
- créer dans la vue *index.pug* un bouton pour chaque type de document ;
- tracer une route pour chaque type de document qui restreigne la sélection des références bibliographiques à afficher.

Étape 5 : ajouter une nouvelle référence bibliographique

Il est temps de vous charger du mécanisme qui permettra d'insérer une nouvelle référence bibliographique dans votre base de données. Il vous faudra pour cela configurer un routeur qui puisse afficher la vue *doc-form.pug* et modifier le fichier *db.json*.

Occupez-vous tout d'abord du fichier principal *app.js* afin d'appeler le routeur *docs.js* que vous allez créer :

```
const docsRouter = require('./routes/docs');
app.use('/docs', docsRouter);
```

Créez à présent un fichier *docs.js* dans votre répertoire *routes* avec la configuration minimale pour afficher la vue *doc-form.pug* avec des valeurs par défaut :

```
const express = require('express');
const router = express.Router();

/* GET doc form. */
router.get('/new', (req, res, next) => {
    res.render('doc-form', {
        title: "Ajout d'une référence",
        action: "./new",
        verb: "Ajouter"
    });
});

module.exports = router;
```

Comme le formulaire est prévu pour fonctionner à la fois en mode insertion et en mode modification, vous devez prévoir de lui transmettre un objet vide :

```
const doc = {
    halId_s: String(),
    label_s: String(),
    docType_s: doctypes,
    publicationDate_tdate: String()
};

res.render('doc-form', {
    title: "Ajout d'une référence",
    action: "./new",
    verb: "Ajouter"
```

```
    doc: doc
});
```

Remarquez l'objet `doctypes` que vous avez transmis comme valeur de la clé `docType_s` du document vide. Il s'agit de la liste des types de documents selon le référentiel de HAL, que vous pouvez instancier en tête de votre script :

```
const doctypes = [
    {
        "code": "ART",
        "type": "Article dans une revue"
    },
    {
        "code": "COMM",
        "type": "Communication dans un congrès"
    },
    {
        "code": "OUV",
        "type": "Ouvrage"
    },
    {
        "code": "COUV",
        "type": "Chapitre d'ouvrage"
    },
    {
        "code": "DOUV",
        "type": "Direction d'ouvrage"
    },
    {
        "code": "OTHER",
        "type": "Autre publication"
    },
    {
        "code": "REPORT",
        "type": "Rapport"
    },
    {
        "code": "THESE",
        "type": "Thèse"
    },
    {
        "code": "HDR",
        "type": "Habilitation à diriger des recherches"
    }
];
```

Modifiez le code de la vue *doc-form.pug* afin d'intégrer ces modifications :

```
select#docType_s.form-select(name="docType_s")
    option(selected disabled)= "Type de document…"
    each doctype in doc.docType_s
        option(value=doctype.code)= doctype.type
```

Il reste à paramétrer une nouvelle route en mode *POST* pour enregistrer les données dans le fichier qui constitue la base de données :

```
/* POST doc form. */
router.post('/new', (req, res, next) => {

    // do something with data

    // redirection to home page
    res.redirect('/');
});
```

Dans cette dernière partie, vous devrez lire le fichier *db.json* au format JSON avec `JSON.parse()` afin de rajouter le nouvel enregistrement saisi. Commencez par récupérer les variables envoyées par le formulaire :

```
var rows;
const doc = req.body;

fs.readFile('./private/db.json', 'utf8', (err, data) => {
    if (err) throw err;
    rows = JSON.parse(data);
    rows.response.docs.push(doc);
});
```

Insérez-les ensuite dans le fichier :

```
fs.writeFile('./private/db.json', JSON.stringify(rows), (error) => {
    if (error) throw error;
});
```

