Web App’Iculture

IUT Du Limousin, département informatique

Module Framework Javascript

Guillaume MAUREL,

Mathieu ETCHEVERRY,

24 Mars 2019

Table des matières :

Introduction

Dans le cadre du module framework Javascript L250, nous avons réalisé une application web visant à aider les apiculteurs dans leur travail. Cette application, réalisé grâce au framework VueJS, permet entre autres de gérer les ruches et les ruchers de ces apiculteurs. Afin de réaliser cette dernière, deux parties sont disponibles pour un utilisateur. La première est le Back-Office : il s'agit en fait de la partie dans laquelle l'utilisateur va administrer et saisir toutes les données. L'autre partie est le Front-Office, destiné à visualiser les données de l'applications selon un certain format.

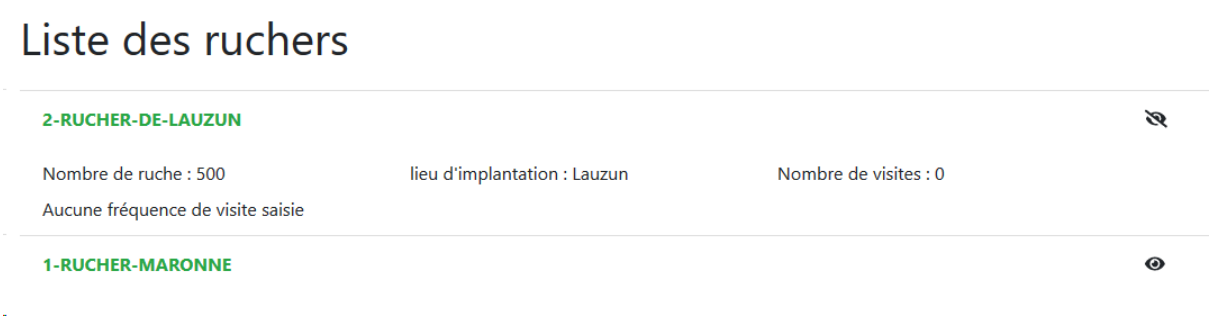
Ainsi, nous verrons dans un premier temps comment fonctionne notre application, puis nous nous attarderons sur l'implémentation de cette dernière. Enfin, nous évoquerons les problèmes que nous avons rencontrés.

I. Fonctionnement de l'application

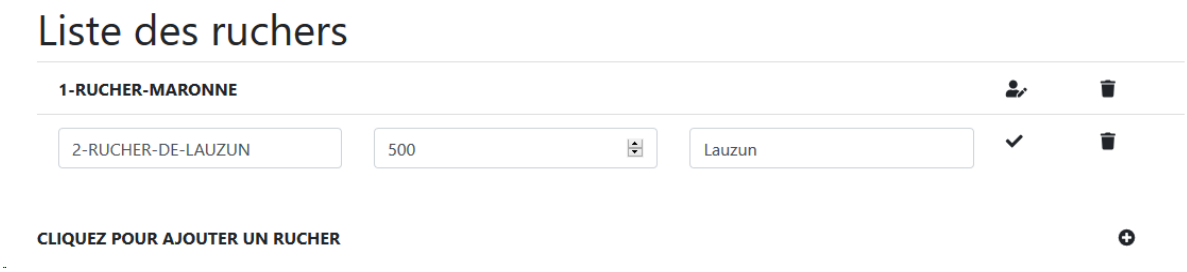
Arborescence de l'application :

Afin de construire notre application, nous avons créé trois pages, que l'on peut regrouper en deux parties : le back-office, correspondant à l'endroit où les données de l'application sont gérées par un utilisateur, et le front-office qui lui correspond à l'endroit où un utilisateur se rend pour visualiser toutes les données. Voici la liste de ces trois pages :

