Comparaison elm/JS dans le cadre du projet ELP

Nous allons ici comparer les langages de programmation elm et Javascript dans le cadre du projet "TC Turtle".

A première vu, le choix d'utiliser elm peut paraître étrange. En effet, une fois le code écrit, il faut le compiler pour transformer le code elm en page HTML/CSS/JS. Mais alors pourquoi utiliser elm si on finit pas obtenir du code javascript ?

Logique de programmation

Elm est un langage purement fonctionnel. Il n'y a donc aucun effet de bord : une même entrée donnera toujours le même résultat. De plus, on utilise de la programmation déclarative : on définit directement ce que l'on souhaite obtenir, tandis que dans les langages de programmation impératifs comme javascript, nous définissons la manière d'arriver à un état. C'est une gymnastique mentale qui est un peu compliquée à comprendre au départ mais qui est très efficace une fois assimilée. Ainsi, il est très rapide de générer des applications web monopage avec Elm. De plus, l'approche fonctionnelle nous permet de réaliser des tâches complexes plus simplement.

Compilation

Le compilateur elm est très puissant. Puisque elm est un langage purement fonctionnel, il n'y a pas d'effets de bord, le compilateur peut donc permettre de détecter un très grand nombre d'erreurs avant l'exécution. De plus, le compilateur d'elm est très "pédagogique". Pour chaque erreur, si elle est simple, il tente d'apporter un correctif (typo/nom proche). Pour les erreurs plus complexes, il redirige vers la page correspondante de la documentation d'elm. En javascript, il n'y a pas de compilateur, la gestion des erreurs se fait donc directement dans la console du navigateur, cette console est beaucoup moins explicite que le compilateur.

Langage avec types statique/dynamique

Dans un langage avec des types statiques comme elm, il est nécessaire de définir explicitement le type de chaque variable. Ces types ne seront pas modifiables par la suite. Dans Javascript, nous n'avons pas cette limitation, les variables n'ont pas de types prédéfinis. Par exemple, nous pouvons initialiser une variable comme un nombre au début du programme et stocker ensuite une chaine de cararctère ou tout autre objet dans cette même variable.

Bien que le typage statique paraisse redondant et lourd à écrire, il permet de supprimer beaucoup de comportements imprévus, notamment lors de l'appel de fonction : si on appelle une fonction en n'utilisant pas le bon type en elm, nous avons une erreur lors de la compilation, tandis qu'en JS nous aurons un comportement incertain lors de l'exécution ou une erreur en production.

Gestion du DOM

En javascript, quand on souhaite modifier la page, on accède au DOM (Document Object Model) directement. Lors de cet accès, on risque d'introduire des erreurs de syntaxe et des comportements indésirés. En Elm, on ne travaille pas directement sur le DOM, on travaille avec une couche d'abstraction en plus qui va générer automatiquement le DOM une fois nos modifications effectuées.

Conclusion

Dans le cadre de ce projet, utiliser elm simplifie la programmation : ce langage n'utilise pas de callback, nous n'avons donc pas d'erreurs liées à la programmation asynchrone. Aussi, cela nous permet de découvrir un autre paradigme de programmation qui a le vent en poupe : les outils de développement déclaratif (Docker, Ansible, Nix) et les langages de programmation fonctionnel (Haskell, Elixir) semblent être de plus en plus utilisés.