Notez que l'identifiant HAL est optionnel dans le formulaire. Votre base de données est en effet prévue pour fonctionner avec des références qui ne seraient pas forcément présentes dans HAL. Il est alors indispensable de mettre au point un mécanisme pour fournir un identifiant unique aux références saisies. Vous pouvez par exemple vous reposer sur la clé `numFound` de votre base de données qui fournit le nombre de documents enregistrés. La route complète devient :

```
/* POST doc form. */
router.post('/new', (req, res, next) => {

    var rows;
    const doc = req.body;

    fs.readFile('./private/db.json', 'utf8', (err, data) => {

        if (err) throw err;
        rows = JSON.parse(data);

        if (!doc.halId_s) doc.halId_s = rows.response.numFound + 1;
        rows.response.docs.push(doc);
        rows.response.numFound += 1;
```

```
        fs.writeFile('./private/db.json', JSON.stringify(rows), (error) => {
            if (error) throw error;
        });

        res.redirect('/');

    });

});
```

Étape 6 : une route pour modifier les références

La route définie au moment de la conception de l'application (`/docs/update/{id}`) doit être capable de fonctionner, comme dans le cas de l'insertion manuelle d'une référence, en mode **PUT** comme en mode **GET**. La première se chargera d'effectuer la mise à jour et de rediriger vers la page d'accueil tandis que la seconde affichera le formulaire de saisie avec les données par défaut.

Une icône pour la modification du fichier

Avant tout, ajoutez une icône qui symbolise l'ouverture du formulaire en mode modification dans la vue *index.pug*. Au clic, elle devra rediriger vers la bonne route.

Le code de départ intègre la librairie *Bootstrap Icons*, aussi vous n'avez qu'à ajouter un élément HTML `i` auquel associer une classe CSS (`bi-pencil-square` par exemple) :

```
ul.list-group.list-group-flush.mt-3
    each doc in docs
        if year == doc.year
            li.d-flex.justify-content-start.gap-1.list-group-item= doc.ref
                a(href="docs/update/" + doc.id)
                    i.bi-pencil-square
```

Afficher le formulaire de saisie

Dans la route *docs.js*, paramétrez votre nouvelle route en mode **GET** et récupérez le paramètre `id` :

```
/* GET update doc form */
router.get('/update/:id', (req, res, next) => {
    const id = req.params.id;
});
```

Configurez ensuite l'objet `doc` par défaut :

```
const doc = {
    halId_s: id,
    label_s: String(),
    docType_s: doctypes,
    publicationDate_tdate: String()
};
```

Et transmettez cet objet à la vue *doc-form.js* :

```
res.render('doc-form', {
    title: "Modifier une référence",
    action: "./" + id,
    verb: "Modifier",
    doc: doc
});
```

Il reste maintenant la partie la plus délicate : récupérer les données de la référence sélectionnée avec le paramètre `id`. L'idée ici est de lire le fichier JSON de la base de données et de comparer le paramètre transmis avec l'identifiant enregistré. Si les deux concordent, vous récupérez les valeurs associées aux clés et les affectez à l'objet `doc`.

La première opération, classique, consiste à convertir le fichier en un objet JSON exploitable :

```
var rows;
fs.readFile('./private/db.json', 'utf8', (err, data) => {
    if (err) throw err;
    rows = JSON.parse(data);
});
```

Reste à parcourir tous les enregistrements, à la recherche du bon, afin de remplir l'objet `doc` par défaut :

```
for (row of rows.response.docs) {
    if (row.halId_s == id) {
        doc.label_s = row.label_s;
        doc.publicationDate_tdate = row.publicationDate_tdate.substring(0,4);
    }
}
```

Une méthode plutôt fastidieuse, non ? Parcourir tous les objets avec un itérateur en espérant tomber rapidement sur le bon… Heureusement, JavaScript intègre une méthode `filter()` pour retourner rapidement un tableau d'objets qui respectent une condition. Dans votre cas, un identifiant étant par essence unique, vous restreindrez la sélection au premier élément du tableau :

```
const result = rows.response.docs.filter(row => row.halId_s == id)[0];
doc.label_s = result.label_s;
doc.publicationDate_tdate = result.publicationDate_tdate.substring(0,4);
```

Gérer l'affichage du type de document par défaut

Un problème persiste : dans la base de données, le type de document est sauvegardé sous une forme codifiée (*COMM*, *OUV* etc.) alors que dans le formulaire l'étiquette est explicite. Pour remédier à cette légère discordance, rajoutez à l'objet `doc` une clé `docTypeSelected` pour identifier le type de document sélectionné :

```
const doc = {
    halId_s: id,
    label_s: String(),
    docType_s: doctypes,
```

```
    publicationDate_tdate: String(),
    docTypeSelected: Object()
  };
```