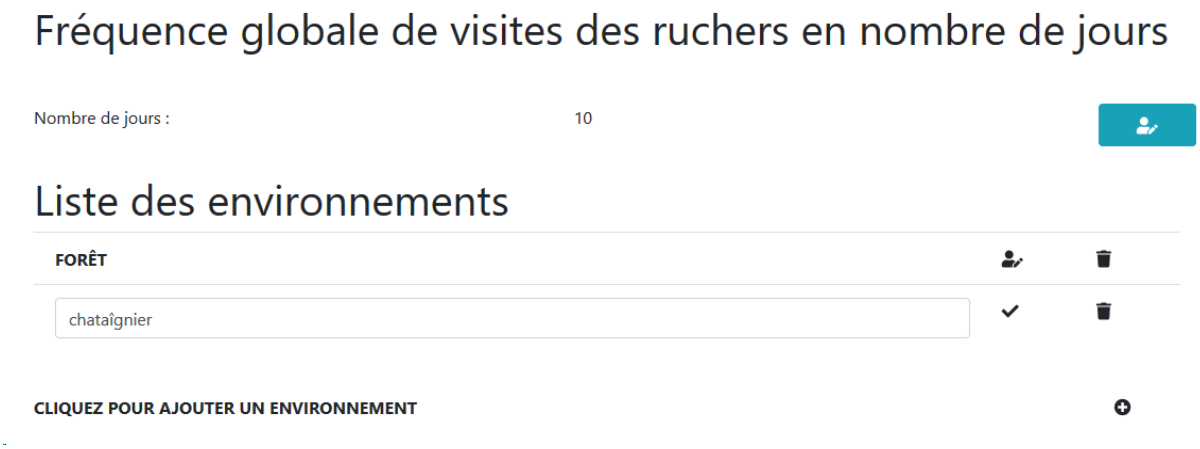
La page "liste des ruchers" : cette page, appartenant au front-office, permet à un utilisateur de visualiser la liste des ruchers sous formes de liste. Lorsque l'on arrive sur cette page, on peut voir une liste correspondant aux identifiants des ruchers saisies dans le back-office. Ces ruchers sont triés selon le nombre de jours écoulés depuis la dernière visite. Cependant, si ces derniers n'ont pas été visité, les ruchers sont triés selon le nombre de jours écoulés depuis la date de la création du rucher. A chaque rucher est associé une couleur : rouge si le nombre de jours entre la dernière visite et aujourd'hui est supérieur à la fréquence définie par l'utilisateur (fréquence par défaut si elle n'existe pas), orange s'il est inférieur à la fréquence mais supérieur à la moitié de la fréquence et verte s'il ne correspond pas à l'un des cas de figure précédent. L'utilisateur a ensuite la possibilité de voir plus en détail les caractéristiques d'un rucher. Pour cela, il doit simplement cliquer sur une icône en forme d'œil. Il a bien sûr la possibilité de cacher ses informations.



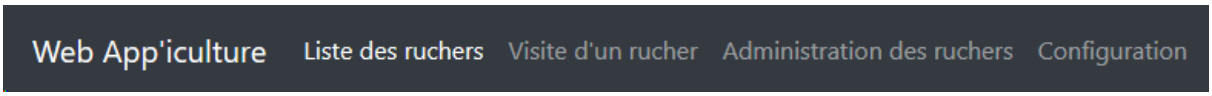
La page “Administration des ruchers” : il s’agit d’une page qui appartient au Back-office. Cette page permet à un utilisateur de saisir, d’éditer, de supprimer et de visualiser des ruchers. Pour l’ajout d’un rucher, un bouton est mis à disposition à l’utilisateur. Lorsqu’il clique dessus, un formulaire apparaît pour saisir les informations du rucher. Il peut alors valider la création du formulaire ou bien annuler cette dernière. Une liste est également présente au-dessus de ce bouton, avec un bouton pour éditer un rucher et un bouton pour le supprimer. Si l’utilisateur souhaite éditer un rucher, un formulaire apparaît à l’emplacement de ce dernier lui permettant de saisir les informations.



La page “Configuration”. Cette page, appartenant au back-office, permet d’initialiser les paramètres par défaut de l’application, à savoir le nombre de jours qui correspond à la fréquence globale des visites de ruchers, et les environnements. Pour réaliser cela, un outil permet à l’utilisateur de saisir le nombre de jours avec un petit formulaire. Il peut à tout moment éditer ce nombre de jours, en cliquant sur un bouton d’édition, ce qui permet de recréer le formulaire avec une zone de saisie. Cette valeur prend alors comme valeur par défaut la précédente valeur. Au niveau de la partie environnement, le principe est exactement le même que celui de la liste des ruchers, seuls les champs à saisir varient.



Au final, on se retrouve avec une arborescence comme cela :



Hébergement

Afin d'héberger notre application, nous utilisons l'outil Github Pages. Il s'agit d'un outil proposé par Github permettant de mettre en ligne un site directement depuis un projet Github. Pour cela, il suffit de créer un repository portant le nom de d'utilisateur suivi de ".github.io". Suite à cela, le contenu de repository github est directement déployé et est accessible sous l'URL https://nom-utilisateur.github.io . Notre application est donc accessible à l'adresse : https://maurel11.github.io/ .

Gestion des données

Afin d'assurer la gestion des données dans notre application, nous avons choisi d'utiliser le LocalStorage proposé par Javascript. En effet, ce dernier nous permet de stocker nos données directement dans le navigateur et les conservent même après la fermeture. Il est ainsi possible de sauvegarder des données dans le LocalStorage ou encore de les récupérer grâce à des fonctions.

