# Exercice de recrutement "Développement frontend Web Components / design system

## Préambule

Cet exercice comporte deux parties.

Dans la première partie, vous serez amené à coder un petit projet Web et le déployer sur Clever Cloud.

Cela nous permettra d'évaluer votre capacité à :

- Utiliser HTML, CSS et JavaScript
- Utiliser Git
- Concevoir des composants simples
- Penser une interface utilisateur

Dans la deuxième partie, vous serez amené à critiquer l'interface et l'expérience utilisateur de la console d'administration de Clever Cloud. En effet, en déployant votre projet, vous aurez l'occasion d'utiliser cette console et vous faire un avis.

Cela nous permettra d'évaluer votre capacité à :

- Analyser une interface utilisateur
- Identifier les points forts et les points faibles
- Apporter des idées d'amélioration

## Partie 1 : le projet

Le but de ce petit projet est de créer une sorte de simulateur de prix pour des applications déployées chez Clever Cloud.

Dans cette vidéo de démo, vous retrouverez une "version zéro" très simpliste et moche du simulateur.

https://www.youtube.com/watch?v=P-E0gJHZG7k

### Initialisation du projet

Vous pouvez utiliser l'outil (bibliothèque/framework) que vous voulez à condition qu'il propose un modèle de composants. En gros, vous pouvez utiliser : des Web Components "vanilla", Lit, Stencil, Svelte, Vue, Preact/React, Angular... Bien que notre bibliothèque de composants utilise Lit, nous ne cherchons pas des personnes expertes de cette technologie. Nous nous intéressons principalement à comment vous isolez vos composants et comment ils communiquent entre eux.

Si l'outil que vous choisissez amène d'autres outils avec lui (Webpack, babel, Vite...), c'est OK, mais ça n'est pas obligatoire du tout. Vous pouvez très bien utiliser une bibliothèque comme Lit directement depuis un CDN.

### Expérience utilisateur

Voici une description du fonctionnement de ce simulateur. N'hésitez pas à regarder la vidéo de démo avant et après avoir lu cette section pour vous faire une idée.

- Au chargement de la page, il faut récupérer les données depuis notre API (voir détails dans la section API et données).
- Dans la colonne de gauche, il faut lister toutes les variants disponibles.
    - Les variants sont un peu les types d'applications supportés par Clever Cloud (Java, Ruby, Python...).
    - Pour chaque variant de la liste, il faut afficher au moins le logo, le nom et un bouton de sélection.
    - Vous pouvez ajouter d'autres informations si cela vous semble pertinent (en vous servant de ce que vous a retourné l'API).
    - Quand l'utilisateur clique sur le bouton de sélection d'un variant, il faut mettre à jour la colonne du milieu avec les flavors de le variant sélectionné.
- Dans la colonne du milieu, il faut lister les flavors (nano, XS, M, 2XL...) de le variant sélectionnée.
    - Les flavors sont les tailles de machine virtuelles proposées par Clever Cloud (nano, XS, M, 2XL...).
    - Pour chaque flavor de la liste, il faut au moins le nom, le nombre de CPU, le nombre de GPU, la RAM et un bouton de sélection.
    - Vous pouvez ajouter d'autres informations si cela vous semble pertinent (en vous servant de ce que vous a retourné l'API).
    - Quand l'utilisateur clique sur le bouton de sélection d'une flavor, il faut ajouter une entrée dans la colonne de droite.
- Dans la colonne de droite, il faut lister les choix de l'utilisateur, c'est une sorte de panier.
    - Pour chaque entrée du panier, il faut au moins le nom de le variant, le nom de la flavor, le prix de la flavor et un bouton de suppression.
    - Vous pouvez ajouter d'autres informations si cela vous semble pertinent (en vous servant de ce que vous a retourné l'API).
    - Quand l'utilisateur clique sur un bouton de suppression, il faut enlever l'entrée correspondante du panier.
- Dans l'en-tête, on retrouve le prix total du panier.
    - À chaque mise à jour du panier (ajout et suppression), il faut mettre à jour le total présent dans l'en-tête.
- L'ensemble de cette interface doit être en anglais.