N’oubliez pas d’adapter le code de votre objet dans la route qui affiche le formulaire en mode insertion :

```
const doc = {
    halId_s: String(),
    label_s: String(),
    docType_s: doctypes,
    publicationDate_tdate: String(),
    docTypeSelected: null
};
```

Il ne reste plus qu’à sélectionner dans l’objet `doctypes`, instancié à l’étape 5, l’étiquette correspondant au code de la référence à modifier. Un nouveau filtre est de rigueur :

```
const doctype = doctypes.filter(dt => dt.code == result.docType_s)[0];
doc.docTypeSelected.code = result.docType_s;
doc.docTypeSelected.type = doctype.type;
```

La vue *doc-form.pug* n’ayant pas été prévue pour ce cas de figure, vous devez encore l’adapter en conséquence :

```
div.mb-3.row
    label.form-label.col-sm-3(for="docType_s")= "*Type de document :"
    div.col-sm-9
        select#docType_s.form-select(name="docType_s")
        if !doc.docTypeSelected
            option(selected disabled)= "Type de document…"
        else
            option(value=doc.docTypeSelected.code selected)= doc.docTypeSelected.type
        each doctype in doc.docType_s
            option(value=doctype.code)= doctype.type
```

Effectuer la modification

La norme HTML ne permettant que deux valeurs dans l’attribut `method` de l’élément `form` (`get` et `post`), vous devez à ce stade paramétrer non pas une route en mode **PUT** mais bien une route en mode **POST** pour opérer la modification de la référence.

Dans cette route, instanciez un objet `doc` grâce au contenu de l’objet `req` de *Node.js* et ouvrez le fichier de la base de données à la recherche de la bonne référence :

```
/* POST update doc form */
router.post('/update/:id', (req, res, next) => {

    const id = req.params.id;
    const doc = req.body;

    // get data
```

```
    var rows;
    fs.readFile('./private/db.json', 'utf8', (err, data) => {
        if (err) throw err;
        rows = JSON.parse(data);
        // find ref among data
        const result = rows.response.docs.filter(row => row.halId_s == id)[0];
    });

    res.redirect('/');

});
```

Plutôt que de supprimer tous les objets dont l'identifiant ne correspond pas, il serait peut-être plus pertinent d'effectuer l'inverse ?

```
// keep all refs but the one to update
const result = rows.response.docs.filter(row => row.halId_s != id);
```

Vous pouvez désormais redéfinir l'ensemble des références à conserver dans la base de données :

```
rows.response.docs = result;
```

Puis ajouter le contenu de l'objet `doc` :

```
rows.response.docs.push(doc);
```

Écrivez le tout dans le fichier !

```
fs.writeFile('./private/db.json', JSON.stringify(rows), (error) => {
    if (error) throw error;
});
```

Étape 7 : une route pour supprimer une référence

À titre d'exercice, configurez la route pour supprimer une référence ! Et pour commencer, n'oubliez pas d'ajouter une icône dans la vue *index.pug* en la cherchant dans la librairie *Bootstrap Icons*.

Conclusions

La gestion d'un fichier JSON en tant que base de données peut sembler fastidieuse. Dans la suite de votre exploration de *Node.js*, vous serez plutôt amené·es à exploiter une base de données relationnelle de type *MySQL*, mais sachez d'ores et déjà que d'autres solutions alternatives existent comme, par exemple, *MongoDB*.

À un tout autre niveau, vous aurez sans doute été dérouté·es par la reprogrammation des routes **PUT** et **DELETE** en banales **GET** et **POST** pour la modification et la suppression des références. Il s'agit là d'une mauvaise pratique. Dans la réalité d'une application pleinement opérationnelle, les formulaires appellent en fait, lors de leur soumission, des méthodes qui exécutent la requête HTTP avec le bon verbe. Aussi, dans les TDs à venir, vous suivrez dorénavant les bonnes pratiques !

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