Architecture du projet

Les outils

Node.js et NPM

L'application est codée en JavaScript par le biais de l'environnement d'exécution Node.js. J'ai en outre eu besoin de nombreux paquets supplémentaires que j'ai installé via NPM (le gestionnaire de paquets officiel de Node.js, concurrencé par Yarn) pour pouvoir répondre à différents problèmes et besoins.

La liste des paquets installés est la suivante :

- Assert : paquet du noyau NPM permettant de faire des contrôles sur des données et de jeter des erreurs s'ils échouent,
  - Documentation : nodejs.org/api/assert.html,
- BCrypt : système de chiffrement de chaines de caractères (typiquement des mots de passe) et de comparaison (notamment pour valider une connexion),
  - Documentation : npmjs.com/package/bcrypt,
- ChargerDossier : module que j'ai créé pour charger l'ensemble des fichiers JavaScript d'un dossier et de ses sous-dossiers,
  - Documentation : npmjs.com/package/@iutlannion/charger-dossier,
- Compression : module permettant de faire de la compression GZIP pour diminuer le poids des réponses,
  - Documentation : npmjs.com/package/compression,
- Connect-Mongo : système d'instanciation de documents MongoDB permettant d'enregistrer les sessions des utilisateurs,
  - Documentation : npmjs.com/package/connect-mongo,
- Csurf : bibliothèque permettant de prendre en charge la sécurité contre les attaques CSRF en générant des jetons de session,
  - Documentation :npmjs.com/package/csurf,
- EJS : langage de création de vues dynamiques permettant non seulement d'afficher des variables mais aussi d'insérer d'autres fichiers et d'exécuter du code JavaScript pour rendre les vues plus flexibles,
  - Documentation : npmjs.com/package/ejs,
- Express : cf *Express*,
- Express-Session : système de gestion de session d'utilisateur,
  - Documentation : npmjs.com/package/express-session,

- Helmet : système de sécurité prévenant notamment le prérequêtage DNS, le détournement de clics, les connexions non sécurisées, les attaques XSS réfléchies, etc.,
    - Documentation : npmjs.com/package/helmet,
- Jasmine : outil d'exécution de batteries de contrôles unitaires léger et programmable,
    - Documentation :
        - jasmine.github.io/setup/nodejs.html,
        - npmjs.com/package/jasmine,
- MongoDB et Mongoose : cf. *MongoDB et Mongoose*,
- Morgan : installé par défaut, permet de générer des messages vers la console indiquant les requêtes reçues par le serveur,
    - Documentation : npmjs.com/package/morgan,
- Multer : outil permettant de gérer les formulaires contenant des fichiers (dont l'encodage est différent),
    - Documentation : npmjs.com/package/multer,
- NodeMailer : outil permettant d'envoyer des courriels,
    - Documentation : nodemailer.com/about,
- Unique-string : outil générant des chaines aléatoires, utilisé pour signer le mouchard de session par mesure de sécurité et générer des mots de passe aléatoires,
    - Documentation : npmjs.com/package/unique-string.

Les autres modules que vous trouverez sont des paquets internes à Node.js, étaient préinstallés avec Express ou sont des dépendances des paquets que j'utilise, c'est-à-dire des paquets nécessaires au fonctionnement sans que je les utilise directement.

Documentation :

- nodejs.org/dist/latest-v14.x/docs/api.

Express

Pour construire l'application, j'ai choisi de me reposer sur un outil préexistant. Parmi ceux dont j'ai pu entendre parler, le plus populaire est de très loin Express. J'ai donc supposé qu'il était un bon choix. Express est un cadriciel non opiniâtre et minimaliste pour NodeJS, il permet de faire du routage très aisément sans être contraignant sur la structure du code, laissant au développeur une liberté totale sur ce point.

Documentation :

- expressjs.com/en/4x/api.html,
- npmjs.com/package/express.

C'est aussi cette liberté qu'on pourrait décrier comme étant à la fois une porte ouverte à une architecture de code chaotique mais aussi à l'absence de conventions entre différents projets reposant sur Express. Le cadriciel Sails.js propose de pallier ces deux problèmes mais j'ai découvert son existence tardivement et j'ai donc préféré rester sur mon application existante plutôt que de tenter une migration. Toutefois, une telle migration vers Sails.js pourrait être une bonne chose pour l'application dans l'avenir.

Lorsqu'une requête (formée principalement par une méthode HTTP et une adresse URI (aussi appelée route) selon la définition de l'architecture REST) est reçue par le serveur, Express va la distribuer à une pile de fonctions qui seront exécutées les unes à la suite des autres, pouvant alors modifier l'objet `requête` ou l'objet `réponse` et générer un retour au client ou une erreur. Ces fonctions sont dites des fonctions intermédiaires (*middlewares* en anglais). Express applique ces fonctions dans l'ordre dans lequel on les lui a fourni, pour toutes les requêtes ou certaines seulement (selon la méthode ou la route) ainsi qu'on lui a précisé. C'est sur la base de ce simple système qu'il peut exécuter des fonctions générales (comme définir la sécurité avec Helmet) pour toutes les requêtes comme des contrôleurs spécifiques à chaque requête pour émettre une réponse particulière au client selon l'adresse URI.

Schéma de la distribution d'une requête dans Express : sohamkamani.com/a1d440ab10d27252ebff47ee98075640/express-routing.svg.

Ainsi, tout ce que fait Express est d'enregistrer des fonctions intermédiaires associées à des routes par des méthodes HTTP lors de son initialisation puis de les exécuter à chaque requête reçue y correspondant.

Bulma

Construire une interface graphique cohérente, esthétique et moderne étant souvent difficile, nous avons décidé d'utiliser une bibliothèque CSS. Parmi toutes celles disponibles comme Bootstrap, Materialize, Foundation, etc., j'ai choisi Bulma pour sa simplicité et ses capacités.

Documentation : bulma.io/documentation.

Bulma manquant de certaines fonctionnalités par défaut, j'ai aussi installé des extensions comme BulmaCheckRadio (qui propose une interface améliorée pour les `<input>` de type `checkbox` et `radio`), BulmaRibbon (qui permet d'ajouter des rubans sur des éléments) et BulmaTooltip (qui permet d'ajouter des infobulles sur des éléments).

Documentation :

- wikiki.github.io/form/checkradio,
- github.com/Wikiki/bulma-ribbon,
- wikiki.github.io/elements/tooltip.

FontAwesome

FontAwesome est une bibliothèque gratuite pour ajouter des icônes sur un site web. Elle permet d'ajouter de la clarté sur une interface graphique et est donc bienvenue.

Documentation :

- fontawesome.com/how-to-use/on-the-web/referencing-icons/basic-use.

MongoDB et Mongoose

MongoDB est un système de gestion de bases de données orienté documents. Il est beaucoup plus flexible que SQL car ne repose pas sur le modèle

relationnel et s'affranchit donc de ses contraintes mais a en contrepartie le défaut de laisser au développeur le soin de contrôler la validité de toutes ses actions. Pour minimiser le risque d'erreur et augmenter notre productivité, j'ai choisi d'utiliser l'ODM Mongoose, connu pour être puissant et offrant de nombreux services supplémentaires, comme la création de modèles, l'instanciation de documents, la définition de pseudopropriétés, etc. Mongoose ne permettant pas de faire de récupérer les pseudopropriétés d'un document sans instanciation du modèle par défaut, j'ai aussi installé le greffon Mongoose-Lean-Virtuals pour ajouter cette fonctionnalité très utile.

Documentation :

- docs.mongodb.com/manual,
    - npmjs.com/package/mongodb,
- mongoosejs.com/docs/guide.html,
    - npmjs.com/package/mongoose.

Vue.js

Vue.js est un cadriciel JavaScript permettant de construire des interfaces graphiques dynamiques et des applications monopage. Je m'en suis peu servi mais il m'a permis d'ajouter quelques briques d'interactivité supplémentaires, notamment une barre de recherche avec sélection par le biais du composant `vue-select` et un système AJAX et un calendrier de sélection de dates ou d'intervalles avec le composant `v-date-picker`.

Documentation :

- fr.vuejs.org/v2/guide,
- vue-select.org,
- vcalendar.io.

NodeBB

Un de nos besoins étant de proposer un système de forums sur l'application et en raison de l'absence de services libres répondant à ce besoin pour PHP – excepté le vieillissant phpBB – et du cout temporel de développement d'un tel service, j'ai décidé d'opter pour le système NodeBB, à l'interface moderne et toujours mis à jour. C'est cette dépendance nécessitant l'installation de NodeJS et MongoDB qui m'a amené à opter aussi pour ces outils pour le développement intégral du site. Le seul concurrent sérieux était Discourse, codé en Ruby avec le cadriciel Ruby on Rails. N'aimant pas cet outil et le sachant sur le déclin, j'ai préféré me tourner vers NodeBB.

Documentation : docs.nodebb.org.

La conception

La spécification des besoins et le diagramme de classes ont été décrits dans un autre document. Cf *Conception du projet*.

L’application

L’architecture

L’application contient deux dossiers et trois fichiers. Les fichiers `package.json` et `package-lock.json` permettent à NPM de gérer le projet et ses dépendances, le premier pour définir ces informations et le second pour conserver des informations sur les installations des paquets (version, adresse de téléchargement et clé d’intégrité). Ces paquets sont enregistrés dans le dossier `node_modules`. Le troisième fichier est le fichier `app.js` et le deuxième dossier est le dossier `application`.

Le fichier `app.js`

Le fichier `app.js` est le noyau de l’application, il est lancé au démarrage (commande NPM `npm run start`). Il lance divers modules, charge le système de session, crée les routes, charge les modèles et traductions ainsi que le gestionnaire d’erreurs et initialise le serveur puis distribue les requêtes.

L’application

Le dossier `application` contient toute la logique de l’application. C’est là que sont les contrôleurs, modèles, vues, traductions, routes, etc.

La configuration

Ce fichier `configuration.js` contient un grand nombre d’informations sur l’application. On y définit notamment si l’application est en cours de développement ou non (ce qui changera la manière dont les erreurs s’affichent), mais aussi les noms des dossiers et fichiers de l’application (le but étant de laisser le loisir de nommer ceux-ci à votre convenance), les paramètres de gestion de session (notamment la durée avant expiration), la répartition des langues et modèles (sur quelles branches du site), la gestion des erreurs au cas par cas, la gestion des requêtes de favicon, etc. Le noyau se base énormément sur ce fichier pour définir de nombreuses informations concernant le serveur et l’application.

Le routage

Les routes sont définies dans le fichier `routes.js`. Le format des routes y est décrit en détails et ne sera donc pas reproduit ici. C’est sur ce fichier que se base le noyau pour définir les routes de l’application.

Les constantes

Le fichier `constantes.js` déclare des constantes de l'application. Il permet avant tout d'alléger les contrôleurs et de les centraliser pour éviter la redondance et l'éparpillement. Elles sont aussi consultées par les modèles et les vues.

Les intermédiaires

Le fichier `intermédiaires.js` déclare des fonctions intermédiaires qui seront traitées par le noyau avant la distribution des requêtes aux contrôleurs.

Les droits

Le fichier droits.js décrit les droits associés à chaque statut d'utilisateur par des booléens attribués à différentes propriétés. Il est exclusivement consulté par le contrôleur `droits` et sert à filtrer les accès aux différentes pages et fonctionnalités.

