Vincent Gagné

Proposition

Mon application s’agira d’un chat en ligne qui supportera ces différents modes de clavardage :

* Un clavardage 1 à 1
* Un clavardage à plusieurs. (il sera possible de rejoindre une room préconcue)

En gros, une application Elm possède 3 concepts de base qui interagissent ensemble.

1. Le “**state**” ou **modèle**

C’est un objet JSON qui représente l'entièreté de l’interface. Le modèle contient toutes les informations dynamiques que contiendra l’application. Dans le cas d’une application avec une liste de produit à vendre, par exemple, le modèle initial pourrait ressembler à ceci:



À noter que dans une application réelle avec du contenu dynamique, le modèle initial serait probablement vide. Il se remplirait avec des données d’une base de donnée ou autre source externe et pourrait devenir très gros. Le modèle en temps réel de l’application n’est pas accessible de l’extérieur et existe seulement durant un rendu de l’application. Ensuite il disparait pour toujours. Des nouveaux modèles sont générés pour faire évoluer l’interface.

1. Ensuite, il y a la **vue**.

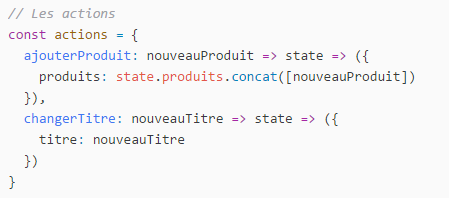
La vue est un ensemble de fonctions “pures” qui retournent du HTML, prenant en paramètre des parties du modèle. Une vue qui affiche le modèle ci-haut pourrait ressembler à ceci:



Cette vue construit une page HTML dynamique en fonction du modèle. Ceci veut dire que, si on veut changer une donnée d’affichage, on a juste à changer sa valeur dans le modèle. Pour ajouter un produit dans la liste HTML par exemple, on a juste à ajouter un élément dans le tableau “produits” du modèle.

1. Finalement, il y a les **actions**.

Étant donné qu’on fait de la programmation fonctionnelle, toutes les variables sont constantes. On ne veut pas qu’elles soient modifiables non plus, car ces variables ne font que décrire l’application initiale et ses composantes. Elles ne représentent pas l’application en temps réel. Alors pour faire évoluer l’application, on utilise un objet d’actions. Les actions sont des fonctions qui ne font que retourner des parties du modèle à mettre à jour. Une action qui mettrait à jour la propriété “titre” du modèle doit simplement retourner un morceau du modèle avec sa nouvelle valeur. Avec une deuxième action de démonstration qui ajoute des produits dans la liste, l’objet d’actions complet ressemblerait à ceci:



La Syntaxe ES6 ici peut être mélangeante, mais en gros, on a des fonctions qui fusionnent des micro objets avec le modèle, le mettant à jour.

Au final, il ne reste plus qu’à démarrer l’application comme ceci:

La magie se trouve ici. HyperApp démarre le premier rendu (appelle notre vue avec le modèle initial) puis commence à automatiquement démarrer un rendu à chaque fois que le modèle est mis à jour avec une action.

Pour garder du beau code propre et rester en programmation fonctionnelle, il y a quelques règles à respecter:

1. Seulement créer des fonctions pures. Ce sont des fonctions qui ont seulement accès à leurs paramètres et retournent des valeurs qui en dépendent. Aucun effet secondaire. Pas le droit d’utiliser des variables externes et pas le droit d’en modifier ou d’en créer. Dans le cadre de HyperApp, il n’y en a aucune de toute façon.
2. Étonnamment, pas le droit d’utiliser l’opérateur “=” dans nos fonctions. L’opérateur signifie qu’une nouvelle variable a été créée ou qu’on a modifié un paramètre, ce qui n’est pas une bonne pratique. On ne fait que retourner des objets et modifier leurs propriétés.
3. Pas de Math.random ou toute autre fonction qui peut créer une valeur non prédictible.
4. Ne jamais interagir avec le DOM, donc pas de “querySelector” ou “getElementById”, en gros, on travail seulement avec nos paramètres.

Au final, la seule chose qui agit avec le DOM est HyperApp. Si on est curieux de savoir ce qui se passe derrière tout ceci, HyperApp est seulement 368 lignes de JavaScript de long, alors c’est assez rapide de comprendre l’ensemble de la librairie et si on le fait, plus aucun secret ou détail ne peut nous surprendre.

Voici quelques exemples d’applications HyperApp :

* Un To-Do list : <https://loteoo.github.io/hyperapp-todolist/>
* Un compteur : <https://codepen.io/jorgebucaran/pen/zNxZLP?editors=0010>