### Structure principale (layout)

Voici les contraintes que nous imposons pour la structure de la page :

- La structure de le la page doit s'adapter à l'écran, mais **SANS** utiliser de media queries.
    - On testera votre page sur un écran minimum de 700px de large par 400px de haut et on agrandira jusqu'à 1920px de large par 1080px de haut.
- La structure de la page comporte un en-tête en haut qui fait toute la largeur.
- Les 3 colonnes en dessous prennent chacune un tiers de la largeur et le reste de la hauteur disponible.
- Les 3 colonnes peuvent avoir des barres de défilement quand c'est nécessaire, mais uniquement en vertical (pas de barre horizontale).
- Seules ces 3 colonnes peuvent avoir des barres de défilement.
- Vous devez utiliser les grilles CSS pour cette structure.

N'hésitez pas à regarder la vidéo de démo pour vous faire une idée.

Comme nous l'avons déjà évoqué, la vidéo de démo présente une "version zéro" très simpliste et moche du simulateur. Elle respecte les contraintes imposées concernant la structure, mais on est loin d'une interface claire et lisible pour les utilisateurs.

À vous de faire les choix de style et de design concernant :

- Les polices, la taille, la graisse, l'italique
- Les couleurs de texte et de fond
- Les espacements (marges et paddings)
- Les bordures et les ombres
- L'alignement des blocs ou du texte
- Le wrapping des blocs ou du texte
- L'affichage d'informations supplémentaires non imposées (comme précisé dans la section *Expérience utilisateur du simulateur*)
- L'affichage des nombres et des devises
- L'utilisation d'icônes ou de logos

Libre à vous de faire quelque chose d'un peu original si vous en avez envie et si cela n'impacte pas l'utilisabilité et l'accessibilité de l'interface.

## Contraintes techniques

Voici les contraintes techniques que nous imposons :

- HTML
  - Pas de contraintes particulières.
- CSS
  - Pas de préprocesseur type SaaS ou Less, uniquement du CSS standard.
  - Pas de Bootstrap, pas de Tailwind, uniquement du code CSS à vous.
  - Pas de media queries, que du design fluide.
  - Utiliser au moins une fois des grilles pour la structure de la page.
  - Utiliser au moins une fois des flexbox pour les composants.
- JavaScript
  - N'importe quelle syntaxe JavaScript standard est autorisée.
  - Si vous êtes poussé à faire du TypeScript par votre framework (comme Angular), c'est OK, mais ça n'est pas obligatoire du tout.
  - Si vous pouvez utiliser des modules ESM (au lieu de CommonJS), c'est mieux.
  - Si vous pensez que c'est utile, vous pouvez ajouter des dépendances JavaScript.
  - Il faudra justifier ces choix.
- Support navigateurs
  - Votre code doit fonctionner sur la dernière version de Chrome.
  - Votre code doit fonctionner sur la dernière version de Firefox.
  - Pour Safari, nous tenons compte du fait qu'il est difficile de tester sans posséder de Mac.
- Git
  - Nous allons lire vos commits, essayez d'en tirer parti.

## Découpage en composants

Il y a 3 composants obligatoires à créer :

- Un composant pour afficher un variant (présent dans la liste de gauche)
- Un composant pour afficher une flavor (présent dans la liste du milieu)
- Un composant pour afficher une entrée du panier (présent dans la liste de droite)

Libre à vous de créer d'autres composants si cela vous semble nécessaire.

## API et données

Un seul appel d'API est nécessaire :

```
GET https://api.clever-cloud.com/v2/products/instances
```

Vous pouvez retrouver une documentation de ce endpoint d'API ici :

https://www.clever-cloud.com/doc/openapi/#get-/products/instances

Comme vous pouvez le constater, cette documentation est très sommaire.

Ce endpoint retourne un tableau d'instances avec une propriété `variant`. C'est cet objet qui nous intéresse pour la liste des variants de la colonne de gauche et notamment les propriétés `name` et `logo`.

Pour chaque instance, vous trouverez également une propriété `flavors`. C'est ce tableau d'objets qui nous intéresse pour la liste des flavors de la colonne du milieu et notamment les propriétés `name`, `cpus`, et `gpus`.

Concernant la RAM d'une flavor, vous pouvez utiliser au choix :

- `mem` qui retourne une valeur en megabytes
- `memory.value` qui retourne une valeur en bytes
- `memory.formatted` qui retourne une valeur en string formattée en MiB

Concernant le prix, il faut utiliser la propriété `price` et multiplier par `41.904` pour avoir le prix en euros (c'est legacy).