Les étudiants, anciens étudiants et enseignants peuvent modifier leur propre page et le service dont ils ont déclaré faire partie, ainsi que l'organisation et les offres qui s'y rattachent. Les administrateurs peuvent tout modifier. De même, ils peuvent supprimer leur propre page mais seul l'administrateur peut supprimer une fiche de membre, d'organisation, de service ou d'offre, excepté son propre compte.

Les contrôles

Le dossier `contrôles` contient les contrôles unitaires effectués par Jasmine sur les `modèles`.

Les contrôleurs

Le dossier `contrôleurs` contient différents scripts ou sous-dossiers définissant des actions logiques de l'application : ils interprètent la requête, vérifient des droits d'accès, peuvent consulter les modèles et chargent les vues. Ils sont uniquement liés aux routes et sont appelés par Express comme des fonctions intermédiaires lorsqu'une requête correspondant à cette route est reçue. Les contrôleurs sont supposés être la fonction finale de la pile et donc générer une réponse au client, mais ils peuvent aussi générer des erreurs qui seront gérées par le noyau pour afficher un message personnalisé en production.

Les contrôleurs actuellement définis correspondent principalement aux différentes branches du site (*accueil*, *annuaire*, *membres*, *organisations*, *services*, *offres*, et *administration*) mais aussi à des domaines abstraits (*établissement*, *institution*, *formation* et *inscription* pour les quels il n'existe pas de vue) et à des actions logiques (*outils*, *droits* et *ajax*). D'autres peuvent être ajoutés ou étendus aisément en créant un dossier ou un fichier JavaScript comme en modifiant les fichiers déjà existants.

Le contrôleur `outils.js` est particulièrement important car il contient de nombreuses bases de code logique qui est le fruit de l'abstraction de plusieurs

contrôleurs au fonctionnement très similaire. De ce fait, les autres contrôleurs se reposent en grande partie dessus pour générer des réponses.

Les modèles

Le dossier `modèles` contient les différents modèles d'accès aux données. L'application ici n'en contient qu'un, codé avec Mongoose dans le dossier du même nom. Il contient un fichier par collection MongoDB, permettant de définir le schéma des documents de la collection, leurs pseudopropriétés et leurs méthodes. Il y a en outre un modèle adapté à la construction de requêtes MongoDB pour la recherche dans l'annuaire et le catalogue des offres ainsi que la consultation des candidats. La plupart des modèles possède aussi les méthodes `obtenir`, `créer` et `modifier` permettant de consulter, de créer un nouveau document ou d'en modifier un déjà existant en base de données, en faisant toutes les vérifications nécessaires à sa validation. Ce sont ces méthodes qui prennent en outre la plus grande place dans ces fichiers car ils assurent aussi en grande partie la sécurité de l'application en contrôlant les saisies.

Les vues

Le dossier `vues` contient toutes les vues nécessaires à l'application. Elles peuvent être rangées en sous-dossiers si nécessaires comme c'est déjà le cas. La plupart sert à afficher des pages entières et les autres à générer des blocs précis de page (en-tête, pied de page, barre de navigation dans l'annuaire, etc.). L'une d'elles permet spécifiquement de créer une méthode forEach pour la classe Object de JavaScript afin de simplifier le parcours d'objet.

Les traductions

Le dossier `traductions` permet de gérer différentes versions du site. Il est lui-même composé de dossiers dont chacun est une version de langue possible. La seule version actuellement disponible est la version française, dans le dossier `fr`. Chaque dossier de langue peut contenir des fichiers ou sous-dossiers JavaScript définissant des propriétés qui sont ensuite utilisées dans les vues pour maintenir l'indépendance de la structure HTML vis-à-vis du contenu linguistique. Ces fichiers sont très importants car les vues ont été pensées pour que toutes les données textuelles statiques soient lues ici et non codées directement dans les vues ou dans la base de données. C'est d'ailleurs pour celà que de nombreuses informations statiques sont représentées en base de données par des clés alphanumériques qui permettent ensuite d'accéder à l'information réelle dans les fichiers de traductions, c'est notamment le cas des noms de secteurs d'activité, types d'emploi ou domaines de l'informatique.

Les ressources

Le dossier `ressources` contient toutes les ressources directement accessibles par le client. Vous y trouverez les feuilles de style, les scripts, les polices, les

images, etc. Chaque page utilisant Vue.js a son propre script pour maintenir la lisibilité et l'autonomie de ceux-ci.

Je n'ai pas jugé nécessaire d'utiliser des extensions telles que SASS ou TypeScript avec WebPack ou Grunt mais ce pourrait être une possibilité d'évolution pour gagner en productivité et que l'environnement Node.js rend très facile à mettre en œuvre.

Les téléversements

Le dossier `téléversements` contient tous les fichiers déposés par les utilisateurs : photographies, logotypes, CVs et offres d'emploi, mais aussi les deux icônes par défaut des membres et organisations.

Les modules

Le dossier `modules` contient des fonctions utiles aux contrôleurs, notamment une fonction permettant de filtrer une chaine de caractères selon un ensemble de caractères autorisés et une fonction permettant de valider une date. Il peut lui aussi contenir des sous-dossiers et des fichiers JavaScript.

Comment…

Modifier…

Les textes statiques

Pour modifier le contenu textuel statique du site (en dehors des données donc), il faut modifier les chaines de caractères définies dans les fichiers JavaScript du dossier `fr` (ou de toute autre traduction), dans le dossier `traductions`.

Vous pouvez aussi en ajouter de nouvelles et les réutiliser ensuite dans les vues, pour celà il suffit d'ajouter un nouveau fichier JavaScript (possiblement dans un sous-dossier) ou de modifier un fichier déjà existant et d'y enregistrer un texte. Dans les vues, il faudra utiliser l'objet `langue` pour y accéder, dont chaque propriété correspond à une ramification. Ainsi, la chaine `libelléPrénom` du fichier `formulaire.js` s'obtient par la variable `langue.formulaire.libelléPrénom`. Il en va de même pour les sous-dossiers, considérés comme une partie du nom de la variable.

L'affichage des vues

La méthode à adopter pour modifier l'affichage des vues dépend de votre finalité.

Si vous souhaitez modifier l'apparence, notamment en réorganisant l'ordre des éléments, vous pouvez soit modifier les fichiers EJS du dossier `ressources/vues`, soit modifier le CSS du dossier `ressources/styles`.

Si vous souhaitez afficher plus d'informations, vous pouvez soit utiliser les informations déjà existantes dans les documents de la base de données chargés dans la

vue, soit les ajouter. En effet, pour générer chaque vue, le contrôleur fait appel à des méthodes du modèle qui lui retournent des documents de la base de données, aux quels s'ajoutent des références imbriquées et des pseudopropriétés.

Il se peut alors que l'information soit déjà fournie et que vous n'ayez qu'à l'afficher comme que vous soyez obligés de l'ajouter dans le document pour pouvoir l'afficher dans une vue.

Dans ce second cas, tout dépend de la nature de cette information. Si cette information peut être calculée à partir de données existantes, il faudra créer dans le modèle une pseudopropriété au moyen de la méthode `Schema.virtual(<nom>).get(<fonction>)`.

Si cette information est disponible dans un document externe, il faudra alors charger aussi ce document ou l'imbriquer au document chargé soit en l'ajoutant comme une information du modèle (déconseillé car les documents existants en base de données ne l'auront pas, cf. *Ajouter… une information*), soit en créer une pseudopropriété référençant un document externe avec `Schema.virtual(<nom>, <référence>)` suivi d'un appel à `Query.populate(<propriété>)` pour que cette pseudopropriété soit remplacée par le document ou la liste des documents correspondants dans la réponse retournée par le modèle. Vous pourrez trouver des exemples de tels codes dans le modèle `mongoose/organisation.js`.

Si au contraire cette information est toute nouvelle, cf. *Ajouter… une information*.

Les droits des utilisateurs

Les droits des utilisateurs sont déclarés dans le fichier `droits.js` et vérifiés par le contrôleur du même nom. Chaque droit est défini comme une propriété dont la valeur est booléenne. Changer ce booléen suffira. Si vous souhaitez ajouter de nouveaux droits, vous pouvez aussi ajouter une nouvelle propriété dans le fichier `droits.js` pour chaque statut et y associer une méthode dans le contrôleurs `droits` ou modifier la méthode vérifier pour retourner directement la valeur de cette propriété.

Les routes de l'application

Vous pouvez modifier les routes dans le fichier `routes.js`, par exemple pour ajouter de nouvelles routes correspondant à de nouvelles fonctionnalités, mais aussi pour en renommer, en déplacer ou en supprimer. Pensez à vérifier que les vues existantes s'adaptent en conséquence, pour éviter que les liens ne correspondent plus aux nouvelles routes.

Les informations de la base de données

Chaque membre, organisation, service et offre a sa propre page que certains membres peuvent modifier selon leurs droits. Pour les autres informations comme les secteurs d'activité, les formations, les établissements ou les institutions, j'ai considéré que l'utilité de l'existence d'une page dédiée était faible et donc j'ai supposé que s'il fallait les modifier, l'administrateur pourrait le faire directement dans la base de données.

Ajouter…

Une nouvelle traduction

Pour ajouter une nouvelle traduction, il faut d'abord copier une traduction existante et traduisez les chaines **sans en modifier les libellés**. Ensuite, il vous suffit d'associer cette traduction à une section du site dans le fichier `configuration.js`. Enfin, il vous est possible de créer des nouvelles routes pour celà, par exemple en ajoutant dans le fichier `routes.js` des routes pour le sous-domaine `/<langue>`.

Une information (ou en supprimer une)

Pour ajouter une information dans l'application, par exemple le deuxième prénom d'un membre, vous devez :

- ajouter cette information dans le schéma du modèle `modèles/mongoose` ou sous la forme d'une pseudopropriété qui peut être construite à partir des propriétés du document ou être une référence vers un ou plusieurs documents externes,
- modifier dans le cas d'une propriété les documents existants dans la base de données, le mieux étant de créer cette propriété et de l'initialiser à `null`,
- ajouter, si cette information est à saisir par l'utilisateur, un champ de formulaire lors de l'inscription ou de la modification d'un objet,
    - il faudra alors ajouter une étape de vérification avec un éventuel rejet si elle ne respecte pas certaines conditions que vous pouvez définir dans la méthode du modèle associé,
- adapter en conséquence les vues pour que cette information soit affichée.
- enfin, ajouter de nouveaux contrôles unitaires.

Pour la supprimer, faites le même chemin en sens inverse.

Si vous souhaitez ajouter une nouvelle information parmi celles statiques comme les secteurs d'activité, les types d'emploi ou les domaines de l'informatique, vous devez l'ajouter dans le dossier `traductions/<langue>/listes` sous le nom de fichier `<catégorie>.js` en lui donnant une clé et une valeur **mais aussi en déclarant cette clé dans le fichier `constantes.js`**, sans quoi elle sera inutilisable car c'est sur les clés déclarées que se reposent les mécanismes de vérification dans les modèles (ils ne doivent pas avoir accès aux traductions).

Une fonctionnalité (ou en modifier une)

Pour ajouter une fonctionnalité, il y a cinq éléments à envisager :

- créer une route correspondant à une requête associé à cette action,

- créer un contrôleur réalisant cette action,
- créer une vue correspondant à cette page,
- créer un modèle ou une méthode sur un modèle préexistant pour manipuler des données,
- créer une collection ou ajouter une propriété sur les documents d'une collection existante.

Toutes ces étapes ne sont pas toujours nécessaires mais sont souvent requises.

Par exemple, pour ajouter une fonction d'envoi de courriel à tous les membres du site pour les informer d'un évènement à l'IUT, vous pouvez envisager de créer la route `/administration/informer` dans le fichier `routes.js`, puis y brancher un ou deux contrôleurs associés aux méthodes HTTP `GET` et `POST`, lesquels pourront alors réaliser la logique de l'action voulue en récupérant des informations du formulaire de la vue et en envoyant des courriels aux membres, dont vous pourrez lire l'adresse mél grâce à une méthode sur le modèle `membre.js`.

Vous pouvez aussi modifier une fonctionnalité existante en modifiant le contrôleur associé.

Note : Pensez à maintenir la qualité du code ! J'ai mis un point d'honneur à pratiquer autant que possible une séparation claire entre les différentes structures logiques de l'application, et il serait dommageable que vous laissiez des textes directement dans les vues, que vous laissiez des constantes en clair dans les contrôleurs ou que vous manipuliez les données hors des modèles.

Changer…

De SGBD

Pour changer le SGBD, il convient de :

- installer un autre SBGD, et probablement un paquet NPM permettant de s'y connecter,
- importer le schéma et les données existantes vers ce SBGD,
- remplacer le modèle par un modèle équivalent, c'est-à-dire fournissant la même structure d'information afin de garantir la pérennité des contrôleurs et des vues ou de modifier ces dernières en conséquence.

De moteur de génération de vues

Pour changer de moteur de génération de vues, il convient de :

- installer le paquet NPM associé à cet autre moteur,
- modifier le fichier `configuration.js` pour définir la propriété `moteurRendu` au nom de ce moteur (le nom du paquet NPM),
- convertir les vues existantes vers ce moteur.

Mettre…

À jour les dépendances

Pour les dépendances NPM, il suffit d'exécuter la commande `npm update`.

Vue.js et ses dépendances sont chargées depuis un CDN donc sont automatiquement mises à jour.

Pour les autres dépendances, elles sont à mettre à jour au cas par cas manuellement. Bulma et ses dépendances peuvent être mises à jour manuellement car il est aisé de trouver les fichiers sources et que leurs mises à jour sont assez rares (il n'y en a eu aucune pour les extensions depuis un an et demi). Bulma est mise à jour plus fréquemment mais ses mises à jour restent mineures.

Dépôts Github des dépendances :

- Bulma : github.com/jgthms/bulma,
  - Ici, le fichier est `css/bulma.min.css`,
- Extensions : github.com/Wikiki/bulma-extensions,
  - Ici, les fichiers sont dans le dossier `dist/css/` sous le nom générique `bulma-<extension>.min.css`.

Pour FontAwesome, il faut procéder ainsi que pour Bulma. Seuls les dossiers `/css` et `/webfonts` sont à conserver car je ne me sers pas des autres. Les mises à jour sont toutefois très rares (4 ce premier semestre 2020).

Adresse de téléchargement de la dépendance :

- fontawesome.com/how-to-use/on-the-web/setup/hosting-font-awesome-yourself.

Sources

En plus des sources citées ci-avant, voici quelques autres sources qui m'ont aidé et m'ont permis de répondre à certaines de mes interrogations et de résoudre certains de mes problèmes :

- Mozilla Developer Network,
  - developer.mozilla.org/fr/docs/Learn/Server-side/Express_Nodejs,
- Express,
  - http://expressjs.com/en/guide/writing-middleware.html,
  - https://expressjs.com/en/advanced/best-practice-performance.html,
- StackOverflow,
- Github,
  - Pages *Issues*,
- Youtube,
  - Grafikart,
- zellwk.com,
  - zellwk.com/blog/async-await,
  - smashingmagazine.com/2018/01/understanding-using-rest-api,

    - zellwk.com/blog/js-promises,
    - zellwk.com/blog/local-mongodb,
    - zellwk.com/blog/mongoose,
        - zellwk.com/blog/mongoose-population,
        - zellwk.com/blog/mongoose-subdocuments,
    - zellwk.com/blog/crud-express-mongodb,
- M. Quiniou.