II. Implémentation

Liste des ruchers :

Afin de construire la page “Liste des ruchers”, nous utilisons un seul composant, intitulé “list”. Ce composant permet l’affichage de l’ensemble des données que l’on choisit et propose plusieurs options afin d’être un maximum réutilisable. La première option est le fait d’activer ou non la corbeille. Si l’on choisit d’activer la corbeille, alors l’utilisateur pourra supprimer un objet directement depuis la liste avec une icône corbeille. La seconde est l’option d’édition : elle permet d’éditer directement un objet en cliquant sur un bouton d’édition ce qui fait apparaître un formulaire où les données saisies, une fois qu’elles sont validées, sont sauvegarder dans le localstorage. La dernière option est la possibilité de voir en détail les informations des données. En cliquant sur un oeil, le détail d’un objet apparaît. Dans notre cas, les deux premières options ne doivent pas être mise en place (la page appartenant au front-office, l’utilisateur ne doit pas modifier les données dans cet espace). Cependant, la troisième est importante. Pour récupérer la liste des objets, on initialise les données de l’application au travers d’un objet “items” chargé à la création de l’application. Cet objet se charge en utilisant le LocalStorage : on récupère pour la page “liste des ruchers” les données relatives aux ruchers présentes dans ce storage. Les données sont ensuite triées par ordre décroissant de date de visite ou date de création. Une couleur est attribuée à chacun des ruchers trouvés après récupération des données selon les conditions citées en première partie. A partir de l’objet “items” récupéré, on affiche une liste. Afin d’accéder au LocalStorage de l’application, on utilise un objet nommé “storage” qui fournit une méthode de récupération d’un objet du storage et une méthode de sauvegarde du storage. Un objet “store” est ensuite chargé de faire le lien entre les données VueJS (le plus souvent, l’objet “items”) et les données du storage. Afin de visualiser le détail d’un objet, une propriété existe sur chaque rucher. Cette dernière, intitulée “show”, est passé à vrai lorsque l’on clique sur le bouton de visualisation. Ainsi, c’est grâce à cette propriété que l’affichage des détails d’un rucher est possible : on se base sur la valeur de “show” pour changer d’affichage.

Pour gérer la couleur des ruchers, on appelle une fonction intitulée “selectColorForItem”. Cette fonction va d’abord regarder si un rucher a déjà été visité ou non. Si c’est le cas, alors on trie le tableau des visites selon leurs dates par ordre décroissant, on récupère ensuite la première visite de ce tableau, et on la compare à la date d’aujourd’hui. La couleur est ensuite assignée selon les conditions citées en première partie.

Administration des ruchers :

Afin de construire la page “administration des ruchers”, on fait appel au composant “list” ainsi qu’au composant “addRucher”. Le composant “list”, déjà cité précédemment, permet d’afficher la liste des ruchers chargés depuis le storage. On choisit de pouvoir éditer et supprimer un objet. Ainsi, on active l’option d’édition et l’option de suppression. Par contre, l’option de visualisation ne sert à rien, elle n’est donc pas activée. Tout comme la page précédente, on récupère les données du LocalStorage grâce à l’objet storage et à ces fonctions, et on les relie ces données à VueJS grâce à l’objet store. Pour le composant addRucher, son fonctionnement est simple. Le but de ce composant est d’ajouter des ruchers directement dans les données de l’application et dans celles du localstorage. Pour cela, une propriété “newEnabled” permet de passer du mode ou l’on voit un bouton indiquant qu’on peut ajouter un rucher au formulaire. Cette propriété est changée au clique sur le bouton précédemment cité et à la validation du formulaire. Lorsque l’utilisateur valide le formulaire, l’objet saisi est enregistré dans les données de l’application, mais également dans le LocalStorage grâce à la fonction “addRucher” de l’objet Store. Ainsi, on voit que le rucher saisi est dynamiquement inséré dans la liste des ruchers. À tout moment, un utilisateur peut annuler la saisi d’un nouveau rucher, grâce au bouton d’annulation.

Configuration :

Afin de construire la page “configuration”, on fait appel au composant “list” précédemment cité avec la même configuration en l’adaptant aux environnements. Il fait également appel au composant “addEnvironnement”, qui reprend également le composant “addRucher” en l’adaptant aux environnements.

Enfin, afin de gérer la fréquence par défaut, on utilise le composant “frequencyTemplate”. Ce composant permet de gérer la fréquence par défaut de visite des ruchers en nombre de jours.

III. Difficultées rencontrées

Pour la liste des difficultés, ils ont surtout concerné l’onglet des visites d’un rucher. Un problème venait de la récupération des informations liés à un rucher. Pour pallier cela, nous avons choisi de mettre les informations destinées à l’onglet de visite d’un rucher dans la page listant les ruchers, à l’aide d’un attribut ‘’collapse’’, ainsi il était prévu d’insérer au sein de chaque rucher un élément faisant apparaître les informations supplémentaires que contenait un rucher. Parmi ces éléments se trouve l’historique de visite, qui est un tableau. Dans l’interface prévu, une date identifierait chaque visite. De la même manière que son parent, un attribut collapse aurait permis de développer les données de chaque visite. Dans un souci d’optimisation, nous voulions faire en sorte d’utiliser le moins possible de composants différents. Ainsi le système de liste, d’édition, d’ajout et de suppression pouvaient aussi servir à la gestion des listes.

Malheureusement, des problèmes liés à l’imbrication des composants à générer beaucoup d’erreurs et rendait l’ensemble bien trop fragile, où certains changements généraient des erreurs en chaînes. Nous avons donc décidé de ne pas implémenter du code à problèmes.

En conclusion de ce projet, notre principal problème a été le choix de notre architecture, qui nous a permis de bien avancer dans ce projet mais qui a fini par nous handicaper à un certain moment de